Bicategories

Eugene W. Stark

Department of Computer Science
Stony Brook University
Stony Brook, New York 11794 USA

March 17, 2025

Abstract

Taking as a starting point the author's previous work ([13] [14]) on developing aspects of category theory in Isabelle/HOL, this article gives a compatible formalization of the notion of "bicategory" and develops a framework within which formal proofs of facts about bicategories can be given. The framework includes a number of basic results, including the Coherence Theorem, the Strictness Theorem, pseudofunctors and biequivalence, and facts about internal equivalences and adjunctions in a bicategory. As a driving application and demonstration of the utility of the framework, it is used to give a formal proof of a theorem, due to Carboni, Kasangian, and Street [4], that characterizes up to biequivalence the bicategories of spans in a category with pullbacks. The formalization effort necessitated the filling-in of many details that were not evident from the brief presentation in the original paper, as well as identifying a few minor corrections along the way.

Revisions made subsequent to the first version of this article added additional material on pseudofunctors, pseudonatural transformations, modifications, and equivalence of bicategories; the main thrust being to give a proof that a pseudofunctor is a biequivalence if and only if it can be extended to an equivalence of bicategories.

Contents

Introduction **4**

Preliminaries **10**
0.1 Isomorphism Classes . . . 11

1 Bicategories **14**
1.1 Weak Composition . . . 16
1.1.1 Definition . . . 16
1.1.2 Hom-Subcategories . . . 19
1.1.3 Weak Units . . . 24
1.1.4 Regularity . . . 29
1.1.5 Associativity . . . 31
1.1.6 Unitors . . . 33
1.1.7 Prebicategories . . . 43
1.2 Horizontal Homs . . . 52
1.2.1 Prebicategories with Homs . . . 56
1.2.2 Choosing Homs . . . 57
1.2.3 Choosing Units . . . 63
1.2.4 Horizontal Composition . . . 64
1.3 Bicategories . . . 72
1.3.1 Categories Induce Bicategories . . . 74
1.3.2 Monoidal Categories induce Bicategories . . . 75
1.3.3 Prebicategories Extend to Bicategories . . . 77
1.4 Bicategories as Prebicategories . . . 88
1.4.1 Bicategories are Prebicategories . . . 88
1.4.2 Vertically Discrete Bicategories are Categories . . . 99
1.4.3 Obtaining the Unitors . . . 102
1.4.4 Further Properties of Bicategories . . . 109
1.5 Concrete Bicategories . . . 118
1.6 Coherence . . . 152
1.6.1 Bicategorical Language . . . 153
1.6.2 Normal Terms . . . 161
1.6.3 Reductions . . . 189

1.6.4 Evaluation . . . 194
1.6.5 Coherence . . . 210
1.7 Canonical Isomorphisms . . . 231
1.7.1 Basic Properties . . . 232
1.7.2 Introduction Rules . . . 236
1.7.3 Rules for Eliminating 'can' . . . 239
1.7.4 Rules for Whiskering . . . 244
1.8 Sub-Bicategories . . . 245
1.8.1 Construction . . . 246
1.8.2 The Sub-bicategory of Endo-arrows of an Object . . . 260
1.9 Internal Equivalences . . . 274
1.9.1 Definition of Equivalence . . . 275
1.9.2 Quasi-Inverses and Equivalence Maps . . . 277
1.9.3 Composing Equivalences . . . 284
1.9.4 Equivalent Objects . . . 287
1.9.5 Transporting Arrows along Equivalences . . . 288
1.10 Pseudofunctors . . . 303
1.10.1 Weak Arrows of Homs . . . 304
1.10.2 Definition of Pseudofunctors . . . 307
1.10.3 Pseudofunctors and Opposite Bicategories . . . 318
1.10.4 Preservation Properties . . . 325
1.10.5 Identity Pseudofunctors . . . 329
1.10.6 Embedding Pseudofunctors . . . 331
1.10.7 Composition of Pseudofunctors . . . 335
1.10.8 Restriction of Pseudofunctors . . . 345
1.10.9 Equivalence Pseudofunctors . . . 352
1.11 Strictness . . . 362
1.11.1 Normal and Strict Bicategories . . . 363
1.11.2 Strictification . . . 366
1.11.3 The Strictness Theorem . . . 398
1.11.4 Pseudofunctors into a Strict Bicategory . . . 428
1.11.5 Internal Equivalences in a Strict Bicategory . . . 432
1.12 Bicategory of Categories . . . 446
1.13 Adjunctions in a Bicategory . . . 469
1.13.1 Adjoint Transpose . . . 472
1.13.2 Preservation Properties for Adjunctions . . . 484
1.13.3 Pseudofunctors and Adjunctions . . . 490
1.13.4 Composition of Adjunctions . . . 504
1.13.5 Choosing Right Adjoints . . . 510
1.13.6 Equivalences Refine to Adjoint Equivalences . . . 511
1.13.7 Uniqueness of Adjoints . . . 527
1.14 Pseudonatural Transformations . . . 537
1.14.1 Definition of Pseudonatural Transformation . . . 537
1.14.2 Identity Pseudonatural Transformation . . . 539

1.14.3 Composite Pseudonatural Transformation 542
1.14.4 Whiskering of Pseudonatural Transformations 562
1.14.5 Pseudonatural Equivalences 583
1.14.6 Pseudonaturally Equivalent Pseudofunctors 693
1.15 Modifications 704
1.16 Equivalence of Bicategories 730
1.16.1 Definition of Equivalence of Bicategories 730
1.16.2 Equivalences Respect Pseudonatural Equivalence 737
1.16.3 Converse of an Equivalence 740
1.16.4 Composition of Equivalences 741
1.16.5 Equivalence with a Dense Sub-bicategory 749
1.16.6 Equivalence Pseudofunctors, Bijective on Objects 856
1.16.7 Equivalence Pseudofunctors Extend to Equivalences of Bicategories 908

2 Bicategories of Spans **941**
2.1 Span Bicategories 941
2.1.1 Spans 941
2.1.2 The Vertical Category of Spans 944
2.1.3 Putting Spans in Homs 952
2.1.4 Horizontal Composite of Spans 954
2.1.5 The Bicategory Span(C) 964
2.1.6 Miscellaneous Formulas 1014
2.1.7 Maps in Span(C) 1026
2.2 Tabulations 1031
2.2.1 Definition of Tabulation 1031
2.2.2 Tabulations yield Factorizations 1043
2.2.3 Tabulation of a Right Adjoint 1068
2.2.4 Preservation by Isomorphisms 1071
2.2.5 Canonical Tabulations 1081
2.2.6 Uniqueness of Tabulations 1082
2.2.7 'Tabulation' is Bicategorical 1089
2.3 Bicategories of Spans 1152
2.3.1 Definition 1152
2.3.2 Span(C) is a Bicategory of Spans 1158
2.3.3 Properties of Bicategories of Spans 1193
2.3.4 Choosing Tabulations 1199
2.3.5 Tabulations in a Bicategory of Spans 1204
2.3.6 The Classifying Category of Maps 1253
2.3.7 Arrows of Tabulations in Maps 1282
2.3.8 Equivalence of B and Span(Maps(B)) 1320

Bibliography **1445**

Introduction

Bicategories, introduced by Bénabou [2], are a generalization of categories in which the sets of arrows between pairs of objects (*i.e.* the "hom-sets") themselves have the structure of categories. In a typical formulation, the definition of bicategories involves three separate kinds of entities: *objects* (or 0-*cells*), *arrows* (or 1-*cells*), and morphisms between arrows (or 2-*cells*). There are two kinds of composition: *vertical* composition, which composes 2-cells within a single hom-category, and *horizontal* composition, which composes 2-cells in "adjacent" hom-categories $\hom(A, B)$ and $\hom(B, C)$. Horizontal composition is required to be functorial with respect to vertical composition; the identification of a 1-cell with the corresponding identity 2-cell then leads to the ability to horizontally compose 1-cells with 2-cells (*i.e.* "whiskering") and to horizontally compose 1-cells with each other. Each hom-category $\hom(A, A)$ is further equipped with an *identity* 1-cell id_A, which serves as a unit for horizontal composition. In a *strict* bicategory, also known as a 2-*category*, the usual unit and associativity laws for horizontal composition are required to hold exactly, or (as it is said) "on the nose". In a general bicategory, these laws are only required to hold "weakly"; that is, up to a collection of (vertical) isomorphisms that satisfy certain *coherence conditions*. A bicategory, all of whose hom-categories are discrete, is essentially an ordinary category. A bicategory with just one object amounts to a monoidal category whose tensor is given by horizontal composition. Alternatively, we may think of bicategories as a generalization of monoidal categories in which the tensor is permitted to be a partial operation, in analogy to the way in which ordinary categories can be considered as a generalization of monoids.

A standard example of a bicategory is **Cat**, the bicategory whose 0-cells are categories, whose 1-cells are functors, and whose 2-cells are natural transformations. This is in fact a 2-category; however, as two categories that are related by an equivalence of categories have the same "categorical" properties, it is often more sensible to consider constructions on categories as given up to equivalence, rather than up to isomorphism, and this leads to considering **Cat** as a bicategory and using bicategorical constructions rather than as a 2-category and using 2-categorical ones. This is one reason for the importance of bicategories: as Street [15] remarks, "In recent years it has become even more obvious that, although the fundamental constructions of set theory are categorical, the fundamental constructions of category theory are bicategorical."

An alternative reason for studying bicategories, which is more aligned with my own personal interests and forms a major reason why I chose to pursue the present project, is that they provide an elegant framework for theories of generalized relations, as has been

shown by Carboni, Walters, Street, and others [4] [6] [5] [3]. Indeed, the category of sets and relations becomes a bicategory by taking the inclusions between relations as 2-cells and thereby becomes an exemplar of the notion bicategory of relations which itself is a specialization of the notion of cartesian bicategory [6] [5]. In the study of the semantics of programming languages containing nondeterministic or concurrent constructs, it is natural to consider the meaning of a program in such a language as some kind of relation between inputs and outputs. Ordinary relations can be used for this purpose in simple situations, but they fail to be adequate for the study of higher-order nondeterministic programs or for concurrent programs that engage in interaction with their environment, so some sort of notion of generalized relation is needed. One is therefore led to try to identify some kind of bicategories of generalized relations as framework suitable for defining the semantics of such programs. One expects these to be instances of cartesian bicategories.

I attempted for a long time to try to develop a semantic framework for a certain class of interactive concurrent programs along the lines outlined above, but ultimately failed to obtain the kind of comprehensive understanding that I was seeking. The basic idea was to try to regard a program as denoting a kind of generalized machine, expressed as some sort of bimodule or two-sided fibration (*cf.* [16] [15]), to be represented as a certain kind of span in an underlying category of "maps", which would correspond to the meanings of deterministic programs. A difficulty with trying to formulate any kind of theory like this is that there quickly gets to be a lot of data and a lot of properties to keep track of, and it was certainly more than I could handle. For example, bicategories have objects, 1-cells, and 2-cells, as well as domains, codomains, composition and identities for both the horizontal and vertical structure. In addition, there are unit and associativity isomorphisms for the weak horizontal composition, as well as their associated coherence conditions. Cartesian bicategories are symmetric monoidal bicategories, which means that there is an additional tensor product, which comes with another set of canonical isomorphisms and coherence conditions. Still more canonical morphisms and coherence conditions are associated with the cartesian structure. Even worse, in order to give a proper account of the computational ideas I was hoping to capture, the underlying category of maps would at least have to be regarded as an ordered category, if not a more general 2-category or bicategory, so the situation starts to become truly daunting.

With so much data and so many properties, it is unusual in the literature to find proofs written out in anything approaching complete detail. To the extent that proofs are given, they often involve additional assumptions made purely for convenience and presentational clarity, such as assuming that the bicategories under consideration are strict when actually they are not, and then discharging these assumptions by appeals to informal arguments such as "the result holds in the general case because we can always replace a non-strict bicategory by an equivalent strict one." This is perhaps fine if you happen to have finely honed insight, but in my case I am always left wondering if something important hasn't been missed or glossed over, and I don't trust very much my own ability to avoid gross errors if I were to work at the same level of detail as the proofs that I see in the literature. So my real motivation for the present project was to try to see whether a proof assistant would actually be useful in carrying out fully formalized,

machine-checkable proofs of some kind of interesting facts about bicategories. I also hoped in the process to develop a better understanding of some concepts that I knew that I hadn't understood very well.

The project described in the present article is divided into two main parts. The first part, which comprises Chapter 1, seeks to develop a formalization of the notion of bicategory using Isabelle/HOL and to prove various facts about bicategories that are required for a subsequent application. Additional goals here are: (1) to be able to make as much use as possible of the formalizations previously created for categories [13] and monoidal categories [14]; (2) to create a plausibly useful framework for future extension; and (3) to better understand some subtleties involved in the definition of bicategory. In this chapter, we give an HOL formalization of bicategories that makes use of and extends the formalization of categories given in [13]. In that previous work, categories were formalized in an "object-free" style in terms of a suitably defined associative partial binary operation of composition on a single type. Elements of the type that behave as units for the composition were called "identities" and the "arrows" were identified as the elements of the type that are composable both on the left and on the right with identities. The identities composable in this way with an arrow were then shown to be uniquely determined, which permitted domain and codomain functions to be defined. This formalization of categories is economical in terms of basic data (only a single partial binary operation is required), but perhaps more importantly, functors and natural transformations need not be defined as structured objects, but instead can be taken to be ordinary functions between types that suitably preserve arrows and composition.

In order to carry forward unchanged the framework developed for categories, for the formalization of bicategories we take as a jumping-off point the somewhat offbeat view of a bicategory as a single global category under vertical composition (the arrows are the 2-cells), which is then equipped with an additional partial binary operation of horizontal composition. This point of view corresponds to thinking of bicategories as generalizations of monoidal categories in which the tensor is allowed to be a partial operation. In a direct generalization of the approach taken for categories, we then show that certain *weak units* with respect to the horizontal composition play the role of 0-cells (the identities with respect to vertical composition play the role of 1-cells) and that we can define the *sources* and *targets* of an arrow as the sets of weak units horizontally composable on the right and on the left with it. We then define a notion of weak associativity for the horizontal composition and arrive at the definition of a *prebicategory*, which consists of a (vertical) category equipped with an associative weak (horizontal) composition, subject to the additional assumption that every vertical arrow has a nonempty set of sources and targets with respect to the horizontal composition. We then show that, to obtain from a prebicategory a structure that satisfies a more traditional-looking definition of a bicategory, all that is necessary is to choose arbitrarily a particular representative source and target for each arrow. Moreover, every bicategory determines a prebicategory by simply forgetting the chosen sources and targets. This development clarifies that an *a priori* assignment of source and target objects for each 2-cell is merely a convenience, rather than an element essential to the notion of bicategory.

Additional highlights of Chapter 1 are as follows:

- As a result of having formalized bicategories essentially as "monoidal categories with partial tensor", we are able to generalize to bicategories, in a mostly straightforward way, the proof of the Coherence Theorem we previously gave for monoidal categories in [14]. We then develop some machinery that enables us to apply the Coherence Theorem to shortcut certain kinds of reasoning involving canonical isomorphisms.

- Using the syntactic setup developed for the proof of the Coherence Theorem, we also give a proof of the Strictness Theorem, which states that every bicategory is biequivalent to a 2-category, its so-called "strictification".

- We define the notions of internal equivalence and internal adjunction in a bicategory and prove a number of basic facts about these notions, including composition of equivalences and adjunctions, and that every equivalence can be refined to an adjoint equivalence.

- We formalize the notion of a pseudofunctor between bicategories, generalizing the notion of a monoidal functor between monoidal categories and we show that pseudofunctors preserve internal equivalences and adjunctions.

- We define a sub-class of pseudofunctors which we call *equivalence pseudofunctors*. Equivalence pseudofunctors are intended to coincide with those pseudofunctors that can be extended to an equivalence of bicategories, but we do not attempt to give an independent definition equivalence of bicategories in the present development. Instead, we establish various properties of equivalence pseudofunctors to provide some confidence that the notion has been formalized correctly. Besides establishing various preservation results, we prove that, given an equivalence pseudofunctor, we may obtain one in the converse direction. For the rest of this article we use the property of two bicategories being connected by an equivalence pseudofunctor as a surrogate for the property of biequivalence, leaving for future work a more proper formulation of equivalence of bicategories and a full verification of the relationship of this notion with equivalence pseudofunctors.

The second part of the project, presented in Chapter 2, is to demonstrate the utility of the framework by giving a formalized proof of a nontrivial theorem about bicategories. For this part, I chose to tackle a theorem of Carboni, Kasangian, and Street ([4], "CKS" for short) which gives axioms that characterize up to equivalence those bicategories whose 1-cells are spans of arrows in an underlying category with pullbacks and whose 2-cells are arrows of spans. The original paper is very short (nine pages in total) and the result I planned to formalize (Theorem 4) was given on the sixth page. I thought I had basically understood this result and that the formalization would not take very long to accomplish, but I definitely underestimated both my prior understanding of the result and the amount of auxiliary material that it would be necessary to formalize before I could complete the main proof. Eventually I did complete the formalization, and in the process filled in what seemed to me to be significant omissions in Carboni, Kasangian, and Street's presentation, as well as correcting some errors of a minor nature.

Highlights of Chapter 2 are the following:

- A formalization of the notion of a category with chosen pullbacks, a proof that this formalization is in agreement with the general definition of limits we gave previously in [13], and the development of some basic properties of a category with pullbacks.

- A construction, given a category C with chosen pullbacks, of the "span bicategory" $\mathrm{Span}(C)$, whose objects are those of the given category, whose 1-cells are spans of arrows of C, and whose 2-cells are arrows of spans. We characterize the maps (the *i.e.* left adjoints) in $\mathrm{Span}(C)$ as exactly those spans whose "input leg" is invertible.

- A formalization of the notion of *tabulation* of a 1-cell in a bicategory and a development of some of its properties. Tabulations are a kind of bicategorical limit introduced by CKS, which can be used to define a kind of biuniversal way of factoring a 1-cell up to isomorphism as the horizontal composition of a map and the adjoint of a map.

- A formalization of *bicategories of spans*, which are bicategories that satisfy three axioms introduced in CKS. We give a formal proof of CKS Theorem 4, which characterizes the bicategories of spans as those bicategories that are biequivalent to a bicategory $\mathrm{Span}(C)$ for some category C with pullbacks. One direction of the proof shows that if C is a category with pullbacks, then $\mathrm{Span}(C)$ satisfies the axioms for a bicategory of spans. Moreover, we show that the notion "bicategory of spans" is preserved under equivalence of bicategories, so that in fact any bicategory biequivalent to one of the form $\mathrm{Span}(C)$ is a bicategory of spans. Conversely, we show that if B is a bicategory of spans, then B is biequivalent to $\mathrm{Span}(\mathrm{Maps}(B))$, where $\mathrm{Maps}(B)$ is the so-called *classifying category* of the maps in B, which has as objects those of B and as arrows the isomorphism classes of maps in B.

  In order to formalize the proof of this result, it was necessary to develop a number of details not mentioned by CKS, including ways of composing tabulations vertically and horizontally, and spelling out a way to choose pullbacks in $\mathrm{Maps}(B)$ so that the tupling of arrows of $\mathrm{Maps}(B)$ obtained using the chosen pullbacks agrees with that obtained through horizontal composition of tabulations. These details were required in order to give the definition of the compositor for an equivalence pseudofunctor SPN from B to $\mathrm{Span}(\mathrm{Maps}(B))$ and establish the necessary coherence conditions.

In the end, I think it can be concluded that Isabelle/HOL can be used with benefit to formalize proofs about bicategories. It is certainly very helpful for keeping track of the data involved and the proof obligations required. For example, in the formalization given here, a total of 99 separate subgoals are involved in proving that a given set of data constitutes a bicategory (only 7 subgoals are required for an ordinary category) and another 29 subgoals must be proved in order to establish a pseudofunctor between two bicategories (only 5 additional subgoals are required for an ordinary functor), but

the proof assistant assumes the burden of keeping track of these proof obligations and presenting them to the human user in a structured, understandable fashion. On the other hand, some of the results proved here still required some lengthy equational "diagram chases" for which the proof assistant (at least so far) didn't provide that much help (aside from checking their correctness). An exception to this was in the case of equational reasoning about expressions constructed purely of canonical isomorphisms, which our formulation of the Coherence Theorem permitted to be carried out automatically by the simplifier. It seems likely, though, that there is still room for more general procedures to be developed in order to allow other currently lengthy chains of equational reasoning to be carried out automatically.

Revision Notes

The original version of this article dates from January, 2020. The current version of this article incorporates revisions made throughout 2020. A number of the changes made in early to mid-2020 consisted of minor improvements and speedups. A more major change made in this period was that the theory "category with pullbacks" was moved to [13], where it more logically belongs.

In late 2020 additional material was added relating to pseudofunctors, pseudonatural transformations, and equivalence of bicategories. The main result shown was that a pseudofunctor is a biequivalence if and only if it can be extended to an equivalence of bicategories. This important result was sidestepped in the original version of this article, but the author felt that it was a glaring omission that should be corrected. Unfortunately, to formalize these results required some rather lengthy calculations in order to establish coherence conditions. These calculations added significantly to the line count of this article, as well as the time and memory required to validate the proofs.

In July, 2021, a "concrete bicategory" construction analogous to the "concrete category" construction in [13] was added. This construction was used to give a construction of the bicategory of categories, functors, and natural transformations, which was then shown to be strict.

Preliminaries

0.1 Isomorphism Classes

The following is a small theory that facilitates working with isomorphism classes of objects in a category.

theory *IsomorphismClass*
imports *Category3.EpiMonoIso Category3.NaturalTransformation*
begin

context *category*
begin

notation *isomorphic* (**infix** ‹≅› *50*)

definition *iso-class* (‹⟦-⟧›)
where *iso-class* $f \equiv \{f'.\ f \cong f'\}$

definition *is-iso-class*
where *is-iso-class* $F \equiv \exists f.\ f \in F \wedge F = \textit{iso-class}\ f$

definition *iso-class-rep*
where *iso-class-rep* $F \equiv \textit{SOME}\ f.\ f \in F$

lemmas *isomorphic-transitive* [*trans*]
lemmas *naturally-isomorphic-transitive* [*trans*]

lemma *inv-in-homI* [*intro*]:
assumes *iso* f **and** «$f : a \to b$»
shows «*inv* $f : b \to a$»
using *assms inv-is-inverse seqE inverse-arrowsE*
by (*metis ide-compE in-homE in-homI*)

lemma *iso-class-is-nonempty*:
assumes *is-iso-class* F
shows $F \neq \{\}$
using *assms is-iso-class-def iso-class-def* **by** *auto*

lemma *iso-class-memb-is-ide*:

assumes *is-iso-class F* **and** $f \in F$
shows *ide f*
  using *assms is-iso-class-def iso-class-def isomorphic-def* **by** *auto*

lemma *ide-in-iso-class*:
assumes *ide f*
shows $f \in [\![f]\!]$
  using *assms iso-class-def is-iso-class-def isomorphic-reflexive* **by** *simp*

lemma *rep-in-iso-class*:
assumes *is-iso-class F*
shows *iso-class-rep* $F \in F$
  using *assms is-iso-class-def iso-class-rep-def someI-ex* [*of* $\lambda f.\ f \in F$]
    ide-in-iso-class
  by *metis*

lemma *is-iso-classI*:
assumes *ide f*
shows *is-iso-class* $[\![f]\!]$
  using *assms iso-class-def is-iso-class-def isomorphic-reflexive* **by** *blast*

lemma *rep-iso-class*:
assumes *ide f*
shows *iso-class-rep* $[\![f]\!] \cong f$
  using *assms is-iso-classI rep-in-iso-class iso-class-def isomorphic-symmetric*
  by *blast*

lemma *iso-class-elems-isomorphic*:
assumes *is-iso-class F* **and** $f \in F$ **and** $f' \in F$
shows $f \cong f'$
  using *assms iso-class-def*
  by (*metis is-iso-class-def isomorphic-symmetric isomorphic-transitive mem-Collect-eq*)

lemma *iso-class-eqI* [*intro*]:
assumes $f \cong g$
shows $[\![f]\!] = [\![g]\!]$
proof −
  have *f*: *ide f*
    using *assms isomorphic-def* **by** *auto*
  have *g*: *ide g*
    using *assms isomorphic-def* **by** *auto*
  have $f \in [\![g]\!]$
    using *assms iso-class-def isomorphic-symmetric* **by** *simp*
  moreover have $g \in [\![g]\!]$
    using *assms g iso-class-def isomorphic-reflexive isomorphic-def* **by** *simp*
  ultimately have $\bigwedge h.\ (h \in [\![f]\!]) = (h \in [\![g]\!])$
    using *assms g iso-class-def* [*of f*] *iso-class-def* [*of g*]
      isomorphic-reflexive isomorphic-symmetric isomorphic-transitive
    by *blast*

```
    thus ?thesis by auto
  qed

  lemma iso-class-eq:
  assumes is-iso-class F and is-iso-class G and F ∩ G ≠ {}
  shows F = G
  proof −
    obtain h where h: h ∈ F ∧ h ∈ G
      using assms by auto
    show ?thesis
      using assms h
      by (metis is-iso-class-def iso-class-elems-isomorphic iso-class-eqI)
  qed

  lemma iso-class-rep [simp]:
  assumes is-iso-class F
  shows ⟦iso-class-rep F⟧ = F
  proof (intro iso-class-eq)
    show is-iso-class ⟦iso-class-rep F⟧
      using assms is-iso-classI iso-class-memb-is-ide rep-in-iso-class by blast
    show is-iso-class F
      using assms by simp
    show ⟦iso-class-rep F⟧ ∩ F ≠ {}
      using assms rep-in-iso-class
      by (meson disjoint-iff-not-equal ide-in-iso-class iso-class-memb-is-ide)
  qed

 end

end
```

Chapter 1

Bicategories

The objective of this section is to construct a formalization of bicategories that is compatible with our previous formulation of categories [13] and that permits us to carry over unchanged as much of the work done on categories as possible. For these reasons, we conceive of a bicategory in what might be regarded as a somewhat unusual fashion. Rather than a traditional development, which would typically define a bicategory to be essentially "a 'category' whose homs themselves have the structure of categories," here we regard a bicategory as "a (vertical) category that has been equipped with a suitable (horizontal) weak composition." Stated another way, we think of a bicategory as a generalization of a monoidal category in which the tensor product is a partial operation, rather than a total one. Our definition of bicategory can thus be summarized as follows: a bicategory is a (vertical) category that has been equipped with idempotent endofunctors *src* and *trg* that assign to each arrow its "source" and "target" subject to certain commutativity constraints, a partial binary operation $\star$ of horizontal composition that is suitably functorial on the "hom-categories" determined by the assignment of sources and targets, "associativity" isomorphisms «$\mathrm{a}[f,\ g,\ h] : (f \star g) \star h \Rightarrow f \star (g \star h)$» for each horizontally composable triple of vertical identities f, g, h, subject to the usual naturality and coherence conditions, and for each "object" a (defined to be an arrow that is its own source and target) a "unit isomorphism" «$\mathrm{i}[a] : a \star a \Rightarrow a$». As is the case for monoidal categories, the unit isomorphisms and associator isomorphisms together enable a canonical definition of left and right "unit" isomorphisms «$\mathrm{l}[f] : a \star f \Rightarrow f$» and «$\mathrm{r}[f] : f \star a \Rightarrow f$» when f is a vertical identity horizontally composable on the left or right by a, and it can be shown that these are the components of natural transformations.

The definition of bicategory just sketched shares with a more traditional version the requirement that assignments of source and target are given as basic data, and these assignments determine horizontal composability in the sense that arrows μ and ν are composable if the chosen source of μ coincides with the chosen target of ν. Thus it appears, at least on its face, that composability of arrows depends on an assignment of sources and targets. We are interested in establishing whether this is essential or whether bicategories can be formalized in a completely "object-free" fashion.

It turns out that we can obtain such an object-free formalization through a rather

direct generalization of the approach we used in the formalization of categories. Specifically, we define a *weak composition* to be a partial binary operation $\star$ on the arrow type of a "vertical" category V, such that the domain of definition of this operation is itself a category (of "horizontally composable pairs of arrows"), the operation is functorial, and it is subject to certain matching conditions which include those satisfied by a category. From the axioms for a weak composition we can prove the existence of "hom-categories", which are subcategories of V consisting of arrows horizontally composable on the left or right by a specified vertical identity. A *weak unit* is defined to be a vertical identity a such that $a \star a \cong a$ and is such that the mappings $a \star$ - and - $\star\ a$ are fully faithful endofunctors of the subcategories of V consisting of the arrows for which they are defined. We define the *sources* of an arrow μ to be the weak units that are horizontally composable with μ on the right, and the *targets* of μ to be the weak units that are horizontally composable with μ on the left. An *associative weak composition* is defined to be a weak composition that is equipped with "associator" isomorphisms «a[f, g, h] : $(f \star g) \star h \Rightarrow f \star (g \star h)$» for horizontally composable vertical identities f, g, h, subject to the usual naturality and coherence conditions. A *prebicategory* is defined to be an associative weak composition for which every arrow has a source and a target. We show that the sets of sources and targets of each arrow in a prebicategory is an isomorphism class of weak units, and that horizontal composability of arrows μ and ν is characterized by the set of sources of μ being equal to the set of targets of ν.

We show that prebicategories are essentially "bicategories without objects". Given a prebicategory, we may choose an arbitrary representative of each isomorphism class of weak units and declare these to be "objects" (this is analogous to choosing a particular unit object in a monoidal category). For each object we may choose a particular *unit isomorphism* «i[a] : $a \star a \Rightarrow a$». This choice, together with the associator isomorphisms, enables a canonical definition of left and right unit isomorphisms «l[f] : $a \star f \Rightarrow f$» and «r[f] : $f \star a \Rightarrow f$» when f is a vertical identity horizontally composable on the left or right by a, and it can be shown that these are the components of natural isomorphisms. We may then define "the source" of an arrow to be the chosen representative of the set of its sources and "the target" to be the chosen representative of the set of its targets. We show that the resulting structure is a bicategory, in which horizontal composability as given by the weak composition coincides with the "syntactic" version determined by the chosen sources and targets. Conversely, a bicategory determines a prebicategory, essentially by forgetting the sources, targets and unit isomorphisms. These results make it clear that the assignment of sources and targets to arrows in a bicategory is basically a convenience and that horizontal composability of arrows is not dependent on a particular choice.

theory *Prebicategory*
imports *Category3.EquivalenceOfCategories Category3.Subcategory IsomorphismClass*
begin

1.1 Weak Composition

In this section we define a locale *weak-composition*, which formalizes a functorial operation of "horizontal" composition defined on an underlying "vertical" category. The definition is expressed without the presumption of the existence of any sort of "objects" that determine horizontal composability. Rather, just as we did in showing that the *partial-magma* locale supported the definition of "identity arrow" as a kind of unit for vertical composition which ultimately served as a basis for the definition of "domain" and "codomain" of an arrow, here we show that the *weak-composition* locale supports a definition of *weak unit* for horizontal composition which can ultimately be used to define the *sources* and *targets* of an arrow with respect to horizontal composition. In particular, the definition of weak composition involves axioms that relate horizontal and vertical composability. As a consequence of these axioms, for any fixed arrow μ, the sets of arrows horizontally composable on the left and on the right with μ form subcategories with respect to vertical composition. We define the sources of μ to be the weak units that are composable with μ on the right, and the targets of μ to be the weak units that are composable with μ on the left. Weak units are then characterized as arrows that are members of the set of their own sources (or, equivalently, of their own targets).

1.1.1 Definition

locale *weak-composition* =
  category V +
  VxV: *product-category V V* +
  VoV: *subcategory VxV.comp* ‹$\lambda\mu\nu$. *fst* $\mu\nu \star$ *snd* $\mu\nu \neq$ *null*› +
  functor VoV.comp V ‹$\lambda\mu\nu$. *fst* $\mu\nu \star$ *snd* $\mu\nu$›
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a comp* (**infixr** ‹⋆› *53*) +
assumes *left-connected*: *seq* $\nu\ \nu' \Longrightarrow \nu \star \mu \neq$ *null* $\longleftrightarrow \nu' \star \mu \neq$ *null*
and *right-connected*: *seq* $\mu\ \mu' \Longrightarrow \nu \star \mu \neq$ *null* $\longleftrightarrow \nu \star \mu' \neq$ *null*
and *match-1*: $[\![\ \nu \star \mu \neq$ *null*; $(\nu \star \mu) \star \tau \neq$ *null* $]\!] \Longrightarrow \mu \star \tau \neq$ *null*
and *match-2*: $[\![\ \nu \star (\mu \star \tau) \neq$ *null*; $\mu \star \tau \neq$ *null* $]\!] \Longrightarrow \nu \star \mu \neq$ *null*
and *match-3*: $[\![\ \mu \star \tau \neq$ *null*; $\nu \star \mu \neq$ *null* $]\!] \Longrightarrow (\nu \star \mu) \star \tau \neq$ *null*
and *match-4*: $[\![\ \mu \star \tau \neq$ *null*; $\nu \star \mu \neq$ *null* $]\!] \Longrightarrow \nu \star (\mu \star \tau) \neq$ *null*
begin

We think of the arrows of the vertical category as "2-cells" and the vertical identities as "1-cells". In the formal development, the predicate *arr* ("arrow") will have its normal meaning with respect to the vertical composition, hence *arr* μ will mean, essentially, "μ is a 2-cell". This is somewhat unfortunate, as it is traditional when discussing bicategories to use the term "arrow" to refer to the 1-cells. However, we are trying to carry over all the formalism that we have already developed for categories and apply it to bicategories with as little change and redundancy as possible. It becomes too confusing to try to repurpose the name *arr* to mean *ide* and to introduce a replacement for the name *arr*, so we will simply tolerate the situation. In informal text, we will prefer the terms "2-cell" and "1-cell" over (vertical) "arrow" and "identity" when there is a chance for confusion.

We do, however, make the following adjustments in notation for *in-hom* so that it is distinguished from the notion *in-hhom* ("in horizontal hom") to be introduced subsequently.

no-notation *in-hom* (‹«- : - → -»›)
notation *in-hom* (‹«- : - ⇒ -»›)

lemma *is-partial-magma*:
shows *partial-magma H*
proof
  show $\exists! n.\ \forall f.\ n \star f = n \wedge f \star n = n$
  proof
    show *1*: $\forall f.\ null \star f = null \wedge f \star null = null$
      using *extensionality VoV.inclusion* $VoV.arr\text{-}char_{SbC}$ **by** *force*
    show $\bigwedge n.\ \forall f.\ n \star f = n \wedge f \star n = n \Longrightarrow n = null$
      using *1* $VoV.arr\text{-}char_{SbC}$ *extensionality not-arr-null* **by** *metis*
  qed
qed

interpretation *H*: *partial-magma H*
  using *is-partial-magma* **by** *auto*
interpretation *H*: *partial-composition H*
  ..

lemma *is-partial-composition*:
shows *partial-composition H*
  ..

Either *match-1* or *match-2* seems essential for the next result, which states that the nulls for the horizontal and vertical compositions coincide.

lemma *null-agreement* [*simp*]:
shows $H.null = null$
  by (*metis VoV.inclusion VxV.not-arr-null match-1 H.null-is-zero*(*1*))

lemma *composable-implies-arr*:
assumes $\nu \star \mu \neq null$
shows $arr\ \mu$ **and** $arr\ \nu$
  using *assms extensionality* $VoV.arr\text{-}char_{SbC}$ *VoV.inclusion* **by** *auto*

lemma *hcomp-null* [*simp*]:
shows $null \star \mu = null$ **and** $\mu \star null = null$
  using *H.null-is-zero* **by** *auto*

lemma $hcomp\text{-}simps_{WC}$ [*simp*]:
assumes $\nu \star \mu \neq null$
shows $arr\ (\nu \star \mu)$ **and** $dom\ (\nu \star \mu) = dom\ \nu \star dom\ \mu$ **and** $cod\ (\nu \star \mu) = cod\ \nu \star cod\ \mu$
  using *assms preserves-arr preserves-dom preserves-cod* $VoV.arr\text{-}char_{SbC}$ *VoV.inclusion*
    VoV.dom-simp VoV.cod-simp
  by *force+*

lemma *ide-hcomp*$_{WC}$:
assumes *ide f* **and** *ide g* **and** $g \star f \neq null$
shows *ide* $(g \star f)$
  using *assms preserves-ide VoV.ide-char*$_{SbC}$ **by** *force*

lemma *hcomp-in-hom*$_{WC}$ [*intro*]:
assumes $\nu \star \mu \neq null$
shows «$\nu \star \mu$: *dom* $\nu \star$ *dom* $\mu \Rightarrow$ *cod* $\nu \star$ *cod* μ»
  using *assms* **by** *auto*

Horizontal composability of arrows is determined by horizontal composability of their domains and codomains (defined with respect to vertical composition).

lemma *hom-connected*:
shows $\nu \star \mu \neq null \longleftrightarrow$ *dom* $\nu \star \mu \neq null$
and $\nu \star \mu \neq null \longleftrightarrow \nu \star$ *dom* $\mu \neq null$
and $\nu \star \mu \neq null \longleftrightarrow$ *cod* $\nu \star \mu \neq null$
and $\nu \star \mu \neq null \longleftrightarrow \nu \star$ *cod* $\mu \neq null$
proof −
  show $\nu \star \mu \neq null \longleftrightarrow$ *dom* $\nu \star \mu \neq null$
    using *left-connected* [*of* ν *dom* ν μ] *composable-implies-arr arr-dom-iff-arr* **by** *force*
  show $\nu \star \mu \neq null \longleftrightarrow$ *cod* $\nu \star \mu \neq null$
    using *left-connected* [*of* *cod* ν ν μ] *composable-implies-arr arr-cod-iff-arr* **by** *force*
  show $\nu \star \mu \neq null \longleftrightarrow \nu \star$ *dom* $\mu \neq null$
    using *right-connected* [*of* μ *dom* μ ν] *composable-implies-arr arr-dom-iff-arr* **by** *force*
  show $\nu \star \mu \neq null \longleftrightarrow \nu \star$ *cod* $\mu \neq null$
    using *right-connected* [*of cod* μ μ ν] *composable-implies-arr arr-cod-iff-arr* **by** *force*
qed

lemma *isomorphic-implies-equicomposable*:
assumes $f \cong g$
shows $\tau \star f \neq null \longleftrightarrow \tau \star g \neq null$
and $f \star \sigma \neq null \longleftrightarrow g \star \sigma \neq null$
  using *assms isomorphic-def hom-connected* **by** *auto*

lemma *interchange*:
assumes *seq* ν μ **and** *seq* τ σ
shows $(\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$
proof −
  have $\mu \star \sigma = null \Longrightarrow$ *?thesis*
    by (*metis assms null-is-zero*(*2*) *dom-comp hom-connected*(*1*−*2*))
  moreover have $\mu \star \sigma \neq null \Longrightarrow$ *?thesis*
  proof −
    assume $\mu\sigma$: $\mu \star \sigma \neq null$
    have *1*: *VoV.arr* (μ, σ)
      using $\mu\sigma$ *VoV.arr-char*$_{SbC}$ **by** *auto*
    have $\nu\tau$: $(\nu, \tau) \in$ *VoV.hom* (*VoV.cod* (μ, σ)) (*VoV.cod* (ν, τ))
    proof −
      have *VoV.arr* (ν, τ)
        using *assms 1 hom-connected VoV.arr-char*$_{SbC}$

by (*elim seqE*, *auto*, *metis*)
thus *?thesis*
using *assms* $\mu\sigma$ $VoV.dom\text{-}char_{SbC}$ $VoV.cod\text{-}char_{SbC}$ **by** *fastforce*
qed
show *?thesis*
proof −
have $VoV.seq\ (\nu,\ \tau)\ (\mu,\ \sigma)$
using *assms 1* $\mu\sigma$ $\nu\tau$ $VoV.seqI$ **by** *blast*
thus *?thesis*
using *assms 1* $\mu\sigma$ $\nu\tau$ $VoV.comp\text{-}char$ *preserves-comp* [*of* $(\nu,\ \tau)$ $(\mu,\ \sigma)$] $VoV.seqI$
by *fastforce*
qed
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *paste-1*:
shows $\nu \star \mu = (cod\ \nu \star \mu) \cdot (\nu \star dom\ \mu)$
using *interchange composable-implies-arr comp-arr-dom comp-cod-arr*
hom-connected(*2*−*3*)
by (*metis null-is-zero*(*2*))

lemma *paste-2*:
shows $\nu \star \mu = (\nu \star cod\ \mu) \cdot (dom\ \nu \star \mu)$
using *interchange composable-implies-arr comp-arr-dom comp-cod-arr*
hom-connected(*1*,*4*)
by (*metis null-is-zero*(*2*))

lemma *whisker-left*:
assumes *seq* ν μ **and** *ide f*
shows $f \star (\nu \cdot \mu) = (f \star \nu) \cdot (f \star \mu)$
using *assms interchange* [*of f f* ν μ] *hom-connected* **by** *auto*

lemma *whisker-right*:
assumes *seq* ν μ **and** *ide f*
shows $(\nu \cdot \mu) \star f = (\nu \star f) \cdot (\mu \star f)$
using *assms interchange* [*of* ν μ *f f*] *hom-connected* **by** *auto*

1.1.2 Hom-Subcategories

definition *left*
where $left\ \tau \equiv \lambda\mu.\ \tau \star \mu \neq null$

definition *right*
where $right\ \sigma \equiv \lambda\mu.\ \mu \star \sigma \neq null$

lemma *right-iff-left*:
shows $right\ \sigma\ \tau \longleftrightarrow left\ \tau\ \sigma$
using *right-def left-def* **by** *simp*

lemma *left-respects-isomorphic*:
assumes $f \cong g$
shows *left* f = *left* g
  using *assms isomorphic-implies-equicomposable left-def* **by** *auto*

lemma *right-respects-isomorphic*:
assumes $f \cong g$
shows *right* f = *right* g
  using *assms isomorphic-implies-equicomposable right-def* **by** *auto*

lemma *left-iff-left-inv*:
assumes *iso* μ
shows *left* τ $\mu \longleftrightarrow$ *left* τ (*inv* μ)
  using *assms left-def inv-in-hom hom-connected*(*2*) *hom-connected*(*4*) [*of* τ *inv* μ]
  by *auto*

lemma *right-iff-right-inv*:
assumes *iso* μ
shows *right* σ $\mu \longleftrightarrow$ *right* σ (*inv* μ)
  using *assms right-def inv-in-hom hom-connected*(*1*) *hom-connected*(*3*) [*of inv* μ σ]
  by *auto*

lemma *left-hom-is-subcategory*:
assumes *arr* μ
shows *subcategory* V (*left* μ)
  using *composable-implies-arr hom-connected*(*2*,*4*)
  apply (*unfold left-def*, *unfold-locales*)
    apply *auto*
  by (*metis cod-comp seqI*)

lemma *right-hom-is-subcategory*:
assumes *arr* μ
shows *subcategory* V (*right* μ)
  using *composable-implies-arr hom-connected*(*1*,*3*)
  apply (*unfold right-def*, *unfold-locales*)
    apply *auto*
  by (*metis cod-comp seqI*)

abbreviation *Left*
where *Left* $a \equiv$ *subcategory.comp* V (*left* a)

abbreviation *Right*
where *Right* $a \equiv$ *subcategory.comp* V (*right* a)

We define operations of composition on the left or right with a fixed 1-cell, and show that such operations are functorial in case that 1-cell is horizontally self-composable.

definition H_L
where $H_L\ g \equiv \lambda\mu.\ g \star \mu$

definition H_R
where $H_R\ f \equiv \lambda\mu.\ \mu \star f$

Note that *match-3* and *match-4* are required for the next results.

lemma *endofunctor-*H_L:
assumes *ide g* **and** $g \star g \neq null$
shows *endofunctor* (*Left g*) (H_L *g*)
proof −
  interpret *L*: *subcategory V* ‹*left g*› **using** *assms left-hom-is-subcategory* **by** *simp*
  have ∗: $\bigwedge\mu.\ L.arr\ \mu \Longrightarrow H_L\ g\ \mu = g \star \mu$
    using *assms* H_L*-def* **by** *simp*
  have *preserves-arr*: $\bigwedge\mu.\ L.arr\ \mu \Longrightarrow L.arr\ (H_L\ g\ \mu)$
    using *assms* ∗ $L.arr\text{-}char_{SbC}$ *left-def match-4* **by** *force*
  show *endofunctor L.comp* (H_L *g*)
    using *assms* ∗
    apply *unfold-locales*
    using H_L*-def* $L.arr\text{-}char_{SbC}$ *L.null-char left-def*
      apply *force*
    using *preserves-arr*
      apply *blast*
      apply (*metis L.dom-simp L.not-arr-null L.null-char* $hcomp\text{-}simps_{WC}$(*2*) *ide-char preserves-arr* H_L*-def*)
  apply (*metis* H_L*-def L.arrE L.cod-simp* $hcomp\text{-}simps_{WC}$(*3*) *ide-char left-def preserves-arr*)
    by (*metis L.comp-def L.comp-simp* $L.seq\text{-}char_{SbC}$ $hcomp\text{-}simps_{WC}$(*1*) *whisker-left preserves-arr*)
qed

lemma *endofunctor-*H_R:
assumes *ide f* **and** $f \star f \neq null$
shows *endofunctor* (*Right f*) (H_R *f*)
proof −
  interpret *R*: *subcategory V* ‹*right f*› **using** *assms right-hom-is-subcategory* **by** *simp*
  have ∗: $\bigwedge\mu.\ R.arr\ \mu \Longrightarrow H_R\ f\ \mu = \mu \star f$
    using *assms* H_R*-def* **by** *simp*
  have *preserves-arr*: $\bigwedge\mu.\ R.arr\ \mu \Longrightarrow R.arr\ (H_R\ f\ \mu)$
    using *assms* ∗ $R.arr\text{-}char_{SbC}$ *right-def match-3* **by** *force*
  show *endofunctor R.comp* (H_R *f*)
    using *assms* ∗
    apply *unfold-locales*
    using H_R*-def* $R.arr\text{-}char_{SbC}$ *R.null-char right-def*
      apply *force*
    using *preserves-arr*
      apply *blast*
      apply (*metis R.dom-simp R.not-arr-null R.null-char* $hcomp\text{-}simps_{WC}$(*2*) *ide-char preserves-arr* H_R*-def*)
      apply (*metis* H_R*-def R.arrE R.cod-simp* $hcomp\text{-}simps_{WC}$(*3*) *ide-char right-def preserves-arr*)
      by (*metis R.comp-def R.comp-simp* $R.seq\text{-}char_{SbC}$ $hcomp\text{-}simps_{WC}$(*1*) *whisker-right*

preserves-arr)
qed

end

locale *left-hom* =
weak-composition V H +
S: *subcategory V* ‹*left* ω›
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a comp* (**infixr** ‹⋆› *53*)
and ω :: *'a* +
assumes *arr-*ω: *arr* ω
begin

no-notation *in-hom* (‹«- : - → -»›)
notation *in-hom* (‹«- : - ⇒ -»›)
notation *S.comp* (**infixr** ‹$\cdot_S$› *55*)
notation *S.in-hom* (‹«- : - $\Rightarrow_S$ -»›)

lemma *right-hcomp-closed*:
assumes «μ : $x \Rightarrow_S y$» **and** «ν : $c \Rightarrow d$» **and** $\mu \star \nu \neq$ *null*
shows «$\mu \star \nu$: $x \star c \Rightarrow_S y \star d$»
using *assms arr-*ω $S.arr\text{-}char_{SbC}$ *S.dom-simp S.cod-simp left-def match-4*
by (*elim S.in-homE*, *intro S.in-homI*) *auto*

lemma *interchange*:
assumes *S.seq* ν μ **and** *S.seq* τ σ **and** $\mu \star \sigma \neq$ *null*
shows $(\nu \cdot_S \mu) \star (\tau \cdot_S \sigma) = (\nu \star \tau) \cdot_S (\mu \star \sigma)$
proof −
have *1*: $\nu \star \tau \neq$ *null*
using *assms hom-connected*(*1*) [*of* ν σ] *hom-connected*(*2*) [*of* ν τ] *hom-connected*(*3*−*4*) *S.dom-simp S.cod-simp*
by *force*
have $(\nu \cdot_S \mu) \star (\tau \cdot_S \sigma) = (\nu \cdot \mu) \star (\tau \cdot \sigma)$
using *assms S.comp-char* $S.seq\text{-}char_{SbC}$ **by** *metis*
also have ... $= (\nu \star \tau) \cdot (\mu \star \sigma)$
using *assms interchange* $S.seq\text{-}char_{SbC}$ $S.arr\text{-}char_{SbC}$ **by** *simp*
also have ... $= (\nu \star \tau) \cdot_S (\mu \star \sigma)$
using *assms 1*
by (*metis* $S.arr\text{-}char_{SbC}$ *S.comp-char* $S.seq\text{-}char_{SbC}$ *ext match-4 left-def*)
finally show *?thesis* **by** *blast*
qed

lemma *inv-char*:
assumes *S.arr* φ **and** *iso* φ
shows *S.inverse-arrows* φ (*inv* φ)
and *S.inv* φ = *inv* φ
proof −
have *1*: *S.arr* (*inv* φ)

```
      using assms S.arr-char_SbC left-iff-left-inv
      by (intro S.arrI_SbC) meson
    show S.inv φ = inv φ
      using assms 1 S.inv-char_SbC S.iso-char_SbC by blast
    thus S.inverse-arrows φ (inv φ)
      using assms 1 S.iso-char_SbC S.inv-is-inverse by metis
  qed

  lemma iso-char:
  assumes S.arr φ
  shows S.iso φ ⟷ iso φ
    using assms S.iso-char_SbC inv-char by auto

end

locale right-hom =
  weak-composition V H +
  S: subcategory V ‹right ω›
for V :: 'a comp        (infixr ‹·› 55)
and H :: 'a comp        (infixr ‹⋆› 53)
and ω :: 'a +
assumes arr-ω: arr ω
begin

  no-notation in-hom      (‹«- : - → -»›)
  notation in-hom         (‹«- : - ⇒ -»›)
  notation S.comp         (infixr ‹·_S› 55)
  notation S.in-hom       (‹«- : - ⇒_S -»›)

  lemma left-hcomp-closed:
  assumes «μ : x ⇒_S y» and «ν : c ⇒ d» and ν ⋆ μ ≠ null
  shows «ν ⋆ μ : c ⋆ x ⇒_S d ⋆ y»
    using assms arr-ω S.arr-char_SbC S.dom-simp S.cod-simp right-def match-3
    by (elim S.in-homE, intro S.in-homI) auto

  lemma interchange:
  assumes S.seq ν μ and S.seq τ σ and μ ⋆ σ ≠ null
  shows (ν ·_S μ) ⋆ (τ ·_S σ) = (ν ⋆ τ) ·_S (μ ⋆ σ)
  proof −
    have 1: ν ⋆ τ ≠ null
      using assms hom-connected(1) [of ν σ] hom-connected(2) [of ν τ] hom-connected(3−4)
            S.dom-simp S.cod-simp
      by fastforce
    have (ν ·_S μ) ⋆ (τ ·_S σ) = (ν · μ) ⋆ (τ · σ)
      using assms S.comp-char S.seq-char_SbC by metis
    also have ... = (ν ⋆ τ) · (μ ⋆ σ)
      using assms interchange S.seq-char_SbC S.arr-char_SbC by simp
    also have ... = (ν ⋆ τ) ·_S (μ ⋆ σ)
      using assms 1
```

by (*metis* $S.arr\text{-}char_{SbC}$ *S.comp-char* $S.seq\text{-}char_{SbC}$ *ext match-3 right-def*)
finally show *?thesis* **by** *blast*
qed

lemma *inv-char*:
assumes $S.arr\ \varphi$ **and** $iso\ \varphi$
shows $S.inverse\text{-}arrows\ \varphi\ (inv\ \varphi)$
and $S.inv\ \varphi = inv\ \varphi$
proof −
have *1*: $S.arr\ (inv\ \varphi)$
using *assms* $S.arr\text{-}char_{SbC}$ *right-iff-right-inv*
by (*intro* $S.arrI_{SbC}$) *meson*
show $S.inv\ \varphi = inv\ \varphi$
using *assms 1* $S.inv\text{-}char_{SbC}$ $S.iso\text{-}char_{SbC}$ **by** *blast*
thus $S.inverse\text{-}arrows\ \varphi\ (inv\ \varphi)$
using *assms 1* $S.iso\text{-}char_{SbC}$ *S.inv-is-inverse* **by** *metis*
qed

lemma *iso-char*:
assumes $S.arr\ \varphi$
shows $S.iso\ \varphi \longleftrightarrow iso\ \varphi$
using *assms* $S.iso\text{-}char_{SbC}$ *inv-char* **by** *auto*

end

1.1.3 Weak Units

We now define a *weak unit* to be an arrow a such that:

1. $a \star a$ is isomorphic to a (and hence a is a horizontally self-composable 1-cell).
2. Horizontal composition on the left with a is a fully faithful endofunctor of the subcategory of arrows that are composable on the left with a.
3. Horizontal composition on the right with a is fully faithful endofunctor of the subcategory of arrows that are composable on the right with a.

context *weak-composition*
begin

definition *weak-unit* :: $'a \Rightarrow bool$
where $weak\text{-}unit\ a \equiv a \star a \cong a \wedge$
$fully\text{-}faithful\text{-}functor\ (Left\ a)\ (Left\ a)\ (H_L\ a) \wedge$
$fully\text{-}faithful\text{-}functor\ (Right\ a)\ (Right\ a)\ (H_R\ a)$

lemma *weak-unit-self-composable*:
assumes *weak-unit a*
shows $ide\ a$ **and** $ide\ (a \star a)$ **and** $a \star a \neq null$
proof −
obtain φ **where** φ: «$\varphi : a \star a \Rightarrow a$» $\wedge\ iso\ \varphi$

using *assms weak-unit-def isomorphic-def* **by** *blast*
have *1*: *arr* φ **using** φ **by** *blast*
show *ide a* **using** φ *ide-cod* **by** *blast*
thus *ide* $(a \star a)$ **using** φ *ide-dom* **by** *force*
thus $a \star a \neq null$ **using** *not-arr-null ideD*(*1*) **by** *metis*
qed

lemma *weak-unit-self-right*:
assumes *weak-unit a*
shows *right a a*
using *assms weak-unit-self-composable right-def* **by** *simp*

lemma *weak-unit-self-left*:
assumes *weak-unit a*
shows *left a a*
using *assms weak-unit-self-composable left-def* **by** *simp*

lemma *weak-unit-in-vhom*:
assumes *weak-unit a*
shows «$a : a \Rightarrow a$»
using *assms weak-unit-self-composable left-def* **by** *auto*

If a is a weak unit, then there exists a "unit isomorphism" «$\iota : a \star a \Rightarrow a$». It need not be unique, but we may choose one arbitrarily.

definition *some-unit*
where *some-unit* $a \equiv$ *SOME* ι. *iso* $\iota \wedge$ «$\iota : a \star a \Rightarrow a$»

lemma *iso-some-unit*:
assumes *weak-unit a*
shows *iso* (*some-unit a*)
and «*some-unit* $a : a \star a \Rightarrow a$»
proof −
let *?P* $= \lambda\iota$. *iso* $\iota \wedge$ «$\iota : a \star a \Rightarrow a$»
have $\exists\iota$. *?P* ι
using *assms weak-unit-def* **by** *auto*
hence *1*: *?P* (*some-unit a*)
using *someI-ex* [*of ?P*] *some-unit-def* **by** *simp*
show *iso* (*some-unit a*) **using** *1* **by** *blast*
show «*some-unit* $a : a \star a \Rightarrow a$» **using** *1* **by** *blast*
qed

The *sources* of an arbitrary arrow μ are the weak units that are composable with μ on the right. Similarly, the *targets* of μ are the weak units that are composable with μ on the left.

definition *sources*
where *sources* $\mu \equiv \{a.$ *weak-unit* $a \wedge \mu \star a \neq null\}$

lemma *sourcesI* [*intro*]:
assumes *weak-unit a* **and** $\mu \star a \neq null$

shows $a \in$ *sources* μ
  using *assms sources-def* **by** *blast*

lemma *sourcesD* [*dest*]:
assumes $a \in$ *sources* μ
shows *ide a* **and** *weak-unit a* **and** $\mu \star a \neq$ *null*
  using *assms sources-def weak-unit-self-composable* **by** *auto*

definition *targets*
where *targets* $\mu \equiv \{b.$ *weak-unit* $b \wedge b \star \mu \neq$ *null*$\}$

lemma *targetsI* [*intro*]:
assumes *weak-unit b* **and** $b \star \mu \neq$ *null*
shows $b \in$ *targets* μ
  using *assms targets-def* **by** *blast*

lemma *targetsD* [*dest*]:
assumes $b \in$ *targets* μ
shows *ide b* **and** *weak-unit b* **and** $b \star \mu \neq$ *null*
  using *assms targets-def weak-unit-self-composable* **by** *auto*

lemma *sources-dom* [*simp*]:
assumes *arr* μ
shows *sources* (*dom* μ) = *sources* μ
  using *assms hom-connected*(*1*) **by** *blast*

lemma *sources-cod* [*simp*]:
assumes *arr* μ
shows *sources* (*cod* μ) = *sources* μ
  using *assms hom-connected*(*3*) **by** *blast*

lemma *targets-dom* [*simp*]:
assumes *arr* μ
shows *targets* (*dom* μ) = *targets* μ
  using *assms hom-connected*(*2*) **by** *blast*

lemma *targets-cod* [*simp*]:
assumes *arr* μ
shows *targets* (*cod* μ) = *targets* μ
  using *assms hom-connected*(*4*) **by** *blast*

lemma *weak-unit-iff-self-source*:
shows *weak-unit a* $\longleftrightarrow$ $a \in$ *sources a*
  using *weak-unit-self-composable* **by** *auto*

lemma *weak-unit-iff-self-target*:
shows *weak-unit b* $\longleftrightarrow$ $b \in$ *targets b*
  using *weak-unit-self-composable* **by** *auto*

abbreviation (*input*) *in-hhom*$_{WC}$ (‹«- : - $\rightarrow_{WC}$ -»›)
where *in-hhom*$_{WC}$ μ f g $\equiv$ *arr* μ $\wedge$ $f \in$ *sources* μ $\wedge$ $g \in$ *targets* μ

lemma *sources-hcomp*:
assumes $\nu \star \mu \neq$ *null*
shows *sources* ($\nu \star \mu$) = *sources* μ
  using *assms match-1 match-3 null-agreement* **by** *blast*

lemma *targets-hcomp*:
assumes $\nu \star \mu \neq$ *null*
shows *targets* ($\nu \star \mu$) = *targets* ν
  using *assms match-2 match-4 null-agreement* **by** *blast*

lemma H_R*-preserved-along-iso*:
assumes *weak-unit* a **and** $a \cong a'$
shows *endofunctor* (*Right* a) (H_R a')
proof −
  have a: *ide* a $\wedge$ *weak-unit* a **using** *assms isomorphic-def* **by** *auto*
  have a': *ide* a' **using** *assms isomorphic-def* **by** *auto*

  interpret R: *subcategory* V ‹*right* a› **using** *a right-hom-is-subcategory* **by** *simp*
  have $*$: $\bigwedge\mu$. *R.arr* $\mu \Longrightarrow H_R$ a' $\mu = \mu \star a'$
    using *assms* H_R*-def* **by** *simp*
  have *preserves-arr*: $\bigwedge\mu$. *R.arr* $\mu \Longrightarrow$ *R.arr* (H_R a' μ)
    using *assms* $a' *$ *R.arr-char*$_{SbC}$ *right-def weak-unit-def weak-unit-self-composable*
      isomorphic-implies-equicomposable R.ide-char match-3 hcomp-simps$_{WC}$(*1*)
      null-agreement
    by *metis*
  show *endofunctor R.comp* (H_R a')
  proof
    show $\bigwedge\mu$. $\neg$ *R.arr* $\mu \Longrightarrow H_R$ a' μ = *R.null*
      using *assms R.arr-char*$_{SbC}$ *R.null-char right-def* H_R*-def null-agreement*
        right-respects-isomorphic
      by *metis*
    fix μ
    assume *R.arr* μ
    hence μ: *R.arr* μ $\wedge$ *arr* μ $\wedge$ *right* a μ $\wedge$ *right* a' μ $\wedge$ $\mu \star a \neq$ *null* $\wedge$ $\mu \star a' \neq$ *null*
    using *assms R.arr-char*$_{SbC}$ *right-respects-isomorphic composable-implies-arr null-agreement*
        right-def
      by *metis*
    show *R.arr* (H_R a' μ) **using** μ *preserves-arr* **by** *blast*
    show *R.dom* (H_R a' μ) = H_R a' (*R.dom* μ)
      using a' $\mu *$ *R.arr-char*$_{SbC}$ *R.dom-char*$_{SbC}$ *preserves-arr hom-connected*(*1*) *right-def*
      by *simp*
    show *R.cod* (H_R a' μ) = H_R a' (*R.cod* μ)
      using a' $\mu *$ *R.arr-char*$_{SbC}$ *R.cod-char*$_{SbC}$ *preserves-arr hom-connected*(*3*) *right-def*
      by *simp*
    next
    fix μ ν

assume $\mu\nu$: *R.seq* ν μ
have μ: *R.arr* $\mu \wedge$ *arr* $\mu \wedge$ *right a* $\mu \wedge$ *right* a' $\mu \wedge \mu \star a \neq$ *null* $\wedge \mu \star a' \neq$ *null*
using *assms* $\mu\nu$ *R.arr-char*$_{SbC}$ *right-respects-isomorphic composable-implies-arr null-agreement right-def*
by (*elim R.seqE*) *metis*
have ν: «ν : *R.cod* $\mu \rightarrow$ *R.cod* ν» $\wedge$ *arr* $\nu \wedge$
right a $\nu \wedge H\ \nu\ a \neq$ *null* $\wedge$ *right* a' $\nu \wedge H\ \nu\ a' \neq$ *null*
by (*metis* $*$ *R.cod-simp R.comp-def R.inclusion R.not-arr-null R.null-char R.seqE* $\mu\nu$ *in-homI preserves-arr right-def*)
show $H_R\ a'$ (*R.comp* ν μ) = *R.comp* ($H_R\ a'\ \nu$) ($H_R\ a'\ \mu$)
proof −
have *1*: *R.arr* ($H_R\ a'\ \nu$)
using ν *preserves-arr* **by** *blast*
have *2*: *seq* ($\nu \star a'$) ($\mu \star a'$)
using a' μ ν *R.arr-char*$_{SbC}$ *R.inclusion R.dom-char*$_{SbC}$ *R.cod-char*$_{SbC}$ *isomorphic-implies-equicomposable*
by *auto*
show *?thesis*
using a' μ ν $\mu\nu$ *1 2 preserves-arr* H_R*-def R.dom-simp R.cod-simp R.comp-char R.seq-char*$_{SbC}$ *R.inclusion whisker-right*
by *metis*
qed
qed
qed

lemma H_L*-preserved-along-iso*:
assumes *weak-unit a* **and** $a \cong a'$
shows *endofunctor* (*Left a*) ($H_L\ a'$)
proof −
have *a*: *ide a* $\wedge$ *weak-unit a* **using** *assms isomorphic-def* **by** *auto*
have a': *ide* a' **using** *assms isomorphic-def* **by** *auto*

interpret *L*: *subcategory V* ‹*left a*› **using** *a left-hom-is-subcategory* **by** *simp*
have $*$: $\bigwedge\mu$. *L.arr* $\mu \Longrightarrow H_L\ a'\ \mu = a' \star \mu$
using *assms* H_L*-def* **by** *simp*
have *preserves-arr*: $\bigwedge\mu$. *L.arr* $\mu \Longrightarrow$ *L.arr* ($H_L\ a'\ \mu$)
using *assms* a' $*$ *L.arr-char*$_{SbC}$ *left-def weak-unit-def weak-unit-self-composable isomorphic-implies-equicomposable L.ide-char match-4 hcomp-simps*$_{WC}$*(1) null-agreement*
by *metis*
show *endofunctor L.comp* ($H_L\ a'$)
proof
show $\bigwedge\mu$. $\neg$ *L.arr* $\mu \Longrightarrow H_L\ a'\ \mu =$ *L.null*
using *assms L.arr-char*$_{SbC}$ *L.null-char left-def* H_L*-def null-agreement left-respects-isomorphic*
by *metis*
fix μ
assume *L.arr* μ
hence μ: *L.arr* $\mu \wedge$ *arr* $\mu \wedge$ *left a* $\mu \wedge$ *left* a' $\mu \wedge a \star \mu \neq$ *null* $\wedge a' \star \mu \neq$ *null*

```
      using assms L.arr-char_SbC left-respects-isomorphic composable-implies-arr null-agreement
            left-def
        by metis
      show L.arr (H_L a' μ) using μ preserves-arr by blast
      show L.dom (H_L a' μ) = H_L a' (L.dom μ)
        using a' μ ∗ L.arr-char_SbC L.dom-char_SbC preserves-arr hom-connected(2) left-def
        by simp
      show L.cod (H_L a' μ) = H_L a' (L.cod μ)
        using a' μ ∗ L.arr-char_SbC L.cod-char_SbC preserves-arr hom-connected(4) left-def
        by simp
      next
      fix μ ν
      assume μν: L.seq ν μ
      have L.arr μ
        using μν by (elim L.seqE, auto)
      hence μ: L.arr μ ∧ arr μ ∧ left a μ ∧ left a' μ ∧ a ⋆ μ ≠ null ∧ a' ⋆ μ ≠ null
      using assms L.arr-char_SbC left-respects-isomorphic composable-implies-arr null-agreement
            left-def
        by metis
      have ν: «ν : L.cod μ ⇒ L.cod ν» ∧ arr ν ∧
              left a ν ∧ a ⋆ ν ≠ null ∧ left a' ν ∧ a' ⋆ ν ≠ null
        by (metis (mono-tags, opaque-lifting) L.arrE L.cod-simp L.seq-char_SbC μν assms(2)
            in-homI seqE left-def left-respects-isomorphic)
      show H_L a' (L.comp ν μ) = L.comp (H_L a' ν) (H_L a' μ)
      proof −
        have 1: L.arr (H_L a' ν)
          using ν preserves-arr by blast
        have 2: seq (a' ⋆ ν) (a' ⋆ μ)
          using a' μ ν L.arr-char_SbC L.inclusion L.dom-char_SbC L.cod-char_SbC
                isomorphic-implies-equicomposable
          by auto
        show ?thesis
          using a' μ ν μν 1 2 preserves-arr H_L-def L.dom-simp L.cod-simp L.comp-char
                L.seq-char_SbC L.inclusion whisker-left
          by metis
      qed
    qed
  qed

end
```

1.1.4 Regularity

We call a weak composition *regular* if $f \star a \cong f$ whenever a is a source of 1-cell f, and $b \star f \cong f$ whenever b is a target of f. A consequence of regularity is that horizontal composability of 2-cells is fully determined by their sets of sources and targets.

```
locale regular-weak-composition =
  weak-composition +
assumes comp-ide-source: ⟦ a ∈ sources f; ide f ⟧ ⟹ f ⋆ a ≅ f
```

and *comp-target-ide*: $[\![\ b \in \mathit{targets}\ f;\ \mathit{ide}\ f\ ]\!] \Longrightarrow b \star f \cong f$
begin

lemma *sources-determine-composability*:
assumes $a \in \mathit{sources}\ \tau$
shows $\tau \star \mu \neq \mathit{null} \longleftrightarrow a \star \mu \neq \mathit{null}$
proof −
have $*$: $\bigwedge\tau.\ \mathit{ide}\ \tau \wedge a \in \mathit{sources}\ \tau \Longrightarrow \tau \star \mu \neq \mathit{null} \longleftrightarrow a \star \mu \neq \mathit{null}$
using *assms comp-ide-source isomorphic-implies-equicomposable match-1 match-3*
by (*meson sourcesD*(*3*))
show *?thesis*
proof −
have $\mathit{arr}\ \tau$ **using** *assms composable-implies-arr* **by** *auto*
thus *?thesis*
using *assms* $*$ [*of dom* τ] *hom-connected*(*1*) **by** *auto*
qed
qed

lemma *targets-determine-composability*:
assumes $b \in \mathit{targets}\ \mu$
shows $\tau \star \mu \neq \mathit{null} \longleftrightarrow \tau \star b \neq \mathit{null}$
proof −
have $*$: $\bigwedge\mu.\ \mathit{ide}\ \mu \wedge b \in \mathit{targets}\ \mu \Longrightarrow \tau \star \mu \neq \mathit{null} \longleftrightarrow \tau \star b \neq \mathit{null}$
using *assms comp-target-ide isomorphic-implies-equicomposable match-2 match-4*
by (*meson targetsD*(*3*))
show *?thesis*
proof −
have $\mathit{arr}\ \mu$ **using** *assms composable-implies-arr* **by** *auto*
thus *?thesis*
using *assms* $*$ [*of dom* μ] *hom-connected*(*2*) **by** *auto*
qed
qed

lemma *composable-if-connected*:
assumes $\mathit{sources}\ \nu \cap \mathit{targets}\ \mu \neq \{\}$
shows $\nu \star \mu \neq \mathit{null}$
using *assms targets-determine-composability* **by** *blast*

lemma *connected-if-composable*:
assumes $\nu \star \mu \neq \mathit{null}$
shows $\mathit{sources}\ \nu = \mathit{targets}\ \mu$
using *assms sources-determine-composability targets-determine-composability* **by** *blast*

lemma *iso-hcomp*$_{RWC}$:
assumes $\mathit{iso}\ \mu$ **and** $\mathit{iso}\ \nu$ **and** $\mathit{sources}\ \nu \cap \mathit{targets}\ \mu \neq \{\}$
shows $\mathit{iso}\ (\nu \star \mu)$
and *inverse-arrows* $(\nu \star \mu)$ $(\mathit{inv}\ \nu \star \mathit{inv}\ \mu)$
proof −
have μ: $\mathit{arr}\ \mu \wedge «\mu : \mathit{dom}\ \mu \Rightarrow \mathit{cod}\ \mu» \wedge$

iso μ $\wedge$ «*inv* μ : *cod* μ $\Rightarrow$ *dom* μ»
using *assms inv-in-hom arr-iff-in-hom iso-is-arr* **by** *auto*
have ν: *arr* ν $\wedge$ «ν : *dom* ν $\Rightarrow$ *cod* ν» $\wedge$
iso ν $\wedge$ «*inv* ν : *cod* ν $\Rightarrow$ *dom* ν»
using *assms inv-in-hom* **by** *blast*
have *1*: *sources* (*inv* ν) $\cap$ *targets* (*inv* μ) $\neq$ {}
using *assms* μ ν *sources-dom sources-cod targets-dom targets-cod arr-inv cod-inv*
by *metis*
show *inverse-arrows* (ν $\star$ μ) (*inv* ν $\star$ *inv* μ)
using *assms 1* μ ν *inv-in-hom inv-is-inverse comp-inv-arr*
interchange [*of inv* ν ν *inv* μ μ] *composable-if-connected*
ide-hcomp$_{WC}$ *sources-dom targets-dom interchange* [*of* ν *inv* ν μ *inv* μ]
ide-hcomp$_{WC}$ *sources-cod targets-cod ide-compE ide-dom comp-arr-inv'*
inverse-arrowsE seqI' inverse-arrowsI
by *metis*
thus *iso* (ν $\star$ μ) **by** *auto*
qed

lemma *inv-hcomp*$_{RWC}$:
assumes *iso* μ **and** *iso* ν **and** *sources* ν $\cap$ *targets* μ $\neq$ {}
shows *inv* (ν $\star$ μ) = *inv* ν $\star$ *inv* μ
using *assms iso-hcomp*$_{RWC}$(*2*) [*of* μ ν] *inverse-arrow-unique* [*of H* ν μ] *inv-is-inverse*
by *auto*

end

1.1.5 Associativity

An *associative weak composition* consists of a weak composition that has been equipped with an *associator* isomorphism: «a[f, g, h] : (f $\star$ g) $\star$ h $\Rightarrow$ f $\star$ g $\star$ h» for each composable triple (f, g, h) of 1-cells, subject to naturality and coherence conditions.

locale *associative-weak-composition* =
weak-composition +
fixes a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
assumes *assoc-in-vhom*$_{AWC}$:
⟦ *ide f*; *ide g*; *ide h*; f $\star$ g $\neq$ *null*; g $\star$ h $\neq$ *null* ⟧ $\Longrightarrow$
«a[f, g, h] : (f $\star$ g) $\star$ h $\Rightarrow$ f $\star$ g $\star$ h»
and *assoc-naturality*$_{AWC}$:
⟦ τ $\star$ μ $\neq$ *null*; μ $\star$ ν $\neq$ *null* ⟧ $\Longrightarrow$
a[*cod* τ, *cod* μ, *cod* ν] $\cdot$ ((τ $\star$ μ) $\star$ ν) = (τ $\star$ μ $\star$ ν) $\cdot$ a[*dom* τ, *dom* μ, *dom* ν]
and *iso-assoc*$_{AWC}$: ⟦ *ide f*; *ide g*; *ide h*; f $\star$ g $\neq$ *null*; g $\star$ h $\neq$ *null* ⟧ $\Longrightarrow$ *iso* a[f, g, h]
and *pentagon*$_{AWC}$:
⟦ *ide f*; *ide g*; *ide h*; *ide k*; *sources f* $\cap$ *targets g* $\neq$ {};
sources g $\cap$ *targets h* $\neq$ {}; *sources h* $\cap$ *targets k* $\neq$ {} ⟧ $\Longrightarrow$
(f $\star$ a[g, h, k]) $\cdot$ a[f, g $\star$ h, k] $\cdot$ (a[f, g, h] $\star$ k) = a[f, g, h $\star$ k] $\cdot$ a[f $\star$ g, h, k]
begin

lemma *assoc-in-hom*$_{AWC}$:
assumes *ide f* **and** *ide g* **and** *ide h*

and $f \star g \neq null$ **and** $g \star h \neq null$
shows *sources* a[f, g, h] = *sources* h **and** *targets* a[f, g, h] = *targets* f
and «a[f, g, h] : $(f \star g) \star h \Rightarrow f \star g \star h$»
proof −
  show *1*: «a[f, g, h] : $(f \star g) \star h \Rightarrow f \star g \star h$»
    using *assms assoc-in-vhom*$_{AWC}$ **by** *simp*
  show *sources* a[f, g, h] = *sources* h
    using *assms 1 sources-dom* [*of* a[f, g, h]] *sources-hcomp match-3*
    by (*elim in-homE*, *auto*)
  show *targets* a[f, g, h] = *targets* f
    using *assms 1 targets-cod* [*of* a[f, g, h]] *targets-hcomp match-4*
    by (*elim in-homE*, *auto*)
qed

lemma *assoc-simps*$_{AWC}$ [*simp*]:
assumes *ide* f **and** *ide* g **and** *ide* h
and $f \star g \neq null$ **and** $g \star h \neq null$
shows *arr* a[f, g, h]
and *dom* a[f, g, h] = $(f \star g) \star h$
and *cod* a[f, g, h] = $f \star g \star h$
proof −
  have *1*: «a[f, g, h] : $(f \star g) \star h \Rightarrow f \star g \star h$»
    using *assms assoc-in-hom*$_{AWC}$ **by** *auto*
  show *arr* a[f, g, h] **using** *1* **by** *auto*
  show *dom* a[f, g, h] = $(f \star g) \star h$ **using** *1* **by** *auto*
  show *cod* a[f, g, h] = $f \star g \star h$ **using** *1* **by** *auto*
qed

lemma *assoc′-in-hom*$_{AWC}$:
assumes *ide* f **and** *ide* g **and** *ide* h
and $f \star g \neq null$ **and** $g \star h \neq null$
shows *sources* (*inv* a[f, g, h]) = *sources* h **and** *targets* (*inv* a[f, g, h]) = *targets* f
and «*inv* a[f, g, h] : $f \star g \star h \Rightarrow (f \star g) \star h$»
proof −
  show *1*: «*inv* a[f, g, h] : $f \star g \star h \Rightarrow (f \star g) \star h$»
    using *assms assoc-in-hom*$_{AWC}$ *iso-assoc*$_{AWC}$ *inv-in-hom* **by** *auto*
  show *sources* (*inv* a[f, g, h]) = *sources* h
    using *assms 1 sources-hcomp* [*of* $f \star g$ h] *sources-cod match-3 null-agreement*
    by (*elim in-homE*, *metis*)
  show *targets* (*inv* a[f, g, h]) = *targets* f
    using *assms 1 targets-hcomp* [*of* f $g \star h$] *targets-dom match-4 null-agreement*
    by (*elim in-homE*, *metis*)
qed

lemma *assoc′-simps*$_{AWC}$ [*simp*]:
assumes *ide* f **and** *ide* g **and** *ide* h
and $f \star g \neq null$ **and** $g \star h \neq null$
shows *arr* (*inv* a[f, g, h])
and *dom* (*inv* a[f, g, h]) = $f \star g \star h$

and *cod* (*inv* a[f, g, h]) = ($f \star g$) $\star$ h
proof −
have *1*: «*inv* a[f, g, h] : $f \star g \star h \Rightarrow (f \star g) \star h$»
using *assms assoc′-in-hom$_{AWC}$* **by** *auto*
show *arr* (*inv* a[f, g, h]) **using** *1* **by** *auto*
show *dom* (*inv* a[f, g, h]) = $f \star g \star h$ **using** *1* **by** *auto*
show *cod* (*inv* a[f, g, h]) = ($f \star g$) $\star$ h **using** *1* **by** *auto*
qed

lemma *assoc′-naturality$_{AWC}$*:
assumes $\tau \star \mu \neq$ *null* **and** $\mu \star \nu \neq$ *null*
shows *inv* a[*cod* τ, *cod* μ, *cod* ν] · ($\tau \star \mu \star \nu$) = (($\tau \star \mu$) $\star$ ν) · *inv* a[*dom* τ, *dom* μ, *dom* ν]
proof −
have $\tau\mu\nu$: *arr* τ ∧ *arr* μ ∧ *arr* ν
using *assms composable-implies-arr* **by** *simp*
have *0*: *dom* $\tau \star$ *dom* $\mu \neq$ *null* ∧ *dom* $\mu \star$ *dom* $\nu \neq$ *null* ∧
cod $\tau \star$ *cod* $\mu \neq$ *null* ∧ *cod* $\mu \star$ *cod* $\nu \neq$ *null*
using *assms* $\tau\mu\nu$ *hom-connected* **by** *simp*
have *1*: «$\tau \star \mu \star \nu$: *dom* $\tau \star$ *dom* $\mu \star$ *dom* $\nu \Rightarrow$ *cod* $\tau \star$ *cod* $\mu \star$ *cod* ν»
using *assms match-4* **by** *auto*
have *2*: «($\tau \star \mu$) $\star$ ν : (*dom* $\tau \star$ *dom* μ) $\star$ *dom* $\nu \Rightarrow$ (*cod* $\tau \star$ *cod* μ) $\star$ *cod* ν»
using *assms match-3* **by** *auto*
have (*inv* a[*cod* τ, *cod* μ, *cod* ν] · ($\tau \star \mu \star \nu$)) · a[*dom* τ, *dom* μ, *dom* ν] = ($\tau \star \mu$) $\star$ ν
proof −
have ($\tau \star \mu$) $\star$ ν = (*inv* a[*cod* τ, *cod* μ, *cod* ν] · a[*cod* τ, *cod* μ, *cod* ν]) · (($\tau \star \mu$) $\star$ ν)
using *0 2* $\tau\mu\nu$ *assoc-in-hom$_{AWC}$ iso-assoc$_{AWC}$ comp-inv-arr inv-is-inverse comp-cod-arr*
by *auto*
thus *?thesis*
using *assms* $\tau\mu\nu$ *0 2 assoc-naturality$_{AWC}$ comp-assoc* **by** *metis*
qed
thus *?thesis*
using *0 1 2* $\tau\mu\nu$ *iso-assoc$_{AWC}$ assoc′-in-hom$_{AWC}$ inv-in-hom invert-side-of-triangle*(*2*)
by *auto*
qed

end

1.1.6 Unitors

For an associative weak composition with a chosen unit isomorphism $\iota : a \star a \Rightarrow a$, where a is a weak unit, horizontal composition on the right by a is a fully faithful endofunctor R of the subcategory of arrows composable on the right with a, and is consequently an endo-equivalence of that subcategory. This equivalence, together with the associator isomorphisms and unit isomorphism ι, canonically associate, with each identity arrow f composable on the right with a, a *right unit* isomorphism «r[f] : $f \star a \Rightarrow f$». These isomorphisms are the components of a natural isomorphism from R to the identity functor.

locale *right-hom-with-unit* =

associative-weak-composition V H a +
right-hom V H a
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a$ *comp* (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and ι :: $'a$
and a :: $'a$ +
assumes *weak-unit-a*: *weak-unit* a
and *ι-in-hom*: «$\iota : a \star a \Rightarrow a$»
and *iso-ι*: *iso* ι
begin

abbreviation R
where $R \equiv H_R\ a$

interpretation R: *endofunctor S.comp R*
using *weak-unit-a weak-unit-self-composable endofunctor-H_R* **by** *simp*
interpretation R: *fully-faithful-functor S.comp S.comp R*
using *weak-unit-a weak-unit-def* **by** *simp*

lemma *fully-faithful-functor-R*:
shows *fully-faithful-functor S.comp S.comp R*
..

definition *runit* (‹r[-]›)
where *runit* $f \equiv$ *THE* μ. «$\mu : R\ f \Rightarrow_S f$» $\wedge$ $R\ \mu = (f \star \iota) \cdot_S$ a$[f, a, a]$

lemma *iso-unit*:
shows *S.iso* ι **and** «$\iota : a \star a \Rightarrow_S a$»
proof −
show «$\iota : a \star a \Rightarrow_S a$»
using *weak-unit-a S.ide-char S.arr-char$_{SbC}$ right-def weak-unit-self-composable S.ideD(1) R.preserves-arr H_R-def S.in-hom-char$_{SbC}$ right-def ι-in-hom S.ideD(1) hom-connected(3) in-homE*
by *metis*
thus *S.iso* ι
using *iso-ι iso-char* **by** *blast*
qed

lemma *characteristic-iso*:
assumes *S.ide f*
shows «a$[f, a, a] : (f \star a) \star a \Rightarrow_S f \star a \star a$»
and «$f \star \iota : f \star a \star a \Rightarrow_S f \star a$»
and «$(f \star \iota) \cdot_S$ a$[f, a, a] : R\ (R\ f) \Rightarrow_S R\ f$»
and *S.iso* $((f \star \iota) \cdot_S$ a$[f, a, a])$
proof −
have f: *S.ide f* $\wedge$ *ide f*
using *assms S.ide-char$_{SbC}$* **by** *simp*
have a: *weak-unit a* $\wedge$ *ide a* $\wedge$ *S.ide a*

using *weak-unit-a S.ide-char$_{SbC}$ weak-unit-def S.arr-char$_{SbC}$ right-def weak-unit-self-composable*
by *metis*
have *fa*: $f \star a \neq null \wedge (f \star a) \star a \neq null \wedge ((f \star a) \star a) \star a \neq null$
using *assms S.ideD(1) R.preserves-arr H_R-def S.not-arr-null S.null-char*
by *metis*
have *aa*: $a \star a \neq null$
using *a S.ideD(1) R.preserves-arr H_R-def S.not-arr-null weak-unit-self-composable*
by *auto*
have *f-ia*: $f \star \iota \neq null$
using *assms S.ide-char right-def S.arr-char$_{SbC}$ hom-connected(4) ι-in-hom* **by** *auto*
show *assoc-in-hom*: «$\mathrm{a}[f, a, a] : (f \star a) \star a \Rightarrow_S f \star a \star a$»
using *a f fa hom-connected(1) [of* $\mathrm{a}[f, a, a]$ *a] S.arr-char$_{SbC}$ right-def match-3 match-4 S.in-hom-char$_{SbC}$ weak-unit-self-composable*
by *auto*
show *1*: «$f \star \iota : f \star a \star a \Rightarrow_S f \star a$»
using *a f fa iso-unit left-hcomp-closed*
by (*simp add*: *f-ia ide-in-hom*)
have *unit-part*: «$f \star \iota : f \star a \star a \Rightarrow_S f \star a$» $\wedge$ *S.iso* $(f \star \iota)$
proof −
have *S.iso* $(f \star \iota)$
using *a f fa f-ia 1 VoV.arr-char$_{SbC}$ VxV.inv-simp inv-in-hom hom-connected(2) [of f inv ι] VoV.arr-char$_{SbC}$ VoV.iso-char$_{SbC}$ preserves-iso iso-char iso-ι S.dom-simp S.cod-simp*
by *auto*
thus *?thesis* **using** *1* **by** *blast*
qed
show *S.iso* $((f \star \iota) \cdot_S \mathrm{a}[f, a, a])$
using *assms a f fa aa hom-connected(1) [of* $\mathrm{a}[f, a, a]$ *a] right-def iso-assoc$_{AWC}$ iso-char S.arr-char$_{SbC}$ unit-part assoc-in-hom isos-compose S.isos-compose S.seqI′* **by** *auto*
show «$(f \star \iota) \cdot_S \mathrm{a}[f, a, a] : R\ (R\ f) \Rightarrow_S R\ f$»
unfolding *H_R-def* **using** *unit-part assoc-in-hom* **by** *blast*
qed

lemma *runit-char*:
assumes *S.ide f*
shows «$\mathrm{r}[f] : R\ f \Rightarrow_S f$» **and** $R\ \mathrm{r}[f] = (f \star \iota) \cdot_S \mathrm{a}[f, a, a]$
and $\exists!\mu.$ «$\mu : R\ f \Rightarrow_S f$» $\wedge$ $R\ \mu = (f \star \iota) \cdot_S \mathrm{a}[f, a, a]$
proof −
let $?P = \lambda\mu.$ «$\mu : R\ f \Rightarrow_S f$» $\wedge$ $R\ \mu = (f \star \iota) \cdot_S \mathrm{a}[f, a, a]$
show $\exists!\mu.\ ?P\ \mu$
proof −
have $\exists\mu.\ ?P\ \mu$
using *assms S.ide-char$_{SbC}$ S.arr-char$_{SbC}$ R.preserves-ide characteristic-iso(3) R.is-full*
by *auto*
moreover have $\forall\mu\ \mu'.\ ?P\ \mu \wedge ?P\ \mu' \longrightarrow \mu = \mu'$
using *R.is-faithful S.in-homE* **by** *metis*
ultimately show *?thesis* **by** *blast*

```
  qed
  hence ?P (THE μ. ?P μ)
    using theI' [of ?P] by fastforce
  hence 1: ?P r[f]
    unfolding runit-def by blast
  show «r[f] : R f ⇒S f» using 1 by fast
  show R r[f] = (f ⋆ ι) ·S a[f, a, a] using 1 by fast
qed

lemma iso-runit:
assumes S.ide f
shows S.iso r[f]
  using assms characteristic-iso(4) runit-char R.reflects-iso by metis

lemma runit-eqI:
assumes «f : a ⇒S b» and «μ : R f ⇒S f»
and R μ = ((f ⋆ ι) ·S a[f, a, a])
shows μ = r[f]
  using assms S.ide-cod runit-char [of f] by blast

lemma runit-naturality:
assumes S.arr μ
shows r[S.cod μ] ·S R μ = μ ·S r[S.dom μ]
proof −
  have 1: «r[S.cod μ] ·S R μ : R (S.dom μ) ⇒S S.cod μ»
    using assms runit-char(1) S.ide-cod by blast
  have 2: S.par (r[S.cod μ] ·S R μ) (μ ·S r[S.dom μ])
    using assms 1 S.ide-dom runit-char(1)
    by (metis S.comp-in-homI' S.in-homE)
  moreover have R (r[S.cod μ] ·S R μ) = R (μ ·S r[S.dom μ])
  proof −
    have 3: «μ ⋆ a ⋆ a : S.dom μ ⋆ a ⋆ a ⇒S S.cod μ ⋆ a ⋆ a»
      using assms weak-unit-a R.preserves-hom H_R-def S.arr-iff-in-hom S.arr-char_SbC
      by (metis match-4 weak-unit-in-vhom weak-unit-self-right S.in-hom-char_SbC
          left-hcomp-closed S.not-arr-null S.null-char)
    have 4: R (r[S.cod μ] ·S R μ) = R r[S.cod μ] ·S R (R μ)
      using assms 1 R.as-nat-trans.preserves-comp-2 by blast
    also have 5: ... = ((S.cod μ ⋆ ι) ·S a[S.cod μ, a, a]) ·S ((μ ⋆ a) ⋆ a)
      using assms R.preserves-arr runit-char S.ide-cod H_R-def by auto
    also have 6: ... = (S.cod μ ⋆ ι) ·S a[S.cod μ, a, a] ·S ((μ ⋆ a) ⋆ a)
      using assms S.comp-assoc by simp
    also have ... = (S.cod μ ⋆ ι) ·S (μ ⋆ a ⋆ a) ·S a[S.dom μ, a, a]
    proof −
      have (μ ⋆ a ⋆ a) ·S a[S.dom μ, a, a] = a[S.cod μ, a, a] ·S ((μ ⋆ a) ⋆ a)
      proof −
        have (μ ⋆ a ⋆ a) ·S a[S.dom μ, a, a] = (μ ⋆ a ⋆ a) · a[S.dom μ, a, a]
          using assms 3 S.ide-dom characteristic-iso(1) S.in-hom-char_SbC
                S.comp-char S.dom-simp
          by fastforce
```

```
      also have ... = a[S.cod μ, a, a] · ((μ ⋆ a) ⋆ a)
        using assms weak-unit-a assoc-naturality_AWC [of μ a a] S.dom-simp S.cod-simp
              weak-unit-self-composable S.arr-char_SbC right-def
        by simp
      also have ... = a[S.cod μ, a, a] ·_S ((μ ⋆ a) ⋆ a)
        using S.in-hom-char_SbC S.comp-char
        by (metis 2 4 5 6 R.preserves-arr S.seq-char_SbC)
      finally show ?thesis by blast
    qed
   thus ?thesis by argo
  qed
  also have ... = ((S.cod μ ⋆ ι) ·_S (μ ⋆ a ⋆ a)) ·_S a[S.dom μ, a, a]
    using S.comp-assoc by auto
  also have ... = ((μ ⋆ a) ·_S (S.dom μ ⋆ ι)) ·_S a[S.dom μ, a, a]
  proof −
    have μ ⋆ a ⋆ a ≠ null
      using 3 S.not-arr-null by auto
    moreover have S.dom μ ⋆ ι ≠ null
      using assms S.not-arr-null
      by (metis S.dom-char_SbC ι-in-hom calculation hom-connected(1−2) in-homE)
    ultimately have (S.cod μ ⋆ ι) ·_S (μ ⋆ a ⋆ a) = (μ ⋆ a) ·_S (S.dom μ ⋆ ι)
      using assms weak-unit-a iso-unit S.comp-arr-dom S.comp-cod-arr
            interchange [of S.cod μ μ ι a ⋆ a] interchange [of μ S.dom μ a ι]
      by auto
    thus ?thesis by argo
  qed
  also have ... = (μ ⋆ a) ·_S (S.dom μ ⋆ ι) ·_S a[S.dom μ, a, a]
    using S.comp-assoc by auto
  also have ... = R μ ·_S R r[S.dom μ]
    using assms runit-char(2) S.ide-dom H_R-def by auto
  also have ... = R (μ ·_S r[S.dom μ])
    using assms S.arr-iff-in-hom [of μ] runit-char(1) S.ide-dom by fastforce
  finally show ?thesis by blast
 qed
 ultimately show r[S.cod μ] ·_S (R μ) = μ ·_S r[S.dom μ]
  using R.is-faithful by blast
qed

abbreviation ɾ
where ɾ μ ≡ if S.arr μ then μ ·_S r[S.dom μ] else null

interpretation ɾ: natural-transformation S.comp S.comp R S.map ɾ
proof −
 interpret ɾ: transformation-by-components S.comp S.comp R S.map runit
  using runit-char(1) runit-naturality by unfold-locales simp-all
 have ɾ.map = ɾ
  using ɾ.extensionality ɾ.map-def ɾ.naturality ɾ.map-simp-ide S.ide-dom S.ide-cod
        S.map-def
  by auto
```

thus *natural-transformation S.comp S.comp R S.map* $\mathfrak{r}$
using $\mathfrak{r}$.*natural-transformation-axioms* **by** *auto*
qed

lemma *natural-transformation*-$\mathfrak{r}$:
shows *natural-transformation S.comp S.comp R S.map* $\mathfrak{r}$ **..**

interpretation $\mathfrak{r}$: *natural-isomorphism S.comp S.comp R S.map* $\mathfrak{r}$
using *S.ide-is-iso iso-runit runit-char*(*1*) *S.isos-compose*
by *unfold-locales force*

lemma *natural-isomorphism*-$\mathfrak{r}$:
shows *natural-isomorphism S.comp S.comp R S.map* $\mathfrak{r}$ **..**

interpretation *R*: *equivalence-functor S.comp S.comp R*
using *natural-isomorphism*-$\mathfrak{r}$ *R.isomorphic-to-identity-is-equivalence* **by** *blast*

lemma *equivalence-functor-R*:
shows *equivalence-functor S.comp S.comp R*
..

lemma *runit-commutes-with-R*:
assumes *S.ide f*
shows r[*R f*] = *R* r[*f*]
using *assms runit-char*(*1*) *R.preserves-hom* [*of* r[*f*] *R f f*]
runit-naturality iso-runit S.iso-is-section S.section-is-mono S.mono-cancel
by (*metis S.in-homE S.seqI'*)

end

Symmetric results hold for the subcategory of all arrows composable on the left with a specified weak unit *b*. This yields the *left unitors*.

locale *left-hom-with-unit* =
associative-weak-composition V H a +
left-hom V H b
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a comp* (**infixr** ‹⋆› *53*)
and a :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›)
and ι :: *'a*
and *b* :: *'a* +
assumes *weak-unit-b*: *weak-unit b*
and ι*-in-hom*: «ι : *b* ⋆ *b* ⇒ *b*»
and *iso-*ι: *iso* ι
begin

abbreviation *L*
where $L \equiv H_L\ b$

interpretation *L*: *endofunctor S.comp L*

using *weak-unit-b weak-unit-self-composable endofunctor-H_L* **by** *simp*
interpretation *L*: *fully-faithful-functor S.comp S.comp L*
using *weak-unit-b weak-unit-def* **by** *simp*

lemma *fully-faithful-functor-L*:
shows *fully-faithful-functor S.comp S.comp L*
..

definition *lunit* (‹l[-]›)
where *lunit f* $\equiv$ *THE* μ. «μ : *L f* $\Rightarrow_S$ *f*» $\wedge$ *L* μ = ($\iota \star f$) $\cdot_S$ (*inv* a[*b*, *b*, *f*])

lemma *iso-unit*:
shows *S.iso* ι **and** «ι : $b \star b \Rightarrow_S b$»
proof −
show «ι : $b \star b \Rightarrow_S b$»
using *weak-unit-b S.ide-char S.arr-char$_{SbC}$ left-def weak-unit-self-composable S.ideD(1) L.preserves-arr H_L-def S.in-hom-char$_{SbC}$ S.arr-char$_{SbC}$ left-def ι-in-hom S.ideD(1) hom-connected(4) in-homE*
by *metis*
thus *S.iso* ι
using *iso-ι iso-char* **by** *blast*
qed

lemma *characteristic-iso*:
assumes *S.ide f*
shows «*inv* a[*b*, *b*, *f*] : $b \star b \star f \Rightarrow_S (b \star b) \star f$»
and «$\iota \star f$: $(b \star b) \star f \Rightarrow_S b \star f$»
and «($\iota \star f$) $\cdot_S$ *inv* a[*b*, *b*, *f*] : *L* (*L f*) $\Rightarrow_S$ *L f*»
and *S.iso* (($\iota \star f$) $\cdot_S$ *inv* a[*b*, *b*, *f*])
proof −
have *f*: *S.ide f* $\wedge$ *ide f*
using *assms S.ide-char$_{SbC}$* **by** *simp*
have *b*: *weak-unit b* $\wedge$ *ide b* $\wedge$ *S.ide b*
using *weak-unit-b S.ide-char$_{SbC}$ weak-unit-def S.arr-char$_{SbC}$ left-def weak-unit-self-composable*
by *metis*
have *bf*: $b \star f \neq$ *null* $\wedge$ $b \star b \star b \star f \neq$ *null*
using *assms S.ideD(1) L.preserves-arr H_L-def S.not-arr-null S.null-char*
by *metis*
have *bb*: $b \star b \neq$ *null*
using *b S.ideD(1) L.preserves-arr H_L-def S.not-arr-null weak-unit-self-composable*
by *auto*
have *ib-f*: $\iota \star f \neq$ *null*
using *assms S.ide-char left-def S.arr-char$_{SbC}$ hom-connected(3) ι-in-hom*
by *auto*
show *assoc-in-hom*: «*inv* a[*b*, *b*, *f*] : $b \star b \star f \Rightarrow_S (b \star b) \star f$»
using *b f bf bb hom-connected(2)* [*of b inv* a[*b*, *b*, *f*]] *left-def*
by (*metis S.arrI$_{SbC}$ S.cod-closed S.in-hom-char$_{SbC}$ assoc′-in-hom$_{AWC}$(3) assoc′-simps$_{AWC}$(2−3)*)
show *1*: «$\iota \star f$: $(b \star b) \star f \Rightarrow_S b \star f$»

```
    using b f bf right-hcomp-closed
    by (simp add: ib-f ide-in-hom iso-unit(2))
  have unit-part: «ι ⋆ f : (b ⋆ b) ⋆ f ⇒S b ⋆ f» ∧ S.iso (ι ⋆ f)
  proof −
    have S.iso (ι ⋆ f)
      using b f bf ib-f 1 VoV.arr-charSbC VxV.inv-simp
          inv-in-hom hom-connected(1) [of inv ι f] VoV.arr-charSbC VoV.iso-charSbC
           preserves-iso iso-char iso-ι S.dom-simp S.cod-simp
      by auto
    thus ?thesis using 1 by blast
  qed
  show S.iso ((ι ⋆ f) ·S inv a[b, b, f])
    using assms b f bf bb hom-connected(2) [of b inv a[b, b, f]] left-def
          isoassocAWC iso-char S.arr-charSbC unit-part assoc-in-hom isos-compose
          S.isos-compose S.seqI′ by auto
  show «(ι ⋆ f) ·S inv a[b, b, f] : L (L f) ⇒S L f»
    unfolding HL-def using unit-part assoc-in-hom by blast
qed

lemma lunit-char:
assumes S.ide f
shows «l[f] : L f ⇒S f» and L l[f] = (ι ⋆ f) ·S inv a[b, b, f]
and ∃!μ. «μ : L f ⇒S f» ∧ L μ = (ι ⋆ f) ·S inv a[b, b, f]
proof −
  let ?P = λμ. «μ : L f ⇒S f» ∧ L μ = (ι ⋆ f) ·S inv a[b, b, f]
  show ∃!μ. ?P μ
  proof −
    have ∃μ. ?P μ
    proof −
      have 1: S.ide f
        using assms S.ide-charSbC S.arr-charSbC by simp
      moreover have S.ide (L f)
        using 1 L.preserves-ide by simp
      ultimately show ?thesis
        using assms characteristic-iso(3) L.is-full by blast
    qed
    moreover have ∀μ μ′. ?P μ ∧ ?P μ′ ⟶ μ = μ′
      using L.is-faithful S.in-homE by metis
    ultimately show ?thesis by blast
  qed
  hence ?P (THE μ. ?P μ)
    using theI′ [of ?P] by fastforce
  hence 1: ?P l[f]
    unfolding lunit-def by blast
  show «l[f] : L f ⇒S f» using 1 by fast
  show L l[f] = (ι ⋆ f) ·S inv a[b, b, f] using 1 by fast
qed

lemma iso-lunit:
```

assumes $S.ide\ f$
shows $S.iso\ \mathrm{l}[f]$
  using *assms characteristic-iso(4) lunit-char L.reflects-iso* **by** *metis*

lemma *lunit-eqI*:
assumes «$f : a \Rightarrow_S b$» **and** «$\mu : L\ f \Rightarrow_S f$»
and $L\ \mu = ((\iota \star f) \cdot_S inv\ \mathrm{a}[b,\ b,\ f])$
shows $\mu = \mathrm{l}[f]$
  using *assms S.ide-cod lunit-char* [*of f*] **by** *blast*

lemma *lunit-naturality*:
assumes $S.arr\ \mu$
shows $\mathrm{l}[S.cod\ \mu] \cdot_S L\ \mu = \mu \cdot_S \mathrm{l}[S.dom\ \mu]$
proof −
  have *1*: «$\mathrm{l}[S.cod\ \mu] \cdot_S L\ \mu : L\ (S.dom\ \mu) \Rightarrow_S S.cod\ \mu$»
    using *assms lunit-char(1)* [*of S.cod μ*] *S.ide-cod* **by** *blast*
  have $S.par\ (\mathrm{l}[S.cod\ \mu] \cdot_S L\ \mu)\ (\mu \cdot_S \mathrm{l}[S.dom\ \mu])$
    using *assms 1 S.ide-dom lunit-char(1)*
    by (*metis S.comp-in-homI′ S.in-homE*)
  moreover have $L\ (\mathrm{l}[S.cod\ \mu] \cdot_S L\ \mu) = L\ (\mu \cdot_S \mathrm{l}[S.dom\ \mu])$
  proof −
    have *2*: «$b \star b \star \mu : b \star b \star S.dom\ \mu \Rightarrow_S b \star b \star S.cod\ \mu$»
      using *assms weak-unit-b L.preserves-hom* H_L*-def S.arr-iff-in-hom* [*of μ*] $S.arr\text{-}char_{SbC}$
      by *simp*
    have *3*: «$(b \star b) \star \mu : (b \star b) \star S.dom\ \mu \Rightarrow_S (b \star b) \star S.cod\ \mu$»
      using *assms weak-unit-b L.preserves-hom* H_L*-def S.arr-iff-in-hom* $S.arr\text{-}char_{SbC}$
      by (*metis match-3 weak-unit-in-vhom weak-unit-self-left* $S.in\text{-}hom\text{-}char_{SbC}$
        S.not-arr-null S.null-char right-hcomp-closed)

    have $L\ (\mathrm{l}[S.cod\ \mu] \cdot_S L\ \mu) = L\ \mathrm{l}[S.cod\ \mu] \cdot_S L\ (L\ \mu)$
      using *assms 1 L.as-nat-trans.preserves-comp-2* **by** *blast*
    also have ... $= ((\iota \star S.cod\ \mu) \cdot_S inv\ \mathrm{a}[b,\ b,\ S.cod\ \mu]) \cdot_S (b \star b \star \mu)$
      using *assms L.preserves-arr lunit-char S.ide-cod* H_L*-def* **by** *auto*
    also have ... $= (\iota \star S.cod\ \mu) \cdot_S inv\ \mathrm{a}[b,\ b,\ S.cod\ \mu] \cdot_S (b \star b \star \mu)$
      using *S.comp-assoc* **by** *auto*
    also have ... $= (\iota \star S.cod\ \mu) \cdot_S ((b \star b) \star \mu) \cdot_S inv\ \mathrm{a}[b,\ b,\ S.dom\ \mu]$
    proof −
      have $inv\ \mathrm{a}[b,\ b,\ S.cod\ \mu] \cdot_S (b \star b \star \mu) = ((b \star b) \star \mu) \cdot_S inv\ \mathrm{a}[b,\ b,\ S.dom\ \mu]$
      proof −
        have $((b \star b) \star \mu) \cdot_S inv\ \mathrm{a}[b,\ b,\ S.dom\ \mu] = ((b \star b) \star \mu) \cdot inv\ \mathrm{a}[b,\ b,\ S.dom\ \mu]$
          using *assms 3* $S.in\text{-}hom\text{-}char_{SbC}$ *S.comp-char* [*of* $(b \star b) \star \mu$ $inv\ \mathrm{a}[b,\ b,\ S.dom\ \mu]$]
          by (*metis S.ide-dom characteristic-iso(1) ext*)
        also have ... $= inv\ \mathrm{a}[b,\ b,\ S.cod\ \mu] \cdot (b \star b \star \mu)$
          using *assms weak-unit-b* $assoc'\text{-}naturality_{AWC}$ [*of b b μ*] *S.dom-simp S.cod-simp*
            weak-unit-self-composable $S.arr\text{-}char_{SbC}$ *left-def*
          by *simp*
        also have ... $= inv\ \mathrm{a}[b,\ b,\ S.cod\ \mu] \cdot_S (b \star b \star \mu)$
          using *assms 2* $S.in\text{-}hom\text{-}char_{SbC}$ *S.comp-char*
          by (*metis S.comp-simp S.ide-cod S.seqI′ characteristic-iso(1)*)

```
      finally show ?thesis by argo
    qed
   thus ?thesis by argo
  qed
  also have ... = ((ι ⋆ S.cod μ) ·S ((b ⋆ b) ⋆ μ)) ·S inv a[b, b, S.dom μ]
    using S.comp-assoc by auto
  also have ... = ((b ⋆ μ) ·S (ι ⋆ S.dom μ)) ·S inv a[b, b, S.dom μ]
  proof −
    have (b ⋆ b) ⋆ μ ≠ null
      using 3 S.not-arr-null by (elim S.in-homE, auto)
    moreover have ι ⋆ S.dom μ ≠ null
      using assms S.not-arr-null
      by (metis S.dom-charSbC ι-in-hom calculation hom-connected(1−2) in-homE)
    ultimately have (ι ⋆ S.cod μ) ·S ((b ⋆ b) ⋆ μ) = (b ⋆ μ) ·S (ι ⋆ S.dom μ)
      using assms weak-unit-b iso-unit S.comp-arr-dom S.comp-cod-arr
            interchange [of ι b ⋆ b S.cod μ μ ] interchange [of b ι μ S.dom μ]
      by auto
    thus ?thesis by argo
  qed
  also have ... = (b ⋆ μ) ·S (ι ⋆ S.dom μ) ·S inv a[b, b, S.dom μ]
    using S.comp-assoc by auto
  also have ... = L μ ·S L l[S.dom μ]
    using assms lunit-char(2) S.ide-dom HL-def by auto
  also have ... = L (μ ·S l[S.dom μ])
    using assms S.arr-iff-in-hom [of μ] lunit-char(1) S.ide-dom S.seqI
    by fastforce
  finally show ?thesis by blast
 qed
 ultimately show l[S.cod μ] ·S L μ = μ ·S l[S.dom μ]
  using L.is-faithful by blast
qed

abbreviation ℓ
where ℓ μ ≡ if S.arr μ then μ ·S l[S.dom μ] else null

interpretation ℓ: natural-transformation S.comp S.comp L S.map ℓ
proof −
 interpret ℓ: transformation-by-components S.comp S.comp L S.map lunit
  using lunit-char(1) lunit-naturality by (unfold-locales, simp-all)
 have ℓ.map = ℓ
  using ℓ.extensionality ℓ.map-def ℓ.naturality ℓ.map-simp-ide S.ide-dom S.ide-cod
        S.map-def
  by auto
 thus natural-transformation S.comp S.comp L S.map ℓ
  using ℓ.natural-transformation-axioms by auto
qed

lemma natural-transformation-ℓ:
shows natural-transformation S.comp S.comp L S.map ℓ ..
```

interpretation $\mathfrak{l}$: *natural-isomorphism S.comp S.comp L S.map* $\mathfrak{l}$
using *S.ide-is-iso iso-lunit lunit-char*(*1*) *S.isos-compose*
by (*unfold-locales*, *force*)

lemma *natural-isomorphism-*$\mathfrak{l}$:
shows *natural-isomorphism S.comp S.comp L S.map* $\mathfrak{l}$ **..**

interpretation *L*: *equivalence-functor S.comp S.comp L*
using *natural-isomorphism-*$\mathfrak{l}$ *L.isomorphic-to-identity-is-equivalence* **by** *blast*

lemma *equivalence-functor-L*:
shows *equivalence-functor S.comp S.comp L*
..

lemma *lunit-commutes-with-L*:
assumes *S.ide f*
shows $\mathfrak{l}[L\ f] = L\ \mathfrak{l}[f]$
using *assms lunit-char*(*1*) *L.preserves-hom* [*of* $\mathfrak{l}[f]$ *L f f*]
lunit-naturality iso-lunit S.iso-is-section S.section-is-mono S.mono-cancel
by (*metis S.in-homE S.seqI'*)

end

1.1.7 Prebicategories

A *prebicategory* is an associative weak composition satisfying the additional assumption that every arrow has a source and a target.

locale *prebicategory* =
associative-weak-composition +
assumes *arr-has-source*: $arr\ \mu \Longrightarrow sources\ \mu \neq \{\}$
and *arr-has-target*: $arr\ \mu \Longrightarrow targets\ \mu \neq \{\}$
begin

lemma *arr-iff-has-src*:
shows $arr\ \mu \longleftrightarrow sources\ \mu \neq \{\}$
using *arr-has-source composable-implies-arr* **by** *auto*

lemma *arr-iff-has-trg*:
shows $arr\ \mu \longleftrightarrow targets\ \mu \neq \{\}$
using *arr-has-target composable-implies-arr* **by** *auto*

end

The horizontal composition of a prebicategory is regular.

sublocale *prebicategory* $\subseteq$ *regular-weak-composition V H*
proof
show $\bigwedge a\ f.\ a \in sources\ f \Longrightarrow ide\ f \Longrightarrow f \star a \cong f$
proof −

```
      fix a f
      assume a: a ∈ sources f and f: ide f
      interpret Right-a: subcategory V ‹right a›
        using a right-hom-is-subcategory weak-unit-self-composable by force
      interpret Right-a: right-hom-with-unit V H a ‹some-unit a› a
        using a iso-some-unit by (unfold-locales, auto)
      show f ⋆ a ≅ f
      proof −
        have Right-a.ide f
          using a f Right-a.ide-char_SbC Right-a.arr-char_SbC right-def by auto
        hence Right-a.iso (Right-a.runit f) ∧ (Right-a.runit f) ∈ Right-a.hom (f ⋆ a) f
          using Right-a.iso-runit Right-a.runit-char(1) H_R-def by simp
        hence iso (Right-a.runit f) ∧ (Right-a.runit f) ∈ hom (f ⋆ a) f
          using Right-a.iso-char Right-a.hom-char by auto
        thus ?thesis using f isomorphic-def by auto
      qed
    qed
    show ⋀b f. b ∈ targets f ⟹ ide f ⟹ b ⋆ f ≅ f
    proof −
      fix b f
      assume b: b ∈ targets f and f: ide f
      interpret Left-b: subcategory V ‹left b›
        using b left-hom-is-subcategory weak-unit-self-composable by force
      interpret Left-b: left-hom-with-unit V H a ‹some-unit b› b
        using b iso-some-unit by (unfold-locales, auto)
      show b ⋆ f ≅ f
      proof −
        have Left-b.ide f
          using b f Left-b.ide-char_SbC Left-b.arr-char_SbC left-def by auto
        hence Left-b.iso (Left-b.lunit f) ∧ (Left-b.lunit f) ∈ Left-b.hom (b ⋆ f) f
          using b f Left-b.iso-lunit Left-b.lunit-char(1) H_L-def by simp
        hence iso (Left-b.lunit f) ∧ (Left-b.lunit f) ∈ hom (b ⋆ f) f
          using Left-b.iso-char Left-b.hom-char by auto
        thus ?thesis using isomorphic-def by auto
      qed
    qed
  qed
```

The regularity allows us to show that, in a prebicategory, all sources of a given arrow are isomorphic, and similarly for targets.

```
  context prebicategory
  begin

    lemma sources-are-isomorphic:
    assumes a ∈ sources μ and a′ ∈ sources μ
    shows a ≅ a′
    proof −
      have μ: arr μ using assms composable-implies-arr by auto
      have ⋀f. ⟦ ide f; a ∈ sources f; a′ ∈ sources f ⟧ ⟹ a ≅ a′
```

using μ *assms*(*1*) *comp-ide-source comp-target-ide* [*of a a'*]
weak-unit-self-composable(*1*) [*of a*] *weak-unit-self-composable*(*1*) [*of a'*]
isomorphic-transitive isomorphic-symmetric
sources-determine-composability sourcesD(*2*−*3*)
by (*metis* (*full-types*) *connected-if-composable*)
moreover have *ide* (*dom* μ) $\wedge$ $a \in$ *sources* (*dom* μ) $\wedge$ $a' \in$ *sources* (*dom* μ)
using *assms* μ *sources-dom* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *targets-are-isomorphic*:
assumes $b \in$ *targets* μ **and** $b' \in$ *targets* μ
shows $b \cong b'$
proof −
have μ: *arr* μ **using** *assms composable-implies-arr* **by** *auto*
have $\bigwedge f.$ $[\![$ *ide* f; $b \in$ *targets* f; $b' \in$ *targets* f $]\!] \Longrightarrow b \cong b'$
by (*metis connected-if-composable sources-are-isomorphic targetsD*(*3*))
moreover have *ide* (*dom* μ) $\wedge$ $b \in$ *targets* (*dom* μ) $\wedge$ $b' \in$ *targets* (*dom* μ)
using *assms* μ *targets-dom* [*of* μ] **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

In fact, we now show that the sets of sources and targets of a 2-cell are isomorphism-closed, and hence are isomorphism classes. We first show that the notion "weak unit" is preserved under isomorphism.

interpretation *H*: *partial-composition H*
using *is-partial-composition* **by** *auto*

lemma *isomorphism-respects-weak-units*:
assumes *weak-unit a* **and** $a \cong a'$
shows *weak-unit a'*
proof −
obtain φ **where** φ: *iso* φ $\wedge$ «$\varphi : a \Rightarrow a'$»
using *assms* **by** *auto*
interpret *Left-a*: *subcategory V* ‹*left a*›
using *assms left-hom-is-subcategory* **by** *fastforce*
interpret *Left-a*: *left-hom-with-unit V H* a ‹*some-unit a*› *a*
using *assms iso-some-unit weak-unit-self-composable*
apply *unfold-locales* **by** *auto*
interpret *Right-a*: *subcategory V right a*
using *assms right-hom-is-subcategory* **by** *fastforce*
interpret *Right-a*: *right-hom-with-unit V H* a ‹*some-unit a*› *a*
using *assms iso-some-unit weak-unit-self-composable*
apply *unfold-locales* **by** *auto*
have *a'*: *ide* $a' \wedge a \star a' \neq$ *null* $\wedge$ $a' \star a \neq$ *null* $\wedge$ $a' \star a' \neq$ *null* $\wedge$
$\varphi \star a' \neq$ *null* $\wedge$ *Left-a.ide a'*
by (*metis* (*no-types*, *lifting*) *Left-a.left-hom-axioms Right-a.weak-unit-a* φ *assms*(*2*)
ide-cod hom-connected(*1*) *in-homE isomorphic-implies-equicomposable*(*1*)
left-def left-hom-def subcategory.ideI$_{SbC}$ *isomorphic-implies-equicomposable*(*2*)

weak-unit-self-composable(*3*))
have *iso*: $a' \star a' \cong a'$
proof −
have *1*: *Right* a' = *Right* a
using *assms right-respects-isomorphic* **by** *simp*
interpret *Right-a'*: *subcategory V* ‹*right* a'›
using *assms right-hom-is-subcategory* **by** *fastforce*

interpret *Ra'*: *endofunctor* ‹*Right* a'› ‹H_R a'›
using *assms* a' *endofunctor-*H_R **by** *auto*
let *?*ψ = *Left-a.lunit* a' · *inv* ($\varphi \star a'$)
have *iso ?*ψ ∧ «*?*ψ : $a' \star a' \Rightarrow a'$»
proof −
have *iso* (*Left-a.lunit* a') ∧ «*Left-a.lunit* a' : $a \star a' \Rightarrow a'$»
using a' *Left-a.lunit-char*(*1*) *Left-a.iso-lunit Left-a.iso-char*
Left-a.in-hom-char$_{SbC}$ H_L*-def*
by *auto*
moreover have *iso* ($\varphi \star a'$) ∧ «$\varphi \star a'$: $a \star a' \Rightarrow a' \star a'$»
using a' φ *1 Right-a'.arr-char*$_{SbC}$ *Right-a'.in-hom-char*$_{SbC}$ *Right-a.iso-char*
right-def Ra'.preserves-iso Ra'.preserves-hom Right-a'.iso-char$_{SbC}$
Ra'.preserves-dom Ra'.preserves-cod Right-a'.arr-iff-in-hom H_R*-def*
by *metis*
ultimately show *?thesis*
using *isos-compose* **by** *blast*
qed
thus *?thesis* **using** *isomorphic-def* **by** *auto*
qed

We show that horizontal composition on the left and right by a' is naturally isomorphic to the identity functor. This follows from the fact that if a is isomorphic to a', then horizontal composition with a is naturally isomorphic to horizontal composition with a', hence the latter is naturally isomorphic to the identity if the former is. This is conceptually simple, but there are tedious composability details to handle.

have *1*: *Left* a' = *Left* a ∧ *Right* a' = *Right* a
using *assms left-respects-isomorphic right-respects-isomorphic* **by** *simp*

interpret *L*: *fully-faithful-functor* ‹*Left* a› ‹*Left* a› ‹H_L a›
using *assms weak-unit-def* **by** *simp*
interpret *L'*: *endofunctor* ‹*Left* a› ‹H_L a'›
using a' *1 endofunctor-*H_L [*of* a'] **by** *auto*
interpret Φ: *natural-isomorphism* ‹*Left* a› ‹*Left* a› ‹H_L a› ‹H_L a'› ‹H_L φ›
proof
fix μ
show ¬ *Left-a.arr* $\mu \Longrightarrow H_L\ \varphi\ \mu$ = *Left-a.null*
using *left-def* φ H_L*-def hom-connected*(*1*) *Left-a.null-char null-agreement*
Left-a.arr-char$_{SbC}$
by *auto*
assume *Left-a.arr* μ
hence μ: *Left-a.arr* μ ∧ *arr* μ ∧ $a \star \mu \neq$ *null*

```
  using Left-a.arr-char_SbC left-def composable-implies-arr by simp
have 2: φ ⋆ μ ≠ null
  using assms φ μ Left-a.arr-char_SbC left-def hom-connected by auto
show Left-a.arr (H_L φ μ)
  using assms 2 φ μ Left-a.arr-char_SbC left-def hom-connected(2) [of a φ]
        weak-unit-self-composable match-4 H_L-def by auto
show Left-a.comp (H_L a′ μ) (H_L φ (Left-a.dom μ)) = H_L φ μ
proof −
  have Left-a.comp (H_L a′ μ) (H_L φ (Left-a.dom μ)) =
        Left-a.comp (a′ ⋆ μ) (φ ⋆ dom μ)
    using assms 1 2 φ μ Left-a.dom-char_SbC left-def H_L-def by simp
  also have ... = (a′ ⋆ μ) · (φ ⋆ dom μ)
  proof −
    have Left-a.seq (a′ ⋆ μ) (φ ⋆ dom μ)
    proof (intro Left-a.seqI)
      show 3: Left-a.arr (φ ⋆ dom μ)
        using assms 2 φ μ Left-a.arr-char_SbC left-def
        by (metis H_L-def L′.preserves-arr hcomp-simps_WC(1) in-homE right-connected
            paste-1)
      show 4: Left-a.arr (a′ ⋆ μ)
        using μ H_L-def L′.preserves-arr by auto
      show Left-a.dom (a′ ⋆ μ) = Left-a.cod (φ ⋆ dom μ)
        using a′ φ μ 2 3 4 Left-a.dom-char_SbC Left-a.cod-char_SbC
        by (metis Left-a.seqE Left-a.seq-char_SbC hcomp-simps_WC(1) in-homE paste-1)
    qed
    thus ?thesis using Left-a.comp-char Left-a.arr-char_SbC left-def by auto
  qed
  also have ... = a′ · φ ⋆ μ · dom μ
    using a′ φ μ interchange hom-connected by auto
  also have ... = φ ⋆ μ
    using φ μ comp-arr-dom comp-cod-arr by auto
  finally show ?thesis using H_L-def by simp
qed
show Left-a.comp (H_L φ (Left-a.cod μ)) (Left-a.L μ) = H_L φ μ
proof −
  have Left-a.comp (H_L φ (Left-a.cod μ)) (Left-a.L μ) = Left-a.comp (φ ⋆ cod μ) (a ⋆ μ)
    using assms 1 2 φ μ Left-a.cod-char_SbC left-def H_L-def by simp
  also have ... = (φ ⋆ cod μ) · (a ⋆ μ)
  proof −
    have Left-a.seq (φ ⋆ cod μ) (a ⋆ μ)
    proof (intro Left-a.seqI)
      show 3: Left-a.arr (φ ⋆ cod μ)
        using φ μ 2 Left-a.arr-char_SbC left-def
        by (metis (no-types, lifting) H_L-def L.preserves-arr hcomp-simps_WC(1)
            in-homE right-connected paste-2)
      show 4: Left-a.arr (a ⋆ μ)
        using assms μ Left-a.arr-char_SbC left-def
        using H_L-def L.preserves-arr by auto
      show Left-a.dom (φ ⋆ cod μ) = Left-a.cod (a ⋆ μ)
```

```
        using assms φ μ 2 3 4 Left-a.dom-char_SbC Left-a.cod-char_SbC
        by (metis Left-a.seqE Left-a.seq-char_SbC hcomp-simps_WC(1) in-homE paste-2)
      qed
      thus ?thesis using Left-a.comp-char Left-a.arr-char_SbC left-def by auto
    qed
    also have ... = φ · a ⋆ cod μ · μ
      using φ μ interchange hom-connected by auto
    also have ... = φ ⋆ μ
      using φ μ comp-arr-dom comp-cod-arr by auto
    finally show ?thesis using H_L-def by simp
  qed
  next
  fix μ
  assume μ: Left-a.ide μ
  have 1: φ ⋆ μ ≠ null
    using assms φ μ Left-a.ide-char Left-a.arr-char_SbC left-def hom-connected by auto
  show Left-a.iso (H_L φ μ)
  proof −
    have iso (φ ⋆ μ)
    proof −
      have a ∈ sources φ ∩ targets μ
        using assms φ μ 1 hom-connected weak-unit-self-composable
              Left-a.ide-char Left-a.arr-char_SbC left-def connected-if-composable
        by auto
      thus ?thesis
        using φ μ Left-a.ide-char_SbC ide-is-iso iso-hcomp_RWC(1) by blast
    qed
    moreover have left a (φ ⋆ μ)
      using assms 1 φ weak-unit-self-composable hom-connected(2) [of a φ]
            left-def match-4 null-agreement
      by auto
    ultimately show ?thesis
      using Left-a.iso-char Left-a.arr-char_SbC left-iff-left-inv Left-a.inv-char H_L-def
      by metis
  qed
qed
interpret L′: equivalence-functor ‹Left a′› ‹Left a′› ‹H_L a′›
proof −
  have naturally-isomorphic (Left a′) (Left a′) (H_L a′) (identity-functor.map (Left a′))
  proof −
    have naturally-isomorphic (Left a) (Left a) (H_L a′)
                              (identity-functor.map (Left a))
      by (meson Left-a.natural-isomorphism-ℓ Φ.natural-isomorphism-axioms
          naturally-isomorphic-def naturally-isomorphic-symmetric
          naturally-isomorphic-transitive)
    thus ?thesis
      using 1 by auto
  qed
  thus equivalence-functor (Left a′) (Left a′) (H_L a′)
```

```
    using 1 L'.isomorphic-to-identity-is-equivalence naturally-isomorphic-def
    by fastforce
qed
```

Now we do the same for R'.

```
interpret R: fully-faithful-functor ‹Right a› ‹Right a› ‹H_R a›
  using assms weak-unit-def by simp
interpret R': endofunctor ‹Right a› ‹H_R a'›
  using a' 1 endofunctor-H_R [of a'] by auto
interpret Ψ: natural-isomorphism ‹Right a› ‹Right a› ‹H_R a› ‹H_R a'› ‹H_R φ›
proof
  fix μ
  show ¬ Right-a.arr μ ⟹ H_R φ μ = Right-a.null
    using right-def φ H_R-def hom-connected Right-a.null-char Right-a.arr-char_SbC
    by auto
  assume Right-a.arr μ
  hence μ: Right-a.arr μ ∧ arr μ ∧ μ ⋆ a ≠ null
    using Right-a.arr-char_SbC right-def composable-implies-arr by simp
  have 2: μ ⋆ φ ≠ null
    using assms φ μ Right-a.arr-char_SbC right-def hom-connected by auto
  show Right-a.arr (H_R φ μ)
    by (metis 2 H_R-def R'.extensionality Right-a.null-char a' match-3)
  show Right-a.comp (H_R a' μ) (H_R φ (Right-a.dom μ)) = H_R φ μ
  proof −
    have Right-a.comp (H_R a' μ) (H_R φ (Right-a.dom μ)) =
          Right-a.comp (μ ⋆ a') (dom μ ⋆ φ)
      using assms 1 2 φ μ Right-a.dom-char_SbC right-def H_R-def by simp
    also have ... = (μ ⋆ a') · (dom μ ⋆ φ)
    proof −
      have Right-a.seq (μ ⋆ a') (dom μ ⋆ φ)
      proof (intro Right-a.seqI)
        show 3: Right-a.arr (dom μ ⋆ φ)
          using assms 2 φ μ Right-a.arr-char_SbC right-def
          by (metis H_R-def R'.preserves-arr hcomp-simps_WC(1) in-homE left-connected
                paste-2)
        show 4: Right-a.arr (μ ⋆ a')
          using μ H_R-def R'.preserves-arr by auto
        show Right-a.dom (μ ⋆ a') = Right-a.cod (dom μ ⋆ φ)
          using a' φ μ 2 3 4 Right-a.dom-char_SbC Right-a.cod-char_SbC
          by (metis Right-a.seqE Right-a.seq-char_SbC hcomp-simps_WC(1) in-homE paste-2)
      qed
      thus ?thesis using Right-a.comp-char Right-a.arr-char_SbC right-def by auto
    qed
    also have ... = μ · dom μ ⋆ a' · φ
      using a' φ μ interchange hom-connected by auto
    also have ... = μ ⋆ φ
      using φ μ comp-arr-dom comp-cod-arr by auto
    finally show ?thesis using H_R-def by simp
  qed
```

```
show Right-a.comp (H_R φ (Right-a.cod μ)) (Right-a.R μ) = H_R φ μ
proof −
  have Right-a.comp (H_R φ (Right-a.cod μ)) (Right-a.R μ)
        = Right-a.comp (cod μ ⋆ φ) (μ ⋆ a)
    using assms 1 2 φ μ Right-a.cod-char_SbC right-def H_R-def by simp
  also have ... = (cod μ ⋆ φ) · (μ ⋆ a)
  proof −
    have Right-a.seq (cod μ ⋆ φ) (μ ⋆ a)
    proof (intro Right-a.seqI)
      show 3: Right-a.arr (cod μ ⋆ φ)
        using φ μ 2 Right-a.arr-char_SbC right-def
        by (metis (no-types, lifting) H_R-def R.preserves-arr hcomp-simps_WC(1)
            in-homE left-connected paste-1)
      show 4: Right-a.arr (μ ⋆ a)
        using assms μ Right-a.arr-char_SbC right-def
        using H_R-def R.preserves-arr by auto
      show Right-a.dom (cod μ ⋆ φ) = Right-a.cod (μ ⋆ a)
        using assms φ μ 2 3 4 Right-a.dom-char_SbC Right-a.cod-char_SbC
        by (metis Right-a.seqE Right-a.seq-char_SbC hcomp-simps_WC(1) in-homE paste-1)
    qed
    thus ?thesis using Right-a.comp-char Right-a.arr-char_SbC right-def by auto
  qed
  also have ... = cod μ · μ ⋆ φ · a
    using φ μ interchange hom-connected by auto
  also have ... = μ ⋆ φ
    using φ μ comp-arr-dom comp-cod-arr by auto
  finally show ?thesis using H_R-def by simp
qed
next
fix μ
assume μ: Right-a.ide μ
have 1: μ ⋆ φ ≠ null
 using assms φ μ Right-a.ide-char Right-a.arr-char_SbC right-def hom-connected by auto
show Right-a.iso (H_R φ μ)
proof −
  have iso (μ ⋆ φ)
  proof −
    have a ∈ targets φ ∩ sources μ
      using assms φ μ 1 hom-connected weak-unit-self-composable
            Right-a.ide-char Right-a.arr-char_SbC right-def connected-if-composable
      by (metis (full-types) IntI targetsI)
    thus ?thesis
      using φ μ Right-a.ide-char_SbC ide-is-iso iso-hcomp_RWC(1) by blast
  qed
  moreover have right a (μ ⋆ φ)
    using assms 1 φ weak-unit-self-composable hom-connected(1) [of φ a]
          right-def match-3 null-agreement
    by auto
  ultimately show ?thesis
```

using *Right-a.iso-char Right-a.arr-char*$_{SbC}$ *right-iff-right-inv*
Right-a.inv-char H_R*-def*
by *metis*
qed
qed
interpret R': *equivalence-functor* ‹*Right* a'› ‹*Right* a'› ‹H_R a'›
proof −
have *naturally-isomorphic* (*Right* a') (*Right* a') (H_R a')
(*identity-functor.map* (*Right* a'))
proof −
have *naturally-isomorphic* (*Right* a) (*Right* a) (H_R a') *Right-a.map*
by (*meson Right-a.natural-isomorphism-*$\mathfrak{r}$ Ψ*.natural-isomorphism-axioms*
naturally-isomorphic-def naturally-isomorphic-symmetric
naturally-isomorphic-transitive)
thus *?thesis*
using *1* **by** *auto*
qed
thus *equivalence-functor* (*Right* a') (*Right* a') (H_R a')
using *1* R'*.isomorphic-to-identity-is-equivalence naturally-isomorphic-def*
by *fastforce*
qed
show *weak-unit* a'
using *weak-unit-def iso* L'*.fully-faithful-functor-axioms* R'*.fully-faithful-functor-axioms*
by *blast*
qed

lemma *sources-iso-closed*:
assumes $a \in$ *sources* μ **and** $a \cong a'$
shows $a' \in$ *sources* μ
using *assms isomorphism-respects-weak-units isomorphic-implies-equicomposable*
by *blast*

lemma *targets-iso-closed*:
assumes $a \in$ *targets* μ **and** $a \cong a'$
shows $a' \in$ *targets* μ
using *assms isomorphism-respects-weak-units isomorphic-implies-equicomposable*
by *blast*

lemma *sources-eqI*:
assumes *sources* $\mu \cap$ *sources* $\nu \neq \{\}$
shows *sources* $\mu =$ *sources* ν
using *assms sources-iso-closed sources-are-isomorphic* **by** *blast*

lemma *targets-eqI*:
assumes *targets* $\mu \cap$ *targets* $\nu \neq \{\}$
shows *targets* $\mu =$ *targets* ν
using *assms targets-iso-closed targets-are-isomorphic* **by** *blast*

The sets of sources and targets of a weak unit are isomorphism classes.

lemma *sources-char*:
assumes *weak-unit a*
shows *sources a* = $\{x.\ x \cong a\}$
using *assms sources-iso-closed weak-unit-iff-self-source sources-are-isomorphic isomorphic-symmetric*
by *blast*

lemma *targets-char*:
assumes *weak-unit a*
shows *targets a* = $\{x.\ x \cong a\}$
using *assms targets-iso-closed weak-unit-iff-self-target targets-are-isomorphic isomorphic-symmetric*
by *blast*

end

1.2 Horizontal Homs

Here we define a locale that axiomatizes a (vertical) category V that has been punctuated into "horizontal homs" by the choice of idempotent endofunctors *src* and *trg* that assign a specific "source" and "target" 1-cell to each of its arrows. The functors *src* and *trg* are also subject to further conditions that constrain how they commute with each other.

locale *horizontal-homs* =
category V +
src: *endofunctor V src* +
trg: *endofunctor V trg*
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *src* :: *'a* ⇒ *'a*
and *trg* :: *'a* ⇒ *'a* +
assumes *ide-src* [*simp*]: *arr μ* ⟹ *ide* (*src μ*)
and *ide-trg* [*simp*]: *arr μ* ⟹ *ide* (*trg μ*)
and *src-src* [*simp*]: *arr μ* ⟹ *src* (*src μ*) = *src μ*
and *trg-trg* [*simp*]: *arr μ* ⟹ *trg* (*trg μ*) = *trg μ*
and *trg-src* [*simp*]: *arr μ* ⟹ *trg* (*src μ*) = *src μ*
and *src-trg* [*simp*]: *arr μ* ⟹ *src* (*trg μ*) = *trg μ*
begin

no-notation *in-hom* (‹«- : - → -»›)
notation *in-hom* (‹«- : - ⇒ -»›)

We define an *object* to be an arrow that is its own source (or equivalently, its own target).

definition *obj*
where *obj a* ≡ *arr a* ∧ *src a* = *a*

lemma *obj-def'*:
shows *obj a* ⟷ *arr a* ∧ *trg a* = *a*
using *trg-src src-trg obj-def* **by** *metis*

lemma *objI-src*:
assumes *arr a* **and** *src a* = *a*
shows *obj a*
  using *assms obj-def* **by** *simp*

lemma *objI-trg*:
assumes *arr a* **and** *trg a* = *a*
shows *obj a*
  using *assms obj-def′* **by** *simp*

lemma *objE* [*elim*]:
assumes *obj a* **and** ⟦ *ide a*; *src a* = *a*; *trg a* = *a* ⟧ ⟹ *T*
shows *T*
  using *assms obj-def obj-def′ ide-src ide-trg* **by** *metis*

lemma *obj-simps* :
assumes *obj a*
shows *arr a* **and** *src a* = *a* **and** *trg a* = *a* **and** *dom a* = *a* **and** *cod a* = *a*
  using *assms* **by** *auto*

lemma *obj-src* [*intro*, *simp*]:
assumes *arr μ*
shows *obj* (*src μ*)
  using *assms objI-src* **by** *auto*

lemma *obj-trg* [*intro*, *simp*]:
assumes *arr μ*
shows *obj* (*trg μ*)
  using *assms objI-trg* **by** *auto*

definition *in-hhom* (‹«- : - → -»›)
where *in-hhom μ a b* ≡ *arr μ* ∧ *src μ* = *a* ∧ *trg μ* = *b*

abbreviation *hhom*
where *hhom a b* ≡ {*μ*. «*μ* : *a* → *b*»}

abbreviation (*input*) $hseq_{HH}$
where $hseq_{HH}$ ≡ λ*μ ν*. *arr μ* ∧ *arr ν* ∧ *src μ* = *trg ν*

lemma *in-hhomI* [*intro*, *simp*]:
assumes *arr μ* **and** *src μ* = *a* **and** *trg μ* = *b*
shows «*μ* : *a* → *b*»
  using *assms in-hhom-def* **by** *auto*

lemma *in-hhomE* [*elim*]:
assumes «*μ* : *a* → *b*»
and ⟦ *arr μ*; *obj a*; *obj b*; *src μ* = *a*; *trg μ* = *b* ⟧ ⟹ *T*

shows T
  using *assms in-hhom-def* **by** *auto*

lemma *ide-in-hom* [*intro*]:
assumes *ide f*
shows «f : *src* $f \rightarrow$ *trg* f» **and** «$f : f \Rightarrow f$»
  using *assms* **by** *auto*

lemma *src-dom* [*simp*]:
shows *src* (*dom* μ) = *src* μ
  by (*metis arr-dom-iff-arr obj-simps*(*4*) *obj-src src.extensionality src.preserves-dom*)

lemma *src-cod* [*simp*]:
shows *src* (*cod* μ) = *src* μ
  by (*metis arr-cod-iff-arr obj-simps*(*5*) *obj-src src.extensionality src.preserves-cod*)

lemma *trg-dom* [*simp*]:
shows *trg* (*dom* μ) = *trg* μ
  by (*metis arr-dom-iff-arr ide-char ide-trg trg.extensionality trg.preserves-dom*)

lemma *trg-cod* [*simp*]:
shows *trg* (*cod* μ) = *trg* μ
  by (*metis arr-cod-iff-arr ide-char ide-trg trg.extensionality trg.preserves-cod*)

lemma *dom-src* [*simp*]:
shows *dom* (*src* μ) = *src* μ
  by (*metis dom-null ideD*(*2*) *ide-src src.extensionality*)

lemma *cod-src* [*simp*]:
shows *cod* (*src* μ) = *src* μ
  by (*metis cod-null ideD*(*3*) *ide-src src.extensionality*)

lemma *dom-trg* [*simp*]:
shows *dom* (*trg* μ) = *trg* μ
  by (*metis dom-null ideD*(*2*) *ide-trg trg.extensionality*)

lemma *cod-trg* [*simp*]:
shows *cod* (*trg* μ) = *trg* μ
  by (*metis cod-null ideD*(*3*) *ide-trg trg.extensionality*)

lemma *vcomp-in-hhom* [*intro*, *simp*]:
assumes *seq* ν μ **and** *src* ν = a **and** *trg* ν = b
shows «$\nu \cdot \mu : a \rightarrow b$»
  using *assms src-cod* [*of* $\nu \cdot \mu$] *trg-cod* [*of* $\nu \cdot \mu$] **by** *auto*

lemma *src-vcomp* [*simp*]:
assumes *seq* ν μ

shows *src* $(\nu \cdot \mu) = src\ \nu$
  using *assms src-cod* [*of* $\nu \cdot \mu$] **by** *auto*

lemma *trg-vcomp* [*simp*]:
assumes *seq* $\nu\ \mu$
shows *trg* $(\nu \cdot \mu) = trg\ \nu$
  using *assms trg-cod* [*of* $\nu \cdot \mu$] **by** *auto*

lemma *vseq-implies-hpar*:
assumes *seq* $\nu\ \mu$
shows *src* $\nu = src\ \mu$ **and** *trg* $\nu = trg\ \mu$
  using *assms src-dom* [*of* $\nu \cdot \mu$] *trg-dom* [*of* $\nu \cdot \mu$] *src-cod* [*of* $\nu \cdot \mu$]
    trg-cod [*of* $\nu \cdot \mu$]
  by *auto*

lemma *vconn-implies-hpar*:
assumes «$\mu : f \Rightarrow g$»
shows *src* $\mu = src\ f$ **and** *trg* $\mu = trg\ f$ **and** *src* $g = src\ f$ **and** *trg* $g = trg\ f$
  using *assms* **by** *auto*

lemma *src-inv* [*simp*]:
assumes *iso* μ
shows *src* (*inv* μ) = *src* μ
  using *assms inv-in-hom iso-is-arr src-dom src-cod iso-inv-iso dom-inv* **by** *metis*

lemma *trg-inv* [*simp*]:
assumes *iso* μ
shows *trg* (*inv* μ) = *trg* μ
  using *assms inv-in-hom iso-is-arr trg-dom trg-cod iso-inv-iso cod-inv* **by** *metis*

lemma *inv-in-hhom* [*intro*, *simp*]:
assumes *iso* μ **and** *src* $\mu = a$ **and** *trg* $\mu = b$
shows «*inv* $\mu : a \to b$»
  using *assms iso-is-arr* **by** *simp*

lemma *hhom-is-subcategory*:
shows *subcategory* V ($\lambda\mu$. «$\mu : a \to b$»)
  using *src-dom trg-dom src-cod trg-cod* **by** (*unfold-locales*, *auto*)

lemma *isomorphic-objects-are-equal*:
assumes *obj* a **and** *obj* b **and** $a \cong b$
shows $a = b$
  using *assms isomorphic-def*
  by (*metis dom-inv in-homE objE src-dom src-inv*)

Having the functors *src* and *trg* allows us to form categories VV and VVV of formally horizontally composable pairs and triples of arrows.

sublocale *VxV*: *product-category* V V **..**
sublocale *VV*: *subcategory* *VxV.comp* ‹$\lambda\mu\nu$. $hseq_{HH}$ (*fst* $\mu\nu$) (*snd* $\mu\nu$)›

by (*unfold-locales*, *auto*)

lemma *subcategory-VV*:
shows *subcategory VxV.comp* ($\lambda\mu\nu$. $hseq_{HH}$ (*fst* $\mu\nu$) (*snd* $\mu\nu$))
..

sublocale *VxVxV*: *product-category V VxV.comp* **..**
sublocale *VVV*: *subcategory VxVxV.comp*
‹$\lambda\tau\mu\nu$. *arr* (*fst* $\tau\mu\nu$) $\wedge$ *VV.arr* (*snd* $\tau\mu\nu$) $\wedge$
src (*fst* $\tau\mu\nu$) = *trg* (*fst* (*snd* $\tau\mu\nu$))›
using $VV.arr\text{-}char_{SbC}$
by (*unfold-locales*, *auto*)

lemma *subcategory-VVV*:
shows *subcategory VxVxV.comp*
($\lambda\tau\mu\nu$. *arr* (*fst* $\tau\mu\nu$) $\wedge$ *VV.arr* (*snd* $\tau\mu\nu$) $\wedge$
src (*fst* $\tau\mu\nu$) = *trg* (*fst* (*snd* $\tau\mu\nu$)))
..

end

1.2.1 Prebicategories with Homs

A *weak composition with homs* consists of a weak composition that is equipped with horizontal homs in such a way that the chosen source and target of each 2-cell μ in fact lie in the set of sources and targets, respectively, of μ, such that horizontal composition respects the chosen sources and targets, and such that if 2-cells μ and ν are horizontally composable, then the chosen target of μ coincides with the chosen source of ν.

locale *weak-composition-with-homs* =
weak-composition +
horizontal-homs +
assumes *src-in-sources*: *arr* $\mu \Longrightarrow$ *src* $\mu \in$ *sources* μ
and *trg-in-targets*: *arr* $\mu \Longrightarrow$ *trg* $\mu \in$ *targets* μ
and *src-hcomp'*: $\nu \star \mu \neq$ *null* $\Longrightarrow$ *src* ($\nu \star \mu$) = *src* μ
and *trg-hcomp'*: $\nu \star \mu \neq$ *null* $\Longrightarrow$ *trg* ($\nu \star \mu$) = *trg* ν
and *seq-if-composable*: $\nu \star \mu \neq$ *null* $\Longrightarrow$ *src* ν = *trg* μ

locale *prebicategory-with-homs* =
prebicategory +
weak-composition-with-homs
begin

lemma $composable\text{-}char_{PBH}$:
shows $\nu \star \mu \neq$ *null* $\longleftrightarrow$ *arr* $\mu \wedge$ *arr* $\nu \wedge$ *src* ν = *trg* μ
using *trg-in-targets src-in-sources composable-if-connected sourcesD*(*3*)
targets-determine-composability seq-if-composable composable-implies-arr
by *metis*

lemma *hcomp-in-hom$_{PBH}$*:
assumes «$\mu : a \rightarrow_{WC} b$» **and** «$\nu : b \rightarrow_{WC} c$»
shows «$\nu \star \mu : a \rightarrow_{WC} c$»
and «$\nu \star \mu : dom\ \nu \star dom\ \mu \Rightarrow cod\ \nu \star cod\ \mu$»
proof −
  show «$\nu \star \mu : a \rightarrow_{WC} c$»
    using *assms sources-determine-composability sources-hcomp targets-hcomp* **by** *auto*
  thus «$\nu \star \mu : dom\ \nu \star dom\ \mu \Rightarrow cod\ \nu \star cod\ \mu$»
    using *assms* **by** *auto*
qed

In a prebicategory with homs, if a is an object (i.e. $src\ a = a$ and $trg\ a = a$), then a is a weak unit. The converse need not hold: there can be weak units that the *src* and *trg* mappings send to other 1-cells in the same isomorphism class.

lemma *obj-is-weak-unit*:
assumes *obj a*
shows *weak-unit a*
proof −
  have $a \in sources\ a$
    using *assms objE src-in-sources ideD(1)* **by** *metis*
  thus *?thesis* **by** *auto*
qed

end

1.2.2 Choosing Homs

Every prebicategory extends to a prebicategory with homs, by choosing an arbitrary representative of each isomorphism class of weak units to serve as an object. "The source" of a 2-cell is defined to be the chosen representative of the set of all its sources (which is an isomorphism class), and similarly for "the target".

context *prebicategory*
begin

definition *rep*
where $rep\ f \equiv SOME\ f'.\ f' \in \{\ f'.\ f \cong f'\ \}$

definition *some-src*
where $some\text{-}src\ \mu \equiv if\ arr\ \mu\ then\ rep\ (SOME\ a.\ a \in sources\ \mu)\ else\ null$

definition *some-trg*
where $some\text{-}trg\ \mu \equiv if\ arr\ \mu\ then\ rep\ (SOME\ b.\ b \in targets\ \mu)\ else\ null$

lemma *isomorphic-ide-rep*:
assumes *ide f*
shows $f \cong rep\ f$
proof −
  have $\exists f'.\ f' \in \{\ f'.\ f \cong f'\ \}$

```
    using assms isomorphic-reflexive by blast
  thus ?thesis using rep-def someI-ex by simp
qed

lemma rep-rep:
assumes ide f
shows rep (rep f) = rep f
proof −
  have rep f ∈ { f'. f ≅ f' }
    using assms isomorphic-ide-rep by simp
  have { f'. f ≅ f' } = { f'. rep f ≅ f' }
  proof −
    have ⋀f'. f ≅ f' ⟷ rep f ≅ f'
    proof
      fix f'
      assume f': f ≅ f'
      show rep f ≅ f'
      proof −
        obtain φ where φ: φ ∈ hom f f' ∧ iso φ
          using f' by auto
        obtain ψ where ψ: ψ ∈ hom f (rep f) ∧ iso ψ
          using assms isomorphic-ide-rep by blast
        have inv ψ ∈ hom (rep f) f ∧ iso (inv ψ)
          using ψ by simp
        hence iso (V φ (inv ψ)) ∧ V φ (inv ψ) ∈ hom (rep f) f'
          using φ isos-compose by auto
        thus ?thesis using isomorphic-def by auto
      qed
      next
      fix f'
      assume f': rep f ≅ f'
      show f ≅ f'
        using assms f' isomorphic-ide-rep isos-compose isomorphic-def
        by (meson isomorphic-transitive)
    qed
    thus ?thesis by auto
  qed
  hence rep (rep f) = (SOME f'. f' ∈ { f'. f ≅ f' })
    using assms rep-def by fastforce
  also have ... = rep f
    using assms rep-def by simp
  finally show ?thesis by simp
qed

lemma some-src-in-sources:
assumes arr μ
shows some-src μ ∈ sources μ
proof −
  have 1: (SOME a. a ∈ sources μ) ∈ sources μ
```

using *assms arr-iff-has-src someI-ex* [*of* $\lambda a.\ a \in$ *sources* μ] **by** *blast*
moreover have *ide* (*SOME a.* $a \in$ *sources* μ)
using *1 weak-unit-self-composable* **by** *auto*
ultimately show *?thesis*
using *assms 1 some-src-def sources-iso-closed isomorphic-ide-rep* **by** *metis*
qed

lemma *some-trg-in-targets*:
assumes *arr* μ
shows *some-trg* $\mu \in$ *targets* μ
proof −
have *1*: (*SOME a.* $a \in$ *targets* μ) $\in$ *targets* μ
using *assms arr-iff-has-trg someI-ex* [*of* $\lambda a.\ a \in$ *targets* μ] **by** *blast*
moreover have *ide* (*SOME a.* $a \in$ *targets* μ)
using *1 weak-unit-self-composable* **by** *auto*
ultimately show *?thesis*
using *assms 1 some-trg-def targets-iso-closed isomorphic-ide-rep* **by** *metis*
qed

lemma *some-src-dom*:
assumes *arr* μ
shows *some-src* (*dom* μ) = *some-src* μ
using *assms some-src-def sources-dom* **by** *simp*

lemma *some-src-cod*:
assumes *arr* μ
shows *some-src* (*cod* μ) = *some-src* μ
using *assms some-src-def sources-cod* **by** *simp*

lemma *some-trg-dom*:
assumes *arr* μ
shows *some-trg* (*dom* μ) = *some-trg* μ
using *assms some-trg-def targets-dom* **by** *simp*

lemma *some-trg-cod*:
assumes *arr* μ
shows *some-trg* (*cod* μ) = *some-trg* μ
using *assms some-trg-def targets-cod* **by** *simp*

lemma *ide-some-src*:
assumes *arr* μ
shows *ide* (*some-src* μ)
using *assms some-src-in-sources weak-unit-self-composable* **by** *blast*

lemma *ide-some-trg*:
assumes *arr* μ
shows *ide* (*some-trg* μ)
using *assms some-trg-in-targets weak-unit-self-composable* **by** *blast*

lemma *some-src-composable*:
assumes $arr\ \tau$
shows $\tau \star \mu \neq null \longleftrightarrow some\text{-}src\ \tau \star \mu \neq null$
  using *assms some-src-in-sources sources-determine-composability* **by** *blast*

lemma *some-trg-composable*:
assumes $arr\ \sigma$
shows $\mu \star \sigma \neq null \longleftrightarrow \mu \star some\text{-}trg\ \sigma \neq null$
  using *assms some-trg-in-targets targets-determine-composability* **by** *blast*

lemma *sources-some-src*:
assumes $arr\ \mu$
shows $sources\ (some\text{-}src\ \mu) = sources\ \mu$
  using *assms sources-determine-composability some-src-in-sources* **by** *blast*

lemma *targets-some-trg*:
assumes $arr\ \mu$
shows $targets\ (some\text{-}trg\ \mu) = targets\ \mu$
  using *assms targets-determine-composability some-trg-in-targets* **by** *blast*

lemma *src-some-src*:
assumes $arr\ \mu$
shows $some\text{-}src\ (some\text{-}src\ \mu) = some\text{-}src\ \mu$
  using *assms some-src-def ide-some-src sources-some-src* **by** *force*

lemma *trg-some-trg*:
assumes $arr\ \mu$
shows $some\text{-}trg\ (some\text{-}trg\ \mu) = some\text{-}trg\ \mu$
  using *assms some-trg-def ide-some-trg targets-some-trg* **by** *force*

lemma *sources-char'*:
assumes $arr\ \mu$
shows $a \in sources\ \mu \longleftrightarrow some\text{-}src\ \mu \cong a$
  using *assms some-src-in-sources sources-iso-closed sources-are-isomorphic* **by** *meson*

lemma *targets-char'*:
assumes $arr\ \mu$
shows $a \in targets\ \mu \longleftrightarrow some\text{-}trg\ \mu \cong a$
  using *assms some-trg-in-targets targets-iso-closed targets-are-isomorphic* **by** *meson*

An arbitrary choice of sources and targets in a prebicategory results in a notion of formal composability that coincides with the actual horizontal composability of the prebicategory.

lemma *composable-char*$_{PB}$:
shows $\tau \star \sigma \neq null \longleftrightarrow arr\ \sigma \wedge arr\ \tau \wedge some\text{-}src\ \tau = some\text{-}trg\ \sigma$
proof
  assume $\sigma\tau$: $\tau \star \sigma \neq null$
  show $arr\ \sigma \wedge arr\ \tau \wedge some\text{-}src\ \tau = some\text{-}trg\ \sigma$
    using $\sigma\tau$ *composable-implies-arr connected-if-composable some-src-def some-trg-def*

by *force*
next
assume $\sigma\tau$: $arr\ \sigma \wedge arr\ \tau \wedge some\text{-}src\ \tau = some\text{-}trg\ \sigma$
show $\tau \star \sigma \neq null$
using $\sigma\tau$ *some-src-in-sources some-trg-composable* **by** *force*
qed

A 1-cell is its own source if and only if it is its own target.

lemma *self-src-iff-self-trg*:
assumes *ide a*
shows $a = some\text{-}src\ a \longleftrightarrow a = some\text{-}trg\ a$
proof
assume *a*: $a = some\text{-}src\ a$
have $weak\text{-}unit\ a \wedge a \star a \neq null$
using *assms a some-src-in-sources* [*of a*] **by** *force*
thus $a = some\text{-}trg\ a$ **using** *a* $composable\text{-}char_{PB}$ **by** *simp*
next
assume *a*: $a = some\text{-}trg\ a$
have $weak\text{-}unit\ a \wedge a \star a \neq null$
using *assms a some-trg-in-targets* [*of a*] **by** *force*
thus $a = some\text{-}src\ a$ **using** *a* $composable\text{-}char_{PB}$ **by** *simp*
qed

lemma *some-trg-some-src*:
assumes $arr\ \mu$
shows $some\text{-}trg\ (some\text{-}src\ \mu) = some\text{-}src\ \mu$
using *assms ide-some-src some-src-def some-trg-def some-src-in-sources sources-char targets-char sources-some-src*
by *force*

lemma *src-some-trg*:
assumes $arr\ \mu$
shows $some\text{-}src\ (some\text{-}trg\ \mu) = some\text{-}trg\ \mu$
using *assms ide-some-trg some-src-def some-trg-def some-trg-in-targets sources-char targets-char targets-some-trg*
by *force*

lemma *some-src-eqI*:
assumes $a \in sources\ \mu$ **and** $some\text{-}src\ a = a$
shows $some\text{-}src\ \mu = a$
using *assms sources-char′ some-src-def some-src-in-sources sources-are-isomorphic isomorphic-symmetric isomorphic-transitive*
by (*metis* $composable\text{-}char_{PB}$ *sourcesD*(*3*))

lemma *some-trg-eqI*:
assumes $b \in targets\ \mu$ **and** $some\text{-}trg\ b = b$
shows $some\text{-}trg\ \mu = b$
using *assms targets-char′ some-trg-def some-trg-in-targets targets-are-isomorphic isomorphic-symmetric isomorphic-transitive*

by (*metis composable-char*$_{PB}$ *targetsD*(*3*))

lemma *some-src-comp*:
assumes $\tau \star \sigma \neq$ *null*
shows *some-src* $(\tau \star \sigma)$ = *some-src* σ
proof (*intro some-src-eqI* [*of some-src* σ $\tau \star \sigma$])
  show *some-src* (*some-src* σ) = *some-src* σ
    using *assms src-some-src composable-implies-arr* **by** *simp*
  show *some-src* $\sigma \in$ *sources* (H τ σ)
    using *assms some-src-in-sources composable-char*$_{PB}$
    by (*simp add*: *sources-hcomp*)
qed

lemma *some-trg-comp*:
assumes $\tau \star \sigma \neq$ *null*
shows *some-trg* $(\tau \star \sigma)$ = *some-trg* τ
proof (*intro some-trg-eqI* [*of some-trg* τ $\tau \star \sigma$])
  show *some-trg* (*some-trg* τ) = *some-trg* τ
    using *assms trg-some-trg composable-implies-arr* **by** *simp*
  show *some-trg* $\tau \in$ *targets* (H τ σ)
    using *assms some-trg-in-targets composable-char*$_{PB}$
    by (*simp add*: *targets-hcomp*)
qed

The mappings that take an arrow to its chosen source or target are endofunctors of the vertical category, which commute with each other in the manner required for horizontal homs.

interpretation S: *endofunctor V some-src*
  using *some-src-def ide-some-src some-src-dom some-src-cod*
  apply *unfold-locales*
    apply *auto*[*4*]
proof −
  fix ν μ
  assume $\mu\nu$: *seq* ν μ
  show *some-src* $(\nu \cdot \mu)$ = *some-src* ν · *some-src* μ
    using $\mu\nu$ *some-src-dom* [*of* $\nu \cdot \mu$] *some-src-dom some-src-cod* [*of* $\nu \cdot \mu$]
      some-src-cod ide-some-src
    by *auto*
qed

interpretation T: *endofunctor V some-trg*
  using *some-trg-def ide-some-trg some-trg-dom some-trg-cod*
  apply *unfold-locales*
    apply *auto*[*4*]
proof −
  fix ν μ
  assume $\mu\nu$: *seq* ν μ
  show *some-trg* $(\nu \cdot \mu)$ = *some-trg* ν · *some-trg* μ
    using $\mu\nu$ *some-trg-dom* [*of* $\nu \cdot \mu$] *some-trg-dom some-trg-cod* [*of* $\nu \cdot \mu$]

some-trg-cod ide-some-trg
by *auto*
qed

interpretation *weak-composition-with-homs V H some-src some-trg*
apply *unfold-locales*
using *some-src-in-sources some-trg-in-targets*
src-some-src trg-some-trg src-some-trg some-trg-some-src
some-src-comp some-trg-comp composable-char$_{PB}$ *ide-some-src ide-some-trg*
by *simp-all*

proposition *extends-to-weak-composition-with-homs*:
shows *weak-composition-with-homs V H some-src some-trg*
..

proposition *extends-to-prebicategory-with-homs*:
shows *prebicategory-with-homs V H* a *some-src some-trg*
..

end

1.2.3 Choosing Units

A *prebicategory with units* is a prebicategory equipped with a choice, for each weak unit a, of a "unit isomorphism" «i[a] : $a \star a \Rightarrow a$».

locale *prebicategory-with-units* =
prebicategory V H a +
weak-composition V H
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a$ *comp* (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›) +
assumes *unit-in-vhom*$_{PBU}$: *weak-unit a* $\Longrightarrow$ «i[a] : $a \star a \Rightarrow a$»
and *iso-unit*$_{PBU}$: *weak-unit a* $\Longrightarrow$ *iso* i[a]
begin

lemma *unit-in-hom*$_{PBU}$:
assumes *weak-unit a*
shows «i[a] : $a \rightarrow_{WC} a$» **and** «i[a] : $a \star a \Rightarrow a$»
proof −
show *1*: «i[a] : $a \star a \Rightarrow a$»
using *assms unit-in-vhom*$_{PBU}$ **by** *auto*
show «i[a] : $a \rightarrow_{WC} a$»
using *assms 1 weak-unit-iff-self-source weak-unit-iff-self-target*
sources-cod [*of* i[a]] *targets-cod* [*of* i[a]]
by (*elim in-homE*, *auto*)
qed

lemma *unit-simps* [*simp*]:

assumes *weak-unit a*
shows *arr* i[*a*] **and** *dom* i[*a*] = *a* ⋆ *a* **and** *cod* i[*a*] = *a*
using *assms unit-in-vhom*$_{PBU}$ **by** *auto*

end

Every prebicategory extends to a prebicategory with units, simply by choosing the unit isomorphisms arbitrarily.

context *prebicategory*
begin

proposition *extends-to-prebicategory-with-units*:
shows *prebicategory-with-units V H* a *some-unit*
using *iso-some-unit* **by** (*unfold-locales*, *auto*)

end

1.2.4 Horizontal Composition

The following locale axiomatizes a (vertical) category V with horizontal homs, which in addition has been equipped with a functorial operation H of horizontal composition from VV to V, assumed to preserve source and target.

locale *horizontal-composition* =
horizontal-homs V src trg +
H: *functor VV.comp V* ‹λμν. *H* (*fst* μν) (*snd* μν)›
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹⋆› *53*)
and *src* :: *'a* ⇒ *'a*
and *trg* :: *'a* ⇒ *'a* +
assumes *src-hcomp*: *arr* (μ ⋆ ν) ⟹ *src* (μ ⋆ ν) = *src* ν
and *trg-hcomp*: *arr* (μ ⋆ ν) ⟹ *trg* (μ ⋆ ν) = *trg* μ
begin

no-notation *in-hom* (‹«- : - → -»›)

H is a partial composition, which shares its null with *V*.

lemma *is-partial-composition*:
shows *partial-composition H* **and** *partial-magma.null H* = *null*
proof −
have *1*: ∀ *f*. *null* ⋆ *f* = *null* ∧ *f* ⋆ *null* = *null*
using *H.extensionality VV.arr-char*$_{SbC}$ *not-arr-null* **by** *auto*
interpret *H*: *partial-composition H*
using *1 VV.arr-char*$_{SbC}$ *H.extensionality not-arr-null*
by *unfold-locales metis*
show *partial-composition H* **..**
show *H.null* = *null*
using *1 H.null-def the1-equality* [*of* λ*n*. ∀ *f*. *n* ⋆ *f* = *n* ∧ *f* ⋆ *n* = *n*]
by *metis*

qed

Note: The following is "almost" $H.seq$, but for that we would need $H.arr = V.arr$. This would be unreasonable to expect, in general, as the definition of $H.arr$ is based on "strict" units rather than weak units. Later we will show that we do have $H.arr = V.arr$ if the vertical category is discrete.

abbreviation *hseq*
where $hseq\ \nu\ \mu \equiv arr\ (\nu \star \mu)$

lemma *hseq-char*:
shows $hseq\ \nu\ \mu \longleftrightarrow arr\ \mu \wedge arr\ \nu \wedge src\ \nu = trg\ \mu$
proof −
  have $hseq\ \nu\ \mu \longleftrightarrow VV.arr\ (\nu, \mu)$
    using $H.extensionality\ H.preserves\text{-}arr$ **by** *force*
  also have ... $\longleftrightarrow arr\ \mu \wedge arr\ \nu \wedge src\ \nu = trg\ \mu$
    using $VV.arr\text{-}char_{SbC}$ **by** *force*
  finally show *?thesis* **by** *blast*
qed

lemma *hseq-char′*:
shows $hseq\ \nu\ \mu \longleftrightarrow \nu \star \mu \neq null$
  using $VV.arr\text{-}char_{SbC}\ H.preserves\text{-}arr\ H.extensionality\ hseq\text{-}char\ [of\ \nu\ \mu]$ **by** *auto*

lemma *hseqI′* [*intro*, *simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \nu = trg\ \mu$
shows $hseq\ \nu\ \mu$
  using *assms hseq-char* **by** *simp*

lemma *hseqI*:
assumes «$\mu : a \rightarrow b$» **and** «$\nu : b \rightarrow c$»
shows $hseq\ \nu\ \mu$
  using *assms hseq-char* **by** *auto*

lemma *hseqE* [*elim*]:
assumes $hseq\ \nu\ \mu$
and $arr\ \mu \Longrightarrow arr\ \nu \Longrightarrow src\ \nu = trg\ \mu \Longrightarrow T$
shows T
  using *assms hseq-char* **by** *simp*

lemma *hcomp-simps* [*simp*]:
assumes $hseq\ \nu\ \mu$
shows $src\ (\nu \star \mu) = src\ \mu$ **and** $trg\ (\nu \star \mu) = trg\ \nu$
and $dom\ (\nu \star \mu) = dom\ \nu \star dom\ \mu$ **and** $cod\ (\nu \star \mu) = cod\ \nu \star cod\ \mu$
  using *assms* $VV.arr\text{-}char_{SbC}$ *src-hcomp* **apply** *blast*
  using *assms* $VV.arr\text{-}char_{SbC}$ *trg-hcomp* **apply** *blast*
  using *assms* $VV.arr\text{-}char_{SbC}\ H.preserves\text{-}dom\ VV.dom\text{-}simp$ **apply** *force*
  using *assms* $VV.arr\text{-}char_{SbC}\ H.preserves\text{-}cod\ VV.cod\text{-}simp$ **by** *force*

lemma *ide-hcomp* [*intro*, *simp*]:

assumes *ide* ν **and** *ide* μ **and** *src* ν = *trg* μ
shows *ide* ($\nu \star \mu$)
using *assms* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *H.preserves-ide* [*of* (ν, μ)] **by** *auto*

lemma *hcomp-in-hhom* [*intro*]:
assumes «$\mu : a \to b$» **and** «$\nu : b \to c$»
shows «$\nu \star \mu : a \to c$»
using *assms hseq-char* **by** *fastforce*

lemma *hcomp-in-hhom′* :
assumes *arr* μ **and** *arr* ν **and** *src* $\mu = a$ **and** *trg* $\nu = c$ **and** *src* ν = *trg* μ
shows «$\nu \star \mu : a \to c$»
using *assms hseq-char* **by** *fastforce*

lemma *hcomp-in-hhomE* [*elim*]:
assumes «$\nu \star \mu : a \to c$»
and $[\![$ *arr* μ; *arr* ν; *src* ν = *trg* μ; *src* $\mu = a$; *trg* $\nu = c$ $]\!] \Longrightarrow T$
shows T
using *assms in-hhom-def* **by** *fastforce*

lemma *hcomp-in-vhom* [*intro*]:
assumes «$\mu : f \Rightarrow g$» **and** «$\nu : h \Rightarrow k$» **and** *src* h = *trg* f
shows «$\nu \star \mu : h \star f \Rightarrow k \star g$»
using *assms* **by** *fastforce*

lemma *hcomp-in-vhom′* :
assumes *hseq* ν μ
and *dom* $\mu = f$ **and** *dom* $\nu = h$ **and** *cod* $\mu = g$ **and** *cod* $\nu = k$
assumes «$\mu : f \Rightarrow g$» **and** «$\nu : h \Rightarrow k$» **and** *src* h = *trg* f
shows «$\nu \star \mu : h \star f \Rightarrow k \star g$»
using *assms* **by** *fastforce*

lemma *hcomp-in-vhomE* [*elim*]:
assumes «$\nu \star \mu : f \Rightarrow g$»
and $[\![$ *arr* μ; *arr* ν; *src* ν = *trg* μ; *src* μ = *src* f; *src* μ = *src* g;
trg ν = *trg* f; *trg* ν = *trg* g $]\!] \Longrightarrow T$
shows T
using *assms in-hom-def*
by (*metis in-homE hseqE src-cod src-dom src-hcomp trg-cod trg-dom trg-hcomp*)

A horizontal composition yields a weak composition by simply forgetting the *src* and *trg* functors.

lemma *match-1*:
assumes $\nu \star \mu \neq null$ **and** $(\nu \star \mu) \star \tau \neq null$
shows $\mu \star \tau \neq null$
using *assms H.extensionality not-arr-null* $VV.arr\text{-}char_{SbC}$ *hseq-char hseq-char′* **by** *auto*

lemma *match-2*:
assumes $\nu \star (\mu \star \tau) \neq null$ **and** $\mu \star \tau \neq null$

shows $\nu \star \mu \neq null$
  using *assms H.extensionality not-arr-null VV.arr-char*$_{SbC}$ *hseq-char hseq-char′* **by** *auto*

lemma *match-3*:
assumes $\mu \star \tau \neq null$ **and** $\nu \star \mu \neq null$
shows $(\nu \star \mu) \star \tau \neq null$
  using *assms H.extensionality not-arr-null VV.arr-char*$_{SbC}$ *hseq-char hseq-char′* **by** *auto*

lemma *match-4*:
assumes $\mu \star \tau \neq null$ **and** $\nu \star \mu \neq null$
shows $\nu \star (\mu \star \tau) \neq null$
  using *assms H.extensionality not-arr-null VV.arr-char*$_{SbC}$ *hseq-char hseq-char′* **by** *auto*

lemma *left-connected*:
assumes *seq* ν ν'
shows $\nu \star \mu \neq null \longleftrightarrow \nu' \star \mu \neq null$
  using *assms H.extensionality not-arr-null VV.arr-char*$_{SbC}$ *hseq-char′*
  by (*metis hseq-char seqE vseq-implies-hpar*(*1*))

lemma *right-connected*:
assumes *seq* μ μ'
shows $H\ \nu\ \mu \neq null \longleftrightarrow H\ \nu\ \mu' \neq null$
  using *assms H.extensionality not-arr-null VV.arr-char*$_{SbC}$ *hseq-char′*
  by (*metis hseq-char seqE vseq-implies-hpar*(*2*))

proposition *is-weak-composition*:
shows *weak-composition V H*
proof −
  have *1*: $(\lambda\mu\nu.\ fst\ \mu\nu \star snd\ \mu\nu \neq null)$
        $= (\lambda\mu\nu.\ arr\ (fst\ \mu\nu) \wedge arr\ (snd\ \mu\nu) \wedge src\ (fst\ \mu\nu) = trg\ (snd\ \mu\nu))$
    using *hseq-char′* **by** *auto*
  interpret *VoV*: *subcategory VxV.comp* ‹$\lambda\mu\nu.\ fst\ \mu\nu \star snd\ \mu\nu \neq null$›
    using *1 VV.subcategory-axioms* **by** *simp*
  interpret *H*: *functor VoV.comp V* ‹$\lambda\mu\nu.\ fst\ \mu\nu \star snd\ \mu\nu$›
    using *H.functor-axioms 1* **by** *simp*
  show *?thesis*
    using *match-1 match-2 match-3 match-4 left-connected right-connected*
    by (*unfold-locales*, *metis*)
qed

interpretation *weak-composition V H*
  using *is-weak-composition* **by** *auto*

It can be shown that $arr\ ((\nu \cdot \mu) \star (\tau \cdot \sigma)) \Longrightarrow (\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$. However, we do not have $arr\ ((\nu \star \tau) \cdot (\mu \star \sigma)) \Longrightarrow (\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$, because it does not follow from $arr\ ((\nu \star \tau) \cdot (\mu \star \sigma))$ that $dom\ \nu = cod\ \mu$ and $dom\ \tau = cod\ \sigma$, only that $dom\ \nu \star dom\ \tau = cod\ \mu \star cod\ \sigma$. So we don't get interchange unconditionally.

lemma *interchange*:

assumes *seq* ν μ **and** *seq* τ σ
shows $(\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$
  using *assms interchange* **by** *simp*

lemma *whisker-right*:
assumes *ide f* **and** *seq* ν μ
shows $(\nu \cdot \mu) \star f = (\nu \star f) \cdot (\mu \star f)$
  using *assms whisker-right* **by** *simp*

lemma *whisker-left*:
assumes *ide f* **and** *seq* ν μ
shows $f \star (\nu \cdot \mu) = (f \star \nu) \cdot (f \star \mu)$
  using *assms whisker-left* **by** *simp*

lemma *inverse-arrows-hcomp*:
assumes *iso* μ **and** *iso* ν **and** *src* ν = *trg* μ
shows *inverse-arrows* $(\nu \star \mu)$ $(inv\ \nu \star inv\ \mu)$
proof −
  show *inverse-arrows* $(\nu \star \mu)$ $(inv\ \nu \star inv\ \mu)$
  proof
    show *ide* $((inv\ \nu \star inv\ \mu) \cdot (\nu \star \mu))$
    proof −
      have $(inv\ \nu \star inv\ \mu) \cdot (\nu \star \mu) = dom\ \nu \star dom\ \mu$
        using *assms interchange iso-is-arr comp-inv-arr′*
        by (*metis arr-dom*)
      thus *?thesis*
        using *assms iso-is-arr* **by** *simp*
    qed
    show *ide* $((\nu \star \mu) \cdot (inv\ \nu \star inv\ \mu))$
    proof −
      have $(\nu \star \mu) \cdot (inv\ \nu \star inv\ \mu) = cod\ \nu \star cod\ \mu$
        using *assms interchange iso-is-arr comp-arr-inv′*
        by (*metis arr-cod*)
      thus *?thesis*
        using *assms iso-is-arr* **by** *simp*
    qed
  qed
qed

lemma *iso-hcomp* [*intro*, *simp*]:
assumes *iso* μ **and** *iso* ν **and** *src* ν = *trg* μ
shows *iso* $(\nu \star \mu)$
  using *assms inverse-arrows-hcomp* **by** *auto*

lemma *hcomp-iso-in-hom* [*intro*]:
assumes *iso-in-hom* μ f g **and** *iso-in-hom* ν h k **and** *src* ν = *trg* μ
shows *iso-in-hom* $(\nu \star \mu)$ $(h \star f)$ $(k \star g)$
  unfolding *iso-in-hom-def*

using *assms hcomp-in-vhom iso-hcomp iso-in-hom-def vconn-implies-hpar*(*1*−*2*) **by** *auto*

lemma *isomorphic-implies-ide*:
assumes $f \cong g$
shows *ide f* **and** *ide g*
using *assms isomorphic-def* **by** *auto*

lemma *hcomp-ide-isomorphic*:
assumes *ide f* **and** $g \cong h$ **and** *src f* = *trg g*
shows $f \star g \cong f \star h$
proof −
obtain μ **where** μ: *iso* $\mu \wedge$ «$\mu : g \Rightarrow h$»
using *assms isomorphic-def* **by** *auto*
have *iso* $(f \star \mu) \wedge$ «$f \star \mu : f \star g \Rightarrow f \star h$»
using *assms* μ *iso-hcomp* **by** *auto*
thus *?thesis*
using *isomorphic-def* **by** *auto*
qed

lemma *hcomp-isomorphic-ide*:
assumes $f \cong g$ **and** *ide h* **and** *src f* = *trg h*
shows $f \star h \cong g \star h$
proof −
obtain μ **where** μ: *iso* $\mu \wedge$ «$\mu : f \Rightarrow g$»
using *assms isomorphic-def* **by** *auto*
have *iso* $(\mu \star h) \wedge$ «$\mu \star h : f \star h \Rightarrow g \star h$»
using *assms* μ *iso-hcomp* **by** *auto*
thus *?thesis*
using *isomorphic-def* **by** *auto*
qed

lemma *isomorphic-implies-hpar*:
assumes $f \cong f'$
shows *ide f* **and** *ide f′* **and** *src f* = *src f′* **and** *trg f* = *trg f′*
using *assms isomorphic-def* **by** *auto*

lemma *inv-hcomp* [*simp*]:
assumes *iso* ν **and** *iso* μ **and** *src* ν = *trg* μ
shows *inv* $(\nu \star \mu)$ = *inv* $\nu \star$ *inv* μ
using *assms inverse-arrow-unique* [*of* $\nu \star \mu$] *inv-is-inverse inverse-arrows-hcomp*
by *auto*

The following define the two ways of using horizontal composition to compose three arrows.

definition *HoHV*
where *HoHV* $\mu \equiv$ *if VVV.arr* μ *then* (*fst* $\mu \star$ *fst* (*snd* μ)) $\star$ *snd* (*snd* μ) *else null*

definition *HoVH*
where *HoVH* $\mu \equiv$ *if VVV.arr* μ *then fst* $\mu \star$ *fst* (*snd* μ) $\star$ *snd* (*snd* μ) *else null*

lemma *functor-HoHV*:
shows *functor VVV.comp V HoHV*
apply *unfold-locales*
using $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VVV.dom\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ *VVV.comp-char HoHV-def*
apply *auto*[4]
proof −
fix *f g*
assume *fg*: *VVV.seq g f*
show *HoHV* (*VVV.comp g f*) = *HoHV g* · *HoHV f*
proof −
have *VxVxV.comp g f* =
(*fst g* · *fst f*, *fst* (*snd g*) · *fst* (*snd f*), *snd* (*snd g*) · *snd* (*snd f*))
using *fg* $VVV.seq\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *VxVxV.comp-char VxV.comp-char*
by (*metis* (*no-types*, *lifting*) $VxV.seqE_{PC}$ $VxVxV.seqE_{PC}$)
hence *HoHV* (*VVV.comp g f*) =
(*fst g* · *fst f* ⋆ *fst* (*snd g*) · *fst* (*snd f*)) ⋆ *snd* (*snd g*) · *snd* (*snd f*)
using *HoHV-def VVV.comp-simp fg* **by** *auto*
also have ... = ((*fst g* ⋆ *fst* (*snd g*)) ⋆ *snd* (*snd g*)) ·
((*fst f* ⋆ *fst* (*snd f*)) ⋆ *snd* (*snd f*))
using *fg* $VVV.seq\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *interchange*
by (*metis* (*no-types*, *lifting*) $VxV.seqE_{PC}$ $VxVxV.seqE_{PC}$ *hseqI′ src-vcomp trg-vcomp*)
also have ... = *HoHV g* · *HoHV f*
using *HoHV-def fg* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed

sublocale *HoHV*: *functor VVV.comp V HoHV*
using *functor-HoHV* **by** *simp*

lemma *functor-HoVH*:
shows *functor VVV.comp V HoVH*
apply *unfold-locales*
using $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VVV.dom\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ *VVV.comp-char HoHV-def HoVH-def*
apply *auto*[4]
proof −
fix *f g*
assume *fg*: *VVV.seq g f*
show *HoVH* (*VVV.comp g f*) = *HoVH g* · *HoVH f*
proof −
have *VxVxV.comp g f* =
(*fst g* · *fst f*, *fst* (*snd g*) · *fst* (*snd f*), *snd* (*snd g*) · *snd* (*snd f*))
using *fg* $VVV.seq\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *VxVxV.comp-char VxV.comp-char*
by (*metis* (*no-types*, *lifting*) $VxV.seqE_{PC}$ $VxVxV.seqE_{PC}$)

```
      hence HoVH (VVV.comp g f) =
          fst g · fst f ⋆ fst (snd g) · fst (snd f) ⋆ snd (snd g) · snd (snd f)
        using HoVH-def VVV.comp-simp fg by auto
      also have ... = (fst g ⋆ fst (snd g) ⋆ snd (snd g)) ·
                      (fst f ⋆ fst (snd f) ⋆ snd (snd f))
        using fg VVV.seq-char_SbC VVV.arr-char_SbC VV.arr-char_SbC interchange
        by (metis (no-types, lifting) VxV.seqE_PC VxVxV.seqE_PC hseqI' src-vcomp trg-vcomp)
      also have ... = HoVH g · HoVH f
     using fg VVV.seq-char_SbC VVV.arr-char_SbC HoVH-def VVV.comp-char VV.arr-char_SbC
        by (metis (no-types, lifting))
      finally show ?thesis by simp
    qed
  qed

  sublocale HoVH: functor VVV.comp V HoVH
    using functor-HoVH by simp
```

The following define horizontal composition of an arrow on the left by its target and on the right by its source.

```
  abbreviation L
  where L ≡ λμ. if arr μ then trg μ ⋆ μ else null

  abbreviation R
  where R ≡ λμ. if arr μ then μ ⋆ src μ else null

  sublocale L: endofunctor V L
    using vseq-implies-hpar(2) whisker-left
    by (unfold-locales, auto)

  lemma endofunctor-L:
  shows endofunctor V L
    ..

  sublocale R: endofunctor V R
    using vseq-implies-hpar(1) whisker-right
    by (unfold-locales, auto)

  lemma endofunctor-R:
  shows endofunctor V R
    ..

 end

end


theory Bicategory
imports Prebicategory Category3.Subcategory Category3.DiscreteCategory
      MonoidalCategory.MonoidalCategory
```

begin

1.3 Bicategories

A *bicategory* is a (vertical) category that has been equipped with a horizontal composition, an associativity natural isomorphism, and for each object a "unit isomorphism", such that horizontal composition on the left by target and on the right by source are fully faithful endofunctors of the vertical category, and such that the usual pentagon coherence condition holds for the associativity.

locale *bicategory* =
  horizontal-composition V H src trg +
  α: *natural-isomorphism VVV.comp V HoHV HoVH*
    ‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))› +
  L: *fully-faithful-functor V V L* +
  R: *fully-faithful-functor V V R*
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹⋆› *53*)
and a :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›)
and i :: *'a* ⇒ *'a* (‹i[-]›)
and *src* :: *'a* ⇒ *'a*
and *trg* :: *'a* ⇒ *'a* +
assumes *unit-in-vhom*: *obj a* ⟹ «i[*a*] : *a* ⋆ *a* ⇒ *a*»
and *iso-unit*: *obj a* ⟹ *iso* i[*a*]
and *pentagon*: ⟦ *ide f*; *ide g*; *ide h*; *ide k*; *src f* = *trg g*; *src g* = *trg h*; *src h* = *trg k* ⟧ ⟹
    (*f* ⋆ a[*g*, *h*, *k*]) · a[*f*, *g* ⋆ *h*, *k*] · (a[*f*, *g*, *h*] ⋆ *k*) = a[*f*, *g*, *h* ⋆ *k*] · a[*f* ⋆ *g*, *h*, *k*]
begin

definition α
where α $\mu\nu\tau$ ≡ a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))

lemma *assoc-in-hom'*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows *in-hhom* a[μ, ν, τ] (*src* τ) (*trg* μ)
and «a[μ, ν, τ] : (*dom* μ ⋆ *dom* ν) ⋆ *dom* τ ⇒ *cod* μ ⋆ *cod* ν ⋆ *cod* τ»
proof −
  show «a[μ, ν, τ] : (*dom* μ ⋆ *dom* ν) ⋆ *dom* τ ⇒ *cod* μ ⋆ *cod* ν ⋆ *cod* τ»
  proof −
    have *1*: *VVV.in-hom* (μ, ν, τ) (*dom* μ, *dom* ν, *dom* τ) (*cod* μ, *cod* ν, *cod* τ)
      using *assms* $VVV.in\text{-}hom\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *auto*
    have «a[μ, ν, τ] : *HoHV* (*dom* μ, *dom* ν, *dom* τ) ⇒ *HoVH* (*cod* μ, *cod* ν, *cod* τ)»
      using *1* α*.preserves-hom* **by** *auto*
    moreover have *HoHV* (*dom* μ, *dom* ν, *dom* τ) = (*dom* μ ⋆ *dom* ν) ⋆ *dom* τ
      using *1 HoHV-def* **by** (*simp add*: $VVV.in\text{-}hom\text{-}char_{SbC}$)
    moreover have *HoVH* (*cod* μ, *cod* ν, *cod* τ) = *cod* μ ⋆ *cod* ν ⋆ *cod* τ
      using *1 HoVH-def* **by** (*simp add*: $VVV.in\text{-}hom\text{-}char_{SbC}$)
    ultimately show *?thesis* **by** *simp*
  qed

thus *in-hhom* a[μ, ν, τ] (*src* τ) (*trg* μ)
using *assms src-cod trg-cod vconn-implies-hpar*(*1*) *vconn-implies-hpar*(*2*) **by** *auto*
qed

lemma *assoc-naturality1*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows a[μ, ν, τ] = ($\mu \star \nu \star \tau$) · a[*dom* μ, *dom* ν, *dom* τ]
using *assms* α.*naturality1* [*of* (μ, ν, τ)] *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VVV.dom-char*$_{SbC}$ *HoVH-def src-dom trg-dom*
by *simp*

lemma *assoc-naturality2*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows a[μ, ν, τ] = a[*cod* μ, *cod* ν, *cod* τ] · (($\mu \star \nu$) $\star$ τ)
using *assms* α.*naturality2* [*of* (μ, ν, τ)] *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VVV.cod-char*$_{SbC}$ *HoHV-def src-dom trg-dom*
by *simp*

lemma *assoc-naturality*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows a[*cod* μ, *cod* ν, *cod* τ] · (($\mu \star \nu$) $\star$ τ) = ($\mu \star \nu \star \tau$) · a[*dom* μ, *dom* ν, *dom* τ]
using *assms* α.*naturality* *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *HoVH-def HoHV-def* *VVV.dom-char*$_{SbC}$ *VVV.cod-char*$_{SbC}$
by *auto*

lemma *assoc-in-hom* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *in-hhom* a[*f*, *g*, *h*] (*src h*) (*trg f*)
and «a[*f*, *g*, *h*] : (*dom f* $\star$ *dom g*) $\star$ *dom h* $\Rightarrow$ *cod f* $\star$ *cod g* $\star$ *cod h*»
using *assms assoc-in-hom'* **apply** *auto*[*1*]
using *assms assoc-in-hom' ideD*(*1*) **by** *metis*

lemma *assoc-simps* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *arr* a[*f*, *g*, *h*]
and *src* a[*f*, *g*, *h*] = *src h* **and** *trg* a[*f*, *g*, *h*] = *trg f*
and *dom* a[*f*, *g*, *h*] = (*dom f* $\star$ *dom g*) $\star$ *dom h*
and *cod* a[*f*, *g*, *h*] = *cod f* $\star$ *cod g* $\star$ *cod h*
using *assms assoc-in-hom* **apply** *auto*
using *assoc-in-hom*(*1*) **by** *auto*

lemma *iso-assoc* [*intro*, *simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *iso* a[*f*, *g*, *h*]
using *assms* α.*components-are-iso* [*of* (*f*, *g*, *h*)] *VVV.ide-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by *simp*

end

1.3.1 Categories Induce Bicategories

In this section we show that a category becomes a bicategory if we take the vertical composition to be discrete, we take the composition of the category as the horizontal composition, and we take the vertical domain and codomain as *src* and *trg*.

```
locale category-as-bicategory = category
begin

  interpretation V: discrete-category ‹Collect arr› null
    using not-arr-null by (unfold-locales, blast)

  abbreviation V
  where V ≡ V.comp

  interpretation src: functor V V dom
    using V.null-char
    by (unfold-locales, simp add: has-domain-iff-arr dom-def, auto)
  interpretation trg: functor V V cod
    using V.null-char
    by (unfold-locales, simp add: has-codomain-iff-arr cod-def, auto)

  interpretation H: horizontal-homs V dom cod
    by (unfold-locales, auto)

  interpretation H: functor H.VV.comp V ‹λμν. fst μν · snd μν›
    apply (unfold-locales)
    using H.VV.arr-char_SbC V.null-char ext
       apply force
    using H.VV.arr-char_SbC V.null-char H.VV.dom-char_SbC H.VV.cod-char_SbC
      apply auto[3]
  proof −
    show ⋀g f. H.VV.seq g f ⟹
              fst (H.VV.comp g f) · snd (H.VV.comp g f) = V (fst g · snd g) (fst f · snd f)
    proof −
      have 0: ⋀f. H.VV.arr f ⟹ V.arr (fst f · snd f)
        using H.VV.arr-char_SbC by auto
      have 1: ⋀f g. V.seq g f ⟹ V.ide f ∧ g = f
        using V.arr-char V.dom-char V.cod-char V.not-arr-null by force
      have 2: ⋀f g. H.VxV.seq g f ⟹ H.VxV.ide f ∧ g = f
        using 1 H.VxV.seq-char by (metis H.VxV.dom-eqI H.VxV.ide-Ide)
      fix f g
      assume fg: H.VV.seq g f
      have 3: H.VV.ide f ∧ f = g
        using fg 2 H.VV.seq-char_SbC H.VV.ide-char_SbC by blast
      show fst (H.VV.comp g f) · snd (H.VV.comp g f) = V (fst g · snd g) (fst f · snd f)
           using fg 0 1 3 H.VV.comp-char H.VV.arr-char_SbC H.VV.ide-char V.arr-char
V.comp-char
             H.VV.comp-arr-ide
        by (metis (no-types, lifting))
```

qed
qed

interpretation H: *horizontal-composition V C dom cod*
by (*unfold-locales*, *auto*)

abbreviation a
where a $f\ g\ h \equiv f \cdot g \cdot h$

interpretation α: *natural-isomorphism H.VVV.comp V H.HoHV H.HoVH*
‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))›
apply *unfold-locales*
using *V.null-char ext*
apply *fastforce*
using *H.HoHV-def H.HoVH-def H.VVV.arr-char*$_{SbC}$ *H.VV.arr-char*$_{SbC}$ *H.VVV.dom-char*$_{SbC}$ *H.VV.dom-char*$_{SbC}$ *H.VVV.cod-char*$_{SbC}$ *H.VV.cod-char*$_{SbC}$ *H.VVV.ide-char comp-assoc*
by *auto*

interpretation *fully-faithful-functor V V H.R*
using *comp-arr-dom* **by** (*unfold-locales*, *auto*)
interpretation *fully-faithful-functor V V H.L*
using *comp-cod-arr* **by** (*unfold-locales*, *auto*)

abbreviation i
where i $\equiv \lambda x.\ x$

proposition *induces-bicategory*:
shows *bicategory V C* a i *dom cod*
apply (*unfold-locales*, *auto simp add*: *comp-assoc*)
using *comp-arr-dom* **by** *fastforce*

end

1.3.2 Monoidal Categories induce Bicategories

In this section we show that our definition of bicategory directly generalizes our definition of monoidal category: a monoidal category becomes a bicategory when equipped with the constant-$\mathcal{I}$ functors as src and trg and ι as the unit isomorphism from $\mathcal{I} \otimes \mathcal{I}$ to $\mathcal{I}$. There is a slight mismatch because the bicategory locale assumes that the associator is given in curried form, whereas for monoidal categories it is given in tupled form. Ultimately, the monoidal category locale should be revised to also use curried form, which ends up being more convenient in most situations.

context *monoidal-category*
begin

interpretation I: *constant-functor C C* $\mathcal{I}$
using *unit-in-hom* **by** *unfold-locales auto*

```
interpretation horizontal-homs C I.map I.map
  by unfold-locales auto

lemma CC-eq-VV:
shows CC.comp = VV.comp
proof −
  have ⋀g f. CC.comp g f = VV.comp g f
  proof −
    fix f g
    show CC.comp g f = VV.comp g f
    proof −
      have CC.seq g f ⟹ CC.comp g f = VV.comp g f
        using VV.comp-char VV.arr-char_SbC CC.seq-char
        by (elim CC.seqE seqE, simp)
      moreover have ¬ CC.seq g f ⟹ CC.comp g f = VV.comp g f
        using VV.seq-char_SbC VV.ext VV.null-char CC.ext
        by (metis (no-types, lifting))
      ultimately show ?thesis by blast
    qed
  qed
  thus ?thesis by blast
qed

lemma CCC-eq-VVV:
shows CCC.comp = VVV.comp
proof −
  have ⋀g f. CCC.comp g f = VVV.comp g f
  proof −
    fix f g
    show CCC.comp g f = VVV.comp g f
    proof −
      have CCC.seq g f ⟹ CCC.comp g f = VVV.comp g f
        by (metis (no-types, lifting) CC.arrE CCC.seqE_PC CC-eq-VV I.map-simp
       I.preserves-reflects-arr VV.seq-char_SbC VVV.arrI_SbC VVV.comp-simp VVV.seq-char_SbC
           trg-vcomp vseq-implies-hpar(1))
      moreover have ¬ CCC.seq g f ⟹ CCC.comp g f = VVV.comp g f
        using VVV.seq-char_SbC VVV.ext VVV.null-char CCC.ext
        by (metis (no-types, lifting))
      ultimately show ?thesis by blast
    qed
  qed
  thus ?thesis by blast
qed

interpretation H: functor VV.comp C ‹λμν. fst μν ⊗ snd μν›
  using CC-eq-VV T.functor-axioms by simp
interpretation H: horizontal-composition C tensor I.map I.map
  by (unfold-locales, simp-all)
```

lemma *HoHV-eq-ToTC*:
shows $H.HoHV = T.ToTC$
  using *H.HoHV-def T.ToTC-def CCC-eq-VVV* **by** *presburger*

lemma *HoVH-eq-ToCT*:
shows $H.HoVH = T.ToCT$
  using *H.HoVH-def T.ToCT-def CCC-eq-VVV* **by** *presburger*

interpretation α: *natural-isomorphism VVV.comp C H.HoHV H.HoVH* α
  using α*.natural-isomorphism-axioms CCC-eq-VVV HoHV-eq-ToTC HoVH-eq-ToCT*
  by *simp*

lemma *R′-eq-R*:
shows $H.R = R$
  using *H.extensionality CC-eq-VV CC.arr-char* **by** *force*

lemma *L′-eq-L*:
shows $H.L = L$
  using *H.extensionality CC-eq-VV CC.arr-char* **by** *force*

interpretation *R′*: *fully-faithful-functor C C H.R*
  using *R′-eq-R R.fully-faithful-functor-axioms* **by** *auto*
interpretation *L′*: *fully-faithful-functor C C H.L*
  using *L′-eq-L L.fully-faithful-functor-axioms* **by** *auto*

lemma *obj-char*:
shows *obj a* $\longleftrightarrow$ $a = \mathcal{I}$
  using *obj-def* [*of a*] *unit-in-hom* **by** *fastforce*

proposition *induces-bicategory*:
shows *bicategory C tensor* ($\lambda\mu\ \nu\ \tau.\ \alpha\ (\mu, \nu, \tau)$) ($\lambda$-. ι) *I.map I.map*
  using *unit-in-hom unit-is-iso pentagon* α*.extensionality*
    α*.naturality1* α*.naturality2 CCC.arrE CCC-eq-VVV*
  by *unfold-locales auto*

end

1.3.3 Prebicategories Extend to Bicategories

In this section, we show that a prebicategory with homs and units extends to a bicategory. The main work is to show that the endofunctors L and R are fully faithful. We take the left and right unitor isomorphisms, which were obtained via local constructions in the left and right hom-subcategories defined by a specified weak unit, and show that in the presence of the chosen sources and targets they are the components of a global natural isomorphisms $\mathfrak{l}$ and $\mathfrak{r}$ from the endofunctors L and R to the identity functor. A consequence is that functors L and R are endo-equivalences, hence fully faithful.

context *prebicategory-with-homs*
begin

Once it is equipped with a particular choice of source and target for each arrow, a prebicategory determines a horizontal composition.

lemma *induces-horizontal-composition*:
shows *horizontal-composition V H src trg*
proof −
  interpret *H*: *functor VV.comp V* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ ⋆ *snd* $\mu\nu$›
  proof −
    have *VV.comp* = *VoV.comp*
      using *composable-char*$_{PBH}$ **by** *meson*
    thus *functor VV.comp V* ($\lambda\mu\nu$. *fst* $\mu\nu$ ⋆ *snd* $\mu\nu$)
      using *functor-axioms* **by** *argo*
  qed
  show *horizontal-composition V H src trg*
    using *src-hcomp′ trg-hcomp′ composable-char*$_{PBH}$ *not-arr-null*
    by (*unfold-locales*; *metis*)
qed

end

sublocale *prebicategory-with-homs* ⊆ *horizontal-composition V H src trg*
  using *induces-horizontal-composition* **by** *auto*

locale *prebicategory-with-homs-and-units* =
  prebicategory-with-units +
  prebicategory-with-homs
begin

no-notation *in-hom* (‹«- : - → -»›)

The next definitions extend the left and right unitors that were defined locally with respect to a particular weak unit, to globally defined versions using the chosen source and target for each arrow.

definition *lunit* (‹l[-]›)
where *lunit f* ≡ *left-hom-with-unit.lunit V H* a i[*trg f*] (*trg f*) *f*

definition *runit* (‹r[-]›)
where *runit f* ≡ *right-hom-with-unit.runit V H* a i[*src f*] (*src f*) *f*

lemma *lunit-in-hom*:
assumes *ide f*
shows «l[*f*] : *src f* $\rightarrow_{WC}$ *trg f*» **and** «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
proof −
  interpret *Left*: *subcategory V* ‹*left* (*trg f*)›
    using *assms left-hom-is-subcategory* **by** *simp*
  interpret *Left*: *left-hom-with-unit V H* a ‹i[*trg f*]› ‹*trg f*›
    using *assms obj-is-weak-unit iso-unit*$_{PBU}$ **by** (*unfold-locales*, *auto*)
  have *0*: *Left.ide f*
    using *assms Left.ide-char*$_{SbC}$ *Left.arr-char*$_{SbC}$ *left-def composable-char*$_{PBH}$ **by** *auto*
  show *1*: «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»

unfolding *lunit-def*
using *assms 0 Left.lunit-char(1) Left.hom-char* H_L*-def* **by** *auto*
show «l[*f*] : *src f* $\rightarrow_{WC}$ *trg f*»
using *1 src-cod trg-cod src-in-sources trg-in-targets*
by (*metis arrI vconn-implies-hpar*)
qed

lemma *runit-in-hom*:
assumes *ide f*
shows «r[*f*] : *src f* $\rightarrow_{WC}$ *trg f*» **and** «r[*f*] : *f* ⋆ *src f* ⇒ *f*»
proof −
interpret *Right*: *subcategory V* ‹*right* (*src f*)›
using *assms right-hom-is-subcategory weak-unit-self-composable* **by** *force*
interpret *Right*: *right-hom-with-unit V H* a ‹i[*src f*]› ‹*src f*›
using *assms obj-is-weak-unit iso-unit*$_{PBU}$ **by** (*unfold-locales*, *auto*)
have *0*: *Right.ide f*
using *assms Right.ide-char*$_{SbC}$ *Right.arr-char*$_{SbC}$ *right-def composable-char*$_{PBH}$ **by** *auto*
show *1*: «r[*f*] : *f* ⋆ *src f* ⇒ *f*»
unfolding *runit-def*
using *assms 0 Right.runit-char(1) Right.hom-char* H_R*-def* **by** *auto*
show «r[*f*] : *src f* $\rightarrow_{WC}$ *trg f*»
using *1 src-cod trg-cod src-in-sources trg-in-targets*
by (*metis arrI vconn-implies-hpar*)
qed

The characterization of the locally defined unitors yields a corresponding characterization of the globally defined versions, by plugging in the chosen source or target for each arrow for the unspecified weak unit in the the local versions.

lemma *lunit-char*:
assumes *ide f*
shows «l[*f*] : *src f* $\rightarrow_{WC}$ *trg f*» **and** «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
and *trg f* ⋆ l[*f*] = (i[*trg f*] ⋆ *f*) · *inv* a[*trg f*, *trg f*, *f*]
and ∃!*μ*. «*μ* : *trg f* ⋆ *f* ⇒ *f*» ∧ *trg f* ⋆ *μ* = (i[*trg f*] ⋆ *f*) · *inv* a[*trg f*, *trg f*, *f*]
proof −
let *?a* = *src f* **and** *?b* = *trg f*
interpret *Left*: *subcategory V* ‹*left ?b*›
using *assms left-hom-is-subcategory weak-unit-self-composable* **by** *force*
interpret *Left*: *left-hom-with-unit V H* a ‹i[*?b*]› *?b*
using *assms obj-is-weak-unit iso-unit*$_{PBU}$ **by** (*unfold-locales*, *auto*)
have *0*: *Left.ide f*
using *assms Left.ide-char*$_{SbC}$ *Left.arr-char*$_{SbC}$ *left-def composable-char*$_{PBH}$ **by** *auto*
show «l[*f*] : *src f* $\rightarrow_{WC}$ *trg f*»
using *assms lunit-in-hom* **by** *simp*
show *A*: «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
using *assms lunit-in-hom* **by** *simp*
show *B*: *?b* ⋆ l[*f*] = (i[*?b*] ⋆ *f*) · *inv* a[*?b*, *?b*, *f*]
unfolding *lunit-def* **using** *0 Left.lunit-char(2)* H_L*-def*
by (*metis Left.comp-simp Left.characteristic-iso(1−2) Left.seqI′*)
show ∃!*μ*. «*μ* : *trg f* ⋆ *f* ⇒ *f*» ∧ *trg f* ⋆ *μ* = (i[*?b*] ⋆ *f*) · *inv* a[*?b*, *?b*, *f*]

```
  proof −
    have 1: hom (trg f ⋆ f) f = Left.hom (Left.L f) f
    proof
      have 1: Left.L f = ?b ⋆ f
      using 0 H_L-def by simp
      show Left.hom (Left.L f) f ⊆ hom (?b ⋆ f) f
        using assms Left.hom-char [of ?b ⋆ f f] H_L-def by simp
      show hom (?b ⋆ f) f ⊆ Left.hom (Left.L f) f
        using assms 1 ide-in-hom composable-char_PBH hom-connected left-def
              Left.hom-char
        by auto
    qed
    let ?P = λμ. Left.in-hom μ (Left.L f) f
    let ?P′ = λμ. «μ : ?b ⋆ f ⇒ f»
    let ?Q = λμ. Left.L μ = (i[?b] ⋆ f) · (inv a[?b, ?b, f])
    let ?R = λμ. ?b ⋆ μ = (i[?b] ⋆ f) · (inv a[?b, ?b, f])
    have 2: ?P = ?P′
      using 0 1 H_L-def Left.hom-char by blast
    moreover have ∀μ. ?P μ ⟶ (?Q μ ⟷ ?R μ)
      using 2 Left.lunit-eqI H_L-def by presburger
    moreover have (∃!μ. ?P μ ∧ ?Q μ)
      using 0 2 A B Left.lunit-char(3) Left.ide-char Left.arr-char_SbC
      by (metis (no-types, lifting) Left.lunit-char(2) calculation(2) lunit-def)
    ultimately show ?thesis by metis
  qed
qed

lemma runit-char:
assumes ide f
shows «r[f] : src f →_WC trg f» and «r[f] : f ⋆ src f ⇒ f»
and r[f] ⋆ src f = (f ⋆ i[src f]) · a[f, src f, src f]
and ∃!μ. «μ : f ⋆ src f ⇒ f» ∧ μ ⋆ src f = (f ⋆ i[src f]) · a[f, src f, src f]
proof −
  let ?a = src f and ?b = trg f
  interpret Right: subcategory V ‹right ?a›
    using assms right-hom-is-subcategory weak-unit-self-composable by force
  interpret Right: right-hom-with-unit V H a ‹i[?a]› ?a
    using assms obj-is-weak-unit iso-unit_PBU by (unfold-locales, auto)
  have 0: Right.ide f
   using assms Right.ide-char_SbC Right.arr-char_SbC right-def composable-char_PBH by auto
  show «r[f] : src f →_WC trg f»
    using assms runit-in-hom by simp
  show A: «r[f] : f ⋆ src f ⇒ f»
    using assms runit-in-hom by simp
  show B: r[f] ⋆ ?a = (f ⋆ i[?a]) · a[f, ?a, ?a]
    unfolding runit-def using 0 Right.runit-char(2) H_R-def
    using Right.comp-simp Right.characteristic-iso(4) Right.iso-is-arr by auto
  show ∃!μ. «μ : f ⋆ src f ⇒ f» ∧ μ ⋆ ?a = (f ⋆ i[?a]) · a[f, ?a, ?a]
  proof −
```

```
    have 1: hom (f ⋆ ?a) f = Right.hom (Right.R f) f
    proof
      have 1: Right.R f = f ⋆ ?a
      using 0 H_R-def by simp
      show Right.hom (Right.R f) f ⊆ hom (f ⋆ ?a) f
        using assms Right.hom-char [of f ⋆ ?a f] H_R-def by simp
      show hom (f ⋆ ?a) f ⊆ Right.hom (Right.R f) f
        using assms 1 ide-in-hom composable-char_PBH hom-connected right-def
              Right.hom-char
        by auto
    qed
    let ?P = λμ. Right.in-hom μ (Right.R f) f
    let ?P' = λμ. «μ : f ⋆ ?a ⇒ f»
    let ?Q = λμ. Right.R μ = (f ⋆ i[?a]) · a[f, ?a, ?a]
    let ?R = λμ. μ ⋆ ?a = (f ⋆ i[?a]) · a[f, ?a, ?a]
    have 2: ?P = ?P'
      using 0 1 H_R-def Right.hom-char by blast
    moreover have ∀μ. ?P μ ⟶ (?Q μ ⟷ ?R μ)
      using 2 Right.runit-eqI H_R-def by presburger
    moreover have (∃!μ. ?P μ ∧ ?Q μ)
      using 0 2 A B Right.runit-char(3) Right.ide-char Right.arr-char_SbC
      by (metis (no-types, lifting) Right.runit-char(2) calculation(2) runit-def)
    ultimately show ?thesis by metis
  qed
qed

lemma lunit-eqI:
assumes ide f and «μ : trg f ⋆ f ⇒ f»
and trg f ⋆ μ = (i[trg f] ⋆ f) · (inv a[trg f, trg f, f])
shows μ = l[f]
  using assms lunit-char(2−4) by blast

lemma runit-eqI:
assumes ide f and «μ : f ⋆ src f ⇒ f»
and μ ⋆ src f = (f ⋆ i[src f]) · a[f, src f, src f]
shows μ = r[f]
  using assms runit-char(2−4) by blast

lemma iso-lunit:
assumes ide f
shows iso l[f]
proof −
  let ?b = trg f
  interpret Left: subcategory V ‹left ?b›
    using assms left-hom-is-subcategory weak-unit-self-composable by force
  interpret Left: left-hom-with-unit V H a ‹i[?b]› ?b
    using assms obj-is-weak-unit iso-unit_PBU by (unfold-locales, auto)
  show ?thesis
  proof −
```

have *0*: *Left.ide f*
using *assms Left.ide-char$_{SbC}$ Left.arr-char$_{SbC}$ left-def composable-char$_{PBH}$* **by** *auto*
thus *?thesis*
unfolding *lunit-def* **using** *Left.iso-lunit Left.iso-char* **by** *blast*
qed
qed

lemma *iso-runit*:
assumes *ide f*
shows *iso* r[*f*]
proof −
let *?a* = *src f*
interpret *Right*: *subcategory V* ‹*right ?a*›
using *assms right-hom-is-subcategory weak-unit-self-composable* **by** *force*
interpret *Right*: *right-hom-with-unit V H* a ‹i[*?a*]› *?a*
using *assms obj-is-weak-unit iso-unit$_{PBU}$* **by** (*unfold-locales*, *auto*)
show *?thesis*
proof −
have *0*: *Right.ide f*
using *assms Right.ide-char$_{SbC}$ Right.arr-char$_{SbC}$ right-def composable-char$_{PBH}$* **by** *auto*
thus *?thesis*
unfolding *runit-def* **using** *Right.iso-runit Right.iso-char* **by** *blast*
qed
qed

lemma *lunit-naturality*:
assumes *arr* μ
shows $\mu \cdot$ l[*dom* μ] = l[*cod* μ] $\cdot$ (*trg* $\mu \star \mu$)
proof −
let *?a* = *src* μ **and** *?b* = *trg* μ
interpret *Left*: *subcategory V* ‹*left ?b*›
using *assms obj-trg left-hom-is-subcategory weak-unit-self-composable* **by** *force*
interpret *Left*: *left-hom-with-unit V H* a ‹i[*?b*]› *?b*
using *assms obj-is-weak-unit iso-unit$_{PBU}$* **by** (*unfold-locales*, *auto*)
interpret *Left.L*: *endofunctor* ‹*Left ?b*› *Left.L*
using *assms endofunctor-H$_L$* [*of ?b*] *weak-unit-self-composable obj-trg obj-is-weak-unit*
by *blast*
have *1*: *Left.in-hom* μ (*dom* μ) (*cod* μ)
using *assms Left.hom-char Left.arr-char$_{SbC}$ left-def composable-char$_{PBH}$ obj-trg* **by** *auto*
have *2*: *Left.in-hom* l[*Left.dom* μ] (*?b* $\star$ *dom* μ) (*dom* μ)
unfolding *lunit-def*
using *assms 1 Left.in-hom-char$_{SbC}$ trg-dom Left.lunit-char*(*1*) *H$_L$-def*
Left.arr-char$_{SbC}$ Left.dom-char$_{SbC}$ Left.ide-dom
by *force*
have *3*: *Left.in-hom* l[*Left.cod* μ] (*?b* $\star$ *cod* μ) (*cod* μ)
unfolding *lunit-def*
using *assms 1 Left.in-hom-char$_{SbC}$ trg-cod Left.lunit-char*(*1*) *H$_L$-def*
Left.cod-char$_{SbC}$ Left.ide-cod

by *force*
have *4*: *Left.in-hom* (*Left.L* μ) (*?b* $\star$ *dom* μ) (*?b* $\star$ *cod* μ)
using *1 Left.L.preserves-hom* [*of* μ *dom* μ *cod* μ] H_L*-def* **by** *auto*
show *?thesis*
by (*metis 1 2* H_L*-def Left.comp-simp Left.in-homE Left.lunit-naturality Left.seqI′ lunit-def trg-cod trg-dom*)
qed

lemma *runit-naturality*:
assumes *arr* μ
shows $\mu \cdot \mathrm{r}[dom\ \mu] = \mathrm{r}[cod\ \mu] \cdot (\mu \star src\ \mu)$
proof −
let *?a* = *src* μ **and** *?b* = *trg* μ
interpret *Right*: *subcategory V* ‹*right ?a*›
using *assms right-hom-is-subcategory weak-unit-self-composable* **by** *force*
interpret *Right*: *right-hom-with-unit V H* a ‹i[*?a*]› *?a*
using *assms obj-is-weak-unit iso-unit*$_{PBU}$ **by** (*unfold-locales*, *auto*)
interpret *Right.R*: *endofunctor* ‹*Right ?a*› *Right.R*
using *assms endofunctor-*H_R [*of ?a*] *weak-unit-self-composable obj-src obj-is-weak-unit*
by *blast*
have *1*: *Right.in-hom* μ (*dom* μ) (*cod* μ)
using *assms Right.hom-char Right.arr-char*$_{SbC}$ *right-def composable-char*$_{PBH}$ **by** *auto*
have *2*: *Right.in-hom* r[*Right.dom* μ] (*dom* $\mu \star$ *?a*) (*dom* μ)
unfolding *runit-def*
using *1 Right.in-hom-char*$_{SbC}$ *trg-dom Right.runit-char*(*1*) [*of Right.dom* μ] H_R*-def Right.arr-char*$_{SbC}$ *Right.dom-char*$_{SbC}$ *Right.ide-dom assms*
by *force*
have *3*: r[*Right.cod* μ] $\in$ *Right.hom* (*cod* $\mu \star$ *?a*) (*cod* μ)
unfolding *runit-def*
using *1 Right.in-hom-char*$_{SbC}$ *trg-cod Right.runit-char*(*1*) [*of Right.cod* μ] H_R*-def Right.cod-char*$_{SbC}$ *Right.ide-cod assms*
by *force*
have *4*: *Right.R* $\mu \in$ *Right.hom* (*dom* $\mu \star$ *?a*) (*cod* $\mu \star$ *?a*)
using *1 Right.R.preserves-hom* [*of* μ *dom* μ *cod* μ] H_R*-def* **by** *auto*
show *?thesis*
by (*metis 1 2* H_R*-def Right.comp-simp Right.in-homE Right.runit-naturality Right.seqI′ runit-def src-cod src-dom*)
qed

interpretation *L*: *endofunctor V L*
using *endofunctor-L* **by** *auto*
interpretation ɩ: *transformation-by-components V V L map lunit*
using *lunit-in-hom lunit-naturality* **by** *unfold-locales auto*
interpretation ɩ: *natural-isomorphism V V L map* ɩ*.map*
using *iso-lunit* **by** *unfold-locales auto*

lemma *natural-isomorphism-*ɩ:
shows *natural-isomorphism V V L map* ɩ*.map*
..

interpretation *L*: *equivalence-functor V V L*
  using *L.isomorphic-to-identity-is-equivalence* $\mathfrak{l}$*.natural-isomorphism-axioms* **by** *simp*

lemma *equivalence-functor-L*:
shows *equivalence-functor V V L*
  ..

lemma *lunit-commutes-with-L*:
assumes *ide f*
shows $\mathrm{l}[L\ f] = L\ \mathrm{l}[f]$
proof −
  have *seq* $\mathrm{l}[f]$ $(L\ \mathrm{l}[f])$
    using *assms lunit-char*(*2*) *L.preserves-hom* **by** *fastforce*
  moreover have *seq* $\mathrm{l}[f]$ $\mathrm{l}[L\ f]$
    using *assms lunit-char*(*2*) *lunit-char*(*2*) [*of L f*] *L.preserves-ide* **by** *auto*
  ultimately show *?thesis*
    using *assms lunit-char*(*2*) [*of f*] *lunit-naturality* [*of* $\mathrm{l}[f]$] *iso-lunit*
      iso-is-section section-is-mono mono-cancel [*of* $\mathrm{l}[f]$ *L* $\mathrm{l}[f]$ $\mathrm{l}[L\ f]$]
    by *auto*
qed

interpretation *R*: *endofunctor V R*
  using *endofunctor-R* **by** *auto*
interpretation $\mathfrak{r}$: *transformation-by-components V V R map runit*
  using *runit-in-hom runit-naturality* **by** *unfold-locales auto*
interpretation $\mathfrak{r}$: *natural-isomorphism V V R map* $\mathfrak{r}$*.map*
  using *iso-runit* **by** *unfold-locales auto*

lemma *natural-isomorphism-*$\mathfrak{r}$:
shows *natural-isomorphism V V R map* $\mathfrak{r}$*.map*
  ..

interpretation *R*: *equivalence-functor V V R*
  using *R.isomorphic-to-identity-is-equivalence* $\mathfrak{r}$*.natural-isomorphism-axioms* **by** *simp*

lemma *equivalence-functor-R*:
shows *equivalence-functor V V R*
  ..

lemma *runit-commutes-with-R*:
assumes *ide f*
shows $\mathrm{r}[R\ f] = R\ \mathrm{r}[f]$
proof −
  have *seq* $\mathrm{r}[f]$ $(R\ \mathrm{r}[f])$
    using *assms runit-char*(*2*) *R.preserves-hom* **by** *fastforce*
  moreover have *seq* $\mathrm{r}[f]$ $\mathrm{r}[R\ f]$
    using *assms runit-char*(*2*) *runit-char*(*2*) [*of R f*] *R.preserves-ide* **by** *auto*
  ultimately show *?thesis*

using *assms runit-char*(*2*) [*of f*] *runit-naturality* [*of* r[*f*]] *iso-runit*
iso-is-section section-is-mono mono-cancel [*of* r[*f*] *R* r[*f*] r[*R f*]]
by *auto*
qed

definition α
where $\alpha\ \mu\ \nu\ \tau \equiv$ *if* $VVV.arr\ (\mu,\ \nu,\ \tau)$ *then*
$(\mu \star \nu \star \tau) \cdot$ a[*dom* μ, *dom* ν, *dom* τ]
else null

lemma α-*ide-simp* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows $\alpha\ f\ g\ h =$ a[f, g, h]
proof −
have $\alpha\ f\ g\ h = (f \star g \star h) \cdot$ a[*dom f*, *dom g*, *dom h*]
using *assms* α-*def* $VVV.arr\text{-}char_{SbC}$ [*of* (f, g, h)] **by** *auto*
also have ... $= (f \star g \star h) \cdot$ a[f, g, h]
using *assms* **by** *simp*
also have ... = a[f, g, h]
using *assms* α-*def assoc-in-hom*$_{AWC}$ *hcomp-in-hom*$_{PBH}$ $VVV.arr\text{-}char_{SbC}$ $VoV.arr\text{-}char_{SbC}$
comp-cod-arr composable-char$_{PBH}$
by *auto*
finally show *?thesis* **by** *simp*
qed

no-notation *in-hom* (‹«- : - → -»›)

lemma *natural-isomorphism*-α:
shows *natural-isomorphism* $VVV.comp$ *V HoHV HoVH*
$(\lambda\mu\nu\tau.\ \alpha\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau)))$
proof −
interpret α: *transformation-by-components* $VVV.comp$ *V HoHV HoVH*
‹$\lambda f.$ a[*fst f*, *fst* (*snd f*), *snd* (*snd f*)]›
proof
show *1*: $\bigwedge x.\ VVV.ide\ x \Longrightarrow$ «a[*fst x*, *fst* (*snd x*), *snd* (*snd x*)] : *HoHV x* ⇒ *HoVH x*»
proof −
fix x
assume x: $VVV.ide\ x$
show «a[*fst x*, *fst* (*snd x*), *snd* (*snd x*)] : *HoHV x* ⇒ *HoVH x*»
proof −
have *ide* (*fst x*) ∧ *ide* (*fst* (*snd x*)) ∧ *ide* (*snd* (*snd x*)) ∧
fst x ⋆ *fst* (*snd x*) ≠ *null* ∧ *fst* (*snd x*) ⋆ *snd* (*snd x*) ≠ *null*
using x $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *composable-char*$_{PBH}$
by *simp*
hence a[*fst x*, *fst* (*snd x*), *snd* (*snd x*)]
∈ *hom* ((*fst x* ⋆ *fst* (*snd x*)) ⋆ *snd* (*snd x*))
(*fst x* ⋆ *fst* (*snd x*) ⋆ *snd* (*snd x*))
using x *assoc-in-hom*$_{AWC}$ **by** *simp*

thus *?thesis*
unfolding *HoHV-def HoVH-def*
using x $VVV.ideD(1)$ **by** *simp*
qed
qed
show $\bigwedge f.\ VVV.arr\ f \Longrightarrow$
$\mathrm{a}[fst\ (VVV.cod\ f),\ fst\ (snd\ (VVV.cod\ f)),\ snd\ (snd\ (VVV.cod\ f))] \cdot HoHV\ f =$
$HoVH\ f \cdot \mathrm{a}[fst\ (VVV.dom\ f),\ fst\ (snd\ (VVV.dom\ f)),\ snd\ (snd\ (VVV.dom\ f))]$
unfolding *HoHV-def HoVH-def*
using $assoc\text{-}naturality_{AWC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VVV.dom\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$
$composable\text{-}char_{PBH}$
by *simp*
qed
interpret α: *natural-isomorphism* $VVV.comp$ V $HoHV$ $HoVH$ $\alpha.map$
proof
fix f
assume f: $VVV.ide\ f$
show $iso\ (\alpha.map\ f)$
proof −
have $fst\ f \star fst\ (snd\ f) \neq null \wedge fst\ (snd\ f) \star snd\ (snd\ f) \neq null$
using f $VVV.ideD(1)$ $VVV.arr\text{-}char_{SbC}$ $[of\ f]$ $VV.arr\text{-}char_{SbC}$ $composable\text{-}char_{PBH}$ **by** *auto*
thus *?thesis*
using f $\alpha.map\text{-}simp\text{-}ide$ $iso\text{-}assoc_{AWC}$ $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ **by** *simp*
qed
qed
have $(\lambda\mu\nu\tau.\ \alpha\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau))) = \alpha.map$
proof
fix $\mu\nu\tau$
have $\neg\ VVV.arr\ \mu\nu\tau \Longrightarrow \alpha\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau)) = \alpha.map\ \mu\nu\tau$
using $\alpha\text{-}def$ $\alpha.map\text{-}def$ **by** *simp*
moreover have $VVV.arr\ \mu\nu\tau \Longrightarrow$
$\alpha\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau)) = \alpha.map\ \mu\nu\tau$
proof −
assume $\mu\nu\tau$: $VVV.arr\ \mu\nu\tau$
have $\alpha\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau)) =$
$(fst\ \mu\nu\tau \star fst\ (snd\ \mu\nu\tau) \star snd\ (snd\ \mu\nu\tau)) \cdot$
$\mathrm{a}[dom\ (fst\ \mu\nu\tau),\ dom\ (fst\ (snd\ \mu\nu\tau)),\ dom\ (snd\ (snd\ \mu\nu\tau))]$
using $\mu\nu\tau$ $\alpha\text{-}def$ **by** *simp*
also have ... $= \mathrm{a}[cod\ (fst\ \mu\nu\tau),\ cod\ (fst\ (snd\ \mu\nu\tau)),\ cod\ (snd\ (snd\ \mu\nu\tau))] \cdot$
$((fst\ \mu\nu\tau \star fst\ (snd\ \mu\nu\tau)) \star snd\ (snd\ \mu\nu\tau))$
using $\mu\nu\tau$ $HoHV\text{-}def$ $HoVH\text{-}def$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $assoc\text{-}naturality_{AWC}$
$composable\text{-}char_{PBH}$
by *simp*
also have ... =
$\mathrm{a}[fst\ (VVV.cod\ \mu\nu\tau),\ fst\ (snd\ (VVV.cod\ \mu\nu\tau)),\ snd\ (snd\ (VVV.cod\ \mu\nu\tau))] \cdot$
$((fst\ \mu\nu\tau \star fst\ (snd\ \mu\nu\tau)) \star snd\ (snd\ \mu\nu\tau))$
using $\mu\nu\tau$ $VVV.arr\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *simp*

```
      also have ... = α.map μντ
        using μντ α.map-def HoHV-def composable-char_PBH by auto
      finally show ?thesis by blast
    qed
    ultimately show α (fst μντ) (fst (snd μντ)) (snd (snd μντ)) = α.map μντ by blast
  qed
  thus ?thesis using α.natural-isomorphism-axioms by simp
qed

proposition induces-bicategory:
shows bicategory V H α i src trg
proof −
  interpret VxVxV: product-category V VxV.comp ..
  interpret VoVoV: subcategory VxVxV.comp
                ‹λτμν. arr (fst τμν) ∧ VV.arr (snd τμν) ∧
                      src (fst τμν) = trg (fst (snd τμν))›
    using subcategory-VVV by blast
  interpret HoHV: functor VVV.comp V HoHV
    using functor-HoHV by blast
  interpret HoVH: functor VVV.comp V HoVH
    using functor-HoVH by blast
  interpret α: natural-isomorphism VVV.comp V HoHV HoVH
              ‹λμντ. α (fst μντ) (fst (snd μντ)) (snd (snd μντ))›
    using natural-isomorphism-α by blast
  interpret L: equivalence-functor V V L
    using equivalence-functor-L by blast
  interpret R: equivalence-functor V V R
    using equivalence-functor-R by blast
  show bicategory V H α i src trg
  proof
    show ⋀a. obj a ⟹ «i[a] : a ⋆ a ⇒ a»
      using obj-is-weak-unit unit-in-vhom_PBU by blast
    show ⋀a. obj a ⟹ iso i[a]
      using obj-is-weak-unit iso-unit_PBU by blast
    show ⋀f g h k. ⟦ ide f; ide g; ide h; ide k;
                     src f = trg g; src g = trg h; src h = trg k ⟧ ⟹
                     (f ⋆ α g h k) · α f (g ⋆ h) k · (α f g h ⋆ k) =
                     α f g (h ⋆ k) · α (f ⋆ g) h k
    proof −
      fix f g h k
      assume f: ide f and g: ide g and h: ide h and k: ide k
      and fg: src f = trg g and gh: src g = trg h and hk: src h = trg k
      have sources f ∩ targets g ≠ {}
        using f g fg src-in-sources [of f] trg-in-targets ideD(1) by auto
      moreover have sources g ∩ targets h ≠ {}
        using g h gh src-in-sources [of g] trg-in-targets ideD(1) by auto
      moreover have sources h ∩ targets k ≠ {}
        using h k hk src-in-sources [of h] trg-in-targets ideD(1) by auto
      moreover have α f g h = a[f, g, h] ∧ α g h k = a[g, h, k]
```

using *f g h k fg gh hk α-ide-simp* **by** *simp*
moreover have $\alpha\ f\ (g \star h)\ k = \mathrm{a}[f,\ g \star h,\ k] \wedge \alpha\ f\ g\ (h \star k) = \mathrm{a}[f,\ g,\ h \star k] \wedge$
$\alpha\ (f \star g)\ h\ k = \mathrm{a}[f \star g,\ h,\ k]$
using *f g h k fg gh hk α-ide-simp preserves-ide hcomp-in-hom*$_{PBH}$*(1)* **by** *simp*
ultimately show $(f \star \alpha\ g\ h\ k) \cdot \alpha\ f\ (g \star h)\ k \cdot (\alpha\ f\ g\ h \star k) =$
$\alpha\ f\ g\ (h \star k) \cdot \alpha\ (f \star g)\ h\ k$
using *f g h k fg gh hk pentagon*$_{AWC}$ [*of f g h k*] *α-ide-simp* **by** *presburger*
qed
qed
qed

end

The following is the main result of this development: Every prebicategory extends to a bicategory, by making an arbitrary choice of representatives of each isomorphism class of weak units and using that to define the source and target mappings, and then choosing an arbitrary isomorphism in $hom\ (a \star a)\ a$ for each weak unit a.

context *prebicategory*
begin

interpretation *prebicategory-with-homs V H* a *some-src some-trg*
using *extends-to-prebicategory-with-homs* **by** *auto*

interpretation *prebicategory-with-units V H* a *some-unit*
using *extends-to-prebicategory-with-units* **by** *auto*

interpretation *prebicategory-with-homs-and-units V H* a *some-unit some-src some-trg* **..**

theorem *extends-to-bicategory*:
shows *bicategory V H α some-unit some-src some-trg*
using *induces-bicategory* **by** *simp*

end

1.4 Bicategories as Prebicategories

1.4.1 Bicategories are Prebicategories

In this section we show that a bicategory determines a prebicategory with homs, whose weak units are exactly those arrows that are isomorphic to their chosen source, or equivalently, to their chosen target. Moreover, the notion of horizontal composability, which in a bicategory is determined by the coincidence of chosen sources and targets, agrees with the version defined for the induced weak composition in terms of nonempty intersections of source and target sets, which is not dependent on any arbitrary choices.

context *bicategory*
begin

no-notation *in-hom* (‹«- : - → -»›)

interpretation α': *inverse-transformation VVV.comp V HoHV HoVH*
‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))› **..**

abbreviation α'
where $\alpha' \equiv \alpha'.map$

definition a′ (‹a^{-1}[-, -, -]›)
where $\mathrm{a}^{-1}[\mu, \nu, \tau] \equiv \alpha'.map\ (\mu, \nu, \tau)$

lemma *assoc′-in-hom′*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows *in-hhom* $\mathrm{a}^{-1}[\mu, \nu, \tau]$ (*src* τ) (*trg* μ)
and «$\mathrm{a}^{-1}[\mu, \nu, \tau]$: *dom* $\mu \star$ *dom* $\nu \star$ *dom* $\tau \Rightarrow$ (*cod* $\mu \star$ *cod* ν) $\star$ *cod* τ»
proof −
  show «$\mathrm{a}^{-1}[\mu, \nu, \tau]$: *dom* $\mu \star$ *dom* $\nu \star$ *dom* $\tau \Rightarrow$ (*cod* $\mu \star$ *cod* ν) $\star$ *cod* τ»
  proof −
    have *1*: *VVV.in-hom* (μ, ν, τ) (*dom* μ, *dom* ν, *dom* τ) (*cod* μ, *cod* ν, *cod* τ)
      using *assms VVV.in-hom-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *auto*
    have «$\mathrm{a}^{-1}[\mu, \nu, \tau]$: *HoVH* (*dom* μ, *dom* ν, *dom* τ) $\Rightarrow$ *HoHV* (*cod* μ, *cod* ν, *cod* τ)»
      using *1* a′*-def* α'*.preserves-hom* **by** *auto*
    moreover have *HoVH* (*dom* μ, *dom* ν, *dom* τ) = *dom* $\mu \star$ *dom* $\nu \star$ *dom* τ
      using *1 HoVH-def* **by** (*simp add*: *VVV.in-hom-char*$_{SbC}$)
    moreover have *HoHV* (*cod* μ, *cod* ν, *cod* τ) = (*cod* $\mu \star$ *cod* ν) $\star$ *cod* τ
      using *1 HoHV-def* **by** (*simp add*: *VVV.in-hom-char*$_{SbC}$)
    ultimately show *?thesis* **by** *simp*
  qed
  thus *in-hhom* $\mathrm{a}^{-1}[\mu, \nu, \tau]$ (*src* τ) (*trg* μ)
    using *assms vconn-implies-hpar*(*1*) *vconn-implies-hpar*(*2*) **by** *auto*
qed

lemma *assoc′-naturality1*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}^{-1}[\mu, \nu, \tau] = ((\mu \star \nu) \star \tau) \cdot \mathrm{a}^{-1}[dom\ \mu, dom\ \nu, dom\ \tau]$
  using *assms* α'*.naturality1* [*of* (μ, ν, τ)] *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
      VVV.dom-char$_{SbC}$ *HoHV-def src-dom trg-dom* a′*-def*
  by *simp*

lemma *assoc′-naturality2*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}^{-1}[\mu, \nu, \tau] = \mathrm{a}^{-1}[cod\ \mu, cod\ \nu, cod\ \tau] \cdot (\mu \star \nu \star \tau)$
  using *assms* α'*.naturality2* [*of* (μ, ν, τ)] *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
      VVV.cod-char$_{SbC}$ *HoVH-def src-dom trg-dom* a′*-def*
  by *simp*

lemma *assoc′-naturality*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ

shows $\mathrm{a}^{-1}[cod\ \mu,\ cod\ \nu,\ cod\ \tau] \cdot (\mu \star \nu \star \tau) = ((\mu \star \nu) \star \tau) \cdot \mathrm{a}^{-1}[dom\ \mu,\ dom\ \nu,\ dom\ \tau]$
  using *assms assoc′-naturality1 assoc′-naturality2* **by** *metis*

lemma *assoc′-in-hom* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *in-hhom* $\mathrm{a}^{-1}[f,\ g,\ h]$ (*src h*) (*trg f*)
and «$\mathrm{a}^{-1}[f,\ g,\ h]$: *dom f* ⋆ *dom g* ⋆ *dom h* ⇒ (*cod f* ⋆ *cod g*) ⋆ *cod h*»
  using *assms assoc′-in-hom′*(*1*−*2*) *ideD*(*1*) **by** *meson*+

lemma *assoc′-simps* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *arr* $\mathrm{a}^{-1}[f,\ g,\ h]$
and *src* $\mathrm{a}^{-1}[f,\ g,\ h]$ = *src h* **and** *trg* $\mathrm{a}^{-1}[f,\ g,\ h]$ = *trg f*
and *dom* $\mathrm{a}^{-1}[f,\ g,\ h]$ = *dom f* ⋆ *dom g* ⋆ *dom h*
and *cod* $\mathrm{a}^{-1}[f,\ g,\ h]$ = (*cod f* ⋆ *cod g*) ⋆ *cod h*
  using *assms assoc′-in-hom* **by** *blast*+

lemma *assoc′-eq-inv-assoc* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows $\mathrm{a}^{-1}[f,\ g,\ h]$ = *inv* $\mathrm{a}[f,\ g,\ h]$
using *assms* $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $\alpha'.map\text{-}ide\text{-}simp$ a'*-def*
  by *auto*

lemma *inverse-assoc-assoc′* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *inverse-arrows* $\mathrm{a}[f,\ g,\ h]$ $\mathrm{a}^{-1}[f,\ g,\ h]$
using *assms* $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $\alpha'.map\text{-}ide\text{-}simp$ $\alpha'.inverts\text{-}components$ a'*-def*
  by *auto*

lemma *iso-assoc′* [*intro*, *simp*]:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows *iso* $\mathrm{a}^{-1}[f,\ g,\ h]$
  using *assms* **by** *simp*

lemma *comp-assoc-assoc′* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows $\mathrm{a}[f,\ g,\ h] \cdot \mathrm{a}^{-1}[f,\ g,\ h] = f \star g \star h$
and $\mathrm{a}^{-1}[f,\ g,\ h] \cdot \mathrm{a}[f,\ g,\ h] = (f \star g) \star h$
  using *assms comp-arr-inv′ comp-inv-arr′* **by** *auto*

lemma *unit-in-hom* [*intro*, *simp*]:
assumes *obj a*
shows «$\mathrm{i}[a] : a \to a$» **and** «$\mathrm{i}[a] : a \star a \Rightarrow a$»
proof −
  show «$\mathrm{i}[a] : a \star a \Rightarrow a$»

```
    using assms unit-in-vhom by simp
  thus «i[a] : a → a»
    using assms
    by (metis arrI in-hhom-def obj-simps(2−3) vconn-implies-hpar(1−4))
qed

interpretation weak-composition V H
  using is-weak-composition by auto

lemma seq-if-composable:
assumes ν ⋆ μ ≠ null
shows src ν = trg μ
  using assms H.extensionality [of (ν, μ)] VV.arr-char_SbC by auto

lemma obj-self-composable:
assumes obj a
shows a ⋆ a ≠ null
and isomorphic (a ⋆ a) a
   apply (metis arr-dom-iff-arr assms in-homE not-arr-null unit-in-vhom)
  by (meson assms iso-unit isomorphic-def unit-in-vhom)

lemma obj-is-weak-unit:
assumes obj a
shows weak-unit a
proof −
  interpret Left-a: subcategory V ‹left a›
    using assms left-hom-is-subcategory by force
  interpret Right-a: subcategory V ‹right a›
    using assms right-hom-is-subcategory by force
```

We know that $H_L\ a$ is fully faithful as a global endofunctor, but the definition of weak unit involves its restriction to a subcategory. So we have to verify that the restriction is also a fully faithful functor.

```
  interpret La: endofunctor ‹Left a› ‹H_L a›
    using assms obj-self-composable endofunctor-H_L [of a] by force
  interpret La: fully-faithful-functor ‹Left a› ‹Left a› ‹H_L a›
  proof
    show ⋀f f'. Left-a.par f f' ⟹ H_L a f = H_L a f' ⟹ f = f'
    proof −
      fix μ μ'
      assume par: Left-a.par μ μ'
      assume eq: H_L a μ = H_L a μ'
      have 1: par μ μ'
        using par Left-a.arr-char_SbC Left-a.dom-char_SbC Left-a.cod-char_SbC left-def
              composable-implies-arr null-agreement
        by metis
      moreover have L μ = L μ'
        using par eq H_L-def Left-a.arr-char_SbC left-def preserves-arr
              assms 1 seq-if-composable [of a μ] not-arr-null seq-if-composable [of a μ']
```

by *auto*
ultimately show $\mu = \mu'$
using *L.is-faithful* **by** *blast*
qed
show $\bigwedge f\ g\ \mu.\ [\![$ *Left-a.ide* f; *Left-a.ide* g; *Left-a.in-hom* μ $(H_L\ a\ f)$ $(H_L\ a\ g)$ $]\!] \Longrightarrow$
$\exists \nu.$ *Left-a.in-hom* $\nu\ f\ g \wedge H_L\ a\ \nu = \mu$
proof −
fix $f\ g\ \mu$
assume f: *Left-a.ide* f **and** g: *Left-a.ide* g
and μ: *Left-a.in-hom* μ $(H_L\ a\ f)$ $(H_L\ a\ g)$
have *1*: $a = trg\ f \wedge a = trg\ g$
using *assms f g Left-a.ide-char Left-a.arr-char*$_{SbC}$ *left-def seq-if-composable* [*of a f*] *seq-if-composable* [*of a g*]
by *auto*
show $\exists \nu.$ *Left-a.in-hom* $\nu\ f\ g \wedge H_L\ a\ \nu = \mu$
proof −
have *2*: $\exists \nu.$ «$\nu : f \Rightarrow g$» $\wedge\ L\ \nu = \mu$
using $f\ g\ \mu$ *1 Left-a.ide-char*$_{SbC}$ H_L*-def L.preserves-reflects-arr Left-a.arr-char*$_{SbC}$ *Left-a.in-hom-char*$_{SbC}$ *L.is-full*
by *force*
obtain ν **where** ν: «$\nu : f \Rightarrow g$» $\wedge\ L\ \nu = \mu$
using *2* **by** *blast*
have *Left-a.arr* ν
using ν *1 trg-dom Left-a.arr-char*$_{SbC}$ *left-def hseq-char'* **by** *fastforce*
moreover have $H_L\ a\ \nu = \mu$
using ν *1 trg-dom* H_L*-def* **by** *auto*
ultimately show *?thesis*
using ν *Left-a.dom-simp Left-a.cod-simp* **by** *blast*
qed
qed
qed
interpret *Ra*: *endofunctor* ‹*Right a*› ‹H_R *a*›
using *assms obj-self-composable endofunctor-*H_R [*of a*] **by** *force*
interpret *Ra*: *fully-faithful-functor* ‹*Right a*› ‹*Right a*› ‹H_R *a*›
proof
show $\bigwedge f\ f'.$ *Right-a.par* $f\ f' \Longrightarrow H_R\ a\ f = H_R\ a\ f' \Longrightarrow f = f'$
proof −
fix $\mu\ \mu'$
assume *par*: *Right-a.par* $\mu\ \mu'$
assume *eq*: $H_R\ a\ \mu = H_R\ a\ \mu'$
have *1*: *par* $\mu\ \mu'$
using *par Right-a.arr-char*$_{SbC}$ *Right-a.dom-char*$_{SbC}$ *Right-a.cod-char*$_{SbC}$ *right-def composable-implies-arr null-agreement*
by *metis*
moreover have $R\ \mu = R\ \mu'$
using *par eq* H_R*-def Right-a.arr-char*$_{SbC}$ *right-def preserves-arr assms 1 seq-if-composable* [*of* μ *a*] *not-arr-null seq-if-composable* [*of* μ' *a*]
by *auto*
ultimately show $\mu = \mu'$

```
      using R.is-faithful by blast
    qed
    show ⋀f g μ. ⟦ Right-a.ide f; Right-a.ide g; Right-a.in-hom μ (H_R a f) (H_R a g) ⟧ ⟹
                 ∃ν. Right-a.in-hom ν f g ∧ H_R a ν = μ
    proof −
      fix f g μ
      assume f: Right-a.ide f and g: Right-a.ide g
      and μ: Right-a.in-hom μ (H_R a f) (H_R a g)
      have 1: a = src f ∧ a = src g
        using assms f g Right-a.ide-char Right-a.arr-char_SbC right-def seq-if-composable
        by auto
      show ∃ν. Right-a.in-hom ν f g ∧ H_R a ν = μ
      proof −
        have 2: ∃ν. «ν : f ⇒ g» ∧ R ν = μ
        using f g μ 1 Right-a.ide-char_SbC H_R-def R.preserves-reflects-arr Right-a.arr-char_SbC
              Right-a.in-hom-char_SbC R.is-full
          by force
        obtain ν where ν: «ν : f ⇒ g» ∧ R ν = μ
          using 2 by blast
        have Right-a.arr ν
          using ν 1 src-dom Right-a.arr-char_SbC right-def hseq-char' by fastforce
        moreover have H_R a ν = μ
          using ν 1 src-dom H_R-def by auto
        ultimately show ?thesis
          using ν Right-a.dom-simp Right-a.cod-simp by blast
      qed
    qed
  qed
  have isomorphic (a ⋆ a) a ∧ a ⋆ a ≠ null
    using assms obj-self-composable unit-in-hom iso-unit isomorphic-def by blast
  thus ?thesis
    using La.fully-faithful-functor-axioms Ra.fully-faithful-functor-axioms weak-unit-def
    by blast
qed

lemma src-in-sources:
assumes arr μ
shows src μ ∈ sources μ
  using assms obj-is-weak-unit R.preserves-arr hseq-char' by auto

lemma trg-in-targets:
assumes arr μ
shows trg μ ∈ targets μ
  using assms obj-is-weak-unit L.preserves-arr hseq-char' by auto

lemma weak-unit-cancel-left:
assumes weak-unit a and ide f and ide g
and a ⋆ f ≅ a ⋆ g
shows f ≅ g
```

proof −
  have *0*: *ide a*
    using *assms weak-unit-def* **by** *force*
  interpret *Left-a*: *subcategory V* ‹*left a*›
    using *0 left-hom-is-subcategory* **by** *simp*
  interpret *Left-a*: *left-hom V H a*
    using *assms weak-unit-self-composable* **by** *unfold-locales auto*
  interpret *La*: *fully-faithful-functor* ‹*Left a*› ‹*Left a*› ‹H_L *a*›
    using *assms weak-unit-def* **by** *fast*
  obtain φ **where** φ: *iso* $\varphi \wedge$ «$\varphi : a \star f \Rightarrow a \star g$»
    using *assms* **by** *blast*
  have *1*: *Left-a.iso* $\varphi \wedge$ *Left-a.in-hom* φ $(a \star f)$ $(a \star g)$
    by (*metis* H_L*-def La.extensionality La.preserves-arr Left-a.dom-closed*
      Left-a.in-hom-char$_{SbC}$ *Left-a.iso-char Left-a.not-arr-null Left-a.null-char* φ *assms*(*4*)
      hom-connected(*4*) *hseq-char′ ide-char′ in-homE isomorphic-implies-ide*(*2*) *left-def*)
  hence *2*: *Left-a.ide* $(a \star f) \wedge$ *Left-a.ide* $(a \star g)$
    using *Left-a.ide-dom* [*of* φ] *Left-a.ide-cod* [*of* φ] *Left-a.dom-simp Left-a.cod-simp*
    by *auto*
  hence *3*: *Left-a.ide f* $\wedge$ *Left-a.ide g*
    by (*metis Left-a.ideI*$_{SbC}$ *Left-a.ide-def Left-a.null-char assms*(*2*) *assms*(*3*) *left-def*)
  obtain ψ **where** ψ: $\psi \in$ *Left-a.hom f g* $\wedge$ $a \star \psi = \varphi$
    using *assms 1 2 3 La.is-full* [*of g f* φ] H_L*-def* **by** *auto*
  have *Left-a.iso* ψ
    using ψ *1* H_L*-def La.reflects-iso* **by** *auto*
  hence *iso* $\psi \wedge$ «$\psi : f \Rightarrow g$»
    using ψ *Left-a.iso-char Left-a.in-hom-char*$_{SbC}$ **by** *auto*
  thus *?thesis* **by** *auto*
qed

lemma *weak-unit-cancel-right*:
assumes *weak-unit a* **and** *ide f* **and** *ide g*
and $f \star a \cong g \star a$
shows $f \cong g$
proof −
  have *0*: *ide a*
    using *assms weak-unit-def* **by** *force*
  interpret *Right-a*: *subcategory V* ‹*right a*›
    using *0 right-hom-is-subcategory* **by** *simp*
  interpret *Right-a*: *right-hom V H a*
    using *assms weak-unit-self-composable* **by** *unfold-locales auto*
  interpret *R*: *fully-faithful-functor* ‹*Right a*› ‹*Right a*› ‹H_R *a*›
    using *assms weak-unit-def* **by** *fast*
  obtain φ **where** φ: *iso* $\varphi \wedge$ *in-hom* φ $(f \star a)$ $(g \star a)$
    using *assms* **by** *blast*
  have *1*: *Right-a.iso* $\varphi \wedge \varphi \in$ *Right-a.hom* $(f \star a)$ $(g \star a)$
  proof
    have $\varphi \star a \neq$ *null*
      by (*metis* φ *assms*(*1*,*4*) *hom-connected*(*3*) *ideD*(*1*) *in-homE isomorphic-implies-ide*(*2*)
        match-3 not-arr-null weak-unit-self-composable(*3*))

```
    thus φ ∈ Right-a.hom (f ⋆ a) (g ⋆ a)
      using φ Right-a.hom-char right-def by simp
    thus Right-a.iso φ
      using φ Right-a.iso-char by auto
  qed
  hence 2: Right-a.ide (f ⋆ a) ∧ Right-a.ide (g ⋆ a)
    using Right-a.ide-dom [of φ] Right-a.ide-cod [of φ] Right-a.dom-simp Right-a.cod-simp
    by auto
  hence 3: Right-a.ide f ∧ Right-a.ide g
  using assms Right-a.ide-char_SbC Right-a.arr-char_SbC right-def Right-a.ide-def Right-a.null-char
    by metis
  obtain ψ where ψ: ψ ∈ Right-a.hom f g ∧ ψ ⋆ a = φ
    using assms 1 2 3 R.is-full [of g f φ] H_R-def by auto
  have Right-a.iso ψ
    using ψ 1 H_R-def R.reflects-iso by auto
  hence iso ψ ∧ «ψ : f ⇒ g»
    using ψ Right-a.iso-char Right-a.in-hom-char_SbC by auto
  thus ?thesis by auto
qed
```

All sources of an arrow (*i.e.* weak units composable on the right with that arrow) are isomorphic to the chosen source, and similarly for targets. That these statements hold was somewhat surprising to me.

```
lemma source-iso-src:
assumes arr μ and a ∈ sources μ
shows a ≅ src μ
proof −
  have 0: ide a
    using assms weak-unit-def by force
  have 1: src a = trg a
    using assms ide-dom sources-def weak-unit-iff-self-target seq-if-composable
          weak-unit-self-composable
    by simp
  obtain φ where φ: iso φ ∧ «φ : a ⋆ a ⇒ a»
    using assms weak-unit-def by blast
  have a ⋆ src a ≅ src a ⋆ src a
  proof −
    have src a ≅ src a ⋆ src a
      using 0 obj-is-weak-unit weak-unit-def isomorphic-symmetric by auto
    moreover have a ⋆ src a ≅ src a
    proof −
      have a ⋆ a ⋆ src a ≅ a ⋆ src a
      proof −
        have iso (φ ⋆ src a) ∧ «φ ⋆ src a : (a ⋆ a) ⋆ src a ⇒ a ⋆ src a»
          using 0 1 φ ide-in-hom(2) by auto
        moreover have iso a⁻¹[a, a, src a] ∧
                      «a⁻¹[a, a, src a] :  a ⋆ a ⋆ src a ⇒ (a ⋆ a) ⋆ src a»
          using 0 1 iso-assoc′ by force
        ultimately show ?thesis
```

```
        using isos-compose isomorphic-def by auto
      qed
      thus ?thesis
        using assms 0 weak-unit-cancel-left by auto
    qed
    ultimately show ?thesis
      using isomorphic-transitive by meson
  qed
  hence a ≅ src a
    using 0 weak-unit-cancel-right [of src a a src a] obj-is-weak-unit by auto
  thus ?thesis using assms seq-if-composable 1 by auto
qed

lemma target-iso-trg:
assumes arr μ and b ∈ targets μ
shows b ≅ trg μ
proof −
  have 0: ide b
    using assms weak-unit-def by force
  have 1: trg μ = src b
    using assms seq-if-composable by auto
  obtain φ where φ: iso φ ∧ «φ : b ⋆ b ⇒ b»
    using assms weak-unit-def by blast
  have trg b ⋆ b ≅ trg b ⋆ trg b
  proof −
    have trg b ≅ trg b ⋆ trg b
      using 0 obj-is-weak-unit weak-unit-def isomorphic-symmetric by auto
    moreover have trg b ⋆ b ≅ trg b
    proof −
      have (trg b ⋆ b) ⋆ b ≅ trg b ⋆ b
      proof −
        have iso (trg b ⋆ φ) ∧ «trg b ⋆ φ : trg b ⋆ b ⋆ b ⇒ trg b ⋆ b»
          using assms 0 1 φ ide-in-hom(2) targetsD(1) weak-unit-self-composable
          apply (intro conjI hcomp-in-vhom) by auto
        moreover have iso a[trg b, b, b] ∧
                      «a[trg b, b, b] : (trg b ⋆ b) ⋆ b ⇒ trg b ⋆ b ⋆ b»
          using assms(2) 0 1 seq-if-composable targetsD(1−2) weak-unit-self-composable
          by auto
        ultimately show ?thesis
          using isos-compose isomorphic-def by auto
      qed
      thus ?thesis
        using assms 0 weak-unit-cancel-right by auto
    qed
    ultimately show ?thesis
      using isomorphic-transitive by meson
  qed
  hence b ≅ trg b
    using 0 weak-unit-cancel-left [of trg b b trg b] obj-is-weak-unit by simp
```

thus *?thesis*
using *assms 0 1 seq-if-composable weak-unit-iff-self-source targetsD*(*1*−*2*) *source-iso-src*
by *simp*
qed

lemma *is-weak-composition-with-homs*:
shows *weak-composition-with-homs V H src trg*
using *src-in-sources trg-in-targets seq-if-composable composable-implies-arr*
by (*unfold-locales*, *simp-all*)

interpretation *weak-composition-with-homs V H src trg*
using *is-weak-composition-with-homs* **by** *auto*

In a bicategory, the notion of composability defined in terms of the chosen sources and targets coincides with the version defined for a weak composition, which does not involve particular choices.

lemma *connected-iff-seq*:
assumes *arr* μ **and** *arr* ν
shows *sources* $\nu \cap$ *targets* $\mu \neq \{\}$ $\longleftrightarrow$ *src* ν = *trg* μ
proof
show *src* ν = *trg* $\mu \Longrightarrow$ *sources* $\nu \cap$ *targets* $\mu \neq \{\}$
using *assms src-in-sources* [*of* ν] *trg-in-targets* [*of* μ] **by** *auto*
show *sources* $\nu \cap$ *targets* $\mu \neq \{\} \Longrightarrow$ *src* ν = *trg* μ
proof −
assume *1*: *sources* $\nu \cap$ *targets* $\mu \neq \{\}$
obtain *a* **where** *a*: $a \in$ *sources* $\nu \cap$ *targets* μ
using *assms 1* **by** *blast*
have μ: *arr* μ
using *a composable-implies-arr* **by** *auto*
have ν: *arr* ν
using *a composable-implies-arr* **by** *auto*
have *2*: $\bigwedge a'.\ a' \in$ *sources* $\nu \Longrightarrow$ *src* a' = *src* $a \wedge$ *trg* a' = *trg* a
by (*metis IntE a seq-if-composable sourcesD*(*2*−*3*) *weak-unit-self-composable*(*3*))
have *src* ν = *src* (*src* ν) **using** ν **by** *simp*
also have ... = *src* (*trg* μ)
using ν *2* [*of src* ν] *src-in-sources a weak-unit-self-composable seq-if-composable*
by *auto*
also have ... = *trg* (*trg* μ) **using** μ **by** *simp*
also have ... = *trg* μ **using** μ **by** *simp*
finally show *src* ν = *trg* μ **by** *blast*
qed
qed

lemma *is-associative-weak-composition*:
shows *associative-weak-composition V H* a
proof −
have *1*: $\bigwedge \nu\ \mu.\ \nu \star \mu \neq$ *null* $\Longrightarrow$ *src* ν = *trg* μ
using *H.extensionality VV.arr-char*$_{SbC}$ **by** *force*
show *associative-weak-composition V H* a

proof
show $\bigwedge f\ g\ h.\ ide\ f \Longrightarrow ide\ g \Longrightarrow ide\ h \Longrightarrow f \star g \neq null \Longrightarrow g \star h \neq null \Longrightarrow$
«a[f, g, h] : $(f \star g) \star h \Rightarrow f \star g \star h$»
using *1* **by** *auto*
show $\bigwedge f\ g\ h.\ ide\ f \Longrightarrow ide\ g \Longrightarrow ide\ h \Longrightarrow f \star g \neq null \Longrightarrow g \star h \neq null \Longrightarrow$
iso a[f, g, h]
using *1 iso-assoc* **by** *presburger*
show $\bigwedge \tau\ \mu\ \nu.\ \tau \star \mu \neq null \Longrightarrow \mu \star \nu \neq null \Longrightarrow$
a[*cod* τ, *cod* μ, *cod* ν] $\cdot$ $((\tau \star \mu) \star \nu) = (\tau \star \mu \star \nu) \cdot$ a[*dom* τ, *dom* μ, *dom* ν]
using *1 assoc-naturality hseq-char hseq-char′* **by** *metis*
show $\bigwedge f\ g\ h\ k.\ ide\ f \Longrightarrow ide\ g \Longrightarrow ide\ h \Longrightarrow ide\ k \Longrightarrow$
sources $f \cap$ *targets* $g \neq \{\} \Longrightarrow$
sources $g \cap$ *targets* $h \neq \{\} \Longrightarrow$
sources $h \cap$ *targets* $k \neq \{\} \Longrightarrow$
$(f \star$ a[g, h, k]$) \cdot$ a[f, $g \star h$, k] $\cdot$ (a[f, g, h] $\star k$) $=$
a[f, g, $h \star k$] $\cdot$ a[$f \star g$, h, k]
using *1 connected-iff-seq pentagon ideD*(*1*) **by** *auto*
qed
qed

interpretation *associative-weak-composition V H* a
using *is-associative-weak-composition* **by** *auto*

theorem *is-prebicategory*:
shows *prebicategory V H* a
using *src-in-sources trg-in-targets* **by** (*unfold-locales*, *auto*)

interpretation *prebicategory V H* a
using *is-prebicategory* **by** *auto*

corollary *is-prebicategory-with-homs*:
shows *prebicategory-with-homs V H* a *src trg*
..

interpretation *prebicategory-with-homs V H* a *src trg*
using *is-prebicategory-with-homs* **by** *auto*

In a bicategory, an arrow is a weak unit if and only if it is isomorphic to its chosen source (or to its chosen target).

lemma *weak-unit-char*:
shows *weak-unit a* $\longleftrightarrow$ $a \cong$ *src a*
and *weak-unit a* $\longleftrightarrow$ $a \cong$ *trg a*
proof −
show *weak-unit a* $\longleftrightarrow$ $a \cong$ *src a*
using *isomorphism-respects-weak-units isomorphic-symmetric*
by (*meson ideD*(*1*) *isomorphic-implies-ide*(*2*) *obj-is-weak-unit obj-src source-iso-src weak-unit-iff-self-source weak-unit-self-composable*(*1*))
show *weak-unit a* $\longleftrightarrow$ $a \cong$ *trg a*
using *isomorphism-respects-weak-units isomorphic-symmetric*

by (*metis* ‹*weak-unit a* = *isomorphic a* (*src a*)› *ideD*(*1*) *isomorphic-implies-hpar*(*3*) *isomorphic-implies-ide*(*1*) *src-trg target-iso-trg weak-unit-iff-self-target*)
qed

interpretation *H*: *partial-composition H*
using *is-partial-composition* **by** *auto*

Every arrow with respect to horizontal composition is also an arrow with respect to vertical composition. The converse is not necessarily true.

lemma *harr-is-varr*:
assumes *H.arr* μ
shows *arr* μ
using *H.arr-def H.codomains-def H.domains-def assms composable-implies-arr*(*1*) *composable-implies-arr*(*2*)
by *auto*

An identity for horizontal composition is also an identity for vertical composition.

lemma *horizontal-identity-is-ide*:
assumes *H.ide* μ
shows *ide* μ
proof −
have μ: *arr* μ
using *assms H.ide-def composable-implies-arr*(*2*) **by** *auto*
hence *1*: $\mu \star$ *dom* $\mu \neq$ *null*
using *assms hom-connected H.ide-def* **by** *auto*
have $\mu \star$ *dom* μ = *dom* μ
using *assms 1 H.ide-def* **by** *simp*
moreover have $\mu \star$ *dom* $\mu = \mu$
using *assms 1 H.ide-def* [*of* μ] *null-agreement*
by (*metis* μ *cod-cod cod-dom hcomp-simps*$_{WC}$(*3*) *ideD*(*2*) *ide-char′ paste-1*)
ultimately have *dom* $\mu = \mu$
by *simp*
thus *?thesis*
using μ **by** (*metis ide-dom*)
qed

Every identity for horizontal composition is a weak unit.

lemma *horizontal-identity-is-weak-unit*:
assumes *H.ide* μ
shows *weak-unit* μ
using *assms weak-unit-char*
by (*metis H.ide-def comp-target-ide horizontal-identity-is-ide ideD*(*1*) *isomorphism-respects-weak-units null-agreement targetsD*(*2*−*3*) *trg-in-targets*)

end

1.4.2 Vertically Discrete Bicategories are Categories

In this section we show that if a bicategory is discrete with respect to vertical composition, then it is a category with respect to horizontal composition. To obtain this result, we

need to establish that the set of arrows for the horizontal composition coincides with the set of arrows for the vertical composition. This is not true for a general bicategory, and even with the assumption that the vertical category is discrete it is not immediately obvious from the definitions. The issue is that the notion "arrow" for the horizontal composition is defined in terms of the existence of "domains" and "codomains" with respect to that composition, whereas the axioms for a bicategory only relate the notion "arrow" for the vertical category to the existence of sources and targets with respect to the horizontal composition. So we have to establish that, under the assumption of vertical discreteness, sources coincide with domains and targets coincide with codomains. We also need the fact that horizontal identities are weak units, which previously required some effort to show.

```
locale vertically-discrete-bicategory =
  bicategory +
assumes vertically-discrete: ide = arr
begin

  interpretation prebicategory-with-homs V H a src trg
    using is-prebicategory-with-homs by auto

  interpretation H: partial-composition H
    using is-partial-composition(1) by auto

  lemma weak-unit-is-horizontal-identity:
  assumes weak-unit a
  shows H.ide a
  proof −
    have a ⋆ a ≠ H.null
      using assms weak-unit-self-composable by simp
    moreover have ⋀f. f ⋆ a ≠ H.null ⟹ f ⋆ a = f
    proof −
      fix f
      assume f ⋆ a ≠ H.null
      hence f ⋆ a ≅ f
        using assms comp-ide-source composable-implies-arr(2) sourcesI vertically-discrete
        by auto
      thus f ⋆ a = f
        using vertically-discrete isomorphic-def by auto
    qed
    moreover have ⋀f. a ⋆ f ≠ H.null ⟹ a ⋆ f = f
    proof −
      fix f
      assume a ⋆ f ≠ H.null
      hence a ⋆ f ≅ f
        using assms comp-target-ide composable-implies-arr(1) targetsI vertically-discrete
        by auto
      thus a ⋆ f = f
        using vertically-discrete isomorphic-def by auto
    qed
```

ultimately show $H.ide\ a$
using $H.ide\text{-}def$ **by** $simp$
qed

lemma $sources\text{-}eq\text{-}domains$:
shows $sources\ \mu = H.domains\ \mu$
using $weak\text{-}unit\text{-}is\text{-}horizontal\text{-}identity\ H.domains\text{-}def\ sources\text{-}def$
$horizontal\text{-}identity\text{-}is\text{-}weak\text{-}unit$
by $auto$

lemma $targets\text{-}eq\text{-}codomains$:
shows $targets\ \mu = H.codomains\ \mu$
using $weak\text{-}unit\text{-}is\text{-}horizontal\text{-}identity\ H.codomains\text{-}def\ targets\text{-}def$
$horizontal\text{-}identity\text{-}is\text{-}weak\text{-}unit$
by $auto$

lemma $arr\text{-}agreement$:
shows $arr = H.arr$
using $arr\text{-}def\ H.arr\text{-}def\ arr\text{-}iff\text{-}has\text{-}src\ arr\text{-}iff\text{-}has\text{-}trg$
$sources\text{-}eq\text{-}domains\ targets\text{-}eq\text{-}codomains$
by $auto$

interpretation H: $category\ H$
proof
show $\bigwedge g\ f.\ g \star f \neq H.null \Longrightarrow H.seq\ g\ f$
using $arr\text{-}agreement\ hcomp\text{-}simps_{WC}(1)$ **by** $auto$
show $\bigwedge f.\ (H.domains\ f \neq \{\}) = (H.codomains\ f \neq \{\})$
using $sources\text{-}eq\text{-}domains\ targets\text{-}eq\text{-}codomains\ arr\text{-}iff\text{-}has\text{-}src\ arr\text{-}iff\text{-}has\text{-}trg$
by $simp$
fix $f\ g\ h$
show $H.seq\ h\ g \Longrightarrow H.seq\ (h \star g)\ f \Longrightarrow H.seq\ g\ f$
using $null\text{-}agreement\ arr\text{-}agreement\ H.not\text{-}arr\text{-}null\ preserves\text{-}arr\ VoV.arr\text{-}char_{SbC}$
by $(metis\ hseq\text{-}char'\ match\text{-}1)$
show $H.seq\ h\ (g \star f) \Longrightarrow H.seq\ g\ f \Longrightarrow H.seq\ h\ g$
using $null\text{-}agreement\ arr\text{-}agreement\ H.not\text{-}arr\text{-}null\ preserves\text{-}arr\ VoV.arr\text{-}char_{SbC}$
by $(metis\ hseq\text{-}char'\ match\text{-}2)$
show $H.seq\ g\ f \Longrightarrow H.seq\ h\ g \Longrightarrow H.seq\ (h \star g)\ f$
using $arr\text{-}agreement\ match\text{-}3\ hseq\text{-}char(1)$ **by** $auto$
show $H.seq\ g\ f \Longrightarrow H.seq\ h\ g \Longrightarrow (h \star g) \star f = h \star g \star f$
proof $-$
assume hg: $H.seq\ h\ g$
assume gf: $H.seq\ g\ f$
have $iso\ \mathrm{a}[h,\ g,\ f] \wedge «\mathrm{a}[h,\ g,\ f] : (h \star g) \star f \Rightarrow h \star g \star f»$
using $hg\ gf\ vertically\text{-}discrete\ arr\text{-}agreement\ hseq\text{-}char\ assoc\text{-}in\text{-}hom\ iso\text{-}assoc$
by $auto$
thus $?thesis$
using $arr\text{-}agreement\ vertically\text{-}discrete$ **by** $auto$
qed
qed

proposition *is-category*:
shows *category H*
..

end

1.4.3 Obtaining the Unitors

We now want to exploit the construction of unitors in a prebicategory with units, to obtain left and right unitors in a bicategory. However, a bicategory is not *a priori* a prebicategory with units, because a bicategory only assigns unit isomorphisms to each *object*, not to each weak unit. In order to apply the results about prebicategories with units to a bicategory, we first need to extend the bicategory to a prebicategory with units, by extending the mapping ι, which provides a unit isomorphism for each object, to a mapping that assigns a unit isomorphism to all weak units. This extension can be made in an arbitrary way, as the values chosen for non-objects ultimately do not affect the components of the unitors at objects.

context *bicategory*
begin

interpretation *prebicategory V H* a
using *is-prebicategory* **by** *auto*

definition i'
where $\mathrm{i}'\ a \equiv \mathit{SOME}\ \varphi.\ \mathit{iso}\ \varphi \wedge \varphi \in \mathit{hom}\ (a \star a)\ a \wedge (\mathit{obj}\ a \longrightarrow \varphi = \mathrm{i}[a])$

lemma i'-*extends*-i:
assumes *weak-unit a*
shows $\mathit{iso}\ (\mathrm{i}'\ a)$ **and** «$\mathrm{i}'\ a : a \star a \Rightarrow a$» **and** $\mathit{obj}\ a \Longrightarrow \mathrm{i}'\ a = \mathrm{i}[a]$
proof −
let $?P = \lambda a\ \varphi.\ \mathit{iso}\ \varphi \wedge$ «$\varphi : a \star a \Rightarrow a$» $\wedge\ (\mathit{obj}\ a \longrightarrow \varphi = \mathrm{i}[a])$
have $\exists \varphi.\ ?P\ a\ \varphi$
by (*metis assms iso-some-unit*(*1*) *iso-some-unit*(*2*) *iso-unit unit-in-vhom*)
hence *1*: $?P\ a\ (\mathrm{i}'\ a)$
using i'-*def someI-ex* [*of* $?P\ a$] **by** *simp*
show $\mathit{iso}\ (\mathrm{i}'\ a)$ **using** *1* **by** *simp*
show «$\mathrm{i}'\ a : a \star a \Rightarrow a$» **using** *1* **by** *simp*
show $\mathit{obj}\ a \Longrightarrow \mathrm{i}'\ a = \mathrm{i}[a]$ **using** *1* **by** *simp*
qed

proposition *extends-to-prebicategory-with-units*:
shows *prebicategory-with-units V H* a i'
using i'-*extends*-i **by** *unfold-locales auto*

interpretation *PB*: *prebicategory-with-units V H* a i'
using *extends-to-prebicategory-with-units* **by** *auto*
interpretation *PB*: *prebicategory-with-homs V H* a *src trg*

using *is-prebicategory-with-homs* **by** *auto*
interpretation *PB*: *prebicategory-with-homs-and-units V H* a i′ *src trg* **..**

proposition *extends-to-prebicategory-with-homs-and-units*:
shows *prebicategory-with-homs-and-units V H* a i′ *src trg*
..

definition *lunit* (‹l[-]›)
where l[*a*] ≡ *PB.lunit a*

definition *runit* (‹r[-]›)
where r[*a*] ≡ *PB.runit a*

abbreviation *lunit′* (‹l^{-1}[-]›)
where $l^{-1}[a] \equiv$ *inv* l[*a*]

abbreviation *runit′* (‹r^{-1}[-]›)
where $r^{-1}[a] \equiv$ *inv* r[*a*]

The characterizations of the left and right unitors that we obtain from locale *prebicategory-with-homs-and-units* mention the arbitarily chosen extension i′, rather than the given i. We want "native versions" for the present context.

lemma *lunit-char*:
assumes *ide f*
shows «l[*f*] : *L f* ⇒ *f*» **and** *L* l[*f*] = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]
and ∃!μ. «μ : *L f* ⇒ *f*» ∧ *L* μ = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]
proof −
  have *1*: *trg* (*PB.lunit f*) = *trg f*
    using *assms PB.lunit-char* [*of f*] *vconn-implies-hpar*(*2*) *vconn-implies-hpar*(*4*)
    by *metis*
  show «l[*f*] : *L f* ⇒ *f*»
    unfolding *lunit-def*
    using *assms PB.lunit-char* **by** *simp*
  show *L* l[*f*] = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]
    unfolding *lunit-def*
    using *assms 1 PB.lunit-char obj-is-weak-unit* i′-*extends*-i **by** *simp*
  let *?P* = λμ. «μ : *L f* ⇒ *f*» ∧ *L* μ = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]
  have *?P* = (λμ. «μ : *trg f* ⋆ *f* ⇒ *f*» ∧
              trg f ⋆ μ = (i′ (*trg f*) ⋆ *f*) · *inv* a[*trg f*, *trg f*, *f*])
  proof −
    have ⋀μ. «μ : *L f* ⇒ *f*» ⟷ «μ : *trg f* ⋆ *f* ⇒ *f*»
      using *assms* **by** *simp*
    moreover have ⋀μ. «μ : *L f* ⇒ *f*» ⟹
              L μ = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*] ⟷
              trg f ⋆ μ = (i′ (*trg f*) ⋆ *f*) · *inv* a[*trg f*, *trg f*, *f*]
      using *calculation obj-is-weak-unit* i′-*extends*-i **by** *auto*
    ultimately show *?thesis* **by** *blast*
  qed
  thus ∃!μ. «μ : *L f* ⇒ *f*» ∧ *L* μ = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]

using *assms PB.lunit-char* **by** *simp*
qed

lemma *lunit-in-hom* [*intro*]:
assumes *ide f*
shows «l[*f*] : *src f* → *trg f*» **and** «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
proof −
  show «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
    using *assms lunit-char* **by** *auto*
  thus «l[*f*] : *src f* → *trg f*»
    by (*metis arrI in-hhomI vconn-implies-hpar*(*1*−*4*))
qed

lemma *lunit-in-vhom* [*simp*]:
assumes *ide f* **and** *trg f* = *b*
shows «l[*f*] : *b* ⋆ *f* ⇒ *f*»
  using *assms* **by** *auto*

lemma *lunit-simps* [*simp*]:
assumes *ide f*
shows *arr* l[*f*] **and** *src* l[*f*] = *src f* **and** *trg* l[*f*] = *trg f*
and *dom* l[*f*] = *trg f* ⋆ *f* **and** *cod* l[*f*] = *f*
  using *assms lunit-in-hom*
      apply *auto*
  using *assms lunit-in-hom*
   apply *blast*
  using *assms lunit-in-hom*
  by *blast*

lemma *runit-char*:
assumes *ide f*
shows «r[*f*] : *R f* ⇒ *f*» **and** *R* r[*f*] = (*f* ⋆ i[*src f*]) · a[*f*, *src f*, *src f*]
and ∃!*μ*. «*μ* : *R f* ⇒ *f*» ∧ *R μ* = (*f* ⋆ i[*src f*]) · a[*f*, *src f*, *src f*]
proof −
  have *1*: *src* (*PB.runit f*) = *src f*
    using *assms PB.runit-char* [*of f*] *vconn-implies-hpar*(*1*) *vconn-implies-hpar*(*3*)
    by *metis*
  show «r[*f*] : *R f* ⇒ *f*»
    unfolding *runit-def*
    using *assms PB.runit-char* **by** *simp*
  show *R* r[*f*] = (*f* ⋆ i[*src f*]) · a[*f*, *src f*, *src f*]
    unfolding *runit-def*
    using *assms 1 PB.runit-char obj-is-weak-unit* i′*-extends*-i **by** *simp*
  let *?P* = λ*μ*. «*μ* : *R f* ⇒ *f*» ∧ *R μ* = (*f* ⋆ i[*src f*]) · a[*f*, *src f*, *src f*]
  have *?P* = (λ*μ*. «*μ* : *f* ⋆ *src f* ⇒ *f*» ∧
              μ ⋆ *src f* = (*f* ⋆ i′ (*src f*)) · a[*f*, *src f*, *src f*])
  proof −
    have ⋀*μ*. «*μ* : *R f* ⇒ *f*» ⟷ «*μ* : *f* ⋆ *src f* ⇒ *f*»
      using *assms* **by** *simp*

moreover have $\bigwedge\mu.$ «μ : $R\ f \Rightarrow f$» $\Longrightarrow$
$R\ \mu = (f \star \mathrm{i}[src\ f]) \cdot \mathrm{a}[f,\ src\ f,\ src\ f] \longleftrightarrow$
$\mu \star src\ f = (f \star \mathrm{i}'\ (src\ f)) \cdot\ \mathrm{a}[f,\ src\ f,\ src\ f]$
using *calculation obj-is-weak-unit* i′*-extends*-i **by** *auto*
ultimately show *?thesis* **by** *blast*
qed
thus $\exists!\mu.$ «μ : $R\ f \Rightarrow f$» $\wedge$ $R\ \mu = (f \star \mathrm{i}[src\ f]) \cdot \mathrm{a}[f,\ src\ f,\ src\ f]$
using *assms PB.runit-char* **by** *simp*
qed

lemma *runit-in-hom* [*intro*]:
assumes *ide f*
shows «r[f] : $src\ f \rightarrow trg\ f$» **and** «r[$f$] : $f \star src\ f \Rightarrow f$»
proof −
show «r[f] : $f \star src\ f \Rightarrow f$»
using *assms runit-char* **by** *auto*
thus «r[f] : $src\ f \rightarrow trg\ f$»
by (*metis arrI in-hhom-def vconn-implies-hpar*(*1*−*4*))
qed

lemma *runit-in-vhom* [*simp*]:
assumes *ide f* **and** $src\ f = a$
shows «r[f] : $f \star a \Rightarrow f$»
using *assms* **by** *auto*

lemma *runit-simps* [*simp*]:
assumes *ide f*
shows *arr* r[f] **and** *src* r[f] = $src\ f$ **and** *trg* r[f] = $trg\ f$
and *dom* r[f] = $f \star src\ f$ **and** *cod* r[f] = f
using *assms runit-in-hom*
apply *auto*
using *assms runit-in-hom*
apply *blast*
using *assms runit-in-hom*
by *blast*

lemma *lunit-eqI*:
assumes *ide f* **and** «μ : $trg\ f \star f \Rightarrow f$»
and $trg\ f \star \mu = (\mathrm{i}[trg\ f] \star f) \cdot \mathrm{a}^{-1}[trg\ f,\ trg\ f,\ f]$
shows μ = l[f]
unfolding *lunit-def*
using *assms PB.lunit-eqI* i′*-extends*-i *trg.preserves-ide obj-is-weak-unit* **by** *simp*

lemma *runit-eqI*:
assumes *ide f* **and** «μ : $f \star src\ f \Rightarrow f$»
and $\mu \star src\ f = (f \star \mathrm{i}[src\ f]) \cdot \mathrm{a}[f,\ src\ f,\ src\ f]$
shows μ = r[f]
unfolding *runit-def*
using *assms PB.runit-eqI* i′*-extends*-i *src.preserves-ide obj-is-weak-unit* **by** *simp*

lemma *lunit-naturality*:
assumes *arr* μ
shows $\mu \cdot \mathrm{l}[dom\ \mu] = \mathrm{l}[cod\ \mu] \cdot (trg\ \mu \star \mu)$
  unfolding *lunit-def*
  using *assms PB.lunit-naturality* **by** *auto*

lemma *runit-naturality*:
assumes *arr* μ
shows $\mu \cdot \mathrm{r}[dom\ \mu] = \mathrm{r}[cod\ \mu] \cdot (\mu \star src\ \mu)$
  unfolding *runit-def*
  using *assms PB.runit-naturality* **by** *auto*

lemma *iso-lunit* [*simp*]:
assumes *ide f*
shows *iso* $\mathrm{l}[f]$
  unfolding *lunit-def*
  using *assms PB.iso-lunit* **by** *blast*

lemma *iso-runit* [*simp*]:
assumes *ide f*
shows *iso* $\mathrm{r}[f]$
  unfolding *runit-def*
  using *assms PB.iso-runit* **by** *blast*

lemma *iso-lunit′* [*simp*]:
assumes *ide f*
shows *iso* $\mathrm{l}^{-1}[f]$
  using *assms iso-lunit* **by** *blast*

lemma *iso-runit′* [*simp*]:
assumes *ide f*
shows *iso* $\mathrm{r}^{-1}[f]$
  using *assms iso-runit* **by** *blast*

lemma *lunit′-in-hom* [*intro*]:
assumes *ide f*
shows «$\mathrm{l}^{-1}[f] : src\ f \rightarrow trg\ f$» **and** «$\mathrm{l}^{-1}[f] : f \Rightarrow trg\ f \star f$»
proof −
  show «$\mathrm{l}^{-1}[f] : f \Rightarrow trg\ f \star f$»
    using *assms lunit-char iso-lunit* **by** *simp*
  thus «$\mathrm{l}^{-1}[f] : src\ f \rightarrow trg\ f$»
    using *assms src-dom trg-dom* **by** *simp*
qed

lemma *lunit′-in-vhom* [*simp*]:
assumes *ide f* **and** $trg\ f = b$
shows «$\mathrm{l}^{-1}[f] : f \Rightarrow b \star f$»
  using *assms* **by** *auto*

lemma *lunit′-simps* [*simp*]:
assumes *ide f*
shows *arr* $\mathrm{l}^{-1}[f]$ **and** *src* $\mathrm{l}^{-1}[f] = src\ f$ **and** *trg* $\mathrm{l}^{-1}[f] = trg\ f$
and *dom* $\mathrm{l}^{-1}[f] = f$ **and** *cod* $\mathrm{l}^{-1}[f] = trg\ f \star f$
  using *assms lunit′-in-hom* **by** *auto*

lemma *runit′-in-hom* [*intro*]:
assumes *ide f*
shows «$\mathrm{r}^{-1}[f] : src\ f \rightarrow trg\ f$» **and** «$\mathrm{r}^{-1}[f] : f \Rightarrow f \star src\ f$»
proof −
  show «$\mathrm{r}^{-1}[f] : f \Rightarrow f \star src\ f$»
    using *assms runit-char iso-runit* **by** *simp*
  thus «$\mathrm{r}^{-1}[f] : src\ f \rightarrow trg\ f$»
    using *src-dom trg-dom*
    by (*simp add*: *assms*)
qed

lemma *runit′-in-vhom* [*simp*]:
assumes *ide f* **and** *src f* = *a*
shows «$\mathrm{r}^{-1}[f] : f \Rightarrow f \star a$»
  using *assms* **by** *auto*

lemma *runit′-simps* [*simp*]:
assumes *ide f*
shows *arr* $\mathrm{r}^{-1}[f]$ **and** *src* $\mathrm{r}^{-1}[f] = src\ f$ **and** *trg* $\mathrm{r}^{-1}[f] = trg\ f$
and *dom* $\mathrm{r}^{-1}[f] = f$ **and** *cod* $\mathrm{r}^{-1}[f] = f \star src\ f$
  using *assms runit′-in-hom* **by** *auto*

interpretation *L*: *endofunctor V L* **..**
interpretation ℓ: *transformation-by-components V V L map lunit*
  using *lunit-in-hom lunit-naturality* **by** *unfold-locales auto*
interpretation ℓ: *natural-isomorphism V V L map* ℓ.*map*
  using *iso-lunit* **by** (*unfold-locales*, *auto*)

lemma *natural-isomorphism-*ℓ:
shows *natural-isomorphism V V L map* ℓ.*map*
  ..

abbreviation ℓ
where ℓ ≡ ℓ.*map*

lemma ℓ-*ide-simp*:
assumes *ide f*
shows ℓ *f* = $\mathrm{l}[f]$
  using *assms* **by** *simp*

interpretation *L*: *equivalence-functor V V L*
  using *L.isomorphic-to-identity-is-equivalence* ℓ.*natural-isomorphism-axioms* **by** *simp*

lemma *equivalence-functor-L*:
shows *equivalence-functor V V L*
  ..

lemma *lunit-commutes-with-L*:
assumes *ide f*
shows $\mathrm{l}[L\ f] = L\ \mathrm{l}[f]$
  unfolding *lunit-def*
  using *assms PB.lunit-commutes-with-L* **by** *blast*

interpretation *R*: *endofunctor V R* **..**
interpretation $\mathfrak{r}$: *transformation-by-components V V R map runit*
  using *runit-in-hom runit-naturality* **by** *unfold-locales auto*
interpretation $\mathfrak{r}$: *natural-isomorphism V V R map* $\mathfrak{r}$.*map*
  using *iso-runit* **by** (*unfold-locales*, *auto*)

lemma *natural-isomorphism-*$\mathfrak{r}$:
shows *natural-isomorphism V V R map* $\mathfrak{r}$.*map*
  ..

abbreviation $\mathfrak{r}$
where $\mathfrak{r} \equiv \mathfrak{r}$.*map*

lemma $\mathfrak{r}$-*ide-simp*:
assumes *ide f*
shows $\mathfrak{r}\ f = \mathrm{r}[f]$
  using *assms* **by** *simp*

interpretation *R*: *equivalence-functor V V R*
  using *R.isomorphic-to-identity-is-equivalence* $\mathfrak{r}$.*natural-isomorphism-axioms* **by** *simp*

lemma *equivalence-functor-R*:
shows *equivalence-functor V V R*
  ..

lemma *runit-commutes-with-R*:
assumes *ide f*
shows $\mathrm{r}[R\ f] = R\ \mathrm{r}[f]$
  unfolding *runit-def*
  using *assms PB.runit-commutes-with-R* **by** *blast*

lemma *lunit′-naturality*:
assumes *arr* μ
shows $(trg\ \mu \star \mu) \cdot \mathrm{l}^{-1}[dom\ \mu] = \mathrm{l}^{-1}[cod\ \mu] \cdot \mu$
  using *assms iso-lunit lunit-naturality invert-opposite-sides-of-square L.preserves-arr L.preserves-cod arr-cod ide-cod ide-dom lunit-simps*(*1*) *lunit-simps*(*4*) *seqI*
  by *presburger*

lemma *runit′-naturality*:
assumes *arr* μ
shows $(\mu \star src\ \mu) \cdot \mathrm{r}^{-1}[dom\ \mu] = \mathrm{r}^{-1}[cod\ \mu] \cdot \mu$
  using *assms iso-runit runit-naturality invert-opposite-sides-of-square R.preserves-arr R.preserves-cod arr-cod ide-cod ide-dom runit-simps(1) runit-simps(4) seqI*
  by *presburger*

lemma *isomorphic-unit-right*:
assumes *ide f*
shows $f \star src\ f \cong f$
  using *assms runit′-in-hom iso-runit′ isomorphic-def isomorphic-symmetric* **by** *blast*

lemma *isomorphic-unit-left*:
assumes *ide f*
shows $trg\ f \star f \cong f$
  using *assms lunit′-in-hom iso-lunit′ isomorphic-def isomorphic-symmetric* **by** *blast*

end

1.4.4 Further Properties of Bicategories

Here we derive further properties of bicategories, now that we have the unitors at our disposal. This section generalizes the corresponding development in theory *MonoidalCategory.MonoidalCategory*, which has some diagrams to illustrate the longer calculations. The present section also includes some additional facts that are now nontrivial due to the partiality of horizontal composition.

context *bicategory*
begin

lemma *unit-simps* [*simp*]:
assumes *obj a*
shows $arr\ \mathrm{i}[a]$ **and** $src\ \mathrm{i}[a] = a$ **and** $trg\ \mathrm{i}[a] = a$
and $dom\ \mathrm{i}[a] = a \star a$ **and** $cod\ \mathrm{i}[a] = a$
  using *assms unit-in-hom* **by** *blast+*

lemma *triangle*:
assumes *ide f* **and** *ide g* **and** $src\ g = trg\ f$
shows $(g \star \mathrm{l}[f]) \cdot \mathrm{a}[g,\ src\ g,\ f] = \mathrm{r}[g] \star f$
proof −
  let $?b = src\ g$
  have $*$: $(g \star \mathrm{l}[?b \star f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] = \mathrm{r}[g] \star ?b \star f$
  proof −
    have *1*: $((g \star \mathrm{l}[?b \star f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f]) \cdot \mathrm{a}[g \star ?b,\ ?b,\ f]$
      $= (\mathrm{r}[g] \star ?b \star f) \cdot \mathrm{a}[g \star ?b,\ ?b,\ f]$
    proof −
      have $((g \star \mathrm{l}[?b \star f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f]) \cdot \mathrm{a}[g \star ?b,\ ?b,\ f]$
        $= (g \star \mathrm{l}[?b \star f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] \cdot \mathrm{a}[g \star ?b,\ ?b,\ f]$
        using *HoVH-def HoHV-def comp-assoc* **by** *auto*

also have
$\ldots = (g \star \mathrm{l}[?b \star f]) \cdot (g \star \mathrm{a}[?b,\ ?b,\ f]) \cdot \mathrm{a}[g,\ ?b \star ?b,\ f] \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms pentagon* **by** *force*
also have
$\ldots = ((g \star \mathrm{l}[?b \star f]) \cdot (g \star \mathrm{a}[?b,\ ?b,\ f])) \cdot \mathrm{a}[g,\ ?b \star ?b,\ f] \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms assoc-in-hom HoVH-def HoHV-def comp-assoc* **by** *auto*
also have
$\ldots = ((g \star ?b \star \mathrm{l}[f]) \cdot (g \star \mathrm{a}[?b,\ ?b,\ f])) \cdot \mathrm{a}[g,\ ?b \star ?b,\ f] \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms lunit-commutes-with-L lunit-in-hom* **by** *force*
also have $\ldots = ((g \star (\mathrm{i}[?b] \star f) \cdot \mathrm{a}^{-1}[?b,\ ?b,\ f]) \cdot (g \star \mathrm{a}[?b,\ ?b,\ f]))$
$\cdot\ \mathrm{a}[g,\ ?b \star ?b,\ f] \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms lunit-char(2)* **by** *force*
also have $\ldots = (g \star ((\mathrm{i}[?b] \star f) \cdot \mathrm{a}^{-1}[?b,\ ?b,\ f]) \cdot \mathrm{a}[?b,\ ?b,\ f])$
$\cdot\ \mathrm{a}[g,\ ?b \star ?b,\ f] \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms interchange* [*of g g* $(\mathrm{i}[?b] \star f) \cdot \mathrm{a}^{-1}[?b,\ ?b,\ f]$ $\mathrm{a}[?b,\ ?b,\ f]$]
by *auto*
also have $\ldots = ((g \star \mathrm{i}[?b] \star f) \cdot \mathrm{a}[g,\ ?b \star ?b,\ f]) \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms comp-arr-dom comp-assoc-assoc′ comp-assoc* **by** *auto*
also have $\ldots = (\mathrm{a}[g,\ ?b,\ f] \cdot ((g \star \mathrm{i}[?b]) \star f)) \cdot (\mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms assoc-naturality* [*of g* $\mathrm{i}[?b]$ *f*] **by** *simp*
also have $\ldots = \mathrm{a}[g,\ ?b,\ f] \cdot ((g \star \mathrm{i}[?b]) \cdot \mathrm{a}[g,\ ?b,\ ?b] \star f)$
using *assms interchange* [*of g* $\star$ $\mathrm{i}[?b]$ $\mathrm{a}[g,\ ?b,\ ?b]$ *f f*] *comp-assoc* **by** *simp*
also have $\ldots = \mathrm{a}[g,\ ?b,\ f] \cdot ((\mathrm{r}[g] \star ?b) \star f)$
using *assms runit-char(2)* **by** *force*
also have $\ldots = (\mathrm{r}[g] \star ?b \star f) \cdot \mathrm{a}[g \star ?b,\ ?b,\ f]$
using *assms assoc-naturality* [*of* $\mathrm{r}[g]$ *?b f*] **by** *auto*
finally show *?thesis* **by** *blast*
qed
show $(g \star \mathrm{l}[?b \star f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] = \mathrm{r}[g] \star ?b \star f$
proof −
have *epi* $\mathrm{a}[g \star ?b,\ ?b,\ f]$
using *assms preserves-ide iso-assoc iso-is-retraction retraction-is-epi* **by** *force*
thus *?thesis*
using *assms 1 epi-cancel* **by** *auto*
qed
qed
have $(g \star \mathrm{l}[f]) \cdot \mathrm{a}[g,\ ?b,\ f] = ((g \star \mathrm{l}[f]) \cdot (g \star \mathrm{l}[?b \star f]) \cdot (g \star ?b \star \mathrm{l}^{-1}[f])) \cdot$
$(g \star ?b \star \mathrm{l}[f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] \cdot ((g \star ?b) \star \mathrm{l}^{-1}[f])$
proof −
have $\mathrm{a}[g,\ ?b,\ f] = (g \star ?b \star \mathrm{l}[f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] \cdot ((g \star ?b) \star \mathrm{l}^{-1}[f])$
proof −
have $\mathrm{a}[g,\ ?b,\ f] = (g \star ?b \star f) \cdot \mathrm{a}[g,\ ?b,\ f]$
using *assms comp-cod-arr* **by** *simp*
have $\mathrm{a}[g,\ ?b,\ f] = ((g \star ?b \star \mathrm{l}[f]) \cdot (g \star ?b \star \mathrm{l}^{-1}[f])) \cdot \mathrm{a}[g,\ ?b,\ f]$
using *assms comp-cod-arr comp-arr-inv′ whisker-left* [*of g*]
whisker-left [*of ?b* $\mathrm{l}[f]$ $\mathrm{l}^{-1}[f]$]
by *simp*
also have $\ldots = (g \star ?b \star \mathrm{l}[f]) \cdot \mathrm{a}[g,\ ?b,\ ?b \star f] \cdot ((g \star ?b) \star \mathrm{l}^{-1}[f])$
using *assms iso-lunit assoc-naturality* [*of g ?b* $\mathrm{l}^{-1}[f]$] *comp-assoc* **by** *force*

```
      finally show ?thesis by blast
    qed
    moreover have g ⋆ l[f] = (g ⋆ l[f]) · (g ⋆ l[?b ⋆ f]) · (g ⋆ ?b ⋆ l⁻¹[f])
    proof −
      have (g ⋆ l[?b ⋆ f]) · (g ⋆ ?b ⋆ l⁻¹[f]) = g ⋆ ?b ⋆ f
      proof −
        have (g ⋆ l[?b ⋆ f]) · (g ⋆ ?b ⋆ l⁻¹[f]) = (g ⋆ ?b ⋆ l[f]) · (g ⋆ ?b ⋆ l⁻¹[f])
          using assms lunit-in-hom lunit-commutes-with-L by simp
        also have ... = g ⋆ ?b ⋆ f
          using assms comp-arr-inv′ whisker-left [of g] whisker-left [of ?b l[f] l⁻¹[f]]
          by simp
        finally show ?thesis by blast
      qed
      thus ?thesis
        using assms comp-arr-dom by auto
    qed
    ultimately show ?thesis by simp
  qed
  also have ... = (g ⋆ l[f]) · (g ⋆ l[?b ⋆ f]) · ((g ⋆ ?b ⋆ l⁻¹[f]) · (g ⋆ ?b ⋆ l[f])) ·
                  a[g, ?b, ?b ⋆ f] · ((g ⋆ ?b) ⋆ l⁻¹[f])
    using comp-assoc by simp
  also have ... = (g ⋆ l[f]) · (g ⋆ l[?b ⋆ f]) · ((g ⋆ ?b ⋆ (?b ⋆ f)) ·
                  a[g, ?b, ?b ⋆ f]) · ((g ⋆ ?b) ⋆ l⁻¹[f])
    using assms iso-lunit comp-inv-arr′ interchange [of g g ?b ⋆ l⁻¹[f] ?b ⋆ l[f]]
          interchange [of ?b ?b l⁻¹[f] l[f]] comp-assoc
    by auto
  also have ... = (g ⋆ l[f]) · ((g ⋆ l[?b ⋆ f]) · a[g, ?b, ?b ⋆ f]) · ((g ⋆ ?b) ⋆ l⁻¹[f])
    using assms comp-cod-arr comp-assoc by auto
  also have ... = r[g] ⋆ f
  proof −
    have r[g] ⋆ f = (g ⋆ l[f]) · (r[g] ⋆ ?b ⋆ f) · ((g ⋆ ?b) ⋆ l⁻¹[f])
    proof −
      have (g ⋆ l[f]) · (r[g] ⋆ ?b ⋆ f) · ((g ⋆ ?b) ⋆ l⁻¹[f])
             = (g ⋆ l[f]) · (r[g] · (g ⋆ ?b) ⋆ (?b ⋆ f) · l⁻¹[f])
        using assms iso-lunit interchange [of r[g] g ⋆ ?b ?b ⋆ f l⁻¹[f]]
        by force
      also have ... = (g ⋆ l[f]) · (r[g] ⋆ l⁻¹[f])
        using assms comp-arr-dom comp-cod-arr by simp
      also have ... = r[g] ⋆ l[f] · l⁻¹[f]
        using assms interchange [of g r[g] l[f] l⁻¹[f]] comp-cod-arr
        by simp
      also have ... = r[g] ⋆ f
        using assms iso-lunit comp-arr-inv′ by simp
      finally show ?thesis by argo
    qed
    thus ?thesis using assms ∗ by argo
  qed
  finally show ?thesis by blast
qed
```

lemma *lunit-hcomp-gen*:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows $(f \star \mathrm{l}[g \star h]) \cdot (f \star \mathrm{a}[trg\ g,\ g,\ h]) = f \star \mathrm{l}[g] \star h$
proof −
  have $((f \star \mathrm{l}[g \star h]) \cdot (f \star \mathrm{a}[trg\ g,\ g,\ h])) \cdot \mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h) =$
    $(f \star (\mathrm{l}[g] \star h)) \cdot \mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h)$
  proof −
    have $((f \star \mathrm{l}[g \star h]) \cdot (f \star \mathrm{a}[trg\ g,\ g,\ h])) \cdot (\mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h)) =$
      $((f \star \mathrm{l}[g \star h]) \cdot \mathrm{a}[f,\ trg\ g,\ g \star h]) \cdot \mathrm{a}[f \star trg\ g,\ g,\ h]$
      using *assms pentagon comp-assoc* **by** *simp*
    also have ... $= (\mathrm{r}[f] \star (g \star h)) \cdot \mathrm{a}[f \star trg\ g,\ g,\ h]$
      using *assms triangle* [*of* $g \star h\ f$] **by** *auto*
    also have ... $= \mathrm{a}[f,\ g,\ h] \cdot ((\mathrm{r}[f] \star g) \star h)$
      using *assms assoc-naturality* [*of* $\mathrm{r}[f]$ g h] **by** *simp*
    also have ... $= (\mathrm{a}[f,\ g,\ h] \cdot ((f \star \mathrm{l}[g]) \star h)) \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h)$
      using *assms triangle interchange* [*of* $f \star \mathrm{l}[g]$ $\mathrm{a}[f,\ trg\ g,\ g]$ h h] *comp-assoc*
      by *auto*
    also have ... $= (f \star (\mathrm{l}[g] \star h)) \cdot (\mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h))$
      using *assms assoc-naturality* [*of* f $\mathrm{l}[g]$ h] *comp-assoc* **by** *simp*
    finally show *?thesis* **by** *blast*
  qed
  moreover have *iso* $(\mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h))$
    using *assms iso-assoc isos-compose* **by** *simp*
  ultimately show *?thesis*
    using *assms iso-is-retraction retraction-is-epi*
      epi-cancel
        [*of* $\mathrm{a}[f,\ trg\ g \star g,\ h] \cdot (\mathrm{a}[f,\ trg\ g,\ g] \star h)$
          $(f \star \mathrm{l}[g \star h]) \cdot (f \star \mathrm{a}[trg\ g,\ g,\ h])$ $f \star \mathrm{l}[g] \star h$]
   by *auto*
qed

lemma *lunit-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows $\mathrm{l}[f \star g] \cdot \mathrm{a}[trg\ f,\ f,\ g] = \mathrm{l}[f] \star g$
and $\mathrm{a}^{-1}[trg\ f,\ f,\ g] \cdot \mathrm{l}^{-1}[f \star g] = \mathrm{l}^{-1}[f] \star g$
and $\mathrm{l}[f \star g] = (\mathrm{l}[f] \star g) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ g]$
and $\mathrm{l}^{-1}[f \star g] = \mathrm{a}[trg\ f,\ f,\ g] \cdot (\mathrm{l}^{-1}[f] \star g)$
proof −
  show *1*: $\mathrm{l}[f \star g] \cdot \mathrm{a}[trg\ f,\ f,\ g] = \mathrm{l}[f] \star g$
  proof −
    have $L\ (\mathrm{l}[f \star g] \cdot \mathrm{a}[trg\ f,\ f,\ g]) = L\ (\mathrm{l}[f] \star g)$
      using *assms interchange* [*of trg f trg f* $\mathrm{l}[f \star g]$ $\mathrm{a}[trg\ f,\ f,\ g]$] *lunit-hcomp-gen*
      by *fastforce*
    thus *?thesis*
      using *assms L.is-faithful* [*of* $\mathrm{l}[f \star g] \cdot \mathrm{a}[trg\ f,\ f,\ g]$ $\mathrm{l}[f] \star g$] **by** *force*
  qed
  show $\mathrm{a}^{-1}[trg\ f,\ f,\ g] \cdot \mathrm{l}^{-1}[f \star g] = \mathrm{l}^{-1}[f] \star g$

proof −
have $\mathrm{a}^{-1}[trg\ f,\ f,\ g] \cdot \mathrm{l}^{-1}[f \star g] = inv\ (\mathrm{l}[f \star g] \cdot \mathrm{a}[trg\ f,\ f,\ g])$
using *assms* **by** (*simp add*: *inv-comp*)
also have ... $= inv\ (\mathrm{l}[f] \star g)$
using *1* **by** *simp*
also have ... $= \mathrm{l}^{-1}[f] \star g$
using *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed
show *2*: $\mathrm{l}[f \star g] = (\mathrm{l}[f] \star g) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ g]$
using *assms 1 invert-side-of-triangle*(*2*) **by** *auto*
show $\mathrm{l}^{-1}[f \star g] = \mathrm{a}[trg\ f,\ f,\ g] \cdot (\mathrm{l}^{-1}[f] \star g)$
proof −
have $\mathrm{l}^{-1}[f \star g] = inv\ ((\mathrm{l}[f] \star g) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ g])$
using *2* **by** *simp*
also have ... $= \mathrm{a}[trg\ f,\ f,\ g] \cdot inv\ (\mathrm{l}[f] \star g)$
using *assms inv-comp* **by** *simp*
also have ... $= \mathrm{a}[trg\ f,\ f,\ g] \cdot (\mathrm{l}^{-1}[f] \star g)$
using *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *runit-hcomp-gen*:
assumes *ide f* **and** *ide g* **and** *ide h*
and $src\ f = trg\ g$ **and** $src\ g = trg\ h$
shows $\mathrm{r}[f \star g] \star h = ((f \star \mathrm{r}[g]) \star h) \cdot (\mathrm{a}[f,\ g,\ src\ g] \star h)$
proof −
have $\mathrm{r}[f \star g] \star h = ((f \star g) \star \mathrm{l}[h]) \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
using *assms triangle* **by** *simp*
also have ... $= (\mathrm{a}^{-1}[f,\ g,\ h] \cdot (f \star g \star \mathrm{l}[h]) \cdot \mathrm{a}[f,\ g,\ src\ g \star h]) \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
using *assms assoc-naturality* [*of f g* $\mathrm{l}[h]$] *invert-side-of-triangle*(*1*)
by *simp*
also have ... $= \mathrm{a}^{-1}[f,\ g,\ h] \cdot (f \star g \star \mathrm{l}[h]) \cdot \mathrm{a}[f,\ g,\ src\ g \star h] \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
using *comp-assoc* **by** *simp*
also have ... $= (\mathrm{a}^{-1}[f,\ g,\ h] \cdot (f \star (\mathrm{r}[g] \star h))) \cdot (f \star \mathrm{a}^{-1}[g,\ src\ g,\ h]) \cdot$
$\mathrm{a}[f,\ g,\ src\ g \star h] \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
using *assms interchange* [*of f f*] *triangle comp-assoc*
invert-side-of-triangle(*2*) [*of* $\mathrm{r}[g] \star h\ g \star \mathrm{l}[h]\ \mathrm{a}[g,\ src\ g,\ h]$]
by *simp*
also have ... $= ((f \star \mathrm{r}[g]) \star h) \cdot \mathrm{a}^{-1}[f,\ g \star src\ g,\ h] \cdot (f \star \mathrm{a}^{-1}[g,\ src\ g,\ h]) \cdot$
$\mathrm{a}[f,\ g,\ src\ g \star h] \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
using *assms assoc′-naturality* [*of f* $\mathrm{r}[g]$ h] *comp-assoc* **by** *simp*
also have ... $= ((f \star \mathrm{r}[g]) \star h) \cdot (\mathrm{a}[f,\ g,\ src\ g] \star h)$
using *assms pentagon* [*of f g src g h*] *iso-assoc inv-hcomp*
invert-side-of-triangle(*1*)
[*of* $\mathrm{a}[f,\ g,\ src\ g \star h] \cdot \mathrm{a}[f \star g,\ src\ g,\ h]\ f \star \mathrm{a}[g,\ src\ g,\ h]$
$\mathrm{a}[f,\ g \star src\ g,\ h] \cdot (\mathrm{a}[f,\ g,\ src\ g] \star h)$]
invert-side-of-triangle(*1*)

$[of\ (f \star \mathrm{a}^{-1}[g,\ src\ g,\ h]) \cdot \mathrm{a}[f,\ g,\ src\ g \star h] \cdot \mathrm{a}[f \star g,\ src\ g,\ h]$
$\mathrm{a}[f,\ g \star src\ g,\ h]\ \mathrm{a}[f,\ g,\ src\ g] \star h]$
by *auto*
finally show *?thesis* **by** *blast*
qed

lemma *runit-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows $\mathrm{r}[f \star g] = (f \star \mathrm{r}[g]) \cdot \mathrm{a}[f,\ g,\ src\ g]$
and $\mathrm{r}^{-1}[f \star g] = \mathrm{a}^{-1}[f,\ g,\ src\ g] \cdot (f \star \mathrm{r}^{-1}[g])$
and $\mathrm{r}[f \star g] \cdot \mathrm{a}^{-1}[f,\ g,\ src\ g] = f \star \mathrm{r}[g]$
and $\mathrm{a}[f,\ g,\ src\ g] \cdot \mathrm{r}^{-1}[f \star g] = f \star \mathrm{r}^{-1}[g]$
proof −
show *1*: $\mathrm{r}[f \star g] = (f \star \mathrm{r}[g]) \cdot \mathrm{a}[f,\ g,\ src\ g]$
using *assms interchange* [*of f* $\star$ r[*g*] a[*f*, *g*, *src g*] *src g src g*]
runit-hcomp-gen [*of f g src g*]
R.is-faithful [*of* $(f \star \mathrm{r}[g]) \cdot (\mathrm{a}[f,\ g,\ src\ g])\ \mathrm{r}[f \star g]$]]
by *simp*
show $\mathrm{r}^{-1}[f \star g] = \mathrm{a}^{-1}[f,\ g,\ src\ g] \cdot (f \star \mathrm{r}^{-1}[g])$
using *assms 1 inv-comp inv-hcomp* **by** *auto*
show *2*: $\mathrm{r}[f \star g] \cdot \mathrm{a}^{-1}[f,\ g,\ src\ g] = f \star \mathrm{r}[g]$
using *assms 1 comp-arr-dom comp-cod-arr comp-assoc hseqI′ comp-assoc-assoc′* **by** *auto*
show $\mathrm{a}[f,\ g,\ src\ g] \cdot \mathrm{r}^{-1}[f \star g] = f \star \mathrm{r}^{-1}[g]$
proof −
have $\mathrm{a}[f,\ g,\ src\ g] \cdot \mathrm{r}^{-1}[f \star g] = inv\ (\mathrm{r}[f \star g] \cdot \mathrm{a}^{-1}[f,\ g,\ src\ g])$
using *assms inv-comp* **by** *simp*
also have ... = *inv* $(f \star \mathrm{r}[g])$
using *2* **by** *simp*
also have ... = $f \star \mathrm{r}^{-1}[g]$
using *assms inv-hcomp* [*of f* r[*g*]] **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *unitor-coincidence*:
assumes *obj a*
shows $\mathrm{l}[a] = \mathrm{i}[a]$ **and** $\mathrm{r}[a] = \mathrm{i}[a]$
proof −
have $R\ \mathrm{l}[a] = R\ \mathrm{i}[a] \wedge R\ \mathrm{r}[a] = R\ \mathrm{i}[a]$
proof −
have $R\ \mathrm{l}[a] = (a \star \mathrm{l}[a]) \cdot \mathrm{a}[a,\ a,\ a]$
using *assms lunit-hcomp* [*of a a*] *lunit-commutes-with-L* [*of a*] **by** *auto*
moreover have $(a \star \mathrm{l}[a]) \cdot \mathrm{a}[a,\ a,\ a] = R\ \mathrm{r}[a]$
using *assms triangle* [*of a a*] **by** *auto*
moreover have $(a \star \mathrm{l}[a]) \cdot \mathrm{a}[a,\ a,\ a] = R\ \mathrm{i}[a]$
proof −
have $(a \star \mathrm{l}[a]) \cdot \mathrm{a}[a,\ a,\ a] = ((\mathrm{i}[a] \star a) \cdot \mathrm{a}^{-1}[a,\ a,\ a]) \cdot \mathrm{a}[a,\ a,\ a]$
using *assms lunit-char*(*2*) **by** *force*
also have ... = $R\ \mathrm{i}[a]$

```
      using assms comp-arr-dom comp-assoc comp-assoc-assoc′
      apply (elim objE)
      by (simp add: assms)
    finally show ?thesis by blast
  qed
  ultimately show ?thesis by argo
qed
moreover have par l[a] i[a] ∧ par r[a] i[a]
  using assms by auto
ultimately have 1: l[a] = i[a] ∧ r[a] = i[a]
  using R.is-faithful by blast
show l[a] = i[a] using 1 by auto
show r[a] = i[a] using 1 by auto
qed

lemma unit-triangle:
assumes obj a
shows i[a] ⋆ a = (a ⋆ i[a]) · a[a, a, a]
and (i[a] ⋆ a) · a⁻¹[a, a, a] = a ⋆ i[a]
proof −
  show 1: i[a] ⋆ a = (a ⋆ i[a]) · a[a, a, a]
    using assms triangle [of a a] unitor-coincidence by auto
  show (i[a] ⋆ a) · a⁻¹[a, a, a] = a ⋆ i[a]
    using assms 1 invert-side-of-triangle(2) [of i[a] ⋆ a a ⋆ i[a] a[a, a, a]]
          assoc′-eq-inv-assoc
    by (metis hseqI′ iso-assoc objE obj-def′ unit-simps(1) unit-simps(2))
qed

lemma hcomp-assoc-isomorphic:
assumes ide f and ide g and ide h and src f = trg g and src g = trg h
shows (f ⋆ g) ⋆ h ≅ f ⋆ g ⋆ h
  using assms assoc-in-hom [of f g h] iso-assoc isomorphic-def by auto

lemma hcomp-arr-obj:
assumes arr μ and obj a and src μ = a
shows μ ⋆ a = r⁻¹[cod μ] · μ · r[dom μ]
and r[cod μ] · (μ ⋆ a) · r⁻¹[dom μ] = μ
proof −
  show μ ⋆ a = r⁻¹[cod μ] · μ · r[dom μ]
    using assms iso-runit runit-naturality comp-cod-arr
    by (metis ide-cod ide-dom invert-side-of-triangle(1) runit-simps(1) runit-simps(5) seqI)
  show r[cod μ] · (μ ⋆ a) · r⁻¹[dom μ] = μ
    using assms iso-runit runit-naturality [of μ] comp-cod-arr
    by (metis ide-dom invert-side-of-triangle(2) comp-assoc runit-simps(1)
        runit-simps(5) seqI)
qed

lemma hcomp-obj-arr:
assumes arr μ and obj b and b = trg μ
```

shows $b \star \mu = \mathrm{l}^{-1}[cod\ \mu] \cdot \mu \cdot \mathrm{l}[dom\ \mu]$
and $\mathrm{l}[cod\ \mu] \cdot (b \star \mu) \cdot \mathrm{l}^{-1}[dom\ \mu] = \mu$
proof −
  show $b \star \mu = \mathrm{l}^{-1}[cod\ \mu] \cdot \mu \cdot \mathrm{l}[dom\ \mu]$
    using *assms iso-lunit lunit-naturality comp-cod-arr*
    by (*metis ide-cod ide-dom invert-side-of-triangle*(*1*) *lunit-simps*(*1*) *lunit-simps*(*5*) *seqI*)
  show $\mathrm{l}[cod\ \mu] \cdot (b \star \mu) \cdot \mathrm{l}^{-1}[dom\ \mu] = \mu$
    using *assms iso-lunit lunit-naturality* [*of* μ] *comp-cod-arr*
    by (*metis ide-dom invert-side-of-triangle*(*2*) *comp-assoc lunit-simps*(*1*)
      lunit-simps(*5*) *seqI*)
qed

lemma *hcomp-reassoc*:
assumes $arr\ \tau$ **and** $arr\ \mu$ **and** $arr\ \nu$
and $src\ \tau = trg\ \mu$ **and** $src\ \mu = trg\ \nu$
shows $(\tau \star \mu) \star \nu = \mathrm{a}^{-1}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot (\tau \star \mu \star \nu) \cdot \mathrm{a}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
and $\tau \star \mu \star \nu = \mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot ((\tau \star \mu) \star \nu) \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
proof −
  show $(\tau \star \mu) \star \nu = \mathrm{a}^{-1}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot (\tau \star \mu \star \nu) \cdot \mathrm{a}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
  proof −
    have $(\tau \star \mu) \star \nu = (\mathrm{a}^{-1}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot \mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu]) \cdot ((\tau \star \mu) \star \nu)$
      using *assms comp-assoc-assoc′*(*2*) *comp-cod-arr* **by** *simp*
    also have ... $= \mathrm{a}^{-1}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot \mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot ((\tau \star \mu) \star \nu)$
      using *comp-assoc* **by** *simp*
    also have ... $= \mathrm{a}^{-1}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot (\tau \star \mu \star \nu) \cdot \mathrm{a}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
      using *assms assoc-naturality* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
  show $\tau \star \mu \star \nu = \mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot ((\tau \star \mu) \star \nu) \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
  proof −
    have $\tau \star \mu \star \nu = (\tau \star \mu \star \nu) \cdot \mathrm{a}[dom\ \tau,\ dom\ \mu,\ dom\ \nu] \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
      using *assms comp-assoc-assoc′*(*1*) *comp-arr-dom* **by** *simp*
    also have ... $= ((\tau \star \mu \star \nu) \cdot \mathrm{a}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]) \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
      using *comp-assoc* **by** *simp*
    also have ... $= (\mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot ((\tau \star \mu) \star \nu)) \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
      using *assms assoc-naturality* **by** *simp*
    also have ... $= \mathrm{a}[cod\ \tau,\ cod\ \mu,\ cod\ \nu] \cdot ((\tau \star \mu) \star \nu) \cdot \mathrm{a}^{-1}[dom\ \tau,\ dom\ \mu,\ dom\ \nu]$
      using *comp-assoc* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
qed

lemma *triangle′*:
assumes $ide\ f$ **and** $ide\ g$ **and** $src\ f = trg\ g$
shows $(f \star \mathrm{l}[g]) = (\mathrm{r}[f] \star g) \cdot \mathrm{a}^{-1}[f,\ src\ f,\ g]$
  using *assms*(*1*−*3*) *invert-side-of-triangle*(*2*) *triangle* **by** *force*

lemma *pentagon′*:
assumes $ide\ f$ **and** $ide\ g$ **and** $ide\ h$ **and** $ide\ k$

and *src f* = *trg g* **and** *src g* = *trg h* **and** *src h* = *trg k*
shows $((\mathrm{a}^{-1}[f,\ g,\ h] \star k) \cdot \mathrm{a}^{-1}[f,\ g \star h,\ k]) \cdot (f \star \mathrm{a}^{-1}[g,\ h,\ k])$
$= \mathrm{a}^{-1}[f \star g,\ h,\ k] \cdot \mathrm{a}^{-1}[f,\ g,\ h \star k]$
proof −
have $((\mathrm{a}^{-1}[f,\ g,\ h] \star k) \cdot \mathrm{a}^{-1}[f,\ g \star h,\ k]) \cdot (f \star \mathrm{a}^{-1}[g,\ h,\ k])$
$= inv\ ((f \star \mathrm{a}[g,\ h,\ k]) \cdot (\mathrm{a}[f,\ g \star h,\ k] \cdot (\mathrm{a}[f,\ g,\ h] \star k)))$
proof −
have $inv\ ((f \star \mathrm{a}[g,\ h,\ k]) \cdot (\mathrm{a}[f,\ g \star h,\ k] \cdot (\mathrm{a}[f,\ g,\ h] \star k))) =$
$inv\ (\mathrm{a}[f,\ g \star h,\ k] \cdot (\mathrm{a}[f,\ g,\ h] \star k)) \cdot inv\ (f \star \mathrm{a}[g,\ h,\ k])$
using *assms inv-comp* [*of* $\mathrm{a}[f,\ g \star h,\ k] \cdot (\mathrm{a}[f,\ g,\ h] \star k)\ f \star \mathrm{a}[g,\ h,\ k]$]
by *force*
also have ... $= (inv\ (\mathrm{a}[f,\ g,\ h] \star k) \cdot inv\ \mathrm{a}[f,\ g \star h,\ k]) \cdot inv\ (f \star \mathrm{a}[g,\ h,\ k])$
using *assms iso-assoc inv-comp* **by** *simp*
also have ... $= ((\mathrm{a}^{-1}[f,\ g,\ h] \star k) \cdot \mathrm{a}^{-1}[f,\ g \star h,\ k]) \cdot (f \star \mathrm{a}^{-1}[g,\ h,\ k])$
using *assms inv-hcomp* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= inv\ (\mathrm{a}[f,\ g,\ h \star k] \cdot \mathrm{a}[f \star g,\ h,\ k])$
using *assms pentagon* **by** *simp*
also have ... $= \mathrm{a}^{-1}[f \star g,\ h,\ k] \cdot \mathrm{a}^{-1}[f,\ g,\ h \star k]$
using *assms inv-comp* **by** *simp*
finally show *?thesis* **by** *auto*
qed

end

The following convenience locale extends *bicategory* by pre-interpreting the various functors and natural transformations.

locale *extended-bicategory* =
bicategory +
L: *equivalence-functor V V L* +
R: *equivalence-functor V V R* +
α: *natural-isomorphism VVV.comp V HoHV HoVH*
‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))› +
α': *inverse-transformation VVV.comp V HoHV HoVH*
‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))› +
$\mathfrak{l}$: *natural-isomorphism V V L map* $\mathfrak{l}$ +
$\mathfrak{l}'$: *inverse-transformation V V L map* $\mathfrak{l}$ +
$\mathfrak{r}$: *natural-isomorphism V V R map* $\mathfrak{r}$ +
$\mathfrak{r}'$: *inverse-transformation V V R map* $\mathfrak{r}$

sublocale *bicategory* ⊆ *extended-bicategory V H* a i *src trg*
proof −
interpret *L*: *equivalence-functor V V L* **using** *equivalence-functor-L* **by** *auto*
interpret *R*: *equivalence-functor V V R* **using** *equivalence-functor-R* **by** *auto*
interpret α': *inverse-transformation VVV.comp V HoHV HoVH*
‹$\lambda\mu\nu\tau$. a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))› **..**
interpret $\mathfrak{l}$: *natural-isomorphism V V L map* $\mathfrak{l}$ **using** *natural-isomorphism-*$\mathfrak{l}$ **by** *auto*
interpret $\mathfrak{l}'$: *inverse-transformation V V L map* $\mathfrak{l}$ **..**

interpret $\mathfrak{r}$: *natural-isomorphism V V R map* $\mathfrak{r}$ **using** *natural-isomorphism-*$\mathfrak{r}$ **by** *auto*
interpret $\mathfrak{r}'$: *inverse-transformation V V R map* $\mathfrak{r}$ **..**
interpret *extended-bicategory V H* a i *src trg* **..**
show *extended-bicategory V H* a i *src trg* **..**
qed

end

1.5 Concrete Bicategories

The locale *concrete-bicategory* defined in this section provides a uniform way to construct a bicategory from extrinsically specified data comprising: a set of *Obj* of "objects", a "hom-category" *Hom A B* for each pair of objects *A* and *B*, an "identity arrow" *Id A* $\in$ *Hom A A* for each object *A*, "horizontal composition" functors *Comp C B A* : *Hom B C* $\times$ *Hom A B* $\rightarrow$ *Hom A C* indexed by triples of objects, together with unit and associativity isomorphisms; the latter subject to naturality and coherence conditions. We show that the bicategory produced by the construction relates to the given data in the expected fashion: the objects of the bicategory are in bijective correspondence with the given set *Obj*, the hom-categories of the bicategory are isomorphic to the given categories *Hom A B*, the horizontal composition of the bicategory agrees with the given compositions *Comp C B A*, and the unit and associativity 2-cells of the bicategory are directly defined in terms of the given unit and associativity isomorphisms.

theory *ConcreteBicategory*
imports *Bicategory.Bicategory*
begin

locale *concrete-bicategory* =
fixes *Obj* :: $'o$ *set*
and *Hom* :: $'o \Rightarrow 'o \Rightarrow 'a$ *comp*
and *Id* :: $'o \Rightarrow 'a$
and *Comp* :: $'o \Rightarrow 'o \Rightarrow 'o \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
and *Unit* :: $'o \Rightarrow 'a$
and *Assoc* :: $'o \Rightarrow 'o \Rightarrow 'o \Rightarrow 'o \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
assumes *category-Hom*: $[\![A \in Obj; B \in Obj]\!] \Longrightarrow category\ (Hom\ A\ B)$
and *binary-functor-Comp*:
$[\![A \in Obj; B \in Obj; C \in Obj]\!]$
$\Longrightarrow binary\text{-}functor\ (Hom\ B\ C)\ (Hom\ A\ B)\ (Hom\ A\ C)\ (\lambda(f, g).\ Comp\ C\ B\ A\ f\ g)$
and *ide-Id*: $A \in Obj \Longrightarrow partial\text{-}composition.ide\ (Hom\ A\ A)\ (Id\ A)$
and *Unit-in-hom*:
$A \in Obj \Longrightarrow$
$partial\text{-}composition.in\text{-}hom\ (Hom\ A\ A)\ (Unit\ A)\ (Comp\ A\ A\ A\ (Id\ A)\ (Id\ A))\ (Id\ A)$
and *iso-Unit*: $A \in Obj \Longrightarrow category.iso\ (Hom\ A\ A)\ (Unit\ A)$
and *natural-isomorphism-Assoc*:
$[\![A \in Obj; B \in Obj; C \in Obj; D \in Obj]\!]$
$\Longrightarrow natural\text{-}isomorphism$
$(product\text{-}category.comp$
$(Hom\ C\ D)\ (product\text{-}category.comp\ (Hom\ B\ C)\ (Hom\ A\ B)))\ (Hom\ A\ D)$

$(\lambda(f,\ g,\ h).\ Comp\ D\ B\ A\ (Comp\ D\ C\ B\ f\ g)\ h)$
$(\lambda(f,\ g,\ h).\ Comp\ D\ C\ A\ f\ (Comp\ C\ B\ A\ g\ h))$
$(\lambda(f,\ g,\ h).\ Assoc\ D\ C\ B\ A\ f\ g\ h)$

and *left-unit-Id*:
$\bigwedge A\ B.\ [\![\ A \in Obj;\ B \in Obj\ ]\!]$
$\Longrightarrow$ *fully-faithful-functor* $(Hom\ A\ B)\ (Hom\ A\ B)$
$(\lambda f.$ *if partial-composition.arr* $(Hom\ A\ B)\ f$
then $Comp\ B\ B\ A\ (Id\ B)\ f$
else partial-magma.null $(Hom\ A\ B))$

and *right-unit-Id*:
$\bigwedge A\ B.\ [\![\ A \in Obj;\ B \in Obj\ ]\!]$
$\Longrightarrow$ *fully-faithful-functor* $(Hom\ A\ B)\ (Hom\ A\ B)$
$(\lambda f.$ *if partial-composition.arr* $(Hom\ A\ B)\ f$
then $Comp\ B\ A\ A\ f\ (Id\ A)$
else partial-magma.null $(Hom\ A\ B))$

and *pentagon*:
$\bigwedge A\ B\ C\ D\ E\ f\ g\ h\ k.$
$[\![\ A \in Obj;\ B \in Obj;\ C \in Obj;\ D \in Obj;\ E \in Obj;$
partial-composition.ide $(Hom\ D\ E)\ f$; *partial-composition.ide* $(Hom\ C\ D)\ g$;
partial-composition.ide $(Hom\ B\ C)\ h$; *partial-composition.ide* $(Hom\ A\ B)\ k\ ]\!] \Longrightarrow$
$Hom\ A\ E\ (Comp\ E\ D\ A\ f\ (Assoc\ D\ C\ B\ A\ g\ h\ k))$
$(Hom\ A\ E\ (Assoc\ E\ D\ B\ A\ f\ (Comp\ D\ C\ B\ g\ h)\ k)$
$(Comp\ E\ B\ A\ (Assoc\ E\ D\ C\ B\ f\ g\ h)\ k)) =$
$Hom\ A\ E\ (Assoc\ E\ D\ C\ A\ f\ g\ (Comp\ C\ B\ A\ h\ k))$
$(Assoc\ E\ C\ B\ A\ (Comp\ E\ D\ C\ f\ g)\ h\ k)$

begin

We first construct the vertical category. Arrows are terms of the form *MkCell A B* μ, where $A \in Obj$, $B \in Obj$, and where μ is an arrow of *Hom A B*. Composition requires agreement of the "source" A and "target" B components, and is then defined in terms of composition within *Hom A B*.

```
datatype ('oo, 'aa) cell =
  Null
| MkCell 'oo 'oo 'aa

abbreviation MkObj :: 'o ⇒ ('o, 'a) cell
where MkObj A ≡ MkCell A A (Id A)

fun Src :: ('o, 'a) cell ⇒ 'o
where Src (MkCell A - -) = A
    | Src - = undefined

fun Trg
where Trg (MkCell - B -) = B
    | Trg - = undefined

fun Map
where Map (MkCell - - F) = F
    | Map - = undefined
```

abbreviation *Cell*
where $Cell\ \mu \equiv \mu \neq Null \wedge Src\ \mu \in Obj \wedge Trg\ \mu \in Obj \wedge$
$partial\text{-}composition.arr\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu)$

definition *vcomp*
where $vcomp\ \mu\ \nu \equiv if\ Cell\ \mu \wedge Cell\ \nu \wedge Src\ \mu = Src\ \nu \wedge Trg\ \mu = Trg\ \nu \wedge$
$partial\text{-}composition.seq\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu)\ (Map\ \nu)$
$then\ MkCell\ (Src\ \mu)\ (Trg\ \mu)\ (Hom\ (Src\ \mu)\ (Trg\ \mu)\ (Map\ \mu)\ (Map\ \nu))$
$else\ Null$

interpretation *partial-composition vcomp*
by (*metis partial-composition-def partial-magma-def vcomp-def*)

lemma *null-char*:
shows $null = Null$
using *null-is-zero*(*1*) *vcomp-def* **by** *metis*

lemma *MkCell-Map*:
assumes $\mu \neq null$
shows $\mu = MkCell\ (Src\ \mu)\ (Trg\ \mu)\ (Map\ \mu)$
using *assms null-char*
by (*metis Map.simps*(*1*) *Src.elims Trg.simps*(*1*))

lemma *ide-char''*:
shows $ide\ \mu \longleftrightarrow Cell\ \mu \wedge partial\text{-}composition.ide\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu)$
proof
show $ide\ \mu \Longrightarrow Cell\ \mu \wedge partial\text{-}composition.ide\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu)$
proof
assume μ: $ide\ \mu$
show *1*: $Cell\ \mu$
by (*metis* μ *ide-def vcomp-def*)
interpret *Hom*: $category\ Hom\ (Src\ \mu)\ (Trg\ \mu)$
using *1 category-Hom* **by** *simp*
let $?\nu = MkCell\ (Src\ \mu)\ (Trg\ \mu)\ (Hom.dom\ (Map\ \mu))$
have $vcomp\ \mu\ ?\nu = MkCell\ (Src\ \mu)\ (Trg\ \mu)\ (Map\ \mu)$
using *1 vcomp-def Hom.comp-arr-dom* **by** *simp*
moreover have $vcomp\ \mu\ ?\nu = ?\nu$
using μ *ide-def null-char*
by (*metis MkCell-Map calculation*)
ultimately show $Hom.ide\ (Map\ \mu)$
using *1 Hom.ide-dom* **by** *fastforce*
qed
show $Cell\ \mu \wedge partial\text{-}composition.ide\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu) \Longrightarrow ide\ \mu$
proof −
assume μ: $Cell\ \mu \wedge partial\text{-}composition.ide\ (Hom\ (Src\ \mu)\ (Trg\ \mu))\ (Map\ \mu)$
interpret *Hom*: $category\ Hom\ (Src\ \mu)\ (Trg\ \mu)$
using μ *category-Hom* **by** *simp*

```
    show ide μ
    proof −
      have vcomp μ μ ≠ null
        using μ vcomp-def null-char by simp
      moreover have ⋀ν. vcomp ν μ ≠ null ⟹ vcomp ν μ = ν
        by (metis (full-types) Hom.comp-arr-ide μ MkCell-Map vcomp-def null-char)
      moreover have ⋀ν. vcomp μ ν ≠ null ⟹ vcomp μ ν = ν
        by (metis Hom.comp-ide-arr MkCell-Map μ null-char vcomp-def)
      ultimately show ?thesis
        unfolding ide-def by simp
    qed
  qed
qed

lemma MkCell-in-domains:
assumes Cell μ
shows MkCell (Src μ) (Trg μ) (partial-composition.dom (Hom (Src μ) (Trg μ)) (Map μ))
        ∈ domains μ
proof −
  interpret Hom: category Hom (Src μ) (Trg μ)
    using assms category-Hom by simp
  let ?ν = MkCell (Src μ) (Trg μ) (Hom.dom (Map μ))
  have ide ?ν
    using assms ide-char'' Hom.arr-dom Hom.ide-dom by simp
  moreover have vcomp μ ?ν = μ
    unfolding vcomp-def
    using assms Hom.comp-arr-dom MkCell-Map null-char by auto
  ultimately show ?thesis
      using domains-def by (simp add: assms null-char)
qed

lemma MkCell-in-codomains:
assumes Cell μ
shows MkCell (Src μ) (Trg μ) (partial-composition.cod (Hom (Src μ) (Trg μ)) (Map μ))
        ∈ codomains μ
proof −
  interpret Hom: category Hom (Src μ) (Trg μ)
    using assms category-Hom by simp
  let ?ν = MkCell (Src μ) (Trg μ) (Hom.cod (Map μ))
  have ide ?ν
    using assms ide-char'' Hom.arr-dom Hom.ide-dom by simp
  moreover have vcomp ?ν μ = μ
    unfolding vcomp-def
    using assms Hom.comp-cod-arr MkCell-Map null-char by auto
  ultimately show ?thesis
      using codomains-def by (simp add: assms null-char)
qed

lemma has-domain-char:
```

shows *domains* $\mu \neq \{\} \longleftrightarrow$ *Cell* μ
proof −
have $\neg$ *Cell* $\mu \Longrightarrow$ *domains* $\mu = \{\}$
using *vcomp-def domains-def null-char* **by** *auto*
moreover have
Cell $\mu \Longrightarrow$ *MkCell* (*Src* μ) (*Trg* μ) (*partial-composition.dom* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ))
$\in$ *domains* μ
using *MkCell-in-domains* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *has-codomain-char*:
shows *codomains* $\mu \neq \{\} \longleftrightarrow$ *Cell* μ
proof −
have $\neg$ *Cell* $\mu \Longrightarrow$ *codomains* $\mu = \{\}$
using *vcomp-def codomains-def null-char* **by** *auto*
moreover have
Cell $\mu \Longrightarrow$ *MkCell* (*Src* μ) (*Trg* μ) (*partial-composition.cod* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ))
$\in$ *codomains* μ
using *MkCell-in-codomains* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *arr-char*:
shows *arr* $\mu \longleftrightarrow$ *Cell* μ
using *arr-def has-domain-char has-codomain-char* **by** *simp*

lemma *ide-char'''*:
shows *ide* $\mu \longleftrightarrow$ *arr* $\mu \wedge$ *partial-composition.ide* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
using *ide-char''* *arr-char* **by** *simp*

lemma *seq-char*:
shows *seq* $\mu\ \nu \longleftrightarrow$ *Cell* $\mu \wedge$ *Cell* $\nu \wedge$ *Src* $\mu =$ *Src* $\nu \wedge$ *Trg* $\mu =$ *Trg* $\nu \wedge$
partial-composition.seq (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ) (*Map* ν)
using *arr-char vcomp-def* **by** *auto*

lemma *vcomp-char*:
shows *vcomp* $\mu\ \nu =$ (*if seq* $\mu\ \nu$ *then*
MkCell (*Src* μ) (*Trg* μ) (*Hom* (*Src* μ) (*Trg* μ) (*Map* μ) (*Map* ν))
else null)
by (*metis null-char seq-char vcomp-def*)

interpretation *category vcomp*
proof
show $\bigwedge g\ f.$ *vcomp* $g\ f \neq$ *null* $\Longrightarrow$ *seq* $g\ f$
using *seq-char null-char vcomp-def* **by** *metis*
show $\bigwedge f.$ (*domains* $f \neq \{\}$) = (*codomains* $f \neq \{\}$)

```
      using has-domain-char has-codomain-char by simp
    show ⋀h g f. ⟦seq h g; seq (vcomp h g) f⟧ ⟹ seq g f
      using vcomp-def
      apply (unfold seq-char, intro conjI)
              apply auto
      by (meson category.match-1 category-Hom)
    show ⋀h g f. ⟦seq h (vcomp g f); seq g f⟧ ⟹ seq h g
      using vcomp-def
      apply (unfold seq-char, intro conjI)
              apply auto
      by (meson category.match-2 category-Hom)
    show ⋀g f h. ⟦seq g f; seq h g⟧ ⟹ seq (vcomp h g) f
      using vcomp-def
      apply (unfold seq-char, intro conjI)
              apply auto
      by (meson category.match-3 category-Hom)
    show ⋀g f h. ⟦seq g f; seq h g⟧ ⟹ vcomp (vcomp h g) f = vcomp h (vcomp g f)
    proof −
      fix f g h
      assume fg: seq g f and gh: seq h g
      interpret Hom: category ‹Hom (Src f) (Trg f)›
        using fg seq-char category-Hom by simp
      have vcomp (vcomp h g) f =
            MkCell (Src f) (Trg f)
                          (Hom (Src f) (Trg f)
                               (Hom (Src f) (Trg f) (Map h) (Map g)) (Map f))
        using fg gh vcomp-char seq-char null-char Hom.match-3 by auto
      also have ... =
            MkCell (Src f) (Trg f)
                  (Hom (Src f) (Trg f) (Map h)
                       (Hom (Src f) (Trg f) (Map g) (Map f)))
        using fg gh seq-char Hom.comp-assoc by simp
      also have ... = vcomp h (vcomp g f)
        using fg gh vcomp-char seq-char null-char Hom.match-4 by auto
      finally show vcomp (vcomp h g) f = vcomp h (vcomp g f)
        by blast
    qed
  qed

  lemma arr-eqI:
  assumes arr f and arr f′
  and Src f = Src f′ and Trg f = Trg f′ and Map f = Map f′
  shows f = f′
    using arr-char MkCell-Map assms null-char by metis

  lemma dom-char:
  shows dom μ = (if arr μ then
                MkCell (Src μ) (Trg μ) (partial-composition.dom (Hom (Src μ) (Trg μ)) (Map
μ))
```

else Null)
by (*metis MkCell-in-domains arr-char dom-in-domains domain-unique has-domain-iff-arr dom-def null-char*)

lemma *cod-char*:
shows *cod* μ = (*if arr* μ *then*
MkCell (*Src* μ) (*Trg* μ) (*partial-composition.cod* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ))
else Null)
by (*metis MkCell-in-codomains arr-char cod-def cod-in-codomains codomain-unique has-codomain-iff-arr null-char*)

lemma *Src-vcomp* [*simp*]:
assumes *seq* μ ν
shows *Src* (*vcomp* μ ν) = *Src* μ
using *assms seq-char vcomp-def* **by** *auto*

lemma *Trg-vcomp* [*simp*]:
assumes *seq* μ ν
shows *Trg* (*vcomp* μ ν) = *Trg* μ
using *assms seq-char vcomp-def* **by** *auto*

lemma *Map-vcomp* [*simp*]:
assumes *seq* μ ν
shows *Map* (*vcomp* μ ν) = *Hom* (*Src* μ) (*Trg* μ) (*Map* μ) (*Map* ν)
using *assms seq-char vcomp-def* **by** *auto*

lemma *arr-MkCell* [*simp*]:
assumes $A \in$ *Obj* **and** $B \in$ *Obj* **and** *partial-composition.arr* (*Hom A B*) *f*
shows *arr* (*MkCell A B f*)
using *assms arr-char* **by** *simp*

lemma *dom-MkCell* [*simp*]:
assumes *arr* (*MkCell A B f*)
shows *dom* (*MkCell A B f*) = *MkCell A B* (*partial-composition.dom* (*Hom A B*) *f*)
using *assms arr-char dom-char* **by** *simp*

lemma *cod-MkCell* [*simp*]:
assumes *arr* (*MkCell A B f*)
shows *cod* (*MkCell A B f*) = *MkCell A B* (*partial-composition.cod* (*Hom A B*) *f*)
using *assms arr-char cod-char* **by** *simp*

lemma *iso-char*:
shows *iso* $\mu \longleftrightarrow$ *arr* $\mu \wedge$ *category.iso* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
proof
assume μ: *iso* μ
have *1*: *arr* μ **using** μ **by** *blast*
interpret *Hom*: *category* ‹*Hom* (*Src* μ) (*Trg* μ)›
using *1 arr-char category-Hom* **by** *simp*

```
    have 2: Hom.iso (Map μ)
    proof −
      obtain ν where ν: inverse-arrows μ ν
        using μ by blast
      have Hom.inverse-arrows (Map μ) (Map ν)
      proof
        show Hom.ide (Hom (Src μ) (Trg μ) (Map μ) (Map ν))
          using ν ide-char'' Src-vcomp Trg-vcomp ideD(1) vcomp-char Map-vcomp
          by (metis inverse-arrowsE)
        show Hom.ide (Hom (Src μ) (Trg μ) (Map ν) (Map μ))
        proof −
          have 1: ide (vcomp ν μ)
            using ν by auto
          hence Hom.ide (Map (vcomp ν μ))
            using ide-char'' Src-vcomp Trg-vcomp ideD(1) seq-char by metis
          thus Hom.ide (Hom (Src μ) (Trg μ) (Map ν) (Map μ))
            using ν 1 vcomp-char Map.simps(1) seq-char ideD(1)
            by (metis (no-types, lifting))
        qed
      qed
      thus ?thesis by auto
    qed
    show arr μ ∧ Hom.iso (Map μ)
      using 1 2 by simp
    next
    assume μ: arr μ ∧ category.iso (Hom (Src μ) (Trg μ)) (Map μ)
    interpret Hom: category ‹Hom (Src μ) (Trg μ)›
      using μ arr-char category-Hom by simp
    obtain f where f: Hom.inverse-arrows (Map μ) f
      using μ by auto
    let ?ν = MkCell (Src μ) (Trg μ) f
    have 1: arr ?ν
      using μ f arr-char by auto
    have inverse-arrows μ (MkCell (Src μ) (Trg μ) f)
      using μ f 1 arr-char ide-char'' vcomp-def
      apply (intro inverse-arrowsI) by auto
    thus iso μ by auto
  qed
```

Next, we equip each arrow with a source and a target, and show that these assignments are functorial.

```
  definition src
  where src μ ≡ if arr μ then MkObj (Src μ) else null

  definition trg
  where trg μ ≡ if arr μ then MkObj (Trg μ) else null

  lemma src-MkCell [simp]:
  assumes arr (MkCell A B f)
```

shows *src* (*MkCell A B f*) = *MkObj A*
  using *assms src-def* **by** *simp*

lemma *trg-MkCell* [*simp*]:
assumes *arr* (*MkCell A B f*)
shows *trg* (*MkCell A B f*) = *MkObj B*
  using *assms trg-def* **by** *simp*

lemma *src-dom*:
assumes *arr* μ
shows *src* (*dom* μ) = *src* μ
  using *assms dom-char src-def arr-char arr-dom* **by** *auto*

lemma *src-cod*:
assumes *arr* μ
shows *src* (*cod* μ) = *src* μ
  using *assms cod-char src-def arr-char arr-cod* **by** *auto*

lemma *trg-dom*:
assumes *arr* μ
shows *trg* (*dom* μ) = *trg* μ
  using *assms dom-char trg-def arr-char arr-dom* **by** *auto*

lemma *trg-cod*:
assumes *arr* μ
shows *trg* (*cod* μ) = *trg* μ
  using *assms cod-char trg-def arr-char arr-cod* **by** *auto*

lemma *Src-src* [*simp*]:
assumes *arr* μ
shows *Src* (*src* μ) = *Src* μ
  using *assms src-def* **by** *simp*

lemma *Trg-src* [*simp*]:
assumes *arr* μ
shows *Trg* (*src* μ) = *Src* μ
  using *assms src-def* **by** *simp*

lemma *Map-src* [*simp*]:
assumes *arr* μ
shows *Map* (*src* μ) = *Id* (*Src* μ)
  using *assms src-def* **by** *simp*

lemma *Src-trg* [*simp*]:
assumes *arr* μ
shows *Src* (*trg* μ) = *Trg* μ
  using *assms trg-def* **by** *simp*

lemma *Trg-trg* [*simp*]:

assumes *arr* μ
shows *Trg* (*trg* μ) = *Trg* μ
  using *assms trg-def* **by** *simp*

lemma *Map-trg* [*simp*]:
assumes *arr* μ
shows *Map* (*trg* μ) = *Id* (*Trg* μ)
  using *assms trg-def* **by** *simp*

lemma *Src-dom* [*simp*]:
assumes *arr* μ
shows *Src* (*dom* μ) = *Src* μ
  using *assms dom-char src-def arr-char arr-dom* **by** *auto*

lemma *Src-cod* [*simp*]:
assumes *arr* μ
shows *Src* (*cod* μ) = *Src* μ
  using *assms src-cod src-def arr-char arr-cod* **by** *auto*

lemma *Trg-dom* [*simp*]:
assumes *arr* μ
shows *Trg* (*dom* μ) = *Trg* μ
  using *assms dom-char trg-def arr-char arr-dom* **by** *auto*

lemma *Trg-cod* [*simp*]:
assumes *arr* μ
shows *Trg* (*cod* μ) = *Trg* μ
  using *assms cod-char trg-def arr-char arr-cod* **by** *auto*

lemma *Map-dom* [*simp*]:
assumes *arr* μ
shows *Map* (*dom* μ) = *partial-composition.dom* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
  using *assms* **by** (*simp add*: *dom-char*)

lemma *Map-cod* [*simp*]:
assumes *arr* μ
shows *Map* (*cod* μ) = *partial-composition.cod* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
  using *assms* **by** (*simp add*: *cod-char*)

lemma *ide-MkObj*:
assumes $A \in$ *Obj*
shows *ide* (*MkObj A*)
  using *assms ide-char′*
  by (*metis Map.simps*(*1*) *Src.simps*(*1*) *Trg.simps*(*1*) *category.ideD*(*1*) *cell.simps*(*2*)
    category-Hom ide-char″ ide-Id)

interpretation *src*: *functor vcomp vcomp src*
  using *src-def arr-char Map.simps*(*1*) *Src.simps*(*1*) *Trg.simps*(*1*) *arr-dom category.ideD*(*1*)
    ide-MkObj src-dom src-cod ide-Id

```
  apply unfold-locales
      apply auto[1]
     apply (simp add: category.ideD(1) category-Hom)
    apply auto[2]
proof -
  fix g :: ('o, 'a) cell and f :: ('o, 'a) cell
  assume fg: seq g f
  thus src (vcomp g f) = vcomp (src g) (src f)
    using arr-char ide-MkObj src-dom src-cod src-def
    by (metis Src-vcomp comp-ide-self seqE)
qed

interpretation trg: functor vcomp vcomp trg
  using trg-def arr-char Map.simps(1) Src.simps(1) Trg.simps(1) arr-dom category.ideD(1)
        ide-MkObj trg-dom trg-cod ide-Id
  apply unfold-locales
      apply auto[1]
     apply (simp add: category.ideD(1) category-Hom)
    apply auto[2]
proof -
  fix g :: ('o, 'a) cell and f :: ('o, 'a) cell
  assume fg: seq g f
  thus trg (vcomp g f) = vcomp (trg g) (trg f)
    using arr-char ide-MkObj trg-dom trg-cod trg-def
    by (metis Trg-vcomp comp-ide-self seqE)
qed

interpretation H: horizontal-homs vcomp src trg
  using ide-MkObj arr-char src-def trg-def src.preserves-arr trg.preserves-arr
  by unfold-locales auto

lemma obj-MkObj:
assumes A ∈ Obj
shows H.obj (MkObj A)
  using assms src-def H.obj-def ide-MkObj by simp

lemma MkCell-in-hom [intro]:
assumes A ∈ Obj and B ∈ Obj and partial-composition.arr (Hom A B) f
shows H.in-hhom (MkCell A B f) (MkObj A) (MkObj B)
and «MkCell A B f : MkCell A B (partial-composition.dom (Hom A B) f)
                    ⇒ MkCell A B (partial-composition.cod (Hom A B) f)»
  using assms by auto
```

Horizontal composition of horizontally composable arrows is now defined by applying the given function *Comp* to the "Map" components.

```
definition hcomp
where hcomp μ ν ≡ if arr μ ∧ arr ν ∧ src μ = trg ν then
                    MkCell (Src ν) (Trg μ) (Comp (Trg μ) (Trg ν) (Src ν) (Map μ) (Map ν))
                  else
```

null

lemma *arr-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *arr* (*hcomp* μ ν)
and *dom* (*hcomp* μ ν) = *hcomp* (*dom* μ) (*dom* ν)
and *cod* (*hcomp* μ ν) = *hcomp* (*cod* μ) (*cod* ν)
proof −
  have *1*: *hcomp* μ ν =
      MkCell (*Src* ν) (*Trg* μ) (*Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν))
    using *assms hcomp-def* **by** *simp*
  have *2*: *Src* μ = *Trg* ν
    using *assms src-def trg-def* **by** *simp*
  interpret *Hom-*μ: *category* ‹*Hom* (*Src* μ) (*Trg* μ)›
    using *assms arr-char category-Hom* **by** *simp*
  interpret *Hom-*ν: *category* ‹*Hom* (*Src* ν) (*Trg* ν)›
    using *assms arr-char category-Hom* **by** *simp*
  interpret *Hom-*$\mu\nu$: *category* ‹*Hom* (*Src* ν) (*Trg* μ)›
    using *assms arr-char category-Hom* **by** *simp*
  interpret *Comp*: *binary-functor*
      ‹*Hom* (*Trg* ν) (*Trg* μ)› ‹*Hom* (*Src* ν) (*Trg* ν)› ‹*Hom* (*Src* ν) (*Trg* μ)›
      ‹λ(*f*, *g*). *Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) *f g*›
    using *assms arr-char 2 binary-functor-Comp* [*of Src* ν *Trg* ν *Trg* μ]
    by *simp*
  have *4*: *Comp.A1xA2.arr* (*Map* μ, *Map* ν)
    using *assms 2 arr-char Comp.A1xA2.arr-char* **by** *simp*
  show *3*: *arr* (*hcomp* μ ν)
    using *assms 1 2 4 arr-char Comp.preserves-arr* [*of* (*Map* μ, *Map* ν)]
    by *simp*
  show *dom* (*hcomp* μ ν) = *hcomp* (*dom* μ) (*dom* ν)
  proof −
    have *dom* (*hcomp* μ ν) =
        MkCell (*Src* ν) (*Trg* μ)
          (*Hom-*$\mu\nu$*.dom* (*Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν)))
      using *1 3 dom-char* **by** *simp*
    moreover have *Hom-*$\mu\nu$*.dom* (*Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν)) =
        Comp (*Trg* μ) (*Trg* ν) (*Src* ν) (*Hom-*μ*.dom* (*Map* μ)) (*Hom-*ν*.dom* (*Map* ν))
      using *2 Comp.preserves-dom* ‹*Comp.A1xA2.arr* (*Map* μ, *Map* ν)› **by** *force*
    ultimately show *?thesis*
      using *assms 2 dom-char hcomp-def arr-dom*
      by *auto metis*
  qed
  show *cod* (*hcomp* μ ν) = *hcomp* (*cod* μ) (*cod* ν)
  proof −
    have *cod* (*hcomp* μ ν) =
        MkCell (*Src* ν) (*Trg* μ)
          (*Hom-*$\mu\nu$*.cod* (*Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν)))
      using *1 3 cod-char* **by** *simp*
    moreover have *Hom-*$\mu\nu$*.cod* (*Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν)) =

Comp (*Trg* μ) (*Trg* ν) (*Src* ν) (*Hom*-μ.*cod* (*Map* μ)) (*Hom*-ν.*cod* (*Map* ν))
using *2 Comp.preserves-cod* ‹*Comp.A1xA2.arr* (*Map* μ, *Map* ν)› **by** *force*
ultimately show *cod* (*hcomp* μ ν) = *hcomp* (*cod* μ) (*cod* ν)
using *assms 2 cod-char hcomp-def arr-cod*
by *auto metis*
qed
qed

lemma *src-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *src* (*hcomp* μ ν) = *src* ν
using *assms hcomp-def src-def arr-hcomp*(*1*) **by** *auto*

lemma *trg-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *trg* (*hcomp* μ ν) = *trg* μ
using *assms hcomp-def trg-def arr-hcomp*(*1*) **by** *auto*

lemma *Src-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Src* (*hcomp* μ ν) = *Src* ν
using *assms hcomp-def* **by** *simp*

lemma *Trg-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Trg* (*hcomp* μ ν) = *Trg* μ
using *assms hcomp-def* **by** *simp*

lemma *Map-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Map* (*hcomp* μ ν) = *Comp* (*Trg* μ) (*Trg* ν) (*Src* ν) (*Map* μ) (*Map* ν)
using *assms hcomp-def* **by** *simp*

lemma *hcomp-vcomp*:
assumes *H.VV.seq g f*
shows *hcomp* (*fst* (*H.VV.comp g f*)) (*snd* (*H.VV.comp g f*)) =
vcomp (*hcomp* (*fst g*) (*snd g*)) (*hcomp* (*fst f*) (*snd f*))
proof −
let *?f1* = *fst f* **and** *?f2* = *snd f* **and** *?g1* = *fst g* **and** *?g2* = *snd g*
have *1*: *Src ?f1* ∈ *Obj* ∧ *Trg ?f1* ∈ *Obj* ∧ *Src ?f2* ∈ *Obj* ∧ *Trg ?f2* ∈ *Obj* ∧
Src ?g1 ∈ *Obj* ∧ *Trg ?g1* ∈ *Obj* ∧ *Src ?g2* ∈ *Obj* ∧ *Trg ?g2* ∈ *Obj*
using *assms arr-char H.VV.arrE* **by** *blast*
interpret *Hom-f1*: *category* ‹*Hom* (*Src ?f1*) (*Trg ?f1*)›
using *assms 1 category-Hom* **by** *simp*
interpret *Hom-f2*: *category* ‹*Hom* (*Src ?f2*) (*Trg ?f2*)›
using *assms 1 category-Hom* **by** *simp*
interpret *Hom-g1*: *category* ‹*Hom* (*Src ?g1*) (*Trg ?g1*)›
using *assms 1 category-Hom* **by** *simp*
interpret *Hom-g2*: *category* ‹*Hom* (*Src ?g2*) (*Trg ?g2*)›

```
  using assms 1 category-Hom by simp
interpret Hom-f: category ‹Hom (Src ?f2) (Trg ?f1)›
  using assms 1 category-Hom by simp
interpret Hom-g: category ‹Hom (Src ?g2) (Trg ?g1)›
  using assms 1 category-Hom by simp
interpret Comp-f: binary-functor ‹Hom (Trg ?f2) (Trg ?f1)› ‹Hom (Src ?f2) (Trg ?f2)›
                ‹Hom (Src ?f2) (Trg ?f1)›
                ‹λ(fa, g). Comp (Trg ?f1) (Trg ?f2) (Src ?f2) fa g›
  using assms 1 arr-char binary-functor-Comp by simp

have hcomp (fst (H.VV.comp g f)) (snd (H.VV.comp g f)) =
     MkCell (Src (snd (H.VV.comp g f))) (Trg (fst (H.VV.comp g f)))
          (Comp (Trg (fst (H.VV.comp g f))) (Trg (snd (H.VV.comp g f)))
               (Src (snd (H.VV.comp g f)))
               (Map (fst (H.VV.comp g f))) (Map (snd (H.VV.comp g f))))
  using assms hcomp-def H.VV.arrE
  by (metis (no-types, lifting))
also have ... = MkCell (Src ?f2) (Trg ?f1)
                    (Hom (Src ?f2) (Trg ?f1)
                         (Comp (Trg ?f1) (Trg ?g2) (Src ?f2) (Map ?g1) (Map ?g2))
                         (Comp (Trg ?f1) (Trg ?f2) (Src ?f2) (Map ?f1) (Map ?f2)))
proof −
  have Src (snd (H.VV.comp g f)) = Src ?f2
    using assms arr-char src-def H.VV.comp-char H.VV.seq-char_SbC
    by (metis (no-types, lifting) H.vseq-implies-hpar(1) Src.simps(1) H.VV.arrE
        H.VV.inclusion H.VxV.comp-arr-dom H.VxV.dom-comp H.VxV.seqE_PC)
  moreover have Trg (fst (H.VV.comp g f)) = Trg ?f1
    by (metis (no-types, lifting) H.VV.comp-arr-dom H.VV.comp-simp H.VV.seq-char_SbC
        H.VxV.arr-char H.VxV.cod-comp H.VxV.comp-cod-arr H.VxV.seqE_PC
        H.vseq-implies-hpar(2) Src-trg assms)
  moreover have
      Comp (Trg (fst (H.VV.comp g f))) (Trg (snd (H.VV.comp g f)))
           (Src (snd (H.VV.comp g f)))
           (Map (fst (H.VV.comp g f))) (Map (snd (H.VV.comp g f))) =
       Hom (Src ?f2) (Trg ?f1)
          (Comp (Trg ?f1) (Trg ?g2) (Src ?f2) (Map ?g1) (Map ?g2))
          (Comp (Trg ?f1) (Trg ?f2) (Src ?f2) (Map ?f1) (Map ?f2))
  proof −
    have Comp (Trg (fst (H.VV.comp g f))) (Trg (snd (H.VV.comp g f)))
              (Src (snd (H.VV.comp g f)))
              (Map (fst (H.VV.comp g f))) (Map (snd (H.VV.comp g f))) =
         Comp (Trg ?g1) (Trg ?g2) (Src ?g2)
              (Map (vcomp ?g1 ?f1)) (Map (vcomp ?g2 ?f2))
      using assms H.VV.comp-char H.VV.arr-char_SbC H.VxV.comp-char by auto
    also have ... = Comp (Trg ?g1) (Trg ?g2) (Src ?g2)
                        (Hom (Src ?g1) (Trg ?g1) (Map ?g1) (Map ?f1))
                        (Hom (Src ?g2) (Trg ?g2) (Map ?g2) (Map ?f2))
      using assms H.VV.seq-char_SbC Map-vcomp H.VxV.seq-char by auto
    also have ... = Hom (Src ?f2) (Trg ?f1)
```

```
                    (Comp (Trg ?f1) (Trg ?g2) (Src ?f2)
                        (Map ?g1) (Map ?g2))
                    (Comp (Trg ?f1) (Trg ?f2) (Src ?f2)
                        (Map ?f1) (Map ?f2))
    proof -
      have 2: Src ?g1 = Trg ?g2
        using assms H.VV.arr-char_SbC [of g] src-def [of ?g1] trg-def [of ?g2] by auto
      have Comp (Trg ?f1) (Trg ?f2) (Src ?f2)
              (Hom (Trg ?g2) (Trg ?g1) (Map ?g1) (Map ?f1))
              (Hom (Src ?g2) (Trg ?g2) (Map ?g2) (Map ?f2)) =
          Hom (Src ?f2) (Trg ?f1)
            (Comp (Trg ?f1) (Trg ?f2) (Src ?f2)
                (Map ?g1) (Map ?g2))
            (Comp (Trg ?f1) (Trg ?f2) (Src ?f2)
                (Map ?f1) (Map ?f2))
      proof -
        have Comp-f.A1xA2.seq (Map ?g1, Map ?g2) (Map ?f1, Map ?f2)
          using assms 2 H.VV.seq-char_SbC
          by (metis (no-types, lifting) Comp-f.A1xA2.seq-char H.VxV.seqE_PC
              fst-conv seq-char snd-conv)
        moreover have
          Comp-f.A1xA2.comp (Map ?g1, Map ?g2) (Map ?f1, Map ?f2) =
           (Hom (Src ?g1) (Trg ?g1) (Map ?g1) (Map ?f1),
            Hom (Src ?g2) (Trg ?g2) (Map ?g2) (Map ?f2))
          using assms 2 H.VV.seq-char_SbC H.VxV.seqE_PC seq-char
          by (metis (no-types, lifting) Comp-f.A1xA2.comp-char fst-conv snd-conv)
        ultimately show ?thesis
          by (metis 2 Comp-f.as-nat-trans.preserves-comp-2 old.prod.case)
      qed
      thus ?thesis using assms 2 H.VV.seq-char_SbC H.VxV.seqE_PC seq-char
        by (metis (no-types, lifting))
    qed
    finally show ?thesis by blast
  qed
  ultimately show ?thesis by blast
qed
also have ... = vcomp (hcomp ?g1 ?g2) (hcomp ?f1 ?f2)
proof -
  have 2: Trg ?g1 = Trg ?f1 ∧ Src ?g2 = Src ?f2 ∧ Src ?f1 = Trg ?f2
    using assms seq-char H.VV.seq-char_SbC H.VxV.seqE_PC
    by (metis (no-types, lifting) H.VV.arr-char_SbC Src-src Src-trg)
  have Hom-f.seq (Comp (Trg ?f1) (Trg ?g2) (Src ?f2)
                    (Map ?g1) (Map ?g2))
                 (Comp (Trg ?f1) (Trg ?f2) (Src ?f2)
                    (Map ?f1) (Map ?f2))
 by (metis (no-types, lifting) 2 Comp-f.A1xA2.seqI_PC Comp-f.preserves-seq H.VV.seq-char_SbC
      H.VxV.seqE_PC arr-char assms case-prod-conv vcomp-def)
  moreover have ?f1 ≠ Null ∧ ?f2 ≠ Null ∧ src ?f1 = trg ?f2 ∧
                ?g1 ≠ Null ∧ ?g2 ≠ Null ∧ src ?g1 = trg ?g2
```

using *assms* $H.VV.arr\text{-}char_{SbC}$ *arr-char assms* **by** *blast*
moreover have *Hom-f1.arr* (*Map ?f1*) ∧ *Hom-f2.arr* (*Map ?f2*)
using *assms seq-char* $H.VV.arrE$ $H.VV.seqE$ *arr-char* **by** *fast*
moreover have *Hom-g1.arr* (*Map ?g1*) ∧ *Hom-g2.arr* (*Map ?g2*)
using *assms seq-char* $H.VV.arrE$ $H.VV.seqE$ *arr-char* **by** *meson*
ultimately show *?thesis*
using *1 2 arr-char hcomp-def vcomp-def* **by** *auto*
qed
finally show *?thesis* **by** *simp*
qed

interpretation *H*: *functor* $H.VV.comp$ *vcomp* ‹$\lambda\mu\nu$. *hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
using *hcomp-def arr-hcomp hcomp-vcomp* $H.VV.arr\text{-}char_{SbC}$ $H.VV.dom\text{-}char_{SbC}$ $H.VV.cod\text{-}char_{SbC}$
by *unfold-locales auto*

interpretation *H*: *horizontal-composition vcomp hcomp src trg*
using *src-hcomp trg-hcomp arr-char hcomp-def null-char*
by *unfold-locales auto*

lemma *Map-obj*:
assumes *H.obj a*
shows *Map a* = *Id* (*Src a*) **and** *Map a* = *Id* (*Trg a*)
using *assms H.obj-def Map-src Map-trg H.obj-simps*(*3*) **by** *metis+*

lemma *MkCell-simps*:
assumes *A* ∈ *Obj* **and** *B* ∈ *Obj* **and** *partial-composition.arr* (*Hom A B*) *f*
shows *arr* (*MkCell A B f*)
and *src* (*MkCell A B f*) = *MkObj A* **and** *trg* (*MkCell A B f*) = *MkObj B*
and *dom* (*MkCell A B f*) = *MkCell A B* (*partial-composition.dom* (*Hom A B*) *f*)
and *cod* (*MkCell A B f*) = *MkCell A B* (*partial-composition.cod* (*Hom A B*) *f*)
using *assms MkCell-in-hom* **by** *auto*

Next, define the associativities and show that they are the components of a natural isomorphism.

definition *assoc*
where *assoc f g h* ≡
if $H.VVV.ide$ (*f*, *g*, *h*) *then*
MkCell (*Src h*) (*Trg f*)
(*Assoc* (*Trg f*) (*Trg g*) (*Trg h*) (*Src h*) (*Map f*) (*Map g*) (*Map h*))
else null

lemma *assoc-in-hom* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows «*assoc f g h* : *hcomp* (*hcomp f g*) *h* ⇒ *hcomp f* (*hcomp g h*)»
proof −
let *?A* = *Src h* **and** *?B* = *Trg h* **and** *?C* = *Trg g* **and** *?D* = *Trg f*
interpret *Hom-f*: *category* ‹*Hom ?C ?D*›
using *assms ide-char arr-char dom-char cod-char category-Hom* **by** *simp*
interpret *Hom-g*: *category* ‹*Hom ?B ?C*›

```
  using assms ide-char arr-char dom-char cod-char category-Hom by simp
interpret Hom-h: category ‹Hom ?A ?B›
  using assms ide-char arr-char dom-char cod-char category-Hom by simp
interpret Hom-gh: product-category ‹Hom ?B ?C› ‹Hom ?A ?B› ..
interpret Hom-f-gh: product-category ‹Hom ?C ?D› Hom-gh.comp ..
interpret Comp-fg: binary-functor ‹Hom ?C ?D› ‹Hom ?B ?C› ‹Hom ?B ?D›
                ‹λ(fa, g). Comp ?D ?C ?B fa g›
  using assms arr-char ide-char'' binary-functor-Comp [of ?B ?C ?D] by simp
interpret α: natural-isomorphism Hom-f-gh.comp ‹Hom (Src h) (Trg f)›
             ‹λ(fa, ga, ha). Comp ?D ?B ?A (Comp ?D ?C ?B fa ga) ha›
             ‹λ(fa, ga, ha). Comp ?D ?C ?A fa (Comp ?C ?B ?A ga ha)›
             ‹λ(fa, ga, ha). Assoc ?D ?C ?B ?A fa ga ha›
  using assms ide-char arr-char natural-isomorphism-Assoc [of ?A ?B ?C ?D]
  by blast
show ?thesis
proof
  have 1: Src f = Trg g ∧ Src g = Trg h
    using assms src-def trg-def by simp
  have 2: Hom-f.ide (Map f) ∧ Hom-g.ide (Map g) ∧ Hom-h.ide (Map h)
    using assms 1 ide-char' [of f] arr-char [of f]
    by (simp add: ide-char'')
  show 3: arr (assoc f g h)
    unfolding assoc-def
 using assms arr-char ide-char'' H.VV.arr-char_SbC H.VVV.arr-char_SbC H.VVV.ide-char_SbC
        src-def trg-def α.preserves-reflects-arr [of (Map f, Map g, Map h)]
        Hom-f-gh.arr-char Hom-gh.arr-char
    by simp
  show dom (assoc f g h) = hcomp (hcomp f g) h
  proof −
    have  arr (MkCell ?B ?D (Comp ?D ?C ?B (Map f) (Map g)))
      by (metis assms(1−2,4) 1 Map-hcomp Src-hcomp Trg-hcomp arr-MkCell arr-char
          arr-hcomp(1) ideD(1))
    moreover have MkObj ?B = trg h
      using assms ide-char'' arr-char MkCell-Map null-char trg-MkCell by metis
    ultimately show ?thesis
      unfolding hcomp-def
      using assms 1 2 3 arr-char ide-char'' assoc-def dom-MkCell H.VV.arr-char_SbC
          H.VVV.arr-char_SbC H.VVV.ide-char_SbC src-MkCell trg-MkCell α.preserves-dom
      by force
  qed
  show cod (assoc f g h) = hcomp f (hcomp g h)
  proof −
    have trg g = MkObj ?C
      using assms ide-char'' arr-char MkCell-Map null-char trg-MkCell by metis
    thus ?thesis
      unfolding hcomp-def
      using assms 2 3 α.preserves-cod src-MkCell trg-MkCell H.hseqI' hcomp-def
            assms arr-char ide-char'' assoc-def cod-MkCell H.VV.arr-char_SbC
            H.VVV.arr-char_SbC H.VVV.ide-char_SbC
```

```
      by force
    qed
  qed
qed

lemma assoc-simps [simp]:
assumes ide f and ide g and ide h and src f = trg g and src g = trg h
shows arr (assoc f g h)
and dom (assoc f g h) = hcomp (hcomp f g) h
and cod (assoc f g h) = hcomp f (hcomp g h)
  using assms assoc-in-hom by auto

lemma assoc-simps' [simp]:
assumes ide f and ide g and ide h and src f = trg g and src g = trg h
shows src (assoc f g h) = src h
and trg (assoc f g h) = trg f
and Src (assoc f g h) = Src h
and Trg (assoc f g h) = Trg f
and Map (assoc f g h) = Assoc (Trg f) (Trg g) (Trg h) (Src h) (Map f) (Map g) (Map h)
proof −
  show src (assoc f g h) = src h
    using assms src-hcomp src-dom src-def src-MkCell assoc-simps(1) assoc-def
    by (metis (no-types, lifting) ideD(1) not-arr-null)
  show trg (assoc f g h) = trg f
    using assms trg-hcomp trg-hcomp trg-cod H.hseqI'
    by (metis assoc-simps(1,3) ideD(1))
  show Src (assoc f g h) = Src h
    using assms Src-hcomp Src-dom
    by (metis (no-types, lifting) Src.simps(1) arr-char assoc-def assoc-simps(1) null-char)
  show Trg (assoc f g h) = Trg f
    using assms Trg-hcomp Trg-dom
    by (metis (no-types, lifting) Trg.simps(1) arr-char assoc-def assoc-simps(1) null-char)
  show Map (assoc f g h) = Assoc (Trg f) (Trg g) (Trg h) (Src h) (Map f) (Map g) (Map h)
    using assms Map-hcomp Map-dom
    by (metis (no-types, lifting) Map.simps(1) arr-char assoc-def assoc-simps(1) null-char)
qed

lemma iso-assoc:
assumes ide f and ide g and ide h and src f = trg g and src g = trg h
shows iso (assoc f g h)
proof −
  let ?A = Src h and ?B = Trg h and ?C = Trg g and ?D = Trg f
  interpret Hom-f: category ‹Hom ?C ?D›
    using assms ide-char arr-char dom-char cod-char category-Hom by simp
  interpret Hom-g: category ‹Hom ?B ?C›
    using assms ide-char arr-char dom-char cod-char category-Hom by simp
  interpret Hom-h: category ‹Hom ?A ?B›
    using assms ide-char arr-char dom-char cod-char category-Hom by simp
  interpret Hom-fg: product-category ‹Hom ?C ?D› ‹Hom ?B ?C› ..
```

interpret *Hom-gh*: *product-category* ‹*Hom ?B ?C*› ‹*Hom ?A ?B*› **..**
interpret *Hom-f-gh*: *product-category* ‹*Hom ?C ?D*› *Hom-gh.comp* **..**
interpret α: *natural-isomorphism Hom-f-gh.comp* ‹*Hom* (*Src h*) (*Trg f*)›
‹λ(*fa*, *ga*, *ha*). *Comp ?D ?B ?A* (*Comp ?D ?C ?B fa ga*) *ha*›
‹λ(*fa*, *ga*, *ha*). *Comp ?D ?C ?A fa* (*Comp ?C ?B ?A ga ha*)›
‹λ(*fa*, *ga*, *ha*). *Assoc ?D ?C ?B ?A fa ga ha*›
using *assms ide-char arr-char natural-isomorphism-Assoc* [*of ?A ?B ?C ?D*]
by *blast*
show *?thesis*
using *assms* α*.components-are-iso* [*of* (*Map f*, *Map g*, *Map h*)]
iso-char $H.VV.arr\text{-}char_{SbC}$ $H.VVV.arr\text{-}char_{SbC}$ *H.VVV.ide-char ide-char''*
by (*simp add*: *src-def trg-def*)
qed

lemma *assoc-naturality*:
assumes *arr f* **and** *arr g* **and** *arr h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *vcomp* (*H.HoVH* (*f*, *g*, *h*)) (*assoc* (*dom f*) (*dom g*) (*dom h*)) =
vcomp (*assoc* (*cod f*) (*cod g*) (*cod h*)) (*H.HoHV* (*f*, *g*, *h*))
proof −
let *?A* = *Src h* **and** *?B* = *Trg h* **and** *?C* = *Trg g* **and** *?D* = *Trg f*
have *1*: *Src f* = *Trg g* ∧ *Src g* = *Trg h*
using *assms src-def trg-def* **by** *simp*
interpret *Hom-f*: *category* ‹*Hom ?C ?D*›
using *assms ide-char arr-char dom-char cod-char category-Hom* **by** *simp*
interpret *Hom-g*: *category* ‹*Hom ?B ?C*›
using *assms ide-char arr-char dom-char cod-char category-Hom* **by** *simp*
interpret *Hom-h*: *category* ‹*Hom ?A ?B*›
using *assms ide-char arr-char dom-char cod-char category-Hom* **by** *simp*
interpret *Hom-fg*: *product-category* ‹*Hom ?C ?D*› ‹*Hom ?B ?C*› **..**
interpret *Hom-gh*: *product-category* ‹*Hom ?B ?C*› ‹*Hom ?A ?B*› **..**
interpret *Hom-fg-h*: *product-category Hom-fg.comp* ‹*Hom ?A ?B*› **..**
interpret *Hom-f-gh*: *product-category* ‹*Hom ?C ?D*› *Hom-gh.comp* **..**
interpret *Hom*: *category* ‹*Hom ?A ?D*›
using *assms ide-char arr-char dom-char cod-char category-Hom* **by** *simp*
interpret *Comp-fg*: *binary-functor* ‹*Hom ?C ?D*› ‹*Hom ?B ?C*› ‹*Hom ?B ?D*›
‹λ(*f'*, *g*). *Comp ?D ?C ?B f' g*›
using *assms arr-char ide-char'' binary-functor-Comp* [*of ?B ?C ?D*] **by** *simp*
interpret *Comp-gh*: *binary-functor* ‹*Hom ?B ?C*› ‹*Hom ?A ?B*› ‹*Hom ?A ?C*›
‹λ(*f'*, *g*). *Comp ?C ?B ?A f' g*›
using *assms arr-char ide-char'' binary-functor-Comp* [*of ?A ?B ?C*] **by** *simp*
interpret *Comp-fg-h*: *binary-functor* ‹*Hom ?B ?D*› ‹*Hom ?A ?B*› ‹*Hom ?A ?D*›
‹λ(*f'*, *g*). *Comp ?D ?B ?A f' g*›
using *assms arr-char ide-char'' binary-functor-Comp* [*of ?A ?B ?D*] **by** *simp*
interpret *Comp-f-gh*: *binary-functor* ‹*Hom ?C ?D*› ‹*Hom ?A ?C*› ‹*Hom ?A ?D*›
‹λ(*f'*, *g*). *Comp ?D ?C ?A f' g*›
using *assms arr-char ide-char'' binary-functor-Comp* [*of ?A ?C ?D*] **by** *simp*
interpret α: *natural-isomorphism Hom-f-gh.comp* ‹*Hom ?A ?D*›
‹λ(*f'*, *g'*, *h'*). *Comp ?D ?B ?A* (*Comp ?D ?C ?B f' g'*) *h'*›
‹λ(*f'*, *g'*, *h'*). *Comp ?D ?C ?A f'* (*Comp ?C ?B ?A g' h'*)›

```
        ‹λ(f′, g′, h′). Assoc ?D ?C ?B ?A f′ g′ h′›
  using assms ide-char arr-char natural-isomorphism-Assoc [of ?A ?B ?C ?D]
  by blast

have ide-Map-dom:
      Hom-f.ide (Map (dom f)) ∧ Hom-g.ide (Map (dom g)) ∧ Hom-h.ide (Map (dom h))
  using assms 1 ide-char′′ arr-char by simp
have ide-Map-cod:
      Hom-f.ide (Map (cod f)) ∧ Hom-g.ide (Map (cod g)) ∧ Hom-h.ide (Map (cod h))
  using assms 1 ide-char′′ arr-char by simp

have vcomp (H.HoVH (f, g, h)) (assoc (dom f) (dom g) (dom h)) =
     vcomp (hcomp f (hcomp g h))
          (MkCell (Src (dom h)) (Trg (dom f))
                (Assoc (Trg (dom f)) (Trg (dom g)) (Trg (dom h)) (Src (dom h))
                      (Map (dom f)) (Map (dom g)) (Map (dom h))))
 using assms 1 H.HoVH-def H.VV.arr-char_SbC H.VVV.arr-char_SbC H.VVV.ide-char_SbC
       assoc-def assoc-in-hom
   by simp
also have ... = MkCell (Src (dom h)) (Trg (dom f))
                    (Hom (Src (dom h)) (Trg (dom f))
                        (Comp (Trg (dom f)) (Trg g) (Src (dom h)) (Map f)
                            (Comp (Trg g) (Trg h) (Src (dom h)) (Map g) (Map h)))
                        (Assoc (Trg (dom f)) (Trg (dom g)) (Trg (dom h)) (Src (dom h))
                            (Map (dom f)) (Map (dom g)) (Map (dom h))))
proof −
  have Hom.seq (Comp ?D ?C ?A (Map f) (Comp ?C ?B ?A (Map g) (Map h)))
            (Assoc ?D (Trg (dom g)) (Trg (dom h)) ?A
                (Map (dom f)) (Map (dom g)) (Map (dom h)))
  proof (intro Hom.seqI)
    show Hom.arr (Assoc ?D (Trg (dom g)) (Trg (dom h)) ?A
                    (Map (dom f)) (Map (dom g)) (Map (dom h)))
    proof −
      have Hom.arr (Assoc ?D (Trg (dom g)) (Trg (dom h)) ?A
                      (Map (dom f)) (Map (dom g)) (Map (dom h)))
        using assms 1 arr-char Trg-dom Src-dom ide-Map-dom
              α.preserves-reflects-arr [of (Map (dom f), Map (dom g), Map (dom h))]
        by simp
      thus ?thesis
        using assms 1 arr-char Src-dom Trg-dom assoc-simps(1−2) assoc-def
              H.VV.ide-char H.VV.arr-char_SbC H.VV.arr-char_SbC ide-Map-dom
        by simp
    qed
    show Hom.arr (Comp ?D ?C ?A (Map f) (Comp ?C ?B ?A (Map g) (Map h)))
    proof −
      have Hom.arr (Comp ?D ?C ?A (Map f) (Comp ?C ?B ?A (Map g) (Map h)))
        using assms 1 arr-char
              Comp-f-gh.preserves-reflects-arr [of (Map f, Comp ?C ?B ?A (Map g) (Map h))]
               Comp-gh.preserves-reflects-arr [of (Map g, Map h)] Src-dom Trg-dom
```

```
      by simp
    thus ?thesis
      using assms 1 arr-char Src-dom Trg-dom by simp
  qed
  show Hom.dom (Comp ?D ?C ?A (Map f) (Comp ?C ?B ?A (Map g) (Map h))) =
        Hom.cod (Assoc ?D (Trg (dom g)) (Trg (dom h)) ?A
                (Map (dom f)) (Map (dom g)) (Map (dom h)))
  proof −
    have Hom.cod (Assoc ?D (Trg (dom g)) (Trg (dom h)) ?A
                  (Map (dom f)) (Map (dom g)) (Map (dom h))) =
          Hom.cod (Map (assoc (dom f) (dom g) (dom h)))
      using Src-dom Trg-dom assms(1−5) assoc-simps′(5) ide-dom src-dom trg-dom
      by presburger
    also have ... = Comp ?D ?C ?A
                        (Map (dom f))
                        (Comp ?C ?B ?A (Map (dom g)) (Map (dom h)))
    proof −
      have dom f ≠ Null ∧ dom g ≠ Null ∧ dom h ≠ Null
        using arr-dom assms not-arr-null null-char by fastforce
      moreover have Hom.cod (Map (assoc (dom f) (dom g) (dom h))) =
                    Comp ?D ?C ?A
                        (Map (dom f))
                        (Comp ?C ?B ?A (Map (dom g)) (Map (dom h)))
        using assms assoc-simps′(5) ide-Map-dom
              α.preserves-cod [of (Map (dom f), Map (dom g), Map (dom h))]
        by simp
      ultimately show ?thesis
     using assms 1 arr-char assoc-def H.VV.ide-char H.VV.arr-char_SbC H.VV.arr-char_SbC
              Src-dom Trg-dom ide-Map-dom null-char assoc-simps′(5)
        by simp
    qed
    also have ... =
              Hom.dom (Comp ?D ?C ?A (Map f) (Comp ?C ?B ?A (Map g) (Map h)))
    proof −
      have arr (MkCell ?A ?C (Comp ?C ?B ?A (Map g) (Map h)))
        using assms 1 arr-char arr-MkCell
              Comp-gh.preserves-reflects-arr [of (Map g, Map h)]
        by simp
      thus ?thesis
        using assms arr-char 1 Map-dom
              Comp-f-gh.preserves-dom [of (Map f, Comp ?C ?B ?A (Map g) (Map h))]
              Comp-gh.preserves-dom [of (Map g, Map h)]
        by simp
    qed
    finally show ?thesis by argo
  qed
qed
thus ?thesis
  using assms(1−5) H.hseqI′ arr-char vcomp-def by auto
```

```
qed
also have ... = MkCell (Src h) (Trg f)
                    (Hom (Src h) (Trg f)
                        (Assoc (Trg f) (Trg (cod g)) (Trg (cod h)) (Src h)
                            (Map (cod f)) (Map (cod g)) (Map (cod h)))
                        (Comp (Trg f) (Trg h) (Src h)
                            (Comp (Trg f) (Trg g) (Src g) (Map f) (Map g)) (Map h)))
  using assms 1 arr-char α.naturality [of (Map f, Map g, Map h)] by simp
also have ... = vcomp (MkCell (Src (cod h)) (Trg (cod f))
                        (Assoc (Trg (cod f)) (Trg (cod g)) (Trg (cod h)) (Src (cod h))
                            (Map (cod f)) (Map (cod g)) (Map (cod h))))
                    (hcomp (hcomp f g) h)
proof −
  have arr (MkCell ?B ?D (Comp ?D ?C ?B (Map f) (Map g)))
    using assms 1 arr-char arr-MkCell
        Comp-fg.preserves-reflects-arr [of (Map f, Map g)]
    by simp
  moreover have
          Hom.arr (Comp ?D ?B ?A (Comp ?D ?C ?B (Map f) (Map g)) (Map h))
    using assms 1 arr-char Comp-fg.preserves-reflects-arr [of (Map f, Map g)]
        Comp-fg-h.preserves-reflects-arr [of (Comp ?D ?C ?B (Map f) (Map g), Map h)]
    by simp
  moreover have Hom.arr (Assoc ?D (Trg (cod g)) (Trg (cod h)) ?A
                            (Map (cod f)) (Map (cod g)) (Map (cod h)))
    using assms 1 arr-char Trg-cod ide-Map-cod
        α.preserves-reflects-arr [of (Map (cod f), Map (cod g), Map (cod h))]
    by simp
  moreover have Hom.seq (Assoc ?D (Trg (cod g)) (Trg (cod h)) ?A
                        (Map (cod f)) (Map (cod g)) (Map (cod h)))
                    (Comp ?D ?B ?A (Comp ?D ?C ?B (Map f) (Map g)) (Map h))
  proof (intro Hom.seqI)
    show Hom.arr (Comp ?D ?B ?A (Comp ?D ?C ?B (Map f) (Map g)) (Map h))
      using assms 1 arr-char Comp-fg.preserves-reflects-arr [of (Map f, Map g)]
          Comp-fg-h.preserves-reflects-arr [of (Comp ?D ?C ?B (Map f) (Map g), Map h)]
      by simp
    show Hom.arr (Assoc ?D (Trg (cod g)) (Trg (cod h)) ?A
                    (Map (cod f)) (Map (cod g)) (Map (cod h)))
      using assms 1 arr-char Trg-cod ide-Map-cod
          α.preserves-reflects-arr [of (Map (cod f), Map (cod g), Map (cod h))]
      by simp
    show Hom.dom (Assoc ?D (Trg (cod g)) (Trg (cod h)) ?A
                (Map (cod f)) (Map (cod g)) (Map (cod h))) =
        Hom.cod (Comp ?D ?B ?A (Comp ?D ?C ?B (Map f) (Map g)) (Map h))
    proof −
      have Hom.dom (Assoc ?D (Trg (cod g)) (Trg (cod h)) ?A
                    (Map (cod f)) (Map (cod g)) (Map (cod h))) =
            Hom.dom (Map (assoc (cod f) (cod g) (cod h)))
        using H.cod-trg Src-cod Trg-cod assms(1−5) assoc-simps′(5) ide-cod src-cod
            trg.preserves-cod
```

by *presburger*
also have ... = *Comp ?D ?B ?A*
(*Comp ?D ?C ?B* (*Map* (*cod f*)) (*Map* (*cod g*))) (*Map* (*cod h*))
proof −
have *cod f* ≠ *Null* ∧ *cod g* ≠ *Null* ∧ *cod h* ≠ *Null*
using *arr-cod assms not-arr-null null-char* **by** *fastforce*
moreover have *Hom.dom* (*Map* (*assoc* (*cod f*) (*cod g*) (*cod h*))) =
Comp ?D ?B ?A
(*Comp ?D ?C ?B* (*Map* (*cod f*)) (*Map* (*cod g*))) (*Map* (*cod h*))
using *assms assoc-simps′*(*5*) *ide-Map-cod*
α.preserves-dom [*of* (*Map* (*cod f*), *Map* (*cod g*), *Map* (*cod h*))]
by *simp*
ultimately show *?thesis*
using *assms 1 arr-char assoc-def H.VV.ide-char H.VV.arr-char$_{SbC}$ H.VV.arr-char$_{SbC}$*
Src-cod Trg-cod ide-Map-cod null-char assoc-simps′(*5*)
by *simp*
qed
also have ... =
Hom.cod (*Comp ?D ?B ?A* (*Comp ?D ?C ?B* (*Map f*) (*Map g*)) (*Map h*))
using *assms arr-char 1 Map-cod arr-MkCell*
Comp-fg-h.preserves-cod [*of* (*Comp ?D ?C ?B* (*Map f*) (*Map g*), *Map h*)]
Comp-fg.preserves-cod [*of* (*Map f*, *Map g*)]
Comp-fg.preserves-reflects-arr [*of* (*Map f*, *Map g*)]
by *simp*
finally show *?thesis* **by** *blast*
qed
qed
ultimately show *?thesis*
using *assms arr-char vcomp-def hcomp-def Src-cod Trg-cod*
H.VV.ide-char H.VV.arr-char$_{SbC}$ H.VV.arr-char$_{SbC}$ src-def trg-def
by *simp*
qed
also have ... = *vcomp* (*assoc* (*cod f*) (*cod g*) (*cod h*)) (*H.HoHV* (*f*, *g*, *h*))
using *assms 1 H.HoHV-def H.VV.arr-char$_{SbC}$ H.VVV.arr-char$_{SbC}$ H.VVV.ide-char$_{SbC}$*
assoc-def assoc-in-hom
by *simp*
finally show *vcomp* (*H.HoVH* (*f*, *g*, *h*)) (*assoc* (*dom f*) (*dom g*) (*dom h*)) =
vcomp (*assoc* (*cod f*) (*cod g*) (*cod h*)) (*H.HoHV* (*f*, *g*, *h*))
by *blast*
qed

interpretation α_0: *transformation-by-components H.VVV.comp vcomp H.HoHV H.HoVH*
‹*λ*(*f*, *g*, *h*). *assoc f g h*›
proof
fix *fgh*
show *H.VVV.ide fgh* ⟹
«*case fgh of* (*f*, *g*, *h*) ⇒ *assoc f g h* : *H.HoHV fgh* ⇒ *H.HoVH fgh*»
using *assoc-in-hom H.HoHV-def H.HoVH-def H.VV.arr-char$_{SbC}$ H.VVV.arr-char$_{SbC}$*
H.VVV.ide-char$_{SbC}$

by (*cases fgh*) *simp*
assume *fgh*: $H.VVV.arr\ fgh$
show *vcomp* (*case* $H.VVV.cod\ fgh$ *of* $(f,\ g,\ h) \Rightarrow assoc\ f\ g\ h$) ($H.HoHV\ fgh$) =
vcomp ($H.HoVH\ fgh$) (*case* $H.VVV.dom\ fgh$ *of* $(f,\ g,\ h) \Rightarrow assoc\ f\ g\ h$)
using *fgh assoc-simps* $H.HoHV$*-def assoc-naturality*
$H.VV.arr\text{-}char_{SbC}$ $H.VVV.arr\text{-}char_{SbC}$ $H.VVV.dom\text{-}char_{SbC}$ $H.VVV.cod\text{-}char_{SbC}$
by (*cases fgh*) *simp*
qed

definition a (‹a[-,-,-]›)
where a $f\ g\ h$ == $\alpha_0.map\ (f,\ g,\ h)$

lemma a-*simp-ide*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows a[f, g, h] =
MkCell (*Src h*) (*Trg f*)
(*Assoc* (*Trg f*) (*Trg g*) (*Trg h*) (*Src h*) (*Map f*) (*Map g*) (*Map h*))
using *assms* a-*def assoc-def assoc-simps′ MkCell-Map not-arr-null*
$\alpha_0.map\text{-}simp\text{-}ide$ $H.VVV.ide\text{-}char_{SbC}$ $H.VV.arr\text{-}char_{SbC}$ $H.VVV.arr\text{-}char_{SbC}$
by *simp*

interpretation α: *natural-isomorphism* $H.VVV.comp$ *vcomp* $H.HoHV$ $H.HoVH$
‹λfgh. a (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*))›
proof −
interpret α: *natural-isomorphism* $H.VVV.comp$ *vcomp* $H.HoHV$ $H.HoVH$ $\alpha_0.map$
using $\alpha_0.map\text{-}simp\text{-}ide$ *iso-assoc* $H.VVV.ide\text{-}char_{SbC}$ $H.VV.arr\text{-}char_{SbC}$ $H.VVV.arr\text{-}char_{SbC}$
by *unfold-locales auto*
show *natural-isomorphism* $H.VVV.comp$ *vcomp* $H.HoHV$ $H.HoVH$
(λfgh. a (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*)))
using α.*natural-isomorphism-axioms* a-*def* **by** *simp*
qed

What remains is to show that horizontal composition with source or target defines fully faithful functors.

interpretation *endofunctor vcomp* $H.L$
using $H.endofunctor\text{-}L$ **by** *auto*
interpretation *endofunctor vcomp* $H.R$
using $H.endofunctor\text{-}R$ **by** *auto*

interpretation R: *fully-faithful-functor vcomp vcomp* $H.R$
proof
show $\bigwedge f\ f'.\ [\![par\ f\ f';\ H.R\ f = H.R\ f']\!] \Longrightarrow f = f'$
proof −
fix $\mu\ \mu'$
assume *par*: $par\ \mu\ \mu'$
and *eq*: $H.R\ \mu = H.R\ \mu'$
have *1*: $Src\ \mu' = Src\ \mu \wedge Trg\ \mu' = Trg\ \mu$
using *par* **by** (*metis Src-dom Trg-dom*)
interpret *Hom*: *category* ‹*Hom* (*Src* μ) (*Trg* μ)›

```
    using par arr-char category-Hom by simp
  let ?R = λf. if Hom.arr f
               then Comp (Trg μ) (Src μ) (Src μ) f (Id (Src μ))
               else Hom.null
  interpret R: fully-faithful-functor ‹Hom (Src μ) (Trg μ)› ‹Hom (Src μ) (Trg μ)› ?R
    using par arr-char right-unit-Id by simp
  have Rμ: H.R μ = MkCell (Src μ) (Trg μ)
                     (Comp (Trg μ) (Src μ) (Src μ) (Map μ) (Id (Src μ)))
    unfolding hcomp-def
    using par src-def null-char H.trg-src src.preserves-arr by simp
  have Rμ': H.R μ' = MkCell (Src μ) (Trg μ)
                       (Comp (Trg μ) (Src μ) (Src μ) (Map μ') (Id (Src μ)))
    unfolding hcomp-def
    using par 1 src-def null-char H.trg-src src.preserves-arr by simp
  have Map μ = Map μ'
    using Rμ Rμ' eq par arr-char eq R.is-faithful
    by (metis 1 Map-cod Map-dom cell.inject)
  thus μ = μ'
    using 1 par MkCell-Map by (metis arr-char null-char)
qed
show ⋀f g ν. ⟦ide f; ide g; «ν : H.R f ⇒ H.R g»⟧ ⟹ ∃μ. «μ : f ⇒ g» ∧ H.R μ = ν
proof −
  fix f g ν
  assume f: ide f and g: ide g and ν: «ν : H.R f ⇒ H.R g»
  interpret Hom: category ‹Hom (Src ν) (Trg ν)›
    using ν arr-char category-Hom by auto
  let ?R = λf. if Hom.arr f
               then Comp (Trg ν) (Src ν) (Src ν) f (Id (Src ν))
               else Hom.null
  interpret R: fully-faithful-functor ‹Hom (Src ν) (Trg ν)› ‹Hom (Src ν) (Trg ν)› ?R
    using ν arr-char right-unit-Id by auto
  have 0: Src f = Src ν ∧ Trg f = Trg ν ∧ Src g = Src ν ∧ Trg g = Trg ν
  proof (intro conjI)
    show Trg f = Trg ν
      using f ν Trg-dom Trg-hcomp by fastforce
    show Trg g = Trg ν
      using g ν Trg-cod Trg-hcomp by fastforce
    show Src f = Src ν
    proof −
      have 1: arr f ∧ dom f = f ∧ cod f = f
        by (meson f ide-char)
      hence hcomp f (src f) = dom ν
        using ν by fastforce
      thus ?thesis
        using 1
        by (metis (no-types) H.trg-src Src.simps(1) Src-hcomp ν dom-char in-homE
            src.preserves-arr src-def)
    qed
    show Src g = Src ν
```

```
  proof −
    have 1: arr g ∧ dom g = g ∧ cod g = g
      by (meson g ide-char)
    hence hcomp g (src g) = cod ν
      using ν by fastforce
    thus ?thesis
      using 1
      by (metis (no-types) H.trg-src Src.simps(1) Src-hcomp ν cod-char in-homE
          src.preserves-arr src-def)
  qed
qed
have 1: Hom.in-hom (Map ν) (?R (Map f)) (?R (Map g))
proof
  show Hom.arr (Map ν)
    using ν arr-char by auto
  show Hom.dom (Map ν) = ?R (Map f)
  proof −
    have 1: arr f ∧ dom f = f ∧ cod f = f
      using f ide-char by blast
    have dom ν = MkCell (Src ν) (Trg ν) (Hom.dom (Map ν))
      by (meson ν dom-char in-homE)
    thus ?thesis
      using 1 H.trg-src ν arr-char hcomp-def src.preserves-arr src-def by fastforce
  qed
  show Hom.cod (Map ν) = ?R (Map g)
  proof −
    have 1: arr g ∧ dom g = g ∧ cod g = g
      using g ide-char by blast
    have cod ν = MkCell (Src ν) (Trg ν) (Hom.cod (Map ν))
      by (meson ν cod-char in-homE)
    thus ?thesis
      using 1 H.trg-src ν arr-char hcomp-def src.preserves-arr src-def by fastforce
  qed
qed
have 2: Hom.ide (Map f) ∧ Hom.ide (Map g)
  using 0 f g arr-char ide-char'' by simp
obtain x where x: Hom.in-hom x (Map f) (Map g) ∧ ?R x = Map ν
  using ν 1 2 R.is-full by blast
have «MkCell (Src ν) (Trg ν) x : f ⇒ g»
proof −
  have arr (MkCell (Src ν) (Trg ν) x)
    using ν arr-char x by auto
  thus ?thesis
    by (metis 0 Hom.in-homE Map.simps(1) Src.simps(1) Trg.simps(1)
            cod-char dom-char f g ide-char in-homI x)
qed
moreover have H.R (MkCell (Src ν) (Trg ν) x) = ν
  using MkCell-Map ν arrI arr-char hcomp-def null-char src.preserves-arr x by auto
ultimately show ∃μ. «μ : f ⇒ g» ∧ H.R μ = ν by auto
```

```
  qed
qed

interpretation L: fully-faithful-functor vcomp vcomp H.L
proof
  show ⋀f f'. ⟦par f f'; H.L f = H.L f'⟧ ⟹ f = f'
  proof −
    fix μ μ'
    assume par: par μ μ'
    and eq: H.L μ = H.L μ'
    have 1: Src μ' = Src μ ∧ Trg μ' = Trg μ
      using par by (metis Src-dom Trg-dom)
    interpret Hom: category ‹Hom (Src μ) (Trg μ)›
      using par arr-char category-Hom by simp
    let ?L = λf. if Hom.arr f
                 then Comp (Trg μ) (Trg μ) (Src μ) (Id (Trg μ)) f
                 else Hom.null
    interpret L: fully-faithful-functor ‹Hom (Src μ) (Trg μ)› ‹Hom (Src μ) (Trg μ)› ?L
      using par arr-char left-unit-Id [of Src μ Trg μ] by simp
    have Lμ: H.L μ = MkCell (Src μ) (Trg μ)
                     (Comp (Trg μ) (Trg μ) (Src μ) (Id (Trg μ)) (Map μ))
      unfolding hcomp-def
      using par trg-def null-char H.src-trg trg.preserves-arr by simp
    have Lμ': H.L μ' = MkCell (Src μ) (Trg μ)
                       (Comp (Trg μ) (Trg μ) (Src μ) (Id (Trg μ)) (Map μ'))
      unfolding hcomp-def
      using par 1 trg-def null-char H.src-trg trg.preserves-arr by simp
    have Map μ = Map μ'
      using Lμ Lμ' eq par arr-char eq L.is-faithful
      by (metis 1 Map.simps(1) Map-cod Map-dom)
    thus μ = μ'
      using 1 par MkCell-Map
      by (metis arr-char null-char)
  qed
  show ⋀f g ν. ⟦ide f; ide g; «ν : H.L f ⇒ H.L g»⟧ ⟹ ∃μ. «μ : f ⇒ g» ∧ H.L μ = ν
  proof −
    fix f g ν
    assume f: ide f and g: ide g and ν: «ν : H.L f ⇒ H.L g»
    interpret Hom: category ‹Hom (Src ν) (Trg ν)›
      using ν arr-char category-Hom by auto
    let ?L = λf. if Hom.arr f
                 then Comp (Trg ν) (Trg ν) (Src ν) (Id (Trg ν)) f
                 else Hom.null
    interpret L: fully-faithful-functor ‹Hom (Src ν) (Trg ν)› ‹Hom (Src ν) (Trg ν)› ?L
      using ν arr-char left-unit-Id by auto
    have 0: Src f = Src ν ∧ Trg f = Trg ν ∧ Src g = Src ν ∧ Trg g = Trg ν
    proof (intro conjI)
      show Src f = Src ν
        using Src-dom Src-hcomp ν f by fastforce
```

```
  show Src g = Src ν
    using ν g Src-cod Src-hcomp by fastforce
  show Trg f = Trg ν
  proof −
    have 1: arr f ∧ dom f = f ∧ cod f = f
      by (meson f ide-char)
    hence hcomp (trg f) f = dom ν
      using ν by fastforce
    thus ?thesis
      using 1
      by (metis (no-types) H.src-trg Trg.simps(1) Trg-hcomp ν dom-char in-homE
          trg.preserves-arr trg-def)
  qed
  show Trg g = Trg ν
  proof −
    have 1: arr g ∧ dom g = g ∧ cod g = g
      by (meson g ide-char)
    hence hcomp (trg g) g = cod ν
      using ν by fastforce
    thus ?thesis
      using 1
      by (metis (no-types) H.src-trg Trg.simps(1) Trg-hcomp ν cod-char in-homE
          trg.preserves-arr trg-def)
  qed
qed
have 1: Hom.in-hom (Map ν) (?L (Map f)) (?L (Map g))
proof
  show Hom.arr (Map ν)
    using ν arr-char by auto
  show Hom.dom (Map ν) = ?L (Map f)
  proof −
    have dom ν = MkCell (Src ν) (Trg ν) (Hom.dom (Map ν))
      using ν dom-char [of ν] by auto
    hence Hom.dom (Map ν) = Map (dom ν)
      by simp
    also have ... = ?L (Map f)
      using 0 f ν left-unit-Id [of Src f Trg f]
      apply (simp add: ide-char'')
      by (metis (no-types, lifting) H.src-trg Map.simps(1) Map-hcomp Trg.simps(1)
          f ide-char in-homE trg.preserves-arr trg-def)
    finally show ?thesis by blast
  qed
  show Hom.cod (Map ν) = ?L (Map g)
  proof −
    have cod ν = MkCell (Src ν) (Trg ν) (Hom.cod (Map ν))
      using ν cod-char [of ν] by auto
    hence Hom.cod (Map ν) = Map (cod ν)
      by simp
    also have ... = ?L (Map g)
```

```
        using 0 g ν left-unit-Id [of Src g Trg g]
        apply (simp add: ide-char'')
        by (metis (no-types, lifting) H.src-trg Map.simps(1) Map-hcomp Trg.simps(1)
            g ide-char in-homE trg.preserves-arr trg-def)
      finally show ?thesis by blast
    qed
  qed
  have 2: Hom.ide (Map f) ∧ Hom.ide (Map g)
    using 0 f g arr-char ide-char'' by simp
  obtain x where x: Hom.in-hom x (Map f) (Map g) ∧ ?L x = Map ν
    using ν 1 2 L.is-full by blast
  have «MkCell (Src ν) (Trg ν) x : f ⇒ g»
  proof −
    have arr (MkCell (Src ν) (Trg ν) x)
      using ν arr-char x by auto
    thus ?thesis
      by (metis 0 Hom.in-homE Map.simps(1) Src.simps(1) Trg.simps(1)
              cod-char dom-char f g ide-char in-homI x)
  qed
  moreover have H.L (MkCell (Src ν) (Trg ν) x) = ν
    using MkCell-Map ν arrI arr-char hcomp-def null-char trg.preserves-arr x by auto
  ultimately show ∃μ. «μ : f ⇒ g» ∧ H.L μ = ν by auto
 qed
qed
```

The unit isomorphisms are defined in terms of the specified function *Unit*.

```
definition i  (‹i[-]›)
where i[a] ≡ MkCell (Src a) (Src a) (Unit (Src a))

lemma i-simps [simp]:
assumes H.obj a
shows Src i[a] = Src a and Trg i[a] = Trg a and Map i[a] = Unit (Src a)
  using assms i-def H.obj-def src-def trg-def
   apply auto
  by (metis Trg.simps(1))
```

The main result: the construction produces a bicategory.

```
proposition induces-bicategory:
shows bicategory vcomp hcomp a i src trg
proof
  fix a
  assume a: H.obj a
  have Src-a: Src a ∈ Obj
    using a ide-char'' by auto
  interpret Hom: category ‹Hom (Src a) (Src a)›
    using Src-a category-Hom by auto
  show «i a : hcomp a a ⇒ a»
  proof −
    have «i a : MkCell (Src a) (Src a) (Hom.dom (Unit (Src a)))
```

$\Rightarrow$ *MkCell* (*Src a*) (*Src a*) (*Hom.cod* (*Unit* (*Src a*)))»
using *a Src-a MkCell-in-hom Unit-in-hom* [*of Src a*] i-*def*
by *simp* (*metis Hom.in-homE*)
moreover have *MkCell* (*Src a*) (*Src a*) (*Hom.cod* (*Unit* (*Src a*))) = *a*
using *a MkCell-Map H.obj-def Map-obj src-def Src-a Unit-in-hom* **by** *force*
moreover have *MkCell* (*Src a*) (*Src a*) (*Hom.dom* (*Unit* (*Src a*))) = *hcomp a a*
using *a H.obj-def Map-hcomp* [*of a a*] *Map-obj Src-a Unit-in-hom Src-hcomp Trg-hcomp*
by (*metis H.objE Hom.in-homE Trg.simps*(*1*) *calculation*(*2*) *hcomp-def*)
ultimately show *?thesis* **by** *simp*
qed
show *iso* (i *a*)
using *a Src-a iso-Unit* [*of Src a*] ‹«i *a* : *hcomp a a* $\Rightarrow$ *a*»› *iso-char* i-*def* **by** *auto*
next
show $\bigwedge$*f g h k*. ⟦*ide f*; *ide g*; *ide h*; *ide k*; *src f* = *trg g*; *src g* = *trg h*; *src h* = *trg k*⟧
$\Longrightarrow$ *vcomp* (*hcomp f* (a *g h k*)) (*vcomp* (a *f* (*hcomp g h*) *k*) (*hcomp* (a *f g h*) *k*)) =
vcomp (a *f g* (*hcomp h k*)) (a (*hcomp f g*) *h k*)
proof (*intro arr-eqI*)
fix *f g h k*
assume *f*: *ide f* **and** *g*: *ide g* **and** *h*: *ide h* **and** *k*: *ide k*
and *fg*: *src f* = *trg g* **and** *gh*: *src g* = *trg h* **and** *hk*: *src h* = *trg k*
have *1*: «*hcomp f* (a *g h k*) :
hcomp f (*hcomp* (*hcomp g h*) *k*) $\Rightarrow$ *hcomp f* (*hcomp g* (*hcomp h k*))»
using *f g h k fg gh hk H.VV.in-hom-char*$_{SbC}$ *H.VV.arr-char*$_{SbC}$
assoc-simps α_0*.map-simp-ide H.VVV.ide-char*$_{SbC}$ *H.VVV.arr-char*$_{SbC}$
H.preserves-hom a-*def*
by *auto*
have *2*: «*hcomp* (a *f g h*) *k* :
hcomp (*hcomp* (*hcomp f g*) *h*) *k* $\Rightarrow$ *hcomp* (*hcomp f* (*hcomp g h*)) *k*»
using *f g h k fg gh hk H.VV.in-hom-char*$_{SbC}$ *H.VV.arr-char*$_{SbC}$
assoc-simps α_0*.map-simp-ide H.VVV.ide-char*$_{SbC}$ *H.VVV.arr-char*$_{SbC}$
H.preserves-hom a-*def*
by *auto*
have *3*: «a *f* (*hcomp g h*) *k* :
hcomp (*hcomp f* (*hcomp g h*)) *k* $\Rightarrow$ *hcomp f* (*hcomp* (*hcomp g h*) *k*)»
using *f g h k fg gh hk H.VV.in-hom-char*$_{SbC}$ *H.VV.arr-char*$_{SbC}$
assoc-simps α_0*.map-simp-ide H.VVV.ide-char*$_{SbC}$ *H.VVV.arr-char*$_{SbC}$
H.preserves-hom a-*def*
by *auto*
have *4*: *seq* (*hcomp f* (a *g h k*)) (*vcomp* (a *f* (*hcomp g h*) *k*) (*hcomp* (a *f g h*) *k*))
using *1 2 3* **by** *auto*
have *5*: *seq* (a *f* (*hcomp g h*) *k*) (*hcomp* (a *f g h*) *k*)
using *2 3* **by** *auto*
have *6*: *seq* (a *f g* (*hcomp h k*)) (a (*hcomp f g*) *h k*)
using *f g h k fg gh hk H.VV.in-hom-char*$_{SbC}$ *H.VV.arr-char*$_{SbC}$
assoc-simps α_0*.map-simp-ide H.VVV.ide-char*$_{SbC}$ *H.VVV.arr-char*$_{SbC}$
H.preserves-hom a-*def*
by *simp*

let *?LHS* = *vcomp* (*hcomp f* (a *g h k*)) (*vcomp* (a *f* (*hcomp g h*) *k*) (*hcomp* (a *f g h*) *k*))

```
      let ?RHS = vcomp (a f g (hcomp h k)) (a (hcomp f g) h k)
      show arr ?LHS
        using 4 by simp
      show arr ?RHS
        using 6 by simp
      show Src ?LHS = Src ?RHS
        using 4 6 f g h k fg gh hk Src-vcomp Src-hcomp
              α0.map-simp-ide H.VVV.ide-char_SbC H.VV.arr-char_SbC H.VVV.arr-char_SbC
              assoc-simps assoc-simps' assoc-def a-def
        by simp
      show Trg ?LHS = Trg ?RHS
        using 4 6 f g h k fg gh hk Trg-vcomp Trg-hcomp
              α0.map-simp-ide H.VVV.ide-char_SbC H.VV.arr-char_SbC H.VVV.arr-char_SbC
              assoc-simps assoc-simps' assoc-def a-def
        by simp
      show Map ?LHS = Map ?RHS
        using 4 5 6 f g h k fg gh hk α0.map-simp-ide H.VVV.ide-char_SbC
              H.VV.arr-char_SbC H.VVV.arr-char_SbC a-def
        apply simp
        using Trg-src Trg-trg pentagon ideD(1) ide-char''
        by (metis (no-types, lifting))
    qed
  qed

  sublocale bicategory vcomp hcomp a i src trg
    using induces-bicategory by simp

end
```

We now establish some correspondences between the constructed bicategory and the originally given data, to provide some assurance that the construction really is doing what we think it is.

```
context concrete-bicategory
begin

  lemma Src-in-Obj:
  assumes arr μ
  shows Src μ ∈ Obj
    using assms arr-char by simp

  lemma Trg-in-Obj:
  assumes arr μ
  shows Trg μ ∈ Obj
    using assms arr-char by simp

  lemma arr-Map:
  assumes arr μ
  shows partial-composition.arr (Hom (Src μ) (Trg μ)) (Map μ)
    using assms arr-char by simp
```

lemma *obj-MkObj-Src*:
assumes *arr* μ
shows *obj* (*MkObj* (*Src* μ))
  using *assms* **by** (*simp add*: *Src-in-Obj obj-MkObj*)

lemma *obj-MkObj-Trg*:
assumes *arr* μ
shows *obj* (*MkObj* (*Trg* μ))
  using *assms* **by** (*simp add*: *Trg-in-Obj obj-MkObj*)

lemma *vcomp-MkCell* [*simp*]:
assumes *arr* (*MkCell A B f*) **and** *arr* (*MkCell A B g*)
and *partial-composition.seq* (*Hom A B*) *f g*
shows *vcomp* (*MkCell A B f*) (*MkCell A B g*) = *MkCell A B* (*Hom A B f g*)
  using *assms*(*1*−*3*) *arr-char vcomp-def* **by** *fastforce*

lemma *hcomp-MkCell* [*simp*]:
assumes *arr* (*MkCell B C f*) **and** *arr* (*MkCell A B g*)
shows *hcomp* (*MkCell B C f*) (*MkCell A B g*) = *MkCell A C* (*Comp C B A f g*)
  by (*simp add*: *assms*(*1*−*2*) *hcomp-def*)

The objects of the constructed bicategory are in bijective correspondence with the originally given set *Obj*, via the inverse mappings *MkObj* and *Src*.

proposition *bij-betw-obj-Obj*:
shows *MkObj* ∈ *Obj* → *Collect obj*
and *Src* ∈ *Collect obj* → *Obj*
and *A* ∈ *Obj* ⟹ *Src* (*MkObj A*) = *A*
and *a* ∈ *Collect obj* ⟹ *MkObj* (*Src a*) = *a*
and *bij-betw MkObj Obj* (*Collect obj*)
  using *obj-MkObj obj-def Src-in-Obj src-def*
    apply *auto*
  by (*intro bij-betwI*) *auto*

lemma *obj-char*:
shows *obj a* ⟷ *Src a* ∈ *Obj* ∧ *a* = *MkCell* (*Src a*) (*Src a*) (*Id* (*Src a*))
  using *Src-in-Obj bij-betw-obj-Obj*(*4*) *obj-MkObj* **by** *force*

lemma *Map-in-Hom*:
assumes *arr* μ
shows *partial-composition.in-hom* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ) (*Map* (*dom* μ)) (*Map* (*cod* μ))
  by (*simp add*: *Src-in-Obj Trg-in-Obj arr-Map assms category.in-homI category-Hom*)

For each pair of objects *a* and *b*, the hom-category *hhom a b* of the constructed bicategory is isomorphic to the originally given category *Hom* (*Src a*) (*Src b*).

proposition *isomorphic-hhom-Hom*:
assumes *obj a* **and** *obj b*
shows *isomorphic-categories*

```
      (subcategory.comp vcomp (λf. f ∈ hhom a b)) (Hom (Src a) (Src b))
proof −
  interpret hom: subcategory vcomp ‹λf. f ∈ hhom a b›
    using assms by unfold-locales auto
  interpret Hom: category ‹Hom (Src a) (Src b)›
    using assms category-Hom Src-in-Obj obj-def by simp
  let ?Map = λμ. if μ ∈ hhom a b then Map μ else Hom.null
  let ?MkCell = λμ. if Hom.arr μ then MkCell (Src a) (Src b) μ else hom.null
  interpret Map: functor hom.comp ‹Hom (Src a) (Src b)› ?Map
  proof
    fix μ
    show ¬ hom.arr μ ⟹ ?Map μ = Hom.null
      using hom.inclusion [of μ] hom.arr-char_SbC by auto
    assume μ: hom.arr μ
    have 0: src μ = a ∧ trg μ = b
      using μ hom.arrE src-def trg-def
      by (metis in-hhomE mem-Collect-eq)
    have 1: arr μ
      using μ hom.inclusion hom.arrE by blast
    have 2: Src μ = Src a ∧ Trg μ = Trg b
      using μ 0
      by (metis Src-src Trg-trg hom.arr-char_SbC hom.inclusion)
    show Hom.arr (?Map μ)
      using 0 1 arr-Map [of μ] μ by auto
    show Hom.dom (?Map μ) = ?Map (hom.dom μ)
    proof −
      have Hom.dom (?Map μ) = Map (dom μ)
        using μ hom.arrE by fastforce
      thus ?thesis
        by (metis μ hom.arrE hom.arr-dom hom.dom-simp)
    qed
    show Hom.cod (?Map μ) = ?Map (hom.cod μ)
    proof −
      have Hom.cod (?Map μ) = Map (cod μ)
        using μ hom.arrE by fastforce
      thus ?thesis
        by (metis μ hom.arrE hom.arr-cod hom.cod-simp)
    qed
    next
    fix μ ν
    assume μν: hom.seq μ ν
    show ?Map (hom.comp μ ν) = Hom (Src a) (Src b) (?Map μ) (?Map ν)
    proof −
      have 1: hom.arr ν ∧ hom.arr μ ∧ seq μ ν
        using μν hom.seq-char_SbC by blast
      hence 2: hom.comp μ ν = vcomp μ ν
        using hom.comp-char by auto
      have 3: μ ∈ hhom a b
        using 1 hom.arrE by blast
```

```
    have Src a = Src μ
      using 3 by (metis Trg-src in-hhomE mem-Collect-eq obj-def)
    moreover have Src b = Trg μ
      using 3 by (metis Trg-src Trg-trg in-hhomE mem-Collect-eq obj-def)
    ultimately show ?thesis
      using 1 2 Map-vcomp μν hom.arrE by presburger
  qed
qed
interpret MkCell: functor ‹Hom (Src a) (Src b)› hom.comp ?MkCell
proof
  fix μ
  show ¬ Hom.arr μ ⟹ ?MkCell μ = hom.null
    by simp
  assume μ: Hom.arr μ
  show 1: hom.arr (?MkCell μ)
    using assms obj-def μ hom.arr-char_SbC arr-MkCell src-def arr-char by auto
  show hom.dom (?MkCell μ) = ?MkCell (Hom.dom μ)
        using assms 1 μ hom.dom-char_SbC src-def arr-char obj-def Src-in-Obj by simp
  show hom.cod (?MkCell μ) = ?MkCell (Hom.cod μ)
        using assms 1 μ hom.cod-char_SbC src-def arr-char obj-def Src-in-Obj by simp
  next
  fix μ ν
  assume μν: Hom.seq μ ν
  have 1: hom.arr (?MkCell μ)
    using assms obj-def μν hom.arr-char_SbC src-def arr-char by auto
  have 2: hom.arr (?MkCell ν)
    using assms obj-def μν hom.arr-char_SbC src-def arr-char by auto
  have 3: hom.dom (?MkCell μ) = hom.cod (?MkCell ν)
    using μν 1 2 hom.dom-char_SbC dom-char hom.cod-char_SbC cod-char arr-char by auto
  have 4: seq (?MkCell μ) (?MkCell ν)
   by (metis 1 2 3 hom.arrE hom.cod-simp hom.comp-closed hom.dom-simp hom.inclusion)
  have hom.comp (?MkCell μ) (?MkCell ν) =
         vcomp (MkCell (Src a) (Src b) μ) (MkCell (Src a) (Src b) ν)
    using μν 1 2 4 hom.comp-char by auto
  also have ... = MkCell (Src a) (Src b) (Hom (Src a) (Src b) μ ν)
    using μν 4 vcomp-char [of MkCell (Src a) (Src b) μ MkCell (Src a) (Src b) ν]
    by auto
  also have ... = ?MkCell (Hom (Src a) (Src b) μ ν)
    using μν by simp
  finally show ?MkCell (Hom (Src a) (Src b) μ ν) = hom.comp (?MkCell μ) (?MkCell ν)
    by simp
qed
interpret inverse-functors hom.comp ‹Hom (Src a) (Src b)› ?MkCell ?Map
proof
  show ?MkCell o ?Map = hom.map
  proof
    fix μ
    have μ ∉ hhom a b ⟹ (?MkCell o ?Map) μ = hom.map μ
      using o-apply hom.extensionality hom.arr-char_SbC by simp
```

```
      moreover have μ ∈ hhom a b ⟹ (?MkCell o ?Map) μ = hom.map μ
      proof −
        assume μ: μ ∈ hhom a b
        have (?MkCell o ?Map) μ = MkCell (Src a) (Src b) (Map μ)
          using μ arr-char src-def trg-def by auto
        also have ... = μ
          using μ MkCell-Map arr-char null-char by auto
        also have ... = hom.map μ
          using μ hom.arrI_SbC hom.map-def by presburger
        finally show (?MkCell o ?Map) μ = hom.map μ
          by simp
      qed
      ultimately show (?MkCell o ?Map) μ = hom.map μ
        by blast
    qed
    show ?Map o ?MkCell = Hom.map
    proof
      fix μ
      have ¬ Hom.arr μ ⟹ (?Map o ?MkCell) μ = Hom.map μ
        using Hom.extensionality hom.null-char by auto
      moreover have Hom.arr μ ⟹ (?Map o ?MkCell) μ = Hom.map μ
      proof −
        assume μ: Hom.arr μ
        have in-hhom (MkCell (Src a) (Src b) μ) a b
          using μ MkCell.preserves-reflects-arr [of μ] hom.arr-char_SbC by simp
        thus (?Map o ?MkCell) μ = Hom.map μ
          using μ by simp
      qed
      ultimately show (?Map o ?MkCell) μ = Hom.map μ
        by blast
    qed
  qed
  show ?thesis ..
qed

end

end
```

1.6 Coherence

theory *Coherence*
imports *Bicategory*
begin

In this section, we generalize to bicategories the proof of the Coherence Theorem that we previously gave for monoidal categories (see *MonoidalCategory.evaluation-map.coherence* in `MonoidalCategory`). As was the case for the previous proof, the current proof takes a syntactic approach. First we define a formal "bicate-

gorical language" of terms constructed using syntactic operators that correspond to the various operations (vertical and horizontal composition, associators and unitors) found in a bicategory. Terms of the language are classified as formal "arrows", "identities", or "objects" according to the syntactic operators used in their formation. A class of terms called "canonical" is also defined in this way. Functions that map "arrows" to their "domain" and "codomain", and to their "source" and "target" are defined by recursion on the structure of terms. Next, we define a notion of "normal form" for terms in this language and we give a recursive definition of a function that maps terms to their normal forms. Normalization moves vertical composition inside of horizontal composition and "flattens" horizontal composition by associating all horizontal compositions to the right. In addition, normalization deletes from a term any horizontal composites involving an arrow and its source or target, replacing such composites by just the arrow itself. We then define a "reduction function" that maps each identity term t to a "canonical" term $t{\downarrow\!\downarrow}$ that connects t with its normal form. The definition of reduction is also recursive, but it is somewhat more complex than normalization in that it involves two mutually recursive functions: one that applies to any identity term and another that applies only to terms that are the horizontal composite of two identity terms.

The next step is to define an "evaluation function" that evaluates terms in a given bicategory (which is left as an unspecified parameter). We show that evaluation respects bicategorical structure: the domain, codomain, source, and target mappings on terms correspond under evaluation to the actual domain, codomain, source and target mappings on the given bicategory, the vertical and horizontal composition on terms correspond to the actual vertical and horizontal composition of the bicategory, and unit and associativity terms evaluate to the actual unit and associativity isomorphisms of the bicategory. In addition, "object terms" evaluate to objects (*i.e.* 0-cells), "identity terms" evaluate to identities (*i.e.* 1-cells), "arrow terms" evaluate to arrows (*i.e.* 2-cells), and "canonical terms" evaluate to canonical isomorphisms. A term is defined to be "coherent" if, roughly speaking, it is a formal arrow whose evaluation commutes with the evaluations of the reductions to normal form of its domain and codomain. We then prove the Coherence Theorem, expressed in the form: "every arrow is coherent." This implies a more classical version of the Coherence Theorem, which says that: "syntactically parallel arrows with the same normal form have equal evaluations".

1.6.1 Bicategorical Language

For the most part, the definition of the "bicategorical language" of terms is a straightforward generalization of the "monoidal language" that we used for monoidal categories. Some modifications are required, however, due to the fact that horizontal composition in a bicategory is a partial operation, whereas the the tensor product in a monoidal category is well-defined for all pairs of arrows. One difference is that we have found it necessary to introduce a new class of primitive terms whose elements represent "formal objects", so that there is some way to identify the source and target of what would otherwise be an empty horizontal composite. This was not an issue for monoidal categories, because the totality of horizontal composition meant that there was no need for syntactically defined

sources and targets. Another difference is what we have chosen for the "generators" of the language and how they are used to form primitive terms. For monoidal categories, we supposed that we were given a category C and the syntax contained a constructor to form a primitive term corresponding to each arrow of C. We assumed a category as the given data, rather than something less structured, such as a graph, because we were primarily interested in the tensor product and the associators and unitors, and were relatively uninterested in the strictly associative and unital composition of the underlying category. For bicategories, we also take the vertical composition as given for the same reasons; however, this is not yet sufficient due to the fact that horizontal composition in a bicategory is a partial operation, in contrast to the tensor product in a monoidal category, which is defined for all pairs of arrows. To deal with this issue, for bicategories we assume that source and target mappings are also given, so that the given data forms a category with "horizontal homs". The given source and target mappings are extended to all terms and used to define when two terms are "formally horizontally composable".

```
locale bicategorical-language =
  category V +
  horizontal-homs V src trg
  for V :: 'a comp                    (infixr ‹·› 55)
  and src :: 'a ⇒ 'a
  and trg :: 'a ⇒ 'a
begin
```

Constructor $Prim_0$ is used to construct "formal objects" and $Prim$ is used to construct primitive terms that are not formal objects.

```
  datatype (discs-sels) 't term =
    Prim₀ 't                          (‹⟨-⟩₀›)
  | Prim 't                           (‹⟨-⟩›)
  | Hcomp 't term 't term             (infixr ‹★› 53)
  | Vcomp 't term 't term             (infixr ‹·› 55)
  | Lunit 't term                     (‹l[-]›)
  | Lunit' 't term                    (‹l⁻¹[-]›)
  | Runit 't term                     (‹r[-]›)
  | Runit' 't term                    (‹r⁻¹[-]›)
  | Assoc 't term 't term 't term     (‹a[-, -, -]›)
  | Assoc' 't term 't term 't term    (‹a⁻¹[-, -, -]›)
```

We define formal domain, codomain, source, and target functions on terms.

```
  primrec Src :: 'a term ⇒ 'a term
  where Src ⟨μ⟩₀ = ⟨μ⟩₀
      | Src ⟨μ⟩ = ⟨src μ⟩₀
      | Src (t ★ u) = Src u
      | Src (t · u) = Src t
      | Src l[t] = Src t
      | Src l⁻¹[t] = Src t
      | Src r[t] = Src t
      | Src r⁻¹[t] = Src t
      | Src a[t, u, v] = Src v
```

$\mid$ *Src* $\mathbf{a}^{-1}[t,\ u,\ v] =$ *Src* v

primrec *Trg* :: $'a$ *term* $\Rightarrow$ $'a$ *term*
where *Trg* $\langle\mu\rangle_0 = \langle\mu\rangle_0$
$\mid$ *Trg* $\langle\mu\rangle = \langle$*trg* $\mu\rangle_0$
$\mid$ *Trg* $(t \star u) =$ *Trg* t
$\mid$ *Trg* $(t \cdot u) =$ *Trg* t
$\mid$ *Trg* $\mathbf{l}[t] =$ *Trg* t
$\mid$ *Trg* $\mathbf{l}^{-1}[t] =$ *Trg* t
$\mid$ *Trg* $\mathbf{r}[t] =$ *Trg* t
$\mid$ *Trg* $\mathbf{r}^{-1}[t] =$ *Trg* t
$\mid$ *Trg* $\mathbf{a}[t,\ u,\ v] =$ *Trg* t
$\mid$ *Trg* $\mathbf{a}^{-1}[t,\ u,\ v] =$ *Trg* t

primrec *Dom* :: $'a$ *term* $\Rightarrow$ $'a$ *term*
where *Dom* $\langle\mu\rangle_0 = \langle\mu\rangle_0$
$\mid$ *Dom* $\langle\mu\rangle = \langle$*dom* $\mu\rangle$
$\mid$ *Dom* $(t \star u) =$ *Dom* $t \star$ *Dom* u
$\mid$ *Dom* $(t \cdot u) =$ *Dom* u
$\mid$ *Dom* $\mathbf{l}[t] =$ *Trg* $t \star$ *Dom* t
$\mid$ *Dom* $\mathbf{l}^{-1}[t] =$ *Dom* t
$\mid$ *Dom* $\mathbf{r}[t] =$ *Dom* $t \star$ *Src* t
$\mid$ *Dom* $\mathbf{r}^{-1}[t] =$ *Dom* t
$\mid$ *Dom* $\mathbf{a}[t,\ u,\ v] = ($*Dom* $t \star$ *Dom* $u) \star$ *Dom* v
$\mid$ *Dom* $\mathbf{a}^{-1}[t,\ u,\ v] =$ *Dom* $t \star ($*Dom* $u \star$ *Dom* $v)$

primrec *Cod* :: $'a$ *term* $\Rightarrow$ $'a$ *term*
where *Cod* $\langle\mu\rangle_0 = \langle\mu\rangle_0$
$\mid$ *Cod* $\langle\mu\rangle = \langle$*cod* $\mu\rangle$
$\mid$ *Cod* $(t \star u) =$ *Cod* $t \star$ *Cod* u
$\mid$ *Cod* $(t \cdot u) =$ *Cod* t
$\mid$ *Cod* $\mathbf{l}[t] =$ *Cod* t
$\mid$ *Cod* $\mathbf{l}^{-1}[t] =$ *Trg* $t \star$ *Cod* t
$\mid$ *Cod* $\mathbf{r}[t] =$ *Cod* t
$\mid$ *Cod* $\mathbf{r}^{-1}[t] =$ *Cod* $t \star$ *Src* t
$\mid$ *Cod* $\mathbf{a}[t,\ u,\ v] =$ *Cod* $t \star ($*Cod* $u \star$ *Cod* $v)$
$\mid$ *Cod* $\mathbf{a}^{-1}[t,\ u,\ v] = ($*Cod* $t \star$ *Cod* $u) \star$ *Cod* v

A term is a "formal arrow" if it is constructed from primitive arrows in such a way that horizontal and vertical composition are applied only to formally composable pairs of terms. The definitions of "formal identity" and "formal object" follow a similar pattern.

primrec *Arr* :: $'a$ *term* $\Rightarrow$ *bool*
where *Arr* $\langle\mu\rangle_0 =$ *obj* μ
$\mid$ *Arr* $\langle\mu\rangle =$ *arr* μ
$\mid$ *Arr* $(t \star u) = ($*Arr* $t \wedge$ *Arr* $u \wedge$ *Src* $t =$ *Trg* $u)$
$\mid$ *Arr* $(t \cdot u) = ($*Arr* $t \wedge$ *Arr* $u \wedge$ *Dom* $t =$ *Cod* $u)$
$\mid$ *Arr* $\mathbf{l}[t] =$ *Arr* t
$\mid$ *Arr* $\mathbf{l}^{-1}[t] =$ *Arr* t
$\mid$ *Arr* $\mathbf{r}[t] =$ *Arr* t

| $Arr\ \mathbf{r}^{-1}[t] = Arr\ t$
| $Arr\ \mathbf{a}[t,\ u,\ v] = (Arr\ t \wedge Arr\ u \wedge Arr\ v \wedge Src\ t = Trg\ u \wedge Src\ u = Trg\ v)$
| $Arr\ \mathbf{a}^{-1}[t,\ u,\ v] = (Arr\ t \wedge Arr\ u \wedge Arr\ v \wedge Src\ t = Trg\ u \wedge Src\ u = Trg\ v)$

primrec $Ide :: 'a\ term \Rightarrow bool$
where $Ide\ \langle\mu\rangle_0 = obj\ \mu$
| $Ide\ \langle\mu\rangle = ide\ \mu$
| $Ide\ (t \star u) = (Ide\ t \wedge Ide\ u \wedge Src\ t = Trg\ u)$
| $Ide\ (t \cdot u) = False$
| $Ide\ \mathbf{l}[t] = False$
| $Ide\ \mathbf{l}^{-1}[t] = False$
| $Ide\ \mathbf{r}[t] = False$
| $Ide\ \mathbf{r}^{-1}[t] = False$
| $Ide\ \mathbf{a}[t,\ u,\ v] = False$
| $Ide\ \mathbf{a}^{-1}[t,\ u,\ v] = False$

primrec $Obj :: 'a\ term \Rightarrow bool$
where $Obj\ \langle\mu\rangle_0 = obj\ \mu$
| $Obj\ \langle\mu\rangle = False$
| $Obj\ (t \star u) = False$
| $Obj\ (t \cdot u) = False$
| $Obj\ \mathbf{l}[t] = False$
| $Obj\ \mathbf{l}^{-1}[t] = False$
| $Obj\ \mathbf{r}[t] = False$
| $Obj\ \mathbf{r}^{-1}[t] = False$
| $Obj\ \mathbf{a}[t,\ u,\ v] = False$
| $Obj\ \mathbf{a}^{-1}[t,\ u,\ v] = False$

abbreviation $HSeq :: 'a\ term \Rightarrow 'a\ term \Rightarrow bool$
where $HSeq\ t\ u \equiv Arr\ t \wedge Arr\ u \wedge Src\ t = Trg\ u$

abbreviation $VSeq :: 'a\ term \Rightarrow 'a\ term \Rightarrow bool$
where $VSeq\ t\ u \equiv Arr\ t \wedge Arr\ u \wedge Dom\ t = Cod\ u$

abbreviation $HPar :: 'a\ term => 'a\ term \Rightarrow bool$
where $HPar\ t\ u \equiv Arr\ t \wedge Arr\ u \wedge Src\ t = Src\ u \wedge Trg\ t = Trg\ u$

abbreviation $VPar :: 'a\ term => 'a\ term \Rightarrow bool$
where $VPar\ t\ u \equiv Arr\ t \wedge Arr\ u \wedge Dom\ t = Dom\ u \wedge Cod\ t = Cod\ u$

abbreviation $HHom :: 'a\ term \Rightarrow 'a\ term \Rightarrow 'a\ term\ set$
where $HHom\ a\ b \equiv \{\ t.\ Arr\ t \wedge Src\ t = a \wedge Trg\ t = b\ \}$

abbreviation $VHom :: 'a\ term \Rightarrow 'a\ term \Rightarrow 'a\ term\ set$
where $VHom\ f\ g \equiv \{\ t.\ Arr\ t \wedge Dom\ t = f \wedge Cod\ t = g\ \}$

lemma *is-Prim0-Src*:
shows $is\text{-}Prim_0\ (Src\ t)$
by (*induct t*; *simp*)

lemma *is-Prim0-Trg*:
shows $\textit{is-Prim}_0$ (*Trg t*)
  by (*induct t*; *simp*)

lemma *Src-Src* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Src* (*Src t*) = *Src t*
  by (*induct t*) *auto*

lemma *Trg-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Trg* (*Trg t*) = *Trg t*
  by (*induct t*) *auto*

lemma *Src-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Src* (*Trg t*) = *Trg t*
  by (*induct t*) *auto*

lemma *Trg-Src* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Trg* (*Src t*) = *Src t*
  by (*induct t*) *auto*

lemma *Dom-Src* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Dom* (*Src t*) = *Src t*
  by (*induct t*) *auto*

lemma *Dom-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Dom* (*Trg t*) = *Trg t*
  by (*induct t*) *auto*

lemma *Cod-Src* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Cod* (*Src t*) = *Src t*
  by (*induct t*) *auto*

lemma *Cod-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Cod* (*Trg t*) = *Trg t*
  by (*induct t*) *auto*

lemma *Src-Dom-Cod*:
shows *Arr t* $\Longrightarrow$ *Src* (*Dom t*) = *Src t* $\wedge$ *Src* (*Cod t*) = *Src t*
   using *src-dom src-cod* **by** (*induct t*) *auto*

lemma *Src-Dom* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Src* (*Dom t*) = *Src t*
  using *Src-Dom-Cod* **by** *blast*

lemma *Src-Cod* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Src* (*Cod t*) = *Src t*
  using *Src-Dom-Cod* **by** *blast*

lemma *Trg-Dom-Cod*:
shows $Arr\ t \Longrightarrow Trg\ (Dom\ t) = Trg\ t \wedge Trg\ (Cod\ t) = Trg\ t$
using *trg-dom trg-cod* **by** (*induct t*) *auto*

lemma *Trg-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Trg\ (Dom\ t) = Trg\ t$
using *Trg-Dom-Cod* **by** *blast*

lemma *Trg-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Trg\ (Cod\ t) = Trg\ t$
using *Trg-Dom-Cod* **by** *blast*

lemma *VSeq-implies-HPar*:
shows $VSeq\ t\ u \Longrightarrow HPar\ t\ u$
using *Src-Dom* [*of t*] *Src-Cod* [*of u*] *Trg-Dom* [*of t*] *Trg-Cod* [*of u*] **by** *auto*

lemma *Dom-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Dom\ (Dom\ t) = Dom\ t$
by (*induct t*, *auto*)

lemma *Cod-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Cod\ (Cod\ t) = Cod\ t$
by (*induct t*, *auto*)

lemma *Dom-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Dom\ (Cod\ t) = Cod\ t$
by (*induct t*, *auto*)

lemma *Cod-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Cod\ (Dom\ t) = Dom\ t$
by (*induct t*, *auto*)

lemma *Obj-implies-Ide* :
shows $Obj\ t \Longrightarrow Ide\ t$
by (*induct t*) *auto*

lemma *Ide-implies-Arr* [*simp*]:
shows $Ide\ t \Longrightarrow Arr\ t$
by (*induct t*, *auto*)

lemma *Dom-Ide*:
shows $Ide\ t \Longrightarrow Dom\ t = t$
by (*induct t*, *auto*)

lemma *Cod-Ide*:
shows $Ide\ t \Longrightarrow Cod\ t = t$
by (*induct t*, *auto*)

lemma *Obj-Src* [*simp*]:

shows *Arr t* $\Longrightarrow$ *Obj* (*Src t*)
  by (*induct t*) *auto*

lemma *Obj-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Obj* (*Trg t*)
  by (*induct t*) *auto*

lemma *Ide-Dom* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Ide* (*Dom t*)
  using *Obj-implies-Ide*
  by (*induct t*, *auto*)

lemma *Ide-Cod* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Ide* (*Cod t*)
  using *Obj-implies-Ide*
  by (*induct t*, *auto*)

lemma *Arr-in-Hom*:
assumes *Arr t*
shows $t \in$ *HHom* (*Src t*) (*Trg t*) **and** $t \in$ *VHom* (*Dom t*) (*Cod t*)
proof −
  have *1*: *Arr t* $\Longrightarrow$ $t \in$ *HHom* (*Src t*) (*Trg t*) $\wedge$ $t \in$ *VHom* (*Dom t*) (*Cod t*)
    by (*induct t*, *auto*)
  show $t \in$ *HHom* (*Src t*) (*Trg t*) **using** *assms 1* **by** *simp*
  show $t \in$ *VHom* (*Dom t*) (*Cod t*) **using** *assms 1* **by** *simp*
qed

lemma *Ide-in-Hom*:
assumes *Ide t*
shows $t \in$ *HHom* (*Src t*) (*Trg t*) **and** $t \in$ *VHom t t*
proof −
  have *1*: *Ide t* $\Longrightarrow$ $t \in$ *HHom* (*Src t*) (*Trg t*) $\wedge$ $t \in$ *VHom t t*
    by (*induct t*, *auto*)
  show $t \in$ *HHom* (*Src t*) (*Trg t*) **using** *assms 1* **by** *simp*
  show $t \in$ *VHom t t* **using** *assms 1* **by** *simp*
qed

lemma *Obj-in-Hom*:
assumes *Obj t*
shows $t \in$ *HHom t t* **and** $t \in$ *VHom t t*
proof −
  have *1*: *Obj t* $\Longrightarrow$ $t \in$ *HHom t t* $\wedge$ $t \in$ *VHom t t*
    by (*induct t*, *auto*)
  show $t \in$ *HHom t t* **using** *assms 1* **by** *simp*
  show $t \in$ *VHom t t* **using** *assms 1* **by** *simp*
qed

A formal arrow is "canonical" if the only primitive arrows used in its construction are objects and identities.

primrec *Can* :: *'a term* ⇒ *bool*
where *Can* $\langle\mu\rangle_0$ = *obj* μ
| *Can* $\langle\mu\rangle$ = *ide* μ
| *Can* ($t \star u$) = (*Can* t ∧ *Can* u ∧ *Src* t = *Trg* u)
| *Can* ($t \cdot u$) = (*Can* t ∧ *Can* u ∧ *Dom* t = *Cod* u)
| *Can* $\mathbf{l}[t]$ = *Can* t
| *Can* $\mathbf{l}^{-1}[t]$ = *Can* t
| *Can* $\mathbf{r}[t]$ = *Can* t
| *Can* $\mathbf{r}^{-1}[t]$ = *Can* t
| *Can* $\mathbf{a}[t, u, v]$ = (*Can* t ∧ *Can* u ∧ *Can* v ∧ *Src* t = *Trg* u ∧ *Src* u = *Trg* v)
| *Can* $\mathbf{a}^{-1}[t, u, v]$ = (*Can* t ∧ *Can* u ∧ *Can* v ∧ *Src* t = *Trg* u ∧ *Src* u = *Trg* v)

lemma *Ide-implies-Can*:
shows *Ide* t ⟹ *Can* t
by (*induct* t, *auto*)

lemma *Can-implies-Arr*:
shows *Can* t ⟹ *Arr* t
by (*induct* t, *auto*)

Canonical arrows can be formally inverted.

primrec *Inv* :: *'a term* ⇒ *'a term*
where *Inv* $\langle\mu\rangle_0$ = $\langle\mu\rangle_0$
| *Inv* $\langle\mu\rangle$ = $\langle$*inv* $\mu\rangle$
| *Inv* ($t \star u$) = (*Inv* $t \star$ *Inv* u)
| *Inv* ($t \cdot u$) = (*Inv* $u \cdot$ *Inv* t)
| *Inv* $\mathbf{l}[t]$ = $\mathbf{l}^{-1}$[*Inv* t]
| *Inv* $\mathbf{l}^{-1}[t]$ = $\mathbf{l}$[*Inv* t]
| *Inv* $\mathbf{r}[t]$ = $\mathbf{r}^{-1}$[*Inv* t]
| *Inv* $\mathbf{r}^{-1}[t]$ = $\mathbf{r}$[*Inv* t]
| *Inv* $\mathbf{a}[t, u, v]$ = $\mathbf{a}^{-1}$[*Inv* t, *Inv* u, *Inv* v]
| *Inv* $\mathbf{a}^{-1}[t, u, v]$ = $\mathbf{a}$[*Inv* t, *Inv* u, *Inv* v]

lemma *Src-Inv* [*simp*]:
shows *Can* t ⟹ *Src* (*Inv* t) = *Src* t
using *Can-implies-Arr VSeq-implies-HPar*
apply (*induct* t, *auto*)
by *metis*

lemma *Trg-Inv* [*simp*]:
shows *Can* t ⟹ *Trg* (*Inv* t) = *Trg* t
using *Can-implies-Arr VSeq-implies-HPar*
apply (*induct* t, *auto*)
by *metis*

lemma *Dom-Inv* [*simp*]:
shows *Can* t ⟹ *Dom* (*Inv* t) = *Cod* t
by (*induct* t, *auto*)

lemma *Cod-Inv* [*simp*]:
shows $Can\ t \Longrightarrow Cod\ (Inv\ t) = Dom\ t$
  by (*induct t*, *auto*)

lemma *Inv-preserves-Ide*:
shows $Ide\ t \Longrightarrow Ide\ (Inv\ t)$
  using *Ide-implies-Can* **by** (*induct t*, *auto*)

lemma *Can-Inv* [*simp*]:
shows $Can\ t \Longrightarrow Can\ (Inv\ t)$
  by (*induct t*, *auto*)

lemma *Inv-in-Hom* [*intro*]:
assumes *Can t*
shows $Inv\ t \in HHom\ (Src\ t)\ (Trg\ t)$ **and** $Inv\ t \in VHom\ (Cod\ t)\ (Dom\ t)$
  using *assms Can-Inv Can-implies-Arr* **by** *simp-all*

lemma *Inv-Ide* [*simp*]:
assumes *Ide a*
shows $Inv\ a = a$
  using *assms* **by** (*induct a*, *auto*)

lemma *Inv-Inv* [*simp*]:
assumes *Can t*
shows $Inv\ (Inv\ t) = t$
  using *assms* **by** (*induct t*, *auto*)

1.6.2 Normal Terms

We call a term "normal" if it is either a formal object or it is constructed from primitive arrows using only horizontal composition associated to the right. Essentially, such terms are (typed) composable sequences of arrows of $(\cdot)$, where the empty list is represented by a formal object and $\star$ is used as the list constructor.

fun $Nml :: 'a\ term \Rightarrow bool$
where $Nml\ \langle\mu\rangle_0 = obj\ \mu$
  $|\ Nml\ \langle\mu\rangle = arr\ \mu$
  $|\ Nml\ (\langle\nu\rangle \star u) = (arr\ \nu \wedge Nml\ u \wedge \neg\ is\text{-}Prim_0\ u \wedge \langle src\ \nu\rangle_0 = Trg\ u)$
  $|\ Nml\ \text{-} = False$

lemma *Nml-HcompD*:
assumes $Nml\ (t \star u)$
shows $\langle un\text{-}Prim\ t\rangle = t$ **and** $arr\ (un\text{-}Prim\ t)$ **and** $Nml\ t$ **and** $Nml\ u$
and $\neg\ is\text{-}Prim_0\ u$ **and** $\langle src\ (un\text{-}Prim\ t)\rangle_0 = Trg\ u$ **and** $Src\ t = Trg\ u$
proof −
  have *1*: $t = \langle un\text{-}Prim\ t\rangle \wedge arr\ (un\text{-}Prim\ t) \wedge Nml\ t \wedge Nml\ u \wedge \neg\ is\text{-}Prim_0\ u \wedge$
    $\langle src\ (un\text{-}Prim\ t)\rangle_0 = Trg\ u$
    using *assms* **by** (*cases t*; *simp*; *cases u*; *simp*)
  show $\langle un\text{-}Prim\ t\rangle = t$ **using** *1* **by** *simp*
  show $arr\ (un\text{-}Prim\ t)$ **using** *1* **by** *simp*

```
  show Nml t using 1 by simp
  show Nml u using 1 by simp
  show ¬ is-Prim₀ u using 1 by simp
  show ⟨src (un-Prim t)⟩₀ = Trg u using 1 by simp
  show Src t = Trg u
    using assms
    apply (cases t) by simp-all
qed

lemma Nml-implies-Arr:
shows Nml t ⟹ Arr t
  by (induct t, auto simp add: Nml-HcompD)

lemma Nml-Src [simp]:
shows Nml t ⟹ Nml (Src t)
  apply (induct t, simp-all)
  using Nml-HcompD by metis

lemma Nml-Trg [simp]:
shows Nml t ⟹ Nml (Trg t)
  apply (induct t, simp-all)
  using Nml-HcompD by metis

lemma Nml-Dom [simp]:
shows Nml t ⟹ Nml (Dom t)
proof (induct t, simp-all add: Nml-HcompD)
  fix u v
  assume I1: Nml (Dom u)
  assume I2: Nml (Dom v)
  assume uv: Nml (u ⋆ v)
  show Nml (Dom u ⋆ Dom v)
  proof −
    have 1: is-Prim (Dom u) ∧ arr (un-Prim (Dom u)) ∧ Dom u = ⟨dom (un-Prim u)⟩
      using uv by (cases u; simp; cases v, simp-all)
    have 2: Nml v ∧ ¬ is-Prim₀ v ∧ ¬ is-Vcomp v ∧ ¬ is-Lunit′ v ∧ ¬ is-Runit′ v
      using uv by (cases u, simp-all; cases v, simp-all)
    have arr (dom (un-Prim u))
      using 1 by fastforce
    moreover have Nml (Dom v) ∧ ¬ is-Prim₀ v
      using 2 I2 by (cases v, simp-all)
    moreover have ⟨src (dom (un-Prim u))⟩₀ = Trg (Dom v)
    proof −
      have Trg (Dom v) = Src (Dom u)
        using uv Nml-implies-Arr by fastforce
      also have ... = ⟨src (dom (un-Prim u))⟩₀
        using 1 by fastforce
      finally show ?thesis by argo
    qed
    moreover have ¬ is-Prim₀ (Dom v)
```

```
      using 2 by (cases v, simp-all)
    ultimately show ?thesis using 1 2 by simp
  qed
qed

lemma Nml-Cod [simp]:
shows Nml t ⟹ Nml (Cod t)
proof (induct t, simp-all add: Nml-HcompD)
  fix u v
  assume I1: Nml (Cod u)
  assume I2: Nml (Cod v)
  assume uv: Nml (u ⋆ v)
  show Nml (Cod u ⋆ Cod v)
  proof −
    have 1: is-Prim (Cod u) ∧ arr (un-Prim (Cod u)) ∧ Cod u = ⟨cod (un-Prim u)⟩
      using uv by (cases u; simp; cases v, simp-all)
    have 2: Nml v ∧ ¬ is-Prim₀ v ∧ ¬ is-Vcomp v ∧ ¬ is-Lunit′ v ∧ ¬ is-Runit′ v
      using uv by (cases u; simp; cases v, simp-all)
    have arr (cod (un-Prim u))
      using 1 by fastforce
    moreover have Nml (Cod v) ∧ ¬ is-Prim₀ v
      using 2 I2 by (cases v, simp-all)
    moreover have ⟨src (cod (un-Prim u))⟩₀ = Trg (Cod v)
    proof −
      have Trg (Cod v) = Src (Cod u)
        using uv Nml-implies-Arr by fastforce
      also have ... = ⟨src (cod (un-Prim u))⟩₀
        using 1 by fastforce
      finally show ?thesis by argo
    qed
    moreover have ¬ is-Prim₀ (Cod v)
      using 2 by (cases v; simp)
    ultimately show ?thesis using 1 2 by simp
  qed
qed

lemma Nml-Inv [simp]:
assumes Can t and Nml t
shows Nml (Inv t)
proof −
  have Can t ∧ Nml t ⟹ Nml (Inv t)
    apply (induct t, simp-all)
  proof −
    fix u v
    assume I1: Nml u ⟹ Nml (Inv u)
    assume I2: Nml v ⟹ Nml (Inv v)
    assume uv: Can u ∧ Can v ∧ Src u = Trg v ∧ Nml (u ⋆ v)
    show Nml (Inv u ⋆ Inv v)
    proof −
```

```
      have u: Arr u ∧ Can u using uv Can-implies-Arr by blast
      have v: Arr v ∧ Can v using uv Can-implies-Arr by blast
      have 1: ⟨un-Prim u⟩ = u ∧ ide (un-Prim u) ∧ Nml u ∧ Nml v ∧ ¬ is-Prim₀ v ∧
              ⟨src (un-Prim u)⟩₀ = Trg v
        using uv Nml-HcompD [of u v] apply simp
        using uv by (cases u, simp-all)
      have 2: ⟨un-Prim (Inv u)⟩ = Inv u ∧ arr (un-Prim (Inv u)) ∧ Nml (Inv u)
        using 1 by (cases u; simp)
      moreover have Nml (Inv v) ∧ ¬ is-Prim₀ (Inv v)
        using 1 I2 by (cases v, simp-all)
      moreover have ⟨src (un-Prim (Inv u))⟩₀ = Trg (Inv v)
        using 1 2 v by (cases u, simp-all)
      ultimately show ?thesis
        by (cases u, simp-all)
    qed
  qed
  thus ?thesis using assms by blast
qed
```

The following function defines a horizontal composition for normal terms. If such terms are regarded as lists, this is just (typed) list concatenation.

```
fun HcompNml (infixr ‹⌊⋆⌋› 53)
where ⟨ν⟩₀ ⌊⋆⌋ u = u
    | t ⌊⋆⌋ ⟨μ⟩₀ = t
    | ⟨ν⟩ ⌊⋆⌋ u = ⟨ν⟩ ⋆ u
    | (t ⋆ u) ⌊⋆⌋ v = t ⌊⋆⌋ (u ⌊⋆⌋ v)
    | t ⌊⋆⌋ u = undefined

lemma HcompNml-Prim [simp]:
assumes ¬ is-Prim₀ t
shows ⟨ν⟩ ⌊⋆⌋ t = ⟨ν⟩ ⋆ t
  using assms by (cases t, simp-all)

lemma HcompNml-term-Prim₀ [simp]:
assumes Src t = Trg ⟨μ⟩₀
shows t ⌊⋆⌋ ⟨μ⟩₀ = t
  using assms by (cases t, simp-all)

lemma HcompNml-Nml:
assumes Nml (t ⋆ u)
shows t ⌊⋆⌋ u = t ⋆ u
  using assms HcompNml-Prim by (metis Nml-HcompD(1) Nml-HcompD(5))

lemma Nml-HcompNml:
assumes Nml t and Nml u and Src t = Trg u
shows Nml (t ⌊⋆⌋ u)
and Dom (t ⌊⋆⌋ u) = Dom t ⌊⋆⌋ Dom u
and Cod (t ⌊⋆⌋ u) = Cod t ⌊⋆⌋ Cod u
and Src (t ⌊⋆⌋ u) = Src u
```

and $Trg\ (t\ \lfloor\star\rfloor\ u) = Trg\ t$
proof −
  have *0*: $\bigwedge u.\ [\![\ Nml\ t;\ Nml\ u;\ Src\ t = Trg\ u\ ]\!] \Longrightarrow$
    $Nml\ (t\ \lfloor\star\rfloor\ u) \wedge Dom\ (t\ \lfloor\star\rfloor\ u) = Dom\ t\ \lfloor\star\rfloor\ Dom\ u \wedge$
      $Cod\ (t\ \lfloor\star\rfloor\ u) = Cod\ t\ \lfloor\star\rfloor\ Cod\ u \wedge$
      $Src\ (t\ \lfloor\star\rfloor\ u) = Src\ u \wedge Trg\ (t\ \lfloor\star\rfloor\ u) = Trg\ t$
  proof (*induct t*, *simp-all add*: *Nml-HcompD*(*1*−*4*))
    fix $\nu :: 'a$ **and** $u :: 'a\ term$
    assume ν: $arr\ \nu$
    assume u: $Nml\ u$
    assume *1*: $\langle src\ \nu\rangle_0 = Trg\ u$
    show $Nml\ (\langle\nu\rangle\ \lfloor\star\rfloor\ u) \wedge Dom\ (\langle\nu\rangle\ \lfloor\star\rfloor\ u) = \langle dom\ \nu\rangle\ \lfloor\star\rfloor\ Dom\ u \wedge$
      $Cod\ (\langle\nu\rangle\ \lfloor\star\rfloor\ u) = \langle cod\ \nu\rangle\ \lfloor\star\rfloor\ Cod\ u \wedge$
      $Src\ (\langle\nu\rangle\ \lfloor\star\rfloor\ u) = Src\ u \wedge Trg\ (\langle\nu\rangle\ \lfloor\star\rfloor\ u) = \langle trg\ \nu\rangle_0$
      using $u\ \nu$ *1* **by** (*cases u*, *simp-all*)
    next
    fix $u\ v\ w$
    assume *I1*: $\bigwedge x.\ Nml\ x \Longrightarrow Src\ v = Trg\ x \Longrightarrow$
      $Nml\ (v\ \lfloor\star\rfloor\ x) \wedge Dom\ (v\ \lfloor\star\rfloor\ x) = Dom\ v\ \lfloor\star\rfloor\ Dom\ x \wedge$
        $Cod\ (v\ \lfloor\star\rfloor\ x) = Cod\ v\ \lfloor\star\rfloor\ Cod\ x \wedge$
        $Src\ (v\ \lfloor\star\rfloor\ x) = Src\ x \wedge Trg\ (v\ \lfloor\star\rfloor\ x) = Trg\ v$
    assume *I2*: $\bigwedge x.\ Nml\ x \Longrightarrow Trg\ u = Trg\ x \Longrightarrow$
      $Nml\ (w\ \lfloor\star\rfloor\ x) \wedge Dom\ (w\ \lfloor\star\rfloor\ x) = Dom\ w\ \lfloor\star\rfloor\ Dom\ x \wedge$
        $Cod\ (w\ \lfloor\star\rfloor\ x) = Cod\ w\ \lfloor\star\rfloor\ Cod\ x \wedge$
        $Src\ (w\ \lfloor\star\rfloor\ x) = Src\ x \wedge Trg\ (w\ \lfloor\star\rfloor\ x) = Trg\ w$
    assume vw: $Nml\ (v \star w)$
    assume u: $Nml\ u$
    assume wu: $Src\ w = Trg\ u$
    show $Nml\ ((v \star w)\ \lfloor\star\rfloor\ u) \wedge$
      $Dom\ ((v \star w)\ \lfloor\star\rfloor\ u) = (Dom\ v \star Dom\ w)\ \lfloor\star\rfloor\ Dom\ u \wedge$
      $Cod\ ((v \star w)\ \lfloor\star\rfloor\ u) = (Cod\ v \star Cod\ w)\ \lfloor\star\rfloor\ Cod\ u \wedge$
      $Src\ ((v \star w)\ \lfloor\star\rfloor\ u) = Src\ u \wedge Trg\ ((v \star w)\ \lfloor\star\rfloor\ u) = Trg\ v$
    proof −
      have v: $v = \langle un\text{-}Prim\ v\rangle \wedge Nml\ v$
        using *vw Nml-implies-Arr Nml-HcompD* **by** *metis*
      have w: $Nml\ w \wedge \neg\ is\text{-}Prim_0\ w \wedge \langle src\ (un\text{-}Prim\ v)\rangle_0 = Trg\ w$
        using *vw* **by** (*simp add*: *Nml-HcompD*)
      have $is\text{-}Prim_0\ u \Longrightarrow$ *?thesis* **by** (*cases u*; *simp add*: *vw wu*)
      moreover have $\neg\ is\text{-}Prim_0\ u \Longrightarrow$ *?thesis*
      proof −
        assume *1*: $\neg\ is\text{-}Prim_0\ u$
        have $Src\ v = Trg\ (w\ \lfloor\star\rfloor\ u)$
          using $u\ v\ w$ *I2* [*of u*] **by** (*cases v*, *simp-all*)
        hence $Nml\ (v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u) \wedge$
          $Dom\ (v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u) = Dom\ v\ \lfloor\star\rfloor\ Dom\ (w\ \lfloor\star\rfloor\ u) \wedge$
          $Cod\ (v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u) = Cod\ v\ \lfloor\star\rfloor\ Cod\ (w\ \lfloor\star\rfloor\ u) \wedge$
          $Src\ (v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u) = Src\ u \wedge Trg\ (v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u) = Trg\ v$
          using $u\ v\ w$ *I1* [*of* $w\ \lfloor\star\rfloor\ u$] *I2* [*of u*] **by** *argo*
        moreover have $v\ \lfloor\star\rfloor\ w\ \lfloor\star\rfloor\ u = (v \star w)\ \lfloor\star\rfloor\ u$

```
        using 1 by (cases u, simp-all)
      moreover have (Dom v ⋆ Dom w) ⌊⋆⌋ Dom u = Dom v ⌊⋆⌋ Dom (w ⌊⋆⌋ u)
        using v w u vw 1 I2 Nml-Dom HcompNml-Prim Nml-HcompD(1) Nml-HcompD(5)
        by (cases u, simp-all)
      moreover have (Cod v ⋆ Cod w) ⌊⋆⌋ Cod u = Cod v ⌊⋆⌋ Cod (w ⌊⋆⌋ u)
        using v w u vw 1 I2 Nml-HcompD(1) Nml-HcompD(5) HcompNml-Prim
        by (cases u, simp-all)
      ultimately show ?thesis
        by argo
    qed
    ultimately show ?thesis by blast
  qed
  next
  fix a u
  assume a: obj a
  assume u: Nml u
  assume au: ⟨a⟩0 = Trg u
  show Nml (Trg u ⌊⋆⌋ u) ∧
       Dom (Trg u ⌊⋆⌋ u) = Dom (Trg u) ⌊⋆⌋ Dom u ∧
       Cod (Trg u ⌊⋆⌋ u) = Cod (Trg u) ⌊⋆⌋ Cod u ∧
       Src (Trg u ⌊⋆⌋ u) = Src u ∧ Trg (Trg u ⌊⋆⌋ u) = Trg (Trg u)
    using au
    by (metis Cod.simps(1) Dom.simps(1) HcompNml.simps(1) Trg.simps(1) u)
 qed
 show Nml (t ⌊⋆⌋ u)  using assms 0 by blast
 show Dom (t ⌊⋆⌋ u) = Dom t ⌊⋆⌋ Dom u using assms 0 by blast
 show Cod (t ⌊⋆⌋ u) = Cod t ⌊⋆⌋ Cod u using assms 0 by blast
 show Src (t ⌊⋆⌋ u) = Src u using assms 0 by blast
 show Trg (t ⌊⋆⌋ u) = Trg t using assms 0 by blast
qed

lemma HcompNml-in-Hom [intro]:
assumes Nml t and Nml u and Src t = Trg u
shows t ⌊⋆⌋ u ∈ HHom (Src u) (Trg t)
and t ⌊⋆⌋ u ∈ VHom (Dom t ⌊⋆⌋ Dom u) (Cod t ⌊⋆⌋ Cod u)
  using assms Nml-HcompNml Nml-implies-Arr by auto

lemma Src-HcompNml:
assumes Nml t and Nml u and Src t = Trg u
shows Src (t ⌊⋆⌋ u) = Src u
  using assms Nml-HcompNml(4) by simp

lemma Trg-HcompNml:
assumes Nml t and Nml u and Src t = Trg u
shows Trg (t ⌊⋆⌋ u) = Trg t
  using assms Nml-HcompNml(5) by simp

lemma Dom-HcompNml:
assumes Nml t and Nml u and Src t = Trg u
```

shows *Dom* $(t \lfloor\star\rfloor\ u) = Dom\ t \lfloor\star\rfloor\ Dom\ u$
  using *assms Nml-HcompNml*(*2*) **by** *simp*

lemma *Cod-HcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows *Cod* $(t \lfloor\star\rfloor\ u) = Cod\ t \lfloor\star\rfloor\ Cod\ u$
  using *assms Nml-HcompNml*(*3*) **by** *simp*

lemma *is-Hcomp-HcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
and $\neg$ *is-Prim*$_0$ *t* **and** $\neg$ *is-Prim*$_0$ *u*
shows *is-Hcomp* $(t \lfloor\star\rfloor\ u)$
proof −
  have $[\![\ \neg$ *is-Hcomp* $(t \lfloor\star\rfloor\ u)$; *Nml t*; *Nml u*; *Src t* = *Trg u*; $\neg$ *is-Prim*$_0$ *t*; $\neg$ *is-Prim*$_0$ *u* $]\!]$
    $\Longrightarrow$ *False*
  proof (*induct t*, *simp-all add*: *Nml-HcompD*)
    fix *a*
    assume *a*: *obj a*
    assume *u*: *Nml u*
    assume *1*: $\neg$ *is-Hcomp u*
    assume *2*: $\neg$ *is-Prim*$_0$ (*Trg u*)
    show *False*
      using *u 1 2* **by** (*cases u*; *simp*)
    next
    fix *v w*
    assume *I2*: $\neg$ *is-Hcomp* $(w \lfloor\star\rfloor\ u) \Longrightarrow$ *False*
    assume *vw*: *Nml* $(v \star w)$
    assume *u*: *Nml u*
    assume *1*: $\neg$ *is-Hcomp* $((v \star w) \lfloor\star\rfloor\ u)$
    assume *2*: $\neg$ *is-Prim*$_0$ *u*
    assume *3*: *Src w* = *Trg u*
    show *False*
    proof −
      have *v*: $v = \langle$*un-Prim v*$\rangle$
        using *vw Nml-HcompD* **by** *metis*
      have *w*: *Nml w* $\wedge$ $\neg$ *is-Prim*$_0$ *w* $\wedge$ $\langle$*src* (*un-Prim v*)$\rangle_0$ = *Trg w*
        using *vw Nml-HcompD* [*of v w*] **by** *blast*
      have $(v \star w) \lfloor\star\rfloor\ u = v \star (w \lfloor\star\rfloor\ u)$
      proof −
        have $(v \star w) \lfloor\star\rfloor\ u = \langle$*un-Prim v*$\rangle \lfloor\star\rfloor\ (w \lfloor\star\rfloor\ u)$
          using *u v 2* **by** (*cases u*, *simp-all*)
        also have ... $= \langle$*un-Prim v*$\rangle \star (w \lfloor\star\rfloor\ u)$
          using *u w I2* **by** *fastforce*
        also have ... $= v \star (w \lfloor\star\rfloor\ u)$
          using *v* **by** *simp*
        finally show *?thesis* **by** *simp*
      qed
      thus *?thesis* **using** *1* **by** *simp*
    qed

qed
thus *?thesis* **using** *assms* **by** *blast*
qed

The following function defines the "dimension" of a term, which is the number of inputs (or outputs) when the term is regarded as an interconnection matrix. For normal terms, this is just the length of the term when regarded as a list of arrows of C. This function is used as a ranking of terms in the subsequent associativity proof.

primrec *dim* :: *'a term* ⇒ *nat*
where *dim* $\langle\mu\rangle_0$ = *0*
| *dim* $\langle\mu\rangle$ = *1*
| *dim* (*t* ⋆ *u*) = (*dim t* + *dim u*)
| *dim* (*t* · *u*) = *dim t*
| *dim* $\mathbf{l}[t]$ = *dim t*
| *dim* $\mathbf{l}^{-1}[t]$ = *dim t*
| *dim* $\mathbf{r}[t]$ = *dim t*
| *dim* $\mathbf{r}^{-1}[t]$ = *dim t*
| *dim* $\mathbf{a}[t, u, v]$ = *dim t* + *dim u* + *dim v*
| *dim* $\mathbf{a}^{-1}[t, u, v]$ = *dim t* + *dim u* + *dim v*

lemma *HcompNml-assoc*:
assumes *Nml t* **and** *Nml u* **and** *Nml v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows (*t* ⌊⋆⌋ *u*) ⌊⋆⌋ *v* = *t* ⌊⋆⌋ (*u* ⌊⋆⌋ *v*)
proof −
have ⋀*n t u v*. ⟦ *dim t* = *n*; *Nml t*; *Nml u*; *Nml v*; *Src t* = *Trg u*; *Src u* = *Trg v* ⟧ ⟹
(*t* ⌊⋆⌋ *u*) ⌊⋆⌋ *v* = *t* ⌊⋆⌋ (*u* ⌊⋆⌋ *v*)
proof −
fix *n*
show ⋀*t u v*. ⟦ *dim t* = *n*; *Nml t*; *Nml u*; *Nml v*; *Src t* = *Trg u*; *Src u* = *Trg v* ⟧ ⟹
(*t* ⌊⋆⌋ *u*) ⌊⋆⌋ *v* = *t* ⌊⋆⌋ (*u* ⌊⋆⌋ *v*)
proof (*induction n rule*: *nat-less-induct*)
fix *n* :: *nat* **and** *t* :: *'a term* **and** *u v*
assume *I*: ∀ *m*<*n*. ∀ *t u v*. *dim t* = *m* ⟶ *Nml t* ⟶ *Nml u* ⟶ *Nml v* ⟶
Src t = *Trg u* ⟶ *Src u* = *Trg v* ⟶
(*t* ⌊⋆⌋ *u*) ⌊⋆⌋ *v* = *t* ⌊⋆⌋ (*u* ⌊⋆⌋ *v*)
assume *dim*: *dim t* = *n*
assume *t*: *Nml t*
assume *u*: *Nml u*
assume *v*: *Nml v*
assume *tu*: *Src t* = *Trg u*
assume *uv*: *Src u* = *Trg v*
show (*t* ⌊⋆⌋ *u*) ⌊⋆⌋ *v* = *t* ⌊⋆⌋ (*u* ⌊⋆⌋ *v*)
proof −
have *is-Prim*$_0$ *t* ⟹ *?thesis* **by** (*cases t*; *simp*)
moreover have ¬*is-Prim*$_0$ *t* ⟹ *is-Prim*$_0$ *u* ⟹ *?thesis*
by (*cases t*; *simp*; *cases u*; *simp*)
moreover have ¬ *is-Prim*$_0$ *t* ⟹ ¬ *is-Prim*$_0$ *u* ⟹ *is-Prim*$_0$ *v* ⟹ *?thesis*
proof −
assume *1*: ¬ *is-Prim*$_0$ *t*

```
    assume 2: ¬ is-Prim0 u
    assume 3: is-Prim0 v
    have ¬is-Prim0 (t ⌊⋆⌋ u)
      using 1 2 t u tu is-Hcomp-HcompNml [of t u]
      by (cases t, simp, cases u, auto)
    thus ?thesis
      using 1 2 3 tu uv by (cases v, simp, cases t ⌊⋆⌋ u, simp-all)
  qed
  moreover have ¬is-Prim0 t ∧ ¬ is-Prim0 u ∧ ¬ is-Prim0 v ∧ is-Prim t ⟹ ?thesis
    using v by (cases t, simp-all, cases u, simp-all; cases v, simp-all)
 moreover have ¬is-Prim0 t ∧ ¬ is-Prim0 u ∧ ¬ is-Prim0 v ∧ is-Hcomp t ⟹ ?thesis
  proof (cases t, simp-all)
    fix w :: 'a term and x :: 'a term
    assume 1: ¬ is-Prim0 u ∧ ¬ is-Prim0 v
    assume 2: t = (w ⋆ x)
    show ((w ⋆ x) ⌊⋆⌋ u) ⌊⋆⌋ v = (w ⋆ x) ⌊⋆⌋ (u ⌊⋆⌋ v)
    proof −
      have w: w = ⟨un-Prim w⟩
        using t 1 2 Nml-HcompD by metis
      have x: Nml x
        using t w 1 2 by (metis Nml.simps(3))
      have ((w ⋆ x) ⌊⋆⌋ u) ⌊⋆⌋ v = (w ⌊⋆⌋ (x ⌊⋆⌋ u)) ⌊⋆⌋ v
        using u v w x 1 2 by (cases u, simp-all)
      also have ... = (w ⋆ (x ⌊⋆⌋ u)) ⌊⋆⌋ v
        using t w u 1 2 HcompNml-Prim is-Hcomp-HcompNml Nml-HcompD
        by (metis Src.simps(3) term.distinct-disc(3) tu)
      also have ... = w ⌊⋆⌋ ((x ⌊⋆⌋ u) ⌊⋆⌋ v)
        using u v w x 1 by (cases u, simp-all; cases v, simp-all)
      also have ... = w ⌊⋆⌋ (x ⌊⋆⌋ (u ⌊⋆⌋ v))
      proof −
        have dim x < dim t
          using 2 w by (cases w; simp)
        moreover have Src x = Trg u ∧ Src u = Trg v
          using tu uv 2 by auto
        ultimately show ?thesis
          using u v x dim I by simp
      qed
      also have ... = (w ⋆ x) ⌊⋆⌋ (u ⌊⋆⌋ v)
      proof −
        have 3: is-Hcomp (u ⌊⋆⌋ v)
          using u v uv 1 is-Hcomp-HcompNml by auto
        obtain u' :: 'a term and v' where uv': u ⌊⋆⌋ v = u' ⋆ v'
          using 3 is-Hcomp-def by blast
        thus ?thesis by simp
      qed
      finally show ?thesis by simp
    qed
  qed
  moreover have is-Prim0 t ∨ is-Prim t ∨ is-Hcomp t
```

```
        using t by (cases t, simp-all)
      ultimately show ?thesis by blast
    qed
  qed
 qed
 thus ?thesis using assms by blast
qed

lemma HcompNml-Trg-Nml:
assumes Nml t
shows Trg t ⌊⋆⌋ t = t
proof −
 have Nml t ⟹ Trg t ⌊⋆⌋ t = t
 proof (induct t, simp-all add: Nml-HcompD)
  fix u v
  assume uv: Nml (u ⋆ v)
  assume I1: Trg u ⌊⋆⌋ u = u
  have 1: Nml u ∧ Nml v ∧ Src u = Trg v
    using uv Nml-HcompD by blast
  have 2: Trg (u ⋆ v) = Trg u
    using uv by auto
  show Trg u ⌊⋆⌋ u ⋆ v = u ⋆ v
  proof −
    have Trg u ⌊⋆⌋ u ⋆ v = Trg u ⌊⋆⌋ u ⌊⋆⌋ v
      using uv HcompNml-Nml by simp
    also have ... = (Trg u ⌊⋆⌋ u) ⌊⋆⌋ v
      using 1 2 HcompNml-assoc Src-Trg Nml-Trg Nml-implies-Arr by simp
    also have ... = u ⋆ v
      using I1 uv HcompNml-Nml by simp
    finally show ?thesis by simp
  qed
 qed
 thus ?thesis using assms by simp
qed

lemma HcompNml-Nml-Src:
assumes Nml t
shows t ⌊⋆⌋ Src t = t
proof −
 have Nml t ⟹ t ⌊⋆⌋ Src t = t
 proof (induct t, simp-all add: Nml-HcompD)
  fix u v
  assume uv: Nml (u ⋆ v)
  assume I2: v ⌊⋆⌋ Src v = v
  have 1: Nml u ∧ Nml v ∧ Src u = Trg v
    using uv Nml-HcompD by blast
  have 2: Src (u ⋆ v) = Src v
    using uv Trg-HcompNml by auto
  show (u ⋆ v) ⌊⋆⌋ Src v = u ⋆ v
```

```
    proof −
      have (u ⋆ v) ⌊⋆⌋ Src v = (u ⌊⋆⌋ v) ⌊⋆⌋ Src v
        using uv HcompNml-Nml by simp
      also have ... = u ⌊⋆⌋ (v ⌊⋆⌋ Src v)
        using 1 2 HcompNml-assoc Trg-Src Nml-Src Nml-implies-Arr by simp
      also have ... = u ⋆ v
        using I2 uv HcompNml-Nml by simp
      finally show ?thesis by simp
    qed
  qed
  thus ?thesis using assms by simp
qed

lemma HcompNml-Obj-Nml:
assumes Obj t and Nml u and Src t = Trg u
shows t ⌊⋆⌋ u = u
  using assms by (cases t, simp-all add: HcompNml-Trg-Nml)

lemma HcompNml-Nml-Obj:
assumes Nml t and Obj u and Src t = Trg u
shows t ⌊⋆⌋ u = t
  using assms by (cases u, simp-all)

lemma Ide-HcompNml:
assumes Ide t and Ide u and Nml t and Nml u and Src t = Trg u
shows Ide (t ⌊⋆⌋ u)
  using assms
  by (metis (mono-tags, lifting) Nml-HcompNml(1) Nml-implies-Arr Dom-HcompNml
      Ide-Dom Ide-in-Hom(2) mem-Collect-eq)

lemma Can-HcompNml:
assumes Can t and Can u and Nml t and Nml u and Src t = Trg u
shows Can (t ⌊⋆⌋ u)
proof −
  have ⋀u. ⟦ Can t ∧ Nml t; Can u ∧ Nml u; Src t = Trg u ⟧ ⟹ Can (t ⌊⋆⌋ u)
  proof (induct t, simp-all add: HcompNml-Trg-Nml HcompNml-Nml-Src)
   show ⋀x u. ide x ∧ arr x ⟹ Can u ∧ Nml u ⟹ ⟨src x⟩0 = Trg u ⟹ Can (⟨x⟩ ⌊⋆⌋ u)
     by (metis Ide.simps(1−2) Ide-implies-Can Can.simps(3) Nml.elims(2)
         Nml.simps(2) HcompNml.simps(12) HcompNml-Prim Ide-HcompNml
         Src.simps(2) term.disc(2))
   show ⋀v w u.
          (⋀u. Nml v ⟹ Can u ∧ Nml u ⟹ Trg w = Trg u ⟹ Can (v ⌊⋆⌋ u)) ⟹
          (⋀ua. Nml w ⟹ Can ua ∧ Nml ua ⟹ Trg u = Trg ua ⟹ Can (w ⌊⋆⌋ ua)) ⟹
          Can v ∧ Can w ∧ Src v = Trg w ∧ Nml (v ⋆ w) ⟹
          Can u ∧ Nml u ⟹ Src w = Trg u ⟹ Can ((v ⋆ w) ⌊⋆⌋ u)
     by (metis Nml-HcompD(3−4) HcompNml-Nml Nml-HcompNml(1)
         HcompNml-assoc Trg-HcompNml)
  qed
  thus ?thesis using assms by blast
```

qed

lemma *Inv-HcompNml*:
assumes $Can\ t$ **and** $Can\ u$ **and** $Nml\ t$ **and** $Nml\ u$ **and** $Src\ t = Trg\ u$
shows $Inv\ (t \lfloor\star\rfloor\ u) = Inv\ t \lfloor\star\rfloor\ Inv\ u$
proof −
  have $\bigwedge u.\ [\![\ Can\ t \wedge Nml\ t;\ Can\ u \wedge Nml\ u;\ Src\ t = Trg\ u\ ]\!]$
    $\Longrightarrow Inv\ (t \lfloor\star\rfloor\ u) = Inv\ t \lfloor\star\rfloor\ Inv\ u$
  proof (*induct t*, *simp-all add*: *HcompNml-Trg-Nml HcompNml-Nml-Src*)
    show $\bigwedge x\ u.\ [\![\ obj\ x;\ Can\ u \wedge Nml\ u;\ \langle x\rangle_0 = Trg\ u\ ]\!] \Longrightarrow Inv\ u = Inv\ (Trg\ u) \lfloor\star\rfloor\ Inv\ u$
      by (*metis HcompNml.simps*(*1*) *Inv.simps*(*1*))
    show $\bigwedge x\ u.\ ide\ x \wedge arr\ x \Longrightarrow Can\ u \wedge Nml\ u \Longrightarrow \langle src\ x\rangle_0 = Trg\ u \Longrightarrow$
      $Inv\ (\langle x\rangle \lfloor\star\rfloor\ u) = \langle x\rangle \lfloor\star\rfloor\ Inv\ u$
      by (*metis Ide.simps*(*2*) *HcompNml.simps*(*2*) *HcompNml-Prim Inv.simps*(*1*,*3*)
        Inv-Ide Inv-Inv is-Prim$_0$*-def*)
    fix *v w u*
    assume *I1*: $\bigwedge x.\ Nml\ v \Longrightarrow Can\ x \wedge Nml\ x \Longrightarrow Trg\ w = Trg\ x \Longrightarrow$
      $Inv\ (v \lfloor\star\rfloor\ x) = Inv\ v \lfloor\star\rfloor\ Inv\ x$
    assume *I2*: $\bigwedge x.\ Nml\ w \Longrightarrow Can\ x \wedge Nml\ x \Longrightarrow Trg\ u = Trg\ x \Longrightarrow$
      $Inv\ (w \lfloor\star\rfloor\ x) = Inv\ w \lfloor\star\rfloor\ Inv\ x$
    assume *vw*: $Can\ v \wedge Can\ w \wedge Src\ v = Trg\ w \wedge Nml\ (v \star w)$
    assume *wu*: $Src\ w = Trg\ u$
    assume *u*: $Can\ u \wedge Nml\ u$
    have *v*: $Can\ v \wedge Nml\ v$
      using *vw Nml-HcompD* **by** *blast*
    have *w*: $Can\ w \wedge Nml\ w$
      using *v vw* **by** (*cases v*, *simp-all*)
    show $Inv\ ((v \star w) \lfloor\star\rfloor\ u) = (Inv\ v \star Inv\ w) \lfloor\star\rfloor\ Inv\ u$
    proof −
      have $is\text{-}Prim_0\ u \Longrightarrow ?thesis$
        apply (*cases u*) **by** *simp-all*
      moreover have $\neg\ is\text{-}Prim_0\ u \Longrightarrow ?thesis$
      proof −
        assume *1*: $\neg\ is\text{-}Prim_0\ u$
        have $Inv\ ((v \star w) \lfloor\star\rfloor\ u) = Inv\ (v \lfloor\star\rfloor\ (w \lfloor\star\rfloor\ u))$
          using *1* **by** (*cases u*, *simp-all*)
        also have ... $= Inv\ v \lfloor\star\rfloor\ Inv\ (w \lfloor\star\rfloor\ u)$
          using *u v w vw wu I1 Nml-HcompNml Can-HcompNml Nml-Inv Can-Inv*
          by *simp*
        also have ... $= Inv\ v \lfloor\star\rfloor\ (Inv\ w \lfloor\star\rfloor\ Inv\ u)$
          using *u v w I2 Nml-HcompNml* **by** *simp*
        also have ... $= (Inv\ v \star Inv\ w) \lfloor\star\rfloor\ Inv\ u$
          using *v 1* **by** (*cases u*, *simp-all*)
        finally show *?thesis* **by** *blast*
      qed
      ultimately show *?thesis* **by** *blast*
    qed
  qed
  thus *?thesis* **using** *assms* **by** *blast*

qed

The following function defines vertical composition for compatible normal terms, by "pushing the composition down" to arrows of V.

fun *VcompNml* :: *'a term* ⇒ *'a term* ⇒ *'a term* (**infixr** ‹⌊·⌋› *55*)
where $\langle\nu\rangle_0$ ⌊·⌋ $u = u$
| $\langle\nu\rangle$ ⌊·⌋ $\langle\mu\rangle = \langle\nu \cdot \mu\rangle$
| $(u \star v)$ ⌊·⌋ $(w \star x) = (u$ ⌊·⌋ $w \star v$ ⌊·⌋ $x)$
| t ⌊·⌋ $\langle\mu\rangle_0 = t$
| t ⌊·⌋ - = *undefined* · *undefined*

Note that the last clause above is not relevant to normal terms. We have chosen a provably non-normal value in order to validate associativity.

lemma *Nml-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Nml* (t ⌊·⌋ u)
and *Dom* (t ⌊·⌋ u) = *Dom u*
and *Cod* (t ⌊·⌋ u) = *Cod t*
proof −
have *0*: ⋀*u*. ⟦ *Nml t*; *Nml u*; *Dom t* = *Cod u* ⟧ ⟹
Nml (t ⌊·⌋ u) ∧ *Dom* (t ⌊·⌋ u) = *Dom u* ∧ *Cod* (t ⌊·⌋ u) = *Cod t*
proof (*induct t*, *simp-all add*: *Nml-HcompD*)
show ⋀*x u*. *obj x* ⟹ *Nml u* ⟹ $\langle x\rangle_0$ = *Cod u* ⟹
Nml (*Cod u* ⌊·⌋ u) ∧ *Dom* (*Cod u* ⌊·⌋ u) = *Dom u* ∧
Cod (*Cod u* ⌊·⌋ u) = *Cod* (*Cod u*)
by (*metis Cod.simps*(*1*) *VcompNml.simps*(*1*))
fix ν u
assume ν: *arr* ν
assume u: *Nml u*
assume *1*: $\langle$*dom* $\nu\rangle$ = *Cod u*
show *Nml* ($\langle\nu\rangle$ ⌊·⌋ u) ∧ *Dom* ($\langle\nu\rangle$ ⌊·⌋ u) = *Dom u* ∧ *Cod* ($\langle\nu\rangle$ ⌊·⌋ u) = $\langle$*cod* $\nu\rangle$
using ν u *1* **by** (*cases u*, *simp-all*)
next
fix u v w
assume *I2*: ⋀*u*. ⟦ *Nml u*; *Dom w* = *Cod u* ⟧ ⟹
Nml (w ⌊·⌋ u) ∧ *Dom* (w ⌊·⌋ u) = *Dom u* ∧ *Cod* (w ⌊·⌋ u) = *Cod w*
assume *vw*: *Nml* ($v \star w$)
have v: *Nml v*
using *vw Nml-HcompD* **by** *blast*
have w: *Nml w*
using *vw Nml-HcompD* **by** *blast*
assume u: *Nml u*
assume *1*: (*Dom* $v \star$ *Dom* w) = *Cod u*
show *Nml* (($v \star w$) ⌊·⌋ u) ∧ *Dom* (($v \star w$) ⌊·⌋ u) = *Dom u* ∧
Cod (($v \star w$) ⌊·⌋ u) = *Cod* $v \star$ *Cod* w
using u v w *1*
proof (*cases u*, *simp-all*)
fix x y
assume *2*: $u = x \star y$

```
      have 4: is-Prim x ∧ x = ⟨un-Prim x⟩ ∧ arr (un-Prim x) ∧ Nml y ∧ ¬ is-Prim₀ y
        using u 2 by (cases x, cases y, simp-all)
      have 5: is-Prim v ∧ v = ⟨un-Prim v⟩ ∧ arr (un-Prim v) ∧ Nml w ∧ ¬ is-Prim₀ w
        using v w vw by (cases v, simp-all)
      have 6: dom (un-Prim v) = cod (un-Prim x) ∧ Dom w = Cod y
      proof −
        have ⟨src (un-Prim v)⟩₀ = Trg w using vw Nml-HcompD [of v w] by simp
        thus dom (un-Prim v) = cod (un-Prim x) ∧ Dom w = Cod y
          using 1 2 4 5 apply (cases u, simp-all)
          by (metis Cod.simps(2) Dom.simps(2) term.simps(2))
      qed
      have (v ⋆ w) ⌊·⌋ u = ⟨un-Prim v · un-Prim x⟩ ⋆ w ⌊·⌋ y
        using 2 4 5 6 VcompNml.simps(2) [of un-Prim v un-Prim x] by simp
      moreover have Nml (⟨un-Prim v · un-Prim x⟩ ⋆ w ⌊·⌋ y)
      proof −
        have Nml (w ⌊·⌋ y)
          using I2 4 5 6 by simp
        moreover have ¬ is-Prim₀ (w ⌊·⌋ y)
          using vw 4 5 6 I2 Nml-Cod Nml-HcompD(5) is-Prim₀-def
          by (metis Cod.simps(1) Cod.simps(3))
        moreover have ⟨src (un-Prim v · un-Prim x)⟩₀ = Trg (w ⌊·⌋ y)
          using vw 4 5 6 I2 Nml-HcompD(6) Nml-implies-Arr
                src.as-nat-trans.naturality1 src.as-nat-trans.preserves-comp-2
                Trg-Cod src-cod
          by (metis seqI)
        ultimately show ?thesis
          using 4 5 6 Nml.simps(3) [of un-Prim v · un-Prim x (w ⌊·⌋ y)]
          by simp
      qed
      ultimately show Nml (v ⌊·⌋ x ⋆ w ⌊·⌋ y) ∧
                      Dom (v ⌊·⌋ x) = Dom x ∧ Dom (w ⌊·⌋ y) = Dom y ∧
                      Cod (v ⌊·⌋ x) = Cod v ∧ Cod (w ⌊·⌋ y) = Cod w
        using 4 5 6 I2
        by (metis (no-types, lifting) Cod.simps(2) Dom.simps(2) VcompNml.simps(2)
            cod-comp dom-comp seqI)
    qed
  qed
  show Nml (t ⌊·⌋ u) using assms 0 by blast
  show Dom (t ⌊·⌋ u) = Dom u using assms 0 by blast
  show Cod (t ⌊·⌋ u) = Cod t using assms 0 by blast
qed

lemma VcompNml-in-Hom [intro]:
assumes Nml t and Nml u and Dom t = Cod u
shows t ⌊·⌋ u ∈ HHom (Src u) (Trg u) and t ⌊·⌋ u ∈ VHom (Dom u) (Cod t)
proof −
  show 1: t ⌊·⌋ u ∈ VHom (Dom u) (Cod t)
    using assms Nml-VcompNml Nml-implies-Arr by simp
  show t ⌊·⌋ u ∈ HHom (Src u) (Trg u)
```

using *assms 1 Src-Dom Trg-Dom Nml-implies-Arr* **by** *fastforce*
qed

lemma *Src-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Src* (*t* ⌊·⌋ *u*) = *Src u*
using *assms VcompNml-in-Hom* **by** *simp*

lemma *Trg-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Trg* (*t* ⌊·⌋ *u*) = *Trg u*
using *assms VcompNml-in-Hom* **by** *simp*

lemma *Dom-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Dom* (*t* ⌊·⌋ *u*) = *Dom u*
using *assms Nml-VcompNml*(*2*) **by** *simp*

lemma *Cod-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Cod* (*t* ⌊·⌋ *u*) = *Cod t*
using *assms Nml-VcompNml*(*3*) **by** *simp*

lemma *VcompNml-Cod-Nml* [*simp*]:
assumes *Nml t*
shows *VcompNml* (*Cod t*) *t* = *t*
proof −
have *Nml t* ⟹ *Cod t* ⌊·⌋ *t* = *t*
apply (*induct t*)
by (*auto simp add*: *Nml-HcompD comp-cod-arr*)
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *VcompNml-Nml-Dom* [*simp*]:
assumes *Nml t*
shows *t* ⌊·⌋ (*Dom t*) = *t*
proof −
have *Nml t* ⟹ *t* ⌊·⌋ *Dom t* = *t*
apply (*induct t*) **by** (*auto simp add*: *Nml-HcompD comp-arr-dom*)
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *VcompNml-Ide-Nml* [*simp*]:
assumes *Nml t* **and** *Ide a* **and** *Dom a* = *Cod t*
shows *a* ⌊·⌋ *t* = *t*
using *assms Ide-in-Hom* **by** *simp*

lemma *VcompNml-Nml-Ide* [*simp*]:
assumes *Nml t* **and** *Ide a* **and** *Dom t* = *Cod a*

```
shows t ⌊·⌋ a = t
  using assms Ide-in-Hom by auto

lemma VcompNml-assoc:
assumes Nml t and Nml u and Nml v
and Dom t = Cod u and Dom u = Cod v
shows (t ⌊·⌋ u) ⌊·⌋ v = t ⌊·⌋ (u ⌊·⌋ v)
proof −
  have ⋀u v. ⟦ Nml t; Nml u; Nml v; Dom t = Cod u; Dom u = Cod v ⟧ ⟹
              (t ⌊·⌋ u) ⌊·⌋ v = t ⌊·⌋ (u ⌊·⌋ v)
  proof (induct t, simp-all)
    show ⋀x u v. obj x ⟹ Nml u ⟹ Nml v ⟹ ⟨x⟩₀ = Cod u ⟹ Dom u = Cod v ⟹
                 u ⌊·⌋ v = Cod u ⌊·⌋ u ⌊·⌋ v
      by (metis VcompNml.simps(1))
    fix f u v
    assume f: arr f
    assume u: Nml u
    assume v: Nml v
    assume 1: ⟨dom f⟩ = Cod u
    assume 2: Dom u = Cod v
    show (⟨f⟩ ⌊·⌋ u) ⌊·⌋ v = ⟨f⟩ ⌊·⌋ (u ⌊·⌋ v)
      using u v f 1 2 comp-assoc
      apply (cases u)
            apply simp-all
      apply (cases v)
      by simp-all
    next
    fix u v w x
    assume I1: ⋀u v. ⟦ Nml w; Nml u; Nml v; Dom w = Cod u; Dom u = Cod v ⟧ ⟹
                    (w ⌊·⌋ u) ⌊·⌋ v = w ⌊·⌋ (u ⌊·⌋ v)
    assume I2: ⋀u v. ⟦ Nml x; Nml u; Nml v; Dom x = Cod u; Dom u = Cod v ⟧ ⟹
                    (x ⌊·⌋ u) ⌊·⌋ v = x ⌊·⌋ (u ⌊·⌋ v)
    assume wx: Nml (w ⋆ x)
    assume u: Nml u
    assume v: Nml v
    assume 1: (Dom w ⋆ Dom x) = Cod u
    assume 2: Dom u = Cod v
    show ((w ⋆ x) ⌊·⌋ u) ⌊·⌋ v = (w ⋆ x) ⌊·⌋ u ⌊·⌋ v
    proof −
      have w: Nml w
        using wx Nml-HcompD by blast
      have x: Nml x
        using wx Nml-HcompD by blast
      have is-Hcomp u
        using u 1 by (cases u) simp-all
      thus ?thesis
        using u v apply (cases u, simp-all, cases v, simp-all)
      proof −
        fix u1 u2 v1 v2
```

```
      assume 3: u = (u1 ⋆ u2)
      assume 4: v = (v1 ⋆ v2)
      show (w ⌊·⌋ u1) ⌊·⌋ v1 = w ⌊·⌋ u1 ⌊·⌋ v1 ∧
           (x ⌊·⌋ u2) ⌊·⌋ v2 = x ⌊·⌋ u2 ⌊·⌋ v2
      proof −
        have Nml u1 ∧ Nml u2
          using u 3 Nml-HcompD by blast
        moreover have Nml v1 ∧ Nml v2
          using v 4 Nml-HcompD by blast
        ultimately show ?thesis using w x I1 I2 1 2 3 4 by simp
      qed
    qed
  qed
  thus ?thesis using assms by blast
qed

lemma Ide-VcompNml:
assumes Ide t and Ide u and Nml t and Nml u and Dom t = Cod u
shows Ide (t ⌊·⌋ u)
proof −
  have ⋀u. ⟦ Ide t; Ide u; Nml t; Nml u; Dom t = Cod u ⟧ ⟹ Ide (VcompNml t u)
    by (induct t, simp-all)
  thus ?thesis using assms by blast
qed

lemma Can-VcompNml:
assumes Can t and Can u and Nml t and Nml u and Dom t = Cod u
shows Can (t ⌊·⌋ u)
proof −
  have ⋀u. ⟦ Can t ∧ Nml t; Can u ∧ Nml u; Dom t = Cod u ⟧ ⟹ Can (t ⌊·⌋ u)
  proof (induct t, simp-all)
    fix t u v
    assume I1: ⋀v. ⟦ Nml t; Can v ∧ Nml v; Dom t = Cod v ⟧ ⟹ Can (t ⌊·⌋ v)
    assume I2: ⋀v. ⟦ Nml u; Can v ∧ Nml v; Dom u = Cod v ⟧ ⟹ Can (u ⌊·⌋ v)
    assume tu: Can t ∧ Can u ∧ Src t = Trg u ∧ Nml (t ⋆ u)
    have t: Can t ∧ Nml t
      using tu Nml-HcompD by blast
    have u: Can u ∧ Nml u
      using tu Nml-HcompD by blast
    assume v: Can v ∧ Nml v
    assume 1: (Dom t ⋆ Dom u) = Cod v
    show Can ((t ⋆ u) ⌊·⌋ v)
    proof −
      have 2: (Dom t ⋆ Dom u) = Cod v using 1 by simp
      show ?thesis
        using v 2
      proof (cases v; simp)
        fix w x
```

```
      assume wx: v = (w ⋆ x)
      have Can w ∧ Nml w using v wx Nml-HcompD Can.simps(3) by blast
      moreover have Can x ∧ Nml x using v wx Nml-HcompD Can.simps(3) by blast
      moreover have Dom t = Cod w using 2 wx by simp
      moreover have ux: Dom u = Cod x using 2 wx by simp
      ultimately show Can (t ⌊·⌋ w) ∧ Can (u ⌊·⌋ x) ∧ Src (t ⌊·⌋ w) = Trg (u ⌊·⌋ x)
        using t u v tu wx I1 I2
        by (metis Nml-HcompD(7) Src-VcompNml Trg-VcompNml)
    qed
  qed
 qed
 thus ?thesis using assms by blast
qed

lemma Inv-VcompNml:
assumes Can t and Can u and Nml t and Nml u and Dom t = Cod u
shows Inv (t ⌊·⌋ u) = Inv u ⌊·⌋ Inv t
proof −
 have ⋀u. ⟦ Can t ∧ Nml t; Can u ∧ Nml u; Dom t = Cod u ⟧ ⟹
        Inv (t ⌊·⌋ u) = Inv u ⌊·⌋ Inv t
 proof (induct t, simp-all)
  show ⋀x u. ⟦ obj x; Can u ∧ Nml u; ⟨x⟩₀ = Cod u ⟧ ⟹ Inv u = Inv u ⌊·⌋ Inv (Cod u)
    by (simp add: Can-implies-Arr)
  show ⋀x u. ⟦ ide x ∧ arr x; Can u ∧ Nml u; ⟨x⟩ = Cod u ⟧ ⟹
            Inv u = Inv u ⌊·⌋ Inv (Cod u)
    by (simp add: Can-implies-Arr)
  fix v w u
  assume vw: Can v ∧ Can w ∧ Src v = Trg w ∧ Nml (v ⋆ w)
  have v: Can v ∧ Nml w
    using vw Nml-HcompD by blast
  have w: Can w ∧ Nml w
    using vw Nml-HcompD by blast
  assume I1: ⋀x. ⟦ Nml v; Can x ∧ Nml x; Dom v = Cod x ⟧ ⟹
              Inv (v ⌊·⌋ x) = Inv x ⌊·⌋ Inv v
  assume I2: ⋀x. ⟦ Nml w; Can x ∧ Nml x; Dom w = Cod x ⟧ ⟹
              Inv (w ⌊·⌋ x) = Inv x ⌊·⌋ Inv w
  assume u: Can u ∧ Nml u
  assume 1: (Dom v ⋆ Dom w) = Cod u
  show Inv ((v ⋆ w) ⌊·⌋ u) = Inv u ⌊·⌋ (Inv v ⋆ Inv w)
    using v 1
  proof (cases w, simp-all)
   show ⋀μ. obj μ ⟹ Dom v ⋆ ⟨μ⟩₀ = Cod u ⟹ w = ⟨μ⟩₀ ⟹ Can v ⟹
          Inv ((v ⋆ ⟨μ⟩₀) ⌊·⌋ u) = Inv u ⌊·⌋ (Inv v ⋆ ⟨μ⟩₀)
    using Nml-HcompD(5) is-Prim₀-def vw by blast
   show ⋀μ. arr μ ⟹ Dom v ⋆ ⟨dom μ⟩ = Cod u ⟹ w = ⟨μ⟩ ⟹ Can v ⟹
          Inv ((v ⋆ ⟨μ⟩) ⌊·⌋ u) = Inv u ⌊·⌋ (Inv v ⋆ ⟨inv μ⟩)
    by (metis Ide.simps(2) Can.simps(2) Nml-HcompD(1) Dom.simps(2) Inv-Ide
          Dom-Inv Nml-Inv ideD(2) inv-ide VcompNml-Cod-Nml VcompNml-Nml-Dom
          u vw)
```

```
      show ⋀y z. Nml (y ⋆ z) ⟹ Dom v ⋆ Dom y ⋆ Dom z = Cod u ⟹
                  w = y ⋆ z ⟹ Can v ⟹
                  Inv ((v ⋆ y ⋆ z) ⌊·⌋ u) = Inv u ⌊·⌋ (Inv v ⋆ Inv y ⋆ Inv z)
      proof −
        fix y z
        assume 2: Nml (y ⋆ z)
        assume yz: w = y ⋆ z
        show Inv ((v ⋆ y ⋆ z) ⌊·⌋ u) = Inv u ⌊·⌋ (Inv v ⋆ Inv y ⋆ Inv z)
          using u vw yz I1 I2 1 2 VcompNml-Nml-Ide
          apply (cases u)
                apply simp-all
          by (metis Nml-HcompD(3−4))
      qed
    qed
  qed
  thus ?thesis using assms by blast
qed

lemma Can-and-Nml-implies-Ide:
assumes Can t and Nml t
shows Ide t
proof −
  have ⟦ Can t; Nml t ⟧ ⟹ Ide t
    apply (induct t) by (simp-all add: Nml-HcompD)
  thus ?thesis using assms by blast
qed

lemma VcompNml-Can-Inv [simp]:
assumes Can t and Nml t
shows t ⌊·⌋ Inv t = Cod t
  using assms Can-and-Nml-implies-Ide Ide-in-Hom by simp

lemma VcompNml-Inv-Can [simp]:
assumes Can t and Nml t
shows Inv t ⌊·⌋ t = Dom t
  using assms Can-and-Nml-implies-Ide Ide-in-Hom by simp
```

The next fact is a syntactic version of the interchange law, for normal terms.

```
lemma VcompNml-HcompNml:
assumes Nml t and Nml u and Nml v and Nml w
and VSeq t v and VSeq u w and Src v = Trg w
shows (t ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = (t ⌊·⌋ v) ⌊⋆⌋ (u ⌊·⌋ w)
proof −
  have ⋀u v w. ⟦ Nml t; Nml u; Nml v; Nml w; VSeq t v; VSeq u w;
                Src t = Trg u; Src v = Trg w ⟧ ⟹
                (t ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = (t ⌊·⌋ v) ⌊⋆⌋ (u ⌊·⌋ w)
  proof (induct t, simp-all)
    fix u v w x
    assume u: Nml u
```

```
assume v: Nml v
assume w: Nml w
assume uw: VSeq u w
show ⋀x. Arr v ∧ ⟨x⟩₀ = Cod v ⟹ (Cod v ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = v ⌊⋆⌋ u ⌊·⌋ w
  using u v w uw by (cases v) simp-all
show ⋀x. ⟦ arr x; Arr v ∧ ⟨dom x⟩ = Cod v; ⟨src x⟩₀ = Trg u; Src v = Trg w ⟧ ⟹
        (⟨x⟩ ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = ⟨x⟩ ⌊·⌋ v ⌊⋆⌋ u ⌊·⌋ w
proof −
  fix x
  assume x: arr x
  assume 1: Arr v ∧ ⟨dom x⟩ = Cod v
  assume tu: ⟨src x⟩₀ = Trg u
  assume vw: Src v = Trg w
  show (⟨x⟩ ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = ⟨x⟩ ⌊·⌋ v ⌊⋆⌋ u ⌊·⌋ w
  proof −
    have 2: v = ⟨un-Prim v⟩ ∧ arr (un-Prim v) using v 1 by (cases v) simp-all
    have is-Prim₀ u ⟹ ?thesis
      using u v w x tu uw vw 1 2 Cod.simps(3) VcompNml-Cod-Nml Dom.simps(2)
        HcompNml-Prim HcompNml-term-Prim₀ Nml-HcompNml(3) HcompNml-Trg-Nml
      apply (cases u)
        apply simp-all
      by (cases w, simp-all add: Src-VcompNml)
    moreover have ¬ is-Prim₀ u ⟹ ?thesis
    proof −
      assume 3: ¬ is-Prim₀ u
      hence 4: ¬ is-Prim₀ w using u w uw by (cases u, simp-all; cases w, simp-all)
      have (⟨x⟩ ⌊⋆⌋ u) ⌊·⌋ (v ⌊⋆⌋ w) = (⟨x⟩ ⋆ u) ⌊·⌋ (v ⋆ w)
      proof −
        have ⟨x⟩ ⌊⋆⌋ u = ⟨x⟩ ⋆ u
          using u x 3 HcompNml-Nml by (cases u, simp-all)
        moreover have v ⌊⋆⌋ w = v ⋆ w
          using w 2 4 HcompNml-Nml by (cases v, simp-all; cases w, simp-all)
        ultimately show ?thesis by simp
      qed
      also have 5: ... = (⟨x⟩ ⌊·⌋ v) ⋆ (u ⌊·⌋ w) by simp
      also have ... = (⟨x⟩ ⌊·⌋ v) ⌊⋆⌋ (u ⌊·⌋ w)
        using x u w uw vw 1 2 3 4 5
              HcompNml-Nml HcompNml-Prim Nml-HcompNml(1)
        by (metis Cod.simps(3) Nml.simps(3) Dom.simps(2) Dom.simps(3)
            Nml-VcompNml(1) tu v)
      finally show ?thesis by blast
    qed
    ultimately show ?thesis by blast
  qed
qed
fix t1 t2
assume t12: Nml (t1 ⋆ t2)
assume I1: ⋀u v w. ⟦ Nml t1; Nml u; Nml v; Nml w;
                     Arr v ∧ Dom t1 = Cod v; VSeq u w;
```

$Trg\ t2 = Trg\ u;\ Src\ v = Trg\ w\ \rrbracket \Longrightarrow$
$(t1\ \lfloor\star\rfloor\ u)\ \lfloor\cdot\rfloor\ (v\ \lfloor\star\rfloor\ w) = t1\ \lfloor\cdot\rfloor\ v\ \lfloor\star\rfloor\ u\ \lfloor\cdot\rfloor\ w$
assume *I2*: $\bigwedge u'\ v\ w.\ \llbracket\ Nml\ t2;\ Nml\ u';\ Nml\ v;\ Nml\ w;$
$Arr\ v \wedge Dom\ t2 = Cod\ v;\ VSeq\ u'\ w;$
$Trg\ u = Trg\ u';\ Src\ v = Trg\ w\ \rrbracket \Longrightarrow$
$(t2\ \lfloor\star\rfloor\ u')\ \lfloor\cdot\rfloor\ (v\ \lfloor\star\rfloor\ w) = (t2\ \lfloor\cdot\rfloor\ v)\ \lfloor\star\rfloor\ (u'\ \lfloor\cdot\rfloor\ w)$
have *t1*: $t1 = \langle un\text{-}Prim\ t1\rangle \wedge arr\ (un\text{-}Prim\ t1) \wedge Nml\ t1$
using *t12* **by** (*cases t1*, *simp-all*)
have *t2*: $Nml\ t2 \wedge \neg is\text{-}Prim_0\ t2$
using *t12* **by** (*cases t1*, *simp-all*)
assume *vw*: $Src\ v = Trg\ w$
assume *tu*: $Src\ t2 = Trg\ u$
assume *1*: $Arr\ t1 \wedge Arr\ t2 \wedge Src\ t1 = Trg\ t2 \wedge Arr\ v \wedge (Dom\ t1 \star Dom\ t2) = Cod\ v$
show $((t1 \star t2)\ \lfloor\star\rfloor\ u)\ \lfloor\cdot\rfloor\ (v\ \lfloor\star\rfloor\ w) = (t1 \star t2)\ \lfloor\cdot\rfloor\ v\ \lfloor\star\rfloor\ u\ \lfloor\cdot\rfloor\ w$
proof −
have $is\text{-}Prim_0\ u \Longrightarrow$ *?thesis*
using *u v w uw tu vw t12 I1 I2 1 Obj-Src*
apply (*cases u*)
apply *simp-all*
by (*cases w*, *simp-all add*: *Src-VcompNml*)
moreover have $\neg\ is\text{-}Prim_0\ u \Longrightarrow$ *?thesis*
proof −
assume *u'*: $\neg\ is\text{-}Prim_0\ u$
hence *w'*: $\neg\ is\text{-}Prim_0\ w$ **using** *u w uw* **by** (*cases u*, *simp-all*; *cases w*, *simp-all*)
show *?thesis*
using *1 v*
proof (*cases v*, *simp-all*)
fix *v1 v2*
assume *v12*: $v = v1 \star v2$
have *v1*: $v1 = \langle un\text{-}Prim\ v1\rangle \wedge arr\ (un\text{-}Prim\ v1) \wedge Nml\ v1$
using *v v12* **by** (*cases v1*, *simp-all*)
have *v2*: $Nml\ v2 \wedge \neg\ is\text{-}Prim_0\ v2$
using *v v12* **by** (*cases v1*, *simp-all*)
have *2*: $v = (\langle un\text{-}Prim\ v1\rangle \star v2)$
using *v1 v12* **by** *simp*
show $((t1 \star t2)\ \lfloor\star\rfloor\ u)\ \lfloor\cdot\rfloor\ ((v1 \star v2)\ \lfloor\star\rfloor\ w) = (t1\ \lfloor\cdot\rfloor\ v1 \star t2\ \lfloor\cdot\rfloor\ v2)\ \lfloor\star\rfloor\ u\ \lfloor\cdot\rfloor\ w$
proof −
have *3*: $(t1 \star t2)\ \lfloor\star\rfloor\ u = t1\ \lfloor\star\rfloor\ (t2\ \lfloor\star\rfloor\ u)$
using *u u'* **by** (*cases u*, *simp-all*)
have *4*: $v\ \lfloor\star\rfloor\ w = v1\ \lfloor\star\rfloor\ v2\ \lfloor\star\rfloor\ w$
proof −
have $Src\ v1 = Trg\ v2$
using *v v12 1 Arr.simps*(*3*) *Nml-HcompD*(*7*) **by** *blast*
moreover have $Src\ v2 = Trg\ w$
using *v v12 vw* **by** *simp*
ultimately show *?thesis*
using *v w v1 v2 v12 2 HcompNml-assoc* [*of v1 v2 w*] *HcompNml-Nml* **by** *metis*
qed
have $((t1 \star t2)\ \lfloor\star\rfloor\ u)\ \lfloor\cdot\rfloor\ ((v1 \star v2)\ \lfloor\star\rfloor\ w)$

```
            = (t1 ⌊⋆⌋ (t2 ⌊⋆⌋ u)) ⌊·⌋ (v1 ⌊⋆⌋ (v2 ⌊⋆⌋ w))
          using 3 4 v12 by simp
        also have ... = (t1 ⌊·⌋ v1) ⌊⋆⌋ ((t2 ⌊⋆⌋ u) ⌊·⌋ (v2 ⌊⋆⌋ w))
        proof −
          have is-Hcomp (t2 ⌊⋆⌋ u)
            using t2 u u′ tu is-Hcomp-HcompNml by auto
          moreover have is-Hcomp (v2 ⌊⋆⌋ w)
            using v2 v12 w w′ vw is-Hcomp-HcompNml by auto
          ultimately show ?thesis
            using u u′ v w t1 v1 t12 v12 HcompNml-Prim
            by (metis VcompNml.simps(2) VcompNml.simps(3) is-Hcomp-def
                term.distinct-disc(3))
        qed
        also have ... = (t1 ⌊·⌋ v1) ⌊⋆⌋ (t2 ⌊·⌋ v2) ⌊⋆⌋ (u ⌊·⌋ w)
          using u w tu uw vw t2 v2 1 2 Nml-implies-Arr I2 by auto
        also have ... = ((t1 ⌊·⌋ v1) ⋆ (t2 ⌊·⌋ v2)) ⌊⋆⌋ (u ⌊·⌋ w)
        proof −
          have ¬is-Prim0 (u ⌊·⌋ w)
            using u w u′ w′ by (cases u, simp-all; cases w, simp-all)
          hence ((t1 ⌊·⌋ v1) ⋆ (t2 ⌊·⌋ v2)) ⌊⋆⌋ (u ⌊·⌋ w)
                  = (t1 ⌊·⌋ v1) ⌊⋆⌋ ((t2 ⌊·⌋ v2) ⌊⋆⌋ (u ⌊·⌋ w))
            by (cases u ⌊·⌋ w) simp-all
          thus ?thesis by presburger
        qed
        finally show ?thesis by blast
      qed
    qed
  qed
  ultimately show ?thesis by blast
 qed
qed
moreover have Src t = Trg u
  using assms Src-Dom Trg-Dom Src-Cod Trg-Cod Nml-implies-Arr by metis
ultimately show ?thesis using assms by simp
qed
```

The following function reduces a formal arrow to normal form.

```
fun Nmlize :: 'a term ⇒ 'a term   (‹⌊-⌋›)
where ⌊⟨μ⟩0⌋ = ⟨μ⟩0
  | ⌊⟨μ⟩⌋ = ⟨μ⟩
  | ⌊t ⋆ u⌋ = ⌊t⌋ ⌊⋆⌋ ⌊u⌋
  | ⌊t · u⌋ = ⌊t⌋ ⌊·⌋ ⌊u⌋
  | ⌊l[t]⌋ = ⌊t⌋
  | ⌊l⁻¹[t]⌋ = ⌊t⌋
  | ⌊r[t]⌋ = ⌊t⌋
  | ⌊r⁻¹[t]⌋ = ⌊t⌋
  | ⌊a[t, u, v]⌋ = (⌊t⌋ ⌊⋆⌋ ⌊u⌋) ⌊⋆⌋ ⌊v⌋
  | ⌊a⁻¹[t, u, v]⌋ = ⌊t⌋ ⌊⋆⌋ (⌊u⌋ ⌊⋆⌋ ⌊v⌋)
```

```
lemma Nml-Nmlize:
assumes Arr t
shows Nml ⌊t⌋ and Src ⌊t⌋ = Src t and Trg ⌊t⌋ = Trg t
and Dom ⌊t⌋ = ⌊Dom t⌋ and Cod ⌊t⌋ = ⌊Cod t⌋
proof −
  have 0: Arr t ⟹ Nml ⌊t⌋ ∧ Src ⌊t⌋ = Src t ∧ Trg ⌊t⌋ = Trg t ∧
                    Dom ⌊t⌋ = ⌊Dom t⌋ ∧ Cod ⌊t⌋ = ⌊Cod t⌋
    using Nml-HcompNml Nml-VcompNml HcompNml-assoc
          Src-VcompNml Trg-VcompNml VSeq-implies-HPar
    apply (induct t)
            apply auto
  proof −
    fix t
    assume 1: Arr t
    assume 2: Nml ⌊t⌋
    assume 3: Src ⌊t⌋ = Src t
    assume 4: Trg ⌊t⌋ = Trg t
    assume 5: Dom ⌊t⌋ = ⌊Dom t⌋
    assume 6: Cod ⌊t⌋ = ⌊Cod t⌋
    have 7: Nml ⌊Dom t⌋
      using 2 5 Nml-Dom by fastforce
    have 8: Trg ⌊t⌋ = ⌊Trg t⌋
      using 1 2 4 Nml-Trg Obj-Trg
      by (metis Nml.elims(2) Nmlize.simps(1) Nmlize.simps(2) Obj.simps(3))
    have 9: Nml ⌊Cod t⌋
      using 2 6 Nml-Cod by fastforce
    have 10: Src ⌊t⌋ = ⌊Src t⌋
      using 1 2 3 Nml-Src Obj-Src
      by (metis Nml.elims(2) Nmlize.simps(1) Nmlize.simps(2) Obj.simps(3))
    show ⌊Dom t⌋ = ⌊Trg t⌋ ⌊⋆⌋ ⌊Dom t⌋
      using 2 5 7 8 Nml-implies-Arr Trg-Dom HcompNml-Trg-Nml by metis
    show ⌊Cod t⌋ = ⌊Trg t⌋ ⌊⋆⌋ ⌊Cod t⌋
      using 2 6 8 9 Nml-implies-Arr Trg-Cod HcompNml-Trg-Nml by metis
    show ⌊Dom t⌋ = ⌊Dom t⌋ ⌊⋆⌋ ⌊Src t⌋
      using 2 5 7 10 Nml-implies-Arr Src-Dom HcompNml-Nml-Src by metis
    show ⌊Cod t⌋ = ⌊Cod t⌋ ⌊⋆⌋ ⌊Src t⌋
      using 2 6 9 10 Nml-implies-Arr Src-Cod HcompNml-Nml-Src by metis
    next
    fix t1 t2 t3
    assume Nml ⌊t1⌋ and Nml ⌊t2⌋ and Nml ⌊t3⌋
    assume Src t1 = Trg t2 and Src t2 = Trg t3
    assume Src ⌊t1⌋ = Trg t2 and Src ⌊t2⌋ = Trg t3
    assume Trg ⌊t1⌋ = Trg t1 and Trg ⌊t2⌋ = Trg t2 and Trg ⌊t3⌋ = Trg t3
    assume Dom ⌊t1⌋ = ⌊Dom t1⌋ and Cod ⌊t1⌋ = ⌊Cod t1⌋ and Dom ⌊t2⌋ = ⌊Dom t2⌋
    and Cod ⌊t2⌋ = ⌊Cod t2⌋ and Dom ⌊t3⌋ = ⌊Dom t3⌋ and Cod ⌊t3⌋ = ⌊Cod t3⌋
    show ⌊Dom t1⌋ ⌊⋆⌋ ⌊Dom t2⌋ ⌊⋆⌋ ⌊Dom t3⌋ =
         (⌊Dom t1⌋ ⌊⋆⌋ ⌊Dom t2⌋) ⌊⋆⌋ ⌊Dom t3⌋
      using Nml-Dom Nml-implies-Arr Src-Dom Trg-Dom
            HcompNml-assoc [of ⌊Dom t1⌋ ⌊Dom t2⌋ ⌊Dom t3⌋]
```

‹*Nml* ⌊*t1*⌋› ‹*Nml* ⌊*t2*⌋› ‹*Nml* ⌊*t3*⌋›
‹*Dom* ⌊*t1*⌋ = ⌊*Dom t1*⌋› ‹*Dom* ⌊*t2*⌋ = ⌊*Dom t2*⌋› ‹*Dom* ⌊*t3*⌋ = ⌊*Dom t3*⌋›
‹*Src* ⌊*t1*⌋ = *Trg t2*› ‹*Trg* ⌊*t2*⌋ = *Trg t2*›
‹*Src* ⌊*t2*⌋ = *Trg t3*› ‹*Trg* ⌊*t3*⌋ = *Trg t3*›
by *metis*
show ⌊*Cod t1*⌋ ⌊⋆⌋ ⌊*Cod t2*⌋ ⌊⋆⌋ ⌊*Cod t3*⌋ = (⌊*Cod t1*⌋ ⌊⋆⌋ ⌊*Cod t2*⌋) ⌊⋆⌋ ⌊*Cod t3*⌋
using *Nml-Cod Nml-implies-Arr Src-Cod Trg-Cod*
HcompNml-assoc [*of* ⌊*Cod t1*⌋ ⌊*Cod t2*⌋ ⌊*Cod t3*⌋]
‹*Nml* ⌊*t1*⌋› ‹*Nml* ⌊*t2*⌋› ‹*Nml* ⌊*t3*⌋›
‹*Cod* ⌊*t1*⌋ = ⌊*Cod t1*⌋› ‹*Cod* ⌊*t2*⌋ = ⌊*Cod t2*⌋› ‹*Cod* ⌊*t3*⌋ = ⌊*Cod t3*⌋›
‹*Src* ⌊*t1*⌋ = *Trg t2*› ‹*Trg* ⌊*t2*⌋ = *Trg t2*›
‹*Src* ⌊*t2*⌋ = *Trg t3*› ‹*Trg* ⌊*t3*⌋ = *Trg t3*›
by *metis*
qed
show *Nml* ⌊*t*⌋ **using** *assms 0* **by** *blast*
show *Src* ⌊*t*⌋ = *Src t* **using** *assms 0* **by** *blast*
show *Trg* ⌊*t*⌋ = *Trg t* **using** *assms 0* **by** *blast*
show *Dom* ⌊*t*⌋ = ⌊*Dom t*⌋ **using** *assms 0* **by** *blast*
show *Cod* ⌊*t*⌋ = ⌊*Cod t*⌋ **using** *assms 0* **by** *blast*
qed

lemma *Nmlize-in-Hom* [*intro*]:
assumes *Arr t*
shows ⌊*t*⌋ ∈ *HHom* (*Src t*) (*Trg t*) **and** ⌊*t*⌋ ∈ *VHom* ⌊*Dom t*⌋ ⌊*Cod t*⌋
using *assms Nml-Nmlize Nml-implies-Arr* **by** *auto*

lemma *Nmlize-Src*:
assumes *Arr t*
shows ⌊*Src t*⌋ = *Src* ⌊*t*⌋ **and** ⌊*Src t*⌋ = *Src t*
proof −
have *1*: *Obj* (*Src t*)
using *assms* **by** *simp*
obtain *a* **where** *a*: *obj a* ∧ *Src t* = ⟨*a*⟩$_0$
using *1* **by** (*cases Src t*, *simp-all*)
show ⌊*Src t*⌋ = *Src t*
using *a* **by** *simp*
thus ⌊*Src t*⌋ = *Src* ⌊*t*⌋
using *assms Nmlize-in-Hom* **by** *simp*
qed

lemma *Nmlize-Trg*:
assumes *Arr t*
shows ⌊*Trg t*⌋ = *Trg* ⌊*t*⌋ **and** ⌊*Trg t*⌋ = *Trg t*
proof −
have *1*: *Obj* (*Trg t*)
using *assms* **by** *simp*
obtain *a* **where** *a*: *obj a* ∧ *Trg t* = ⟨*a*⟩$_0$
using *1* **by** (*cases Trg t*, *simp-all*)
show ⌊*Trg t*⌋ = *Trg t*

```
    using a by simp
  thus ⌊Trg t⌋ = Trg ⌊t⌋
    using assms Nmlize-in-Hom by simp
qed

lemma Nmlize-Dom:
assumes Arr t
shows ⌊Dom t⌋ = Dom ⌊t⌋
  using assms Nmlize-in-Hom by simp

lemma Nmlize-Cod:
assumes Arr t
shows ⌊Cod t⌋ = Cod ⌊t⌋
  using assms Nmlize-in-Hom by simp

lemma Ide-Nmlize-Ide:
assumes Ide t
shows Ide ⌊t⌋
proof −
  have Ide t ⟹ Ide ⌊t⌋
    using Ide-HcompNml Nml-Nmlize
    by (induct t, simp-all)
  thus ?thesis using assms by blast
qed

lemma Ide-Nmlize-Can:
assumes Can t
shows Ide ⌊t⌋
proof −
  have Can t ⟹ Ide ⌊t⌋
    using Can-implies-Arr Ide-HcompNml Nml-Nmlize Ide-VcompNml Nml-HcompNml
    apply (induct t)
      apply (auto simp add: Dom-Ide Cod-Ide)
    by (metis Ide-VcompNml)
  thus ?thesis using assms by blast
qed

lemma Can-Nmlize-Can:
assumes Can t
shows Can ⌊t⌋
  using assms Ide-Nmlize-Can Ide-implies-Can by auto

lemma Nmlize-Nml [simp]:
assumes Nml t
shows ⌊t⌋ = t
proof −
  have Nml t ⟹ ⌊t⌋ = t
    apply (induct t, simp-all)
    using HcompNml-Prim Nml-HcompD by metis
```

thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *Nmlize-Nmlize* [*simp*]:
assumes *Arr t*
shows $\lfloor\lfloor t\rfloor\rfloor = \lfloor t\rfloor$
using *assms Nml-Nmlize Nmlize-Nml* **by** *blast*

lemma *Nmlize-Hcomp*:
assumes *Arr t* **and** *Arr u*
shows $\lfloor t \star u\rfloor = \lfloor\lfloor t\rfloor \star \lfloor u\rfloor\rfloor$
using *assms Nmlize-Nmlize* **by** *simp*

lemma *Nmlize-Hcomp-Obj-Arr* [*simp*]:
assumes *Arr u*
shows $\lfloor\langle b\rangle_0 \star u\rfloor = \lfloor u\rfloor$
using *assms* **by** *simp*

lemma *Nmlize-Hcomp-Arr-Obj* [*simp*]:
assumes *Arr t* **and** *Src t* $= \langle a\rangle_0$
shows $\lfloor t \star \langle a\rangle_0\rfloor = \lfloor t\rfloor$
using *assms HcompNml-Nml-Src Nmlize-in-Hom* **by** *simp*

lemma *Nmlize-Hcomp-Prim-Arr* [*simp*]:
assumes *Arr u* **and** $\neg$ *is-Prim*$_0$ $\lfloor u\rfloor$
shows $\lfloor\langle\mu\rangle \star u\rfloor = \langle\mu\rangle \star \lfloor u\rfloor$
using *assms* **by** *simp*

lemma *Nmlize-Hcomp-Hcomp*:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\lfloor(t \star u) \star v\rfloor = \lfloor\lfloor t\rfloor \star (\lfloor u\rfloor \star \lfloor v\rfloor)\rfloor$
using *assms Nml-Nmlize Nmlize-Nmlize* **by** (*simp add*: *HcompNml-assoc*)

lemma *Nmlize-Hcomp-Hcomp'*:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\lfloor t \star u \star v\rfloor = \lfloor\lfloor t\rfloor \star \lfloor u\rfloor \star \lfloor v\rfloor\rfloor$
using *assms Nml-Nmlize Nmlize-Nmlize* **by** (*simp add*: *HcompNml-assoc*)

lemma *Nmlize-Vcomp-Cod-Arr*:
assumes *Arr t*
shows $\lfloor Cod\ t \cdot t\rfloor = \lfloor t\rfloor$
proof −
have $Arr\ t \Longrightarrow \lfloor Cod\ t \cdot t\rfloor = \lfloor t\rfloor$
proof (*induct t*, *simp-all*)
show $\bigwedge x.\ arr\ x \Longrightarrow cod\ x \cdot x = x$
using *comp-cod-arr* **by** *blast*
fix *t1 t2*
show $\bigwedge t1\ t2.\ \lfloor Cod\ t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t1\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor Cod\ t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor = \lfloor t2\rfloor \Longrightarrow HSeq\ t1\ t2 \Longrightarrow$
$(\lfloor Cod\ t1\rfloor\ \lfloor\star\rfloor\ \lfloor Cod\ t2\rfloor)\ \lfloor\cdot\rfloor\ (\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor) = \lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor$

using *VcompNml-HcompNml Ide-Cod Nml-Nmlize Nmlize-in-Hom*
by *simp*
show $\lfloor Cod\ t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t1\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor Cod\ t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor = \lfloor t2\rfloor \Longrightarrow VSeq\ t1\ t2 \Longrightarrow$
$\lfloor Cod\ t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor = \lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor$
using *VcompNml-assoc* [*of* $\lfloor Cod\ t1\rfloor\ \lfloor t1\rfloor\ \lfloor t2\rfloor$] *Ide-Cod Nml-Nmlize*
by *simp*
next
show $\bigwedge t.\ \lfloor Cod\ t\rfloor\ \lfloor\cdot\rfloor\ \lfloor t\rfloor = \lfloor t\rfloor \Longrightarrow Arr\ t \Longrightarrow (\lfloor Trg\ t\rfloor\ \lfloor\star\rfloor\ \lfloor Cod\ t\rfloor)\ \lfloor\cdot\rfloor\ \lfloor t\rfloor = \lfloor t\rfloor$
by (*metis Arr.simps*(*6*) *Cod.simps*(*6*) *Nmlize.simps*(*3*) *Nmlize.simps*(*6*) *Nmlize-Cod*)
show $\bigwedge t.\ \lfloor Cod\ t\rfloor\ \lfloor\cdot\rfloor\ \lfloor t\rfloor = \lfloor t\rfloor \Longrightarrow Arr\ t \Longrightarrow (\lfloor Cod\ t\rfloor\ \lfloor\star\rfloor\ \lfloor Src\ t\rfloor)\ \lfloor\cdot\rfloor\ \lfloor t\rfloor = \lfloor t\rfloor$
by (*simp add*: *Nml-Nmlize*(*1*) *Nml-Nmlize*(*2*) *Nmlize-Src*(*2*) *HcompNml-Nml-Obj*)
show $\bigwedge t1\ t2\ t3.\ \lfloor Cod\ t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t1\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor Cod\ t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor = \lfloor t2\rfloor \Longrightarrow$
$\lfloor Cod\ t3\rfloor\ \lfloor\cdot\rfloor\ \lfloor t3\rfloor = \lfloor t3\rfloor \Longrightarrow$
$Arr\ t1 \wedge Arr\ t2 \wedge Arr\ t3 \wedge Src\ t1 = Trg\ t2 \wedge Src\ t2 = Trg\ t3 \Longrightarrow$
$(\lfloor Cod\ t1\rfloor\ \lfloor\star\rfloor\ \lfloor Cod\ t2\rfloor\ \lfloor\star\rfloor\ \lfloor Cod\ t3\rfloor)\ \lfloor\cdot\rfloor\ ((\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor)\ \lfloor\star\rfloor\ \lfloor t3\rfloor) =$
$(\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor)\ \lfloor\star\rfloor\ \lfloor t3\rfloor$
using *VcompNml-HcompNml Ide-Cod HcompNml-in-Hom Nml-HcompNml Nml-Nmlize Nmlize-in-Hom HcompNml-assoc*
by *simp*
show $\bigwedge t1\ t2\ t3.\ \lfloor Cod\ t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t1\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor Cod\ t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor = \lfloor t2\rfloor \Longrightarrow$
$\lfloor Cod\ t3\rfloor\ \lfloor\cdot\rfloor\ \lfloor t3\rfloor = \lfloor t3\rfloor \Longrightarrow$
$Arr\ t1 \wedge Arr\ t2 \wedge Arr\ t3 \wedge Src\ t1 = Trg\ t2 \wedge Src\ t2 = Trg\ t3 \Longrightarrow$
$((\lfloor Cod\ t1\rfloor\ \lfloor\star\rfloor\ \lfloor Cod\ t2\rfloor)\ \lfloor\star\rfloor\ \lfloor Cod\ t3\rfloor)\ \lfloor\cdot\rfloor\ (\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor\ \lfloor\star\rfloor\ \lfloor t3\rfloor) =$
$\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor\ \lfloor\star\rfloor\ \lfloor t3\rfloor$
using *VcompNml-HcompNml Ide-Cod HcompNml-in-Hom Nml-HcompNml Nml-Nmlize Nmlize-in-Hom HcompNml-assoc*
by *simp*
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *Nmlize-Vcomp-Arr-Dom*:
assumes $Arr\ t$
shows $\lfloor t \cdot Dom\ t\rfloor = \lfloor t\rfloor$
proof −
have $Arr\ t \Longrightarrow \lfloor t \cdot Dom\ t\rfloor = \lfloor t\rfloor$
proof (*induct t*, *simp-all*)
show $\bigwedge x.\ arr\ x \Longrightarrow x \cdot local.dom\ x = x$
using *comp-arr-dom* **by** *blast*
fix *t1 t2*
show $\lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor Dom\ t1\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor Dom\ t2\rfloor = \lfloor t2\rfloor \Longrightarrow HSeq\ t1\ t2 \Longrightarrow$
$(\lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor)\ \lfloor\cdot\rfloor\ (\lfloor Dom\ t1\rfloor\ \lfloor\star\rfloor\ \lfloor Dom\ t2\rfloor) = \lfloor t1\rfloor\ \lfloor\star\rfloor\ \lfloor t2\rfloor$
using *VcompNml-HcompNml Ide-Dom HcompNml-in-Hom Nml-HcompNml Nml-Nmlize Nmlize-in-Hom HcompNml-assoc*
by *simp*
show $\lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor Cod\ t2\rfloor = \lfloor t1\rfloor \Longrightarrow \lfloor t2\rfloor\ \lfloor\cdot\rfloor\ \lfloor Dom\ t2\rfloor = \lfloor t2\rfloor \Longrightarrow VSeq\ t1\ t2 \Longrightarrow$
$(\lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor)\ \lfloor\cdot\rfloor\ \lfloor Dom\ t2\rfloor = \lfloor t1\rfloor\ \lfloor\cdot\rfloor\ \lfloor t2\rfloor$

using *VcompNml-assoc* [*of* $\lfloor t1 \rfloor$ $\lfloor t2 \rfloor$ $\lfloor Dom\ t2 \rfloor$] *Ide-Dom Nml-Nmlize*
by *simp*
next
show $\bigwedge t.\ \lfloor t \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t \rfloor = \lfloor t \rfloor \Longrightarrow Arr\ t \Longrightarrow \lfloor t \rfloor\ \lfloor \cdot \rfloor\ (\lfloor Trg\ t \rfloor\ \lfloor \star \rfloor\ \lfloor Dom\ t \rfloor) = \lfloor t \rfloor$
by (*simp add*: *Nml-Nmlize*(*1*) *Nml-Nmlize*(*3*) *Nmlize-Trg*(*2*)
HcompNml-Obj-Nml bicategorical-language.Ide-Dom
bicategorical-language-axioms)
show $\bigwedge t.\ \lfloor t \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t \rfloor = \lfloor t \rfloor \Longrightarrow Arr\ t \Longrightarrow \lfloor t \rfloor\ \lfloor \cdot \rfloor\ (\lfloor Dom\ t \rfloor\ \lfloor \star \rfloor\ \lfloor Src\ t \rfloor) = \lfloor t \rfloor$
by (*simp add*: *Nml-Nmlize*(*1*) *Nml-Nmlize*(*2*) *Nmlize-Src*(*2*) *HcompNml-Nml-Obj*)
show $\bigwedge t1\ t2\ t3.\ \lfloor t1 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t1 \rfloor = \lfloor t1 \rfloor \Longrightarrow \lfloor t2 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t2 \rfloor = \lfloor t2 \rfloor \Longrightarrow$
$\lfloor t3 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t3 \rfloor = \lfloor t3 \rfloor \Longrightarrow$
$Arr\ t1 \wedge Arr\ t2 \wedge Arr\ t3 \wedge Src\ t1 = Trg\ t2 \wedge Src\ t2 = Trg\ t3 \Longrightarrow$
$((\lfloor t1 \rfloor\ \lfloor \star \rfloor\ \lfloor t2 \rfloor)\ \lfloor \star \rfloor\ \lfloor t3 \rfloor)\ \lfloor \cdot \rfloor\ ((\lfloor Dom\ t1 \rfloor\ \lfloor \star \rfloor\ \lfloor Dom\ t2 \rfloor)\ \lfloor \star \rfloor\ \lfloor Dom\ t3 \rfloor) =$
$(\lfloor t1 \rfloor\ \lfloor \star \rfloor\ \lfloor t2 \rfloor)\ \lfloor \star \rfloor\ \lfloor t3 \rfloor$
using *VcompNml-HcompNml Ide-Dom HcompNml-in-Hom Nml-HcompNml*
Nml-Nmlize Nmlize-in-Hom HcompNml-assoc
by *simp*
show $\bigwedge t1\ t2\ t3.\ \lfloor t1 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t1 \rfloor = \lfloor t1 \rfloor \Longrightarrow \lfloor t2 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t2 \rfloor = \lfloor t2 \rfloor \Longrightarrow$
$\lfloor t3 \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t3 \rfloor = \lfloor t3 \rfloor \Longrightarrow$
$Arr\ t1 \wedge Arr\ t2 \wedge Arr\ t3 \wedge Src\ t1 = Trg\ t2 \wedge Src\ t2 = Trg\ t3 \Longrightarrow$
$(\lfloor t1 \rfloor\ \lfloor \star \rfloor\ \lfloor t2 \rfloor\ \lfloor \star \rfloor\ \lfloor t3 \rfloor)\ \lfloor \cdot \rfloor\ (\lfloor Dom\ t1 \rfloor\ \lfloor \star \rfloor\ \lfloor Dom\ t2 \rfloor\ \lfloor \star \rfloor\ \lfloor Dom\ t3 \rfloor) =$
$\lfloor t1 \rfloor\ \lfloor \star \rfloor\ \lfloor t2 \rfloor\ \lfloor \star \rfloor\ \lfloor t3 \rfloor$
using *VcompNml-HcompNml Ide-Dom HcompNml-in-Hom Nml-HcompNml*
Nml-Nmlize Nmlize-in-Hom HcompNml-assoc
by *simp*
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *Nmlize-Inv*:
assumes *Can t*
shows $\lfloor Inv\ t \rfloor = Inv\ \lfloor t \rfloor$
proof −
have $Can\ t \Longrightarrow \lfloor Inv\ t \rfloor = Inv\ \lfloor t \rfloor$
proof (*induct t*, *simp-all*)
fix *u v*
assume *I1*: $\lfloor Inv\ u \rfloor = Inv\ \lfloor u \rfloor$
assume *I2*: $\lfloor Inv\ v \rfloor = Inv\ \lfloor v \rfloor$
show $Can\ u \wedge Can\ v \wedge Src\ u = Trg\ v \Longrightarrow Inv\ \lfloor u \rfloor\ \lfloor \star \rfloor\ Inv\ \lfloor v \rfloor = Inv\ (\lfloor u \rfloor\ \lfloor \star \rfloor\ \lfloor v \rfloor)$
using *Inv-HcompNml Nml-Nmlize Can-implies-Arr Can-Nmlize-Can*
I1 I2
by *simp*
show $Can\ u \wedge Can\ v \wedge Dom\ u = Cod\ v \Longrightarrow Inv\ \lfloor v \rfloor\ \lfloor \cdot \rfloor\ Inv\ \lfloor u \rfloor = Inv\ (\lfloor u \rfloor\ \lfloor \cdot \rfloor\ \lfloor v \rfloor)$
using *Inv-VcompNml Nml-Nmlize Can-implies-Arr Nmlize-in-Hom Can-Nmlize-Can*
I1 I2
by *simp*
fix *w*
assume *I3*: $\lfloor Inv\ w \rfloor = Inv\ \lfloor w \rfloor$
assume *uvw*: $Can\ u \wedge Can\ v \wedge Can\ w \wedge Src\ u = Trg\ v \wedge Src\ v = Trg\ w$

```
    show Inv ⌊u⌋ ⌊⋆⌋ (Inv ⌊v⌋ ⌊⋆⌋ Inv ⌊w⌋) = Inv ((⌊u⌋ ⌊⋆⌋ ⌊v⌋) ⌊⋆⌋ ⌊w⌋)
      using uvw I1 I2 I3
            Inv-HcompNml Nml-Nmlize Can-implies-Arr Can-Nmlize-Can
            Nml-HcompNml Can-HcompNml HcompNml-assoc
      by simp
    show (Inv ⌊u⌋ ⌊⋆⌋ Inv ⌊v⌋) ⌊⋆⌋ Inv ⌊w⌋ = Inv (⌊u⌋ ⌊⋆⌋ (⌊v⌋ ⌊⋆⌋ ⌊w⌋))
      using uvw I1 I2 I3
            Inv-HcompNml Nml-Nmlize Can-implies-Arr Can-Nmlize-Can
            Nml-HcompNml Can-HcompNml HcompNml-assoc Can-Inv
      by simp
  qed
  thus ?thesis using assms by blast
qed
```

1.6.3 Reductions

Function *red* defined below takes a formal identity t to a canonical arrow $f\downarrow \in Hom\ f\ \lfloor f \rfloor$. The auxiliary function *red2* takes a pair $(f,\ g)$ of normalized formal identities and produces a canonical arrow $f \Downarrow g \in Hom\ (f \star g)\ \lfloor f \star g \rfloor$.

```
fun red2                    (infixr ‹⇓› 53)
where ⟨b⟩₀ ⇓ u = l[u]
    | ⟨f⟩ ⇓ ⟨a⟩₀ = r[⟨f⟩]
    | ⟨f⟩ ⇓ u = ⟨f⟩ ⋆ u
    | (t ⋆ u) ⇓ ⟨a⟩₀ = r[t ⋆ u]
    | (t ⋆ u) ⇓ v = (t ⇓ ⌊u ⋆ v⌋) · (t ⋆ (u ⇓ v)) · a[t, u, v]
    | t ⇓ u = undefined

fun red                     (‹-↓› [56] 56)
where ⟨f⟩₀↓ = ⟨f⟩₀
    | ⟨f⟩↓ = ⟨f⟩
    | (t ⋆ u)↓ = (if Nml (t ⋆ u) then t ⋆ u else (⌊t⌋ ⇓ ⌊u⌋) · (t↓ ⋆ u↓))
    | t↓ = undefined

lemma red-Nml [simp]:
assumes Nml a
shows a↓ = a
  using assms by (cases a, simp-all)

lemma red2-Nml:
assumes Nml (a ⋆ b)
shows a ⇓ b = a ⋆ b
proof −
  have a: a = ⟨un-Prim a⟩
    using assms Nml-HcompD by metis
  have b: Nml b ∧ ¬ is-Prim₀ b
    using assms Nml-HcompD by metis
  show ?thesis using a b
    apply (cases b)
      apply simp-all
```

```
    apply (metis red2.simps(3))
   by (metis red2.simps(4))
qed

lemma Can-red2:
assumes Ide a and Nml a and Ide b and Nml b and Src a = Trg b
shows Can (a ⇓ b)
and a ⇓ b ∈ VHom (a ★ b) ⌊a ★ b⌋
proof −
  have 0: ⋀b. ⟦ Ide a ∧ Nml a; Ide b ∧ Nml b; Src a = Trg b ⟧ ⟹
              Can (a ⇓ b) ∧ a ⇓ b ∈ VHom (a ★ b) ⌊a ★ b⌋
  proof (induct a, simp-all add: HcompNml-Nml-Src HcompNml-Trg-Nml)
    fix x b
    show Ide b ∧ Nml b ⟹ Can (Trg b ⇓ b) ∧ Arr (Trg b ⇓ b) ∧
                         Dom (Trg b ⇓ b) = Trg b ★ b ∧ Cod (Trg b ⇓ b) = b
      using Ide-implies-Can Ide-in-Hom Nmlize-Nml
      apply (cases b, simp-all)
    proof −
      fix u v
      assume uv: Ide u ∧ Ide v ∧ Src u = Trg v ∧ Nml (u ★ v)
      show Can (Trg u ⇓ (u ★ v)) ∧ Arr (Trg u ⇓ (u ★ v)) ∧
           Dom (Trg u ⇓ (u ★ v)) = Trg u ★ u ★ v ∧
           Cod (Trg u ⇓ (u ★ v)) = u ★ v
        using uv Ide-implies-Can Can-implies-Arr Ide-in-Hom
        by (cases u, simp-all)
    qed
    show ide x ∧ arr x ⟹ Ide b ∧ Nml b ⟹ ⟨src x⟩₀ = Trg b ⟹
         Can (⟨x⟩ ⇓ b) ∧ Arr (⟨x⟩ ⇓ b) ∧ Dom (⟨x⟩ ⇓ b) = ⟨x⟩ ★ b ∧ Cod (⟨x⟩ ⇓ b) =
         ⟨x⟩ ⌊★⌋ b
        using Ide-implies-Can Can-implies-Arr Nmlize-Nml Ide-in-Hom
        by (cases b, simp-all)
    next
    fix u v w
    assume uv: Ide u ∧ Ide v ∧ Src u = Trg v ∧ Nml (u ★ v)
    assume vw: Src v = Trg w
    assume w: Ide w ∧ Nml w
    assume I1: ⋀w. ⟦ Nml u; Ide w ∧ Nml w; Trg v = Trg w ⟧ ⟹
                   Can (u ⇓ w) ∧ Arr (u ⇓ w) ∧
                   Dom (u ⇓ w) = u ★ w ∧ Cod (u ⇓ w) = u ⌊★⌋ w
    assume I2: ⋀x. ⟦ Nml v; Ide x ∧ Nml x; Trg w = Trg x ⟧ ⟹
                   Can (v ⇓ x) ∧ Arr (v ⇓ x) ∧
                   Dom (v ⇓ x) = v ★ x ∧ Cod (v ⇓ x) = v ⌊★⌋ x
    show Can ((u ★ v) ⇓ w) ∧ Arr ((u ★ v) ⇓ w) ∧
         Dom ((u ★ v) ⇓ w) = (u ★ v) ★ w ∧
         Cod ((u ★ v) ⇓ w) = (⌊u⌋ ⌊★⌋ ⌊v⌋) ⌊★⌋ w
    proof −
      have u: Nml u ∧ Ide u
        using uv Nml-HcompD by blast
      have v: Nml v ∧ Ide v
```

using *uv Nml-HcompD* **by** *blast*
have *is-Prim*$_0$ *w* $\Longrightarrow$ *?thesis*
proof −
assume *1*: *is-Prim*$_0$ *w*
have *2*: $(u \star v) \Downarrow w = \mathbf{r}[u \star v]$
using *1* **by** (*cases w*, *simp-all*)
have *3*: *Can* $(u \Downarrow v) \wedge$ *Arr* $(u \Downarrow v) \wedge$ *Dom* $(u \Downarrow v) = u \star v \wedge$ *Cod* $(u \Downarrow v) = u \star v$
using *u v uv 1 2 I1 Nmlize-Nml Nmlize.simps*(*3*) **by** *metis*
hence *4*: *VSeq* $(u \Downarrow v)$ $\mathbf{r}[u \star v]$
using *uv*
by (*metis* (*mono-tags*, *lifting*) *Arr.simps*(*7*) *Cod.simps*(*3*) *Cod.simps*(*7*)
Nml-implies-Arr Ide-in-Hom(*2*) *mem-Collect-eq*)
have *Can* $((u \star v) \Downarrow w)$
using *1 2 3 4 uv* **by** (*simp add*: *Ide-implies-Can*)
moreover have *Dom* $((u \star v) \Downarrow w) = (u \star v) \star w$
using *1 2 3 4 u v w uv vw I1 Ide-in-Hom Nml-HcompNml Ide-in-Hom*
apply (*cases w*) **by** *simp-all*
moreover have *Cod* $((u \star v) \Downarrow w) = \lfloor (u \star v) \star w \rfloor$
using *1 2 3 4 uv*
using *Nmlize-Nml* **apply** (*cases w*, *simp-all*)
by (*metis Nmlize.simps*(*3*) *Nmlize-Nml HcompNml.simps*(*3*))
ultimately show *?thesis* **using** *w Can-implies-Arr* **by** (*simp add*: *1 uv*)
qed
moreover have $\neg$ *is-Prim*$_0$ *w* $\Longrightarrow$ *?thesis*
proof −
assume *1*: $\neg$ *is-Prim*$_0$ *w*
have *2*: $(u \star v) \Downarrow w = (u \Downarrow \lfloor v \star w \rfloor) \cdot (u \star v \Downarrow w) \cdot \mathbf{a}[u, v, w]$
using *1 u v uv w* **by** (*cases w*; *simp*)
have *3*: *Can* $(u \Downarrow \lfloor v \star w \rfloor) \wedge$ *Dom* $(u \Downarrow \lfloor v \star w \rfloor) = u \star \lfloor v \star w \rfloor \wedge$
Cod $(u \Downarrow \lfloor v \star w \rfloor) = \lfloor u \star (v \star w) \rfloor$
proof −
have *Can* $(u \Downarrow \lfloor v \star w \rfloor) \wedge$ *Dom* $(u \Downarrow \lfloor v \star w \rfloor) = u \star \lfloor v \star w \rfloor \wedge$
Cod $(u \Downarrow \lfloor v \star w \rfloor) = u \lfloor \star \rfloor \lfloor v \star w \rfloor$
using *w uv Ide-HcompNml Nml-HcompNml*(*1*)
apply (*cases u*, *simp-all*)
using *u v vw I1 Nmlize-in-Hom*(*1*) [*of* $v \star w$] *Nml-Nmlize Ide-Nmlize-Ide*
by *simp*
moreover have $u \lfloor \star \rfloor \lfloor v \star w \rfloor = \lfloor u \star (v \star w) \rfloor$
using *uv u w Nmlize-Hcomp Nmlize-Nmlize Nml-implies-Arr* **by** *simp*
ultimately show *?thesis* **by** *presburger*
qed
have *4*: *Can* $(v \Downarrow w) \wedge$ *Dom* $(v \Downarrow w) = v \star w \wedge$ *Cod* $(v \Downarrow w) = \lfloor v \star w \rfloor$
using *v w vw 1 2 I2* **by** *simp*
hence *5*: *Src* $(v \Downarrow w) =$ *Src* $w \wedge$ *Trg* $(v \Downarrow w) =$ *Trg* v
using *Src-Dom Trg-Dom Can-implies-Arr* **by** *fastforce*
have *Can* $(u \star (v \Downarrow w)) \wedge$ *Dom* $(u \star (v \Downarrow w)) = u \star (v \star w) \wedge$
Cod $(u \star (v \Downarrow w)) = u \star \lfloor v \star w \rfloor$
using *u uv vw 4 5 Ide-implies-Can Ide-in-Hom* **by** *simp*
moreover have $\lfloor u \star \lfloor v \star w \rfloor \rfloor = \lfloor u \star v \rfloor \lfloor \star \rfloor \lfloor w \rfloor$

```
        proof −
          have ⌊u ⋆ ⌊v ⋆ w⌋⌋ = u ⌊⋆⌋ (v ⌊⋆⌋ w)
            using u v w 4
            by (metis Ide-Dom Can-implies-Arr Ide-implies-Arr
                Nml-Nmlize(1) Nmlize.simps(3) Nmlize-Nml)
          also have ... = (u ⌊⋆⌋ v) ⌊⋆⌋ w
            using u v w uv vw HcompNml-assoc by metis
          also have ... = ⌊u ⋆ v⌋ ⌊⋆⌋ ⌊w⌋
            using u v w by (metis Nmlize.simps(3) Nmlize-Nml)
          finally show ?thesis by blast
        qed
        moreover have Can a[u, v, w] ∧ Dom a[u, v, w] = (u ⋆ v) ⋆ w ∧
                                  Cod a[u, v, w] = u ⋆ (v ⋆ w)
          using uv vw w Ide-implies-Can Ide-in-Hom by auto
        ultimately show ?thesis
          using uv w 2 3 4 Nml-implies-Arr Nmlize-Nmlize Ide-implies-Can
                Nmlize-Nml Ide-Dom Can-implies-Arr
          by (metis Can.simps(4) Cod.simps(4) Dom.simps(4) Nmlize.simps(3))
      qed
      ultimately show ?thesis by blast
    qed
  qed
  show Can (a ⇓ b) using assms 0 by blast
  show a ⇓ b ∈ VHom (a ⋆ b) ⌊a ⋆ b⌋ using 0 assms by blast
qed

lemma red2-in-Hom [intro]:
assumes Ide u and Nml u and Ide v and Nml v and Src u = Trg v
shows u ⇓ v ∈ HHom (Src v) (Trg u) and u ⇓ v ∈ VHom (u ⋆ v) ⌊u ⋆ v⌋
proof −
  show 1: u ⇓ v ∈ VHom (u ⋆ v) ⌊u ⋆ v⌋
    using assms Can-red2 Can-implies-Arr by simp
  show u ⇓ v ∈ HHom (Src v) (Trg u)
    using assms 1 Src-Dom [of u ⇓ v] Trg-Dom [of u ⇓ v] Can-red2 Can-implies-Arr by simp
qed

lemma red2-simps [simp]:
assumes Ide u and Nml u and Ide v and Nml v and Src u = Trg v
shows Src (u ⇓ v) = Src v and Trg (u ⇓ v) = Trg u
and Dom (u ⇓ v) = u ⋆ v and Cod (u ⇓ v) = ⌊u ⋆ v⌋
  using assms red2-in-Hom by auto

lemma Can-red:
assumes Ide u
shows Can (u↓) and u↓ ∈ VHom u ⌊u⌋
proof −
  have 0: Ide u ⟹ Can (u↓) ∧ u↓ ∈ VHom u ⌊u⌋
  proof (induct u, simp-all add: Dom-Ide Cod-Ide)
    fix v w
```

assume *v*: $Can\ (v\downarrow) \wedge Arr\ (v\downarrow) \wedge Dom\ (v\downarrow) = v \wedge Cod\ (v\downarrow) = \lfloor v \rfloor$
assume *w*: $Can\ (w\downarrow) \wedge Arr\ (w\downarrow) \wedge Dom\ (w\downarrow) = w \wedge Cod\ (w\downarrow) = \lfloor w \rfloor$
assume *vw*: $Ide\ v \wedge Ide\ w \wedge Src\ v = Trg\ w$
show $(Nml\ (v \star w) \longrightarrow$
$Can\ v \wedge Can\ w \wedge v \star w = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor) \wedge$
$(\neg\ Nml\ (v \star w) \longrightarrow$
$Can\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge$
$Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Arr\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge$
$Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Cod\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor)$
proof
show $Nml\ (v \star w) \longrightarrow$
$Can\ v \wedge Can\ w \wedge v \star w = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor$
using *vw Nml-HcompD Ide-implies-Can Dom-Inv VcompNml-Ide-Nml Inv-Ide Nmlize.simps(3) Nmlize-Nml*
by *metis*
show $\neg\ Nml\ (v \star w) \longrightarrow$
$Can\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge$
$Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Arr\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge$
$Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Cod\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor$
proof
assume *1*: $\neg\ Nml\ (v \star w)$
have $Can\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor)$
using *v w vw Can-red2 Nml-Nmlize Ide-Nmlize-Ide Nml-HcompNml Ide-HcompNml*
by *simp*
moreover have $Src\ (v\downarrow) = Trg\ (w\downarrow)$
using *v w vw Src-Dom Trg-Dom* **by** *metis*
moreover have $Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Cod\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor$
using *v w vw Can-red2 Nml-Nmlize Ide-Nmlize-Ide* **by** *simp*
ultimately show $Can\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge$
$Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge Arr\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) \wedge$
$Src\ (v\downarrow) = Trg\ (w\downarrow) \wedge Dom\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor \star \lfloor w \rfloor \wedge$
$Cod\ (\lfloor v \rfloor \Downarrow \lfloor w \rfloor) = \lfloor v \rfloor\ \lfloor \star \rfloor\ \lfloor w \rfloor$
using *Can-implies-Arr* **by** *blast*
qed
qed
qed
show $Can\ (u\downarrow)$ **using** *assms 0* **by** *blast*
show $u\downarrow \in VHom\ u\ \lfloor u \rfloor$ **using** *assms 0* **by** *blast*
qed

lemma *red-in-Hom* [*intro*]:
assumes $Ide\ t$
shows $t\downarrow \in HHom\ (Src\ t)\ (Trg\ t)$ **and** $t\downarrow \in VHom\ t\ \lfloor t \rfloor$
proof −
show *1*: $t\downarrow \in VHom\ t\ \lfloor t \rfloor$
using *assms Can-red Can-implies-Arr* **by** *simp*
show $t\downarrow \in HHom\ (Src\ t)\ (Trg\ t)$
using *assms 1 Src-Dom* [*of* $t\downarrow$] *Trg-Dom* [*of* $t\downarrow$] *Can-red Can-implies-Arr* **by** *simp*
qed

lemma *red-simps* [*simp*]:
assumes *Ide t*
shows *Src* ($t\downarrow$) = *Src t* **and** *Trg* ($t\downarrow$) = *Trg t*
and *Dom* ($t\downarrow$) = *t* **and** *Cod* ($t\downarrow$) = $\lfloor t \rfloor$
  using *assms red-in-Hom* **by** *auto*

lemma *red-Src*:
assumes *Ide t*
shows *Src* $t\downarrow$ = *Src t*
  using *assms is-Prim0-Src* [*of t*]
  by (*cases Src t*, *simp-all*)

lemma *red-Trg*:
assumes *Ide t*
shows *Trg* $t\downarrow$ = *Trg t*
  using *assms is-Prim0-Trg* [*of t*]
  by (*cases Trg t*, *simp-all*)

lemma *Nmlize-red* [*simp*]:
assumes *Ide t*
shows $\lfloor t\downarrow \rfloor = \lfloor t \rfloor$
  using *assms Can-red Ide-Nmlize-Can Nmlize-in-Hom Ide-in-Hom* **by** *fastforce*

lemma *Nmlize-red2* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows $\lfloor t \Downarrow u \rfloor = \lfloor t \star u \rfloor$
  using *assms Can-red2 Ide-Nmlize-Can Nmlize-in-Hom* [*of t* $\Downarrow$ *u*] *red2-in-Hom Ide-in-Hom*
  by *simp*

end

1.6.4 Evaluation

The following locale is concerned with the evaluation of terms of the bicategorical language determined by C, src_C, and trg_C in a bicategory (V, H, a, i, *src*, *trg*), given a source and target-preserving functor from C to V.

locale *evaluation-map* =
  C: *horizontal-homs C* src_C trg_C +
  bicategorical-language C src_C trg_C +
  bicategory V H a i *src trg* +
  E: *functor C V E*
  for *C* :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
  and src_C :: *'c* ⇒ *'c*
  and trg_C :: *'c* ⇒ *'c*
  and *V* :: *'b comp* (**infixr** ‹·› *55*)
  and *H* :: *'b comp* (**infixr** ‹⋆› *53*)
  and a :: *'b* ⇒ *'b* ⇒ *'b* ⇒ *'b* (‹a[-, -, -]›)
  and i :: *'b* ⇒ *'b* (‹i[-]›)

and *src* :: $'b \Rightarrow 'b$
and *trg* :: $'b \Rightarrow 'b$
and E :: $'c \Rightarrow 'b$ +
assumes *preserves-src*: $E\ (src_C\ x) = src\ (E\ x)$
and *preserves-trg*: $E\ (trg_C\ x) = trg\ (E\ x)$
begin

notation *Nmlize* (‹⌊-⌋›)
notation *HcompNml* (**infixr** ‹⌊⋆⌋› *53*)
notation *VcompNml* (**infixr** ‹⌊·⌋› *55*)
notation *red* (‹-↓› [*56*] *56*)
notation *red2* (**infixr** ‹⇓› *53*)

primrec *eval* :: $'c$ *term* $\Rightarrow 'b$ (‹⦃-⦄›)
where $⦃\langle f\rangle_0⦄ = E\ f$
| $⦃\langle f\rangle⦄ = E\ f$
| $⦃t \star u⦄ = ⦃t⦄ \star ⦃u⦄$
| $⦃t \cdot u⦄ = ⦃t⦄ \cdot ⦃u⦄$
| $⦃\mathbf{l}[t]⦄ = \mathfrak{l}\ ⦃t⦄$
| $⦃\mathbf{l}^{-1}[t]⦄ = \mathfrak{l}'.map\ ⦃t⦄$
| $⦃\mathbf{r}[t]⦄ = \mathfrak{r}\ ⦃t⦄$
| $⦃\mathbf{r}^{-1}[t]⦄ = \mathfrak{r}'.map\ ⦃t⦄$
| $⦃\mathbf{a}[t,\ u,\ v]⦄ = \alpha\ (⦃t⦄,\ ⦃u⦄,\ ⦃v⦄)$
| $⦃\mathbf{a}^{-1}[t,\ u,\ v]⦄ = \alpha'.map\ (⦃t⦄,\ ⦃u⦄,\ ⦃v⦄)$

lemma *preserves-obj*:
assumes $C.obj\ a$
shows $obj\ (E\ a)$
proof (*unfold obj-def*)
show $arr\ (E\ a) \wedge src\ (E\ a) = E\ a$
proof
show $arr\ (E\ a)$ **using** *assms* $C.obj$-*simps* **by** *simp*
have $src\ (E\ a) = E\ (src_C\ a)$
using *assms preserves-src* **by** *metis*
also have ... $= E\ a$
using *assms* $C.obj$-*simps* **by** *simp*
finally show $src\ (E\ a) = E\ a$ **by** *simp*
qed
qed

lemma *eval-in-hom′*:
shows $Arr\ t \Longrightarrow «⦃t⦄ : ⦃Src\ t⦄ \rightarrow ⦃Trg\ t⦄» \wedge «⦃t⦄ : ⦃Dom\ t⦄ \Rightarrow ⦃Cod\ t⦄»$
proof (*induct t*)
show $\bigwedge x.\ Arr\ \langle x\rangle_0 \Longrightarrow$
$«⦃\langle x\rangle_0⦄ : ⦃Src\ \langle x\rangle_0⦄ \rightarrow ⦃Trg\ \langle x\rangle_0⦄» \wedge «⦃\langle x\rangle_0⦄ : ⦃Dom\ \langle x\rangle_0⦄ \Rightarrow ⦃Cod\ \langle x\rangle_0⦄»$
apply (*simp add*: *preserves-src preserves-trg*)
using *preserves-src preserves-trg* $C.objE$
by (*metis* (*full-types*) $C.obj\text{-}def'$ *E.preserves-arr E.preserves-ide in-hhom-def*

ide-in-hom(*2*))
show $\bigwedge x.$ *Arr* $\langle x\rangle \Longrightarrow$
«$\{\!|\langle x\rangle|\!\}$: $\{\!|$*Src* $\langle x\rangle|\!\} \rightarrow \{\!|$*Trg* $\langle x\rangle|\!\}$» $\wedge$ «$\{\!|\langle x\rangle|\!\}$: $\{\!|$*Dom* $\langle x\rangle|\!\} \Rightarrow \{\!|$*Cod* $\langle x\rangle|\!\}$»
by (*auto simp add*: *preserves-src preserves-trg*)
show $\bigwedge$*t1 t2*.
(*Arr t1* $\Longrightarrow$ «$\{\!|$*t1*$|\!\}$: $\{\!|$*Src t1*$|\!\} \rightarrow \{\!|$*Trg t1*$|\!\}$» $\wedge$ «$\{\!|$*t1*$|\!\}$: $\{\!|$*Dom t1*$|\!\} \Rightarrow \{\!|$*Cod t1*$|\!\}$») $\Longrightarrow$
(*Arr t2* $\Longrightarrow$ «$\{\!|$*t2*$|\!\}$: $\{\!|$*Src t2*$|\!\} \rightarrow \{\!|$*Trg t2*$|\!\}$» $\wedge$ «$\{\!|$*t2*$|\!\}$: $\{\!|$*Dom t2*$|\!\} \Rightarrow \{\!|$*Cod t2*$|\!\}$») $\Longrightarrow$
Arr (*t1* $\star$ *t2*) $\Longrightarrow$
«$\{\!|$*t1* $\star$ *t2*$|\!\}$: $\{\!|$*Src* (*t1* $\star$ *t2*)$|\!\} \rightarrow \{\!|$*Trg* (*t1* $\star$ *t2*)$|\!\}$» $\wedge$
«$\{\!|$*t1* $\star$ *t2*$|\!\}$: $\{\!|$*Dom* (*t1* $\star$ *t2*)$|\!\} \Rightarrow \{\!|$*Cod* (*t1* $\star$ *t2*)$|\!\}$»
using *hcomp-in-hhom in-hhom-def vconn-implies-hpar*(*1*) *vconn-implies-hpar*(*2*) **by** *auto*
show $\bigwedge$*t1 t2*.
(*Arr t1* $\Longrightarrow$ «$\{\!|$*t1*$|\!\}$: $\{\!|$*Src t1*$|\!\} \rightarrow \{\!|$*Trg t1*$|\!\}$» $\wedge$ «$\{\!|$*t1*$|\!\}$: $\{\!|$*Dom t1*$|\!\} \Rightarrow \{\!|$*Cod t1*$|\!\}$») $\Longrightarrow$
(*Arr t2* $\Longrightarrow$ «$\{\!|$*t2*$|\!\}$: $\{\!|$*Src t2*$|\!\} \rightarrow \{\!|$*Trg t2*$|\!\}$» $\wedge$ «$\{\!|$*t2*$|\!\}$: $\{\!|$*Dom t2*$|\!\} \Rightarrow \{\!|$*Cod t2*$|\!\}$») $\Longrightarrow$
Arr (*t1* $\cdot$ *t2*) $\Longrightarrow$
«$\{\!|$*t1* $\cdot$ *t2*$|\!\}$: $\{\!|$*Src* (*t1* $\cdot$ *t2*)$|\!\} \rightarrow \{\!|$*Trg* (*t1* $\cdot$ *t2*)$|\!\}$» $\wedge$
«$\{\!|$*t1* $\cdot$ *t2*$|\!\}$: $\{\!|$*Dom* (*t1* $\cdot$ *t2*)$|\!\} \Rightarrow \{\!|$*Cod* (*t1* $\cdot$ *t2*)$|\!\}$»
using *VSeq-implies-HPar seqI′* **by** *auto*
show $\bigwedge t.$ (*Arr t* $\Longrightarrow$ «$\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$») $\Longrightarrow$
Arr $\mathfrak{l}[t] \Longrightarrow$
«$\{\!|\mathfrak{l}[t]|\!\}$: $\{\!|$*Src* $\mathfrak{l}[t]|\!\} \rightarrow \{\!|$*Trg* $\mathfrak{l}[t]|\!\}$» $\wedge$ «$\{\!|\mathfrak{l}[t]|\!\}$: $\{\!|$*Dom* $\mathfrak{l}[t]|\!\} \Rightarrow \{\!|$*Cod* $\mathfrak{l}[t]|\!\}$»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t*
assume *t*: «$\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$»
assume *1*: *Arr t*
show «$\mathfrak{l}$ $\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$ «$\mathfrak{l}$ $\{\!|t|\!\}$: $\{\!|$*Trg t*$|\!\} \star \{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$»
proof −
have *src* ($\mathfrak{l}$ $\{\!|t|\!\}$) = $\{\!|$*Src t*$|\!\}$
using *t 1*
by (*metis* (*no-types*, *lifting*) $\mathfrak{l}$.*preserves-cod arr-cod in-hhomE map-simp src-cod*)
moreover have *trg* ($\mathfrak{l}$ $\{\!|t|\!\}$) = $\{\!|$*Trg t*$|\!\}$
using *t 1*
by (*metis* (*no-types*, *lifting*) $\mathfrak{l}$.*preserves-cod arr-cod in-hhomE map-simp trg-cod*)
moreover have «$\mathfrak{l}$ $\{\!|t|\!\}$: $\{\!|$*Trg t*$|\!\} \star \{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$»
using *t 1*
apply (*elim conjE in-hhomE*)
by (*intro in-homI*, *auto*)
ultimately show *?thesis* **by** *auto*
qed
qed
show $\bigwedge t.$ (*Arr t* $\Longrightarrow$ «$\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$») $\Longrightarrow$
Arr $\mathfrak{l}^{-1}[t] \Longrightarrow$
«$\{\!|\mathfrak{l}^{-1}[t]|\!\}$: $\{\!|$*Src* $\mathfrak{l}^{-1}[t]|\!\} \rightarrow \{\!|$*Trg* $\mathfrak{l}^{-1}[t]|\!\}$» $\wedge$
«$\{\!|\mathfrak{l}^{-1}[t]|\!\}$: $\{\!|$*Dom* $\mathfrak{l}^{-1}[t]|\!\} \Rightarrow \{\!|$*Cod* $\mathfrak{l}^{-1}[t]|\!\}$»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t*
assume *t*: «$\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\} \Rightarrow \{\!|$*Cod t*$|\!\}$»
assume *1*: *Arr t*
show «$\mathfrak{l}'$.*map* $\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\} \rightarrow \{\!|$*Trg t*$|\!\}$» $\wedge$

«$\mathfrak{l}'$.*map* ⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Trg t*⦄ ⋆ ⦃*Cod t*⦄»
proof −
have *src* ($\mathfrak{l}'$.*map* ⦃*t*⦄) = ⦃*Src t*⦄
using *t 1* $\mathfrak{l}'$.*preserves-dom arr-dom map-simp* $\mathfrak{l}'$.*preserves-reflects-arr src-dom*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have *trg* ($\mathfrak{l}'$.*map* ⦃*t*⦄) = ⦃*Trg t*⦄
using *t 1* $\mathfrak{l}'$.*preserves-dom arr-dom map-simp* $\mathfrak{l}'$.*preserves-reflects-arr trg-dom*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have «$\mathfrak{l}'$.*map* ⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Trg t*⦄ ⋆ ⦃*Cod t*⦄»
using *t 1* $\mathfrak{l}'$.*preserves-hom*
apply (*intro in-homI*)
apply *auto*[*1*]
apply *fastforce*
by *fastforce*
ultimately show *?thesis* **by** *blast*
qed
qed
show ⋀*t*. (*Arr t* ⟹ «⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧ «⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Cod t*⦄») ⟹
Arr **r**[*t*] ⟹
«⦃**r**[*t*]⦄ : ⦃*Src* **r**[*t*]⦄ → ⦃*Trg* **r**[*t*]⦄» ∧ «⦃**r**[*t*]⦄ : ⦃*Dom* **r**[*t*]⦄ ⇒ ⦃*Cod* **r**[*t*]⦄»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t*
assume *t*: «⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧ «⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Cod t*⦄»
assume *1*: *Arr t*
show «$\mathfrak{r}$ ⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧ «$\mathfrak{r}$ ⦃*t*⦄ : ⦃*Dom t*⦄ ⋆ ⦃*Src t*⦄ ⇒ ⦃*Cod t*⦄»
proof −
have *src* ($\mathfrak{r}$ ⦃*t*⦄) = ⦃*Src t*⦄
using *t 1* $\mathfrak{r}$.*preserves-cod arr-cod map-simp* $\mathfrak{r}$.*preserves-reflects-arr src-cod*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have *trg* ($\mathfrak{r}$ ⦃*t*⦄) = ⦃*Trg t*⦄
using *t 1* $\mathfrak{r}$.*preserves-cod arr-cod map-simp* $\mathfrak{r}$.*preserves-reflects-arr trg-cod*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have «$\mathfrak{r}$ ⦃*t*⦄ : ⦃*Dom t*⦄ ⋆ ⦃*Src t*⦄ ⇒ ⦃*Cod t*⦄»
using *t 1* **by** *force*
ultimately show *?thesis* **by** *blast*
qed
qed
show ⋀*t*. (*Arr t* ⟹ «⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧ «⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Cod t*⦄») ⟹
Arr $\mathbf{r}^{-1}$[*t*] ⟹
«⦃$\mathbf{r}^{-1}$[*t*]⦄ : ⦃*Src* $\mathbf{r}^{-1}$[*t*]⦄ → ⦃*Trg* $\mathbf{r}^{-1}$[*t*]⦄» ∧
«⦃$\mathbf{r}^{-1}$[*t*]⦄ : ⦃*Dom* $\mathbf{r}^{-1}$[*t*]⦄ ⇒ ⦃*Cod* $\mathbf{r}^{-1}$[*t*]⦄»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t*
assume *t*: «⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧ «⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Cod t*⦄»
assume *1*: *Arr t*
show «$\mathfrak{r}'$.*map* ⦃*t*⦄ : ⦃*Src t*⦄ → ⦃*Trg t*⦄» ∧
«$\mathfrak{r}'$.*map* ⦃*t*⦄ : ⦃*Dom t*⦄ ⇒ ⦃*Cod t*⦄ ⋆ ⦃*Src t*⦄»
proof −
have *src* ($\mathfrak{r}'$.*map* ⦃*t*⦄) = ⦃*Src t*⦄

using *t 1* $\mathfrak{r}'$*.preserves-dom arr-dom map-simp* $\mathfrak{r}'$*.preserves-reflects-arr src-dom*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have *trg* ($\mathfrak{r}'$*.map* $\{\!|t|\!\}$) = $\{\!|$*Trg t*$|\!\}$
using *t 1* $\mathfrak{r}'$*.preserves-dom arr-dom map-simp* $\mathfrak{r}'$*.preserves-reflects-arr trg-dom*
by (*metis* (*no-types*, *lifting*) *in-hhomE*)
moreover have «$\mathfrak{r}'$*.map* $\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\}$ ⇒ $\{\!|$*Cod t*$|\!\}$ ⋆ $\{\!|$*Src t*$|\!\}$»
using *t 1 src-cod arr-cod* $\mathfrak{r}'$*.preserves-hom* [*of* $\{\!|t|\!\}$ $\{\!|$*Dom t*$|\!\}$ $\{\!|$*Cod t*$|\!\}$]
apply (*elim conjE in-hhomE*)
by (*intro in-homI*, *auto*)
ultimately show *?thesis* **by** *blast*
qed
qed
show ⋀*t u v*.
(*Arr t* ⟹ «$\{\!|t|\!\}$: $\{\!|$*Src t*$|\!\}$ → $\{\!|$*Trg t*$|\!\}$» ∧ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\}$ ⇒ $\{\!|$*Cod t*$|\!\}$») ⟹
(*Arr u* ⟹ «$\{\!|u|\!\}$: $\{\!|$*Src u*$|\!\}$ → $\{\!|$*Trg u*$|\!\}$» ∧ «$\{\!|u|\!\}$: $\{\!|$*Dom u*$|\!\}$ ⇒ $\{\!|$*Cod u*$|\!\}$») ⟹
(*Arr v* ⟹ «$\{\!|v|\!\}$: $\{\!|$*Src v*$|\!\}$ → $\{\!|$*Trg v*$|\!\}$» ∧ «$\{\!|v|\!\}$: $\{\!|$*Dom v*$|\!\}$ ⇒ $\{\!|$*Cod v*$|\!\}$») ⟹
Arr **a**[*t*, *u*, *v*] ⟹
«$\{\!|$**a**[*t*, *u*, *v*]$|\!\}$: $\{\!|$*Src* **a**[*t*, *u*, *v*]$|\!\}$ → $\{\!|$*Trg* **a**[*t*, *u*, *v*]$|\!\}$» ∧
«$\{\!|$**a**[*t*, *u*, *v*]$|\!\}$: $\{\!|$*Dom* **a**[*t*, *u*, *v*]$|\!\}$ ⇒ $\{\!|$*Cod* **a**[*t*, *u*, *v*]$|\!\}$»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t u v*
assume *t*: «$\{\!|t|\!\}$: $\{\!|$*Trg u*$|\!\}$ → $\{\!|$*Trg t*$|\!\}$» ∧ «$\{\!|t|\!\}$: $\{\!|$*Dom t*$|\!\}$ ⇒ $\{\!|$*Cod t*$|\!\}$»
assume *u*: «$\{\!|u|\!\}$: $\{\!|$*Trg v*$|\!\}$ → $\{\!|$*Trg u*$|\!\}$» ∧ «$\{\!|u|\!\}$: $\{\!|$*Dom u*$|\!\}$ ⇒ $\{\!|$*Cod u*$|\!\}$»
assume *v*: «$\{\!|v|\!\}$: $\{\!|$*Src v*$|\!\}$ → $\{\!|$*Trg v*$|\!\}$» ∧ «$\{\!|v|\!\}$: $\{\!|$*Dom v*$|\!\}$ ⇒ $\{\!|$*Cod v*$|\!\}$»
assume *tuv*: *Arr t* ∧ *Arr u* ∧ *Arr v* ∧ *Src t* = *Trg u* ∧ *Src u* = *Trg v*
show «α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) : $\{\!|$*Src v*$|\!\}$ → $\{\!|$*Trg t*$|\!\}$» ∧
«α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) :
($\{\!|$*Dom t*$|\!\}$ ⋆ $\{\!|$*Dom u*$|\!\}$) ⋆ $\{\!|$*Dom v*$|\!\}$ ⇒ $\{\!|$*Cod t*$|\!\}$ ⋆ $\{\!|$*Cod u*$|\!\}$ ⋆ $\{\!|$*Cod v*$|\!\}$»
proof −
have *1*: *VVV.in-hom* ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)
($\{\!|$*Dom t*$|\!\}$, $\{\!|$*Dom u*$|\!\}$, $\{\!|$*Dom v*$|\!\}$) ($\{\!|$*Cod t*$|\!\}$, $\{\!|$*Cod u*$|\!\}$, $\{\!|$*Cod v*$|\!\}$)
proof −
have ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) ∈
VxVxV.hom ($\{\!|$*Dom t*$|\!\}$, $\{\!|$*Dom u*$|\!\}$, $\{\!|$*Dom v*$|\!\}$) ($\{\!|$*Cod t*$|\!\}$, $\{\!|$*Cod u*$|\!\}$, $\{\!|$*Cod v*$|\!\}$)
using *t u v tuv* **by** *simp*
moreover have ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) ∈
{$\tau\mu\nu$. *arr* (*fst* $\tau\mu\nu$) ∧ *VV.arr* (*snd* $\tau\mu\nu$) ∧
src (*fst* $\tau\mu\nu$) = *trg* (*fst* (*snd* $\tau\mu\nu$))}
using *t u v tuv* **by** *fastforce*
ultimately show *?thesis*
using *VVV.hom-char*
[*of* ($\{\!|$*Dom t*$|\!\}$, $\{\!|$*Dom u*$|\!\}$, $\{\!|$*Dom v*$|\!\}$) ($\{\!|$*Cod t*$|\!\}$, $\{\!|$*Cod u*$|\!\}$, $\{\!|$*Cod v*$|\!\}$)]
by *blast*
qed
have *4*: *VVV.arr* ($\{\!|$*Dom t*$|\!\}$, $\{\!|$*Dom u*$|\!\}$, $\{\!|$*Dom v*$|\!\}$)
using *1 VVV.ide-dom VVV.dom-simp* **by** (*elim VVV.in-homE*) *force*
have *5*: *VVV.arr* ($\{\!|$*Cod t*$|\!\}$, $\{\!|$*Cod u*$|\!\}$, $\{\!|$*Cod v*$|\!\}$)
using *1 VVV.ide-cod VVV.cod-simp* $VVV.in\text{-}hom\text{-}char_{SbC}$ **by** *blast*
have *2*: «α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) :

$(\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\}) \star \{\!|Dom\ v|\!\} \Rightarrow \{\!|Cod\ t|\!\} \star \{\!|Cod\ u|\!\} \star \{\!|Cod\ v|\!\}$»
using *1 4 5 HoHV-def HoVH-def* α*-def*
α*.preserves-hom* [*of* ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) ($\{\!|Dom\ t|\!\}$, $\{\!|Dom\ u|\!\}$, $\{\!|Dom\ v|\!\}$)
($\{\!|Cod\ t|\!\}$, $\{\!|Cod\ u|\!\}$, $\{\!|Cod\ v|\!\}$)]
by *simp*
have *3*: «α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$) : $\{\!|Src\ v|\!\} \rightarrow \{\!|Trg\ t|\!\}$»
proof
show *arr* (α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$))
using *2* **by** *auto*
show *src* (α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)) = $\{\!|Src\ v|\!\}$
proof −
have *src* (α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)) = *src* (($\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\}$) $\star$ $\{\!|Dom\ v|\!\}$)
using *2 src-dom* [*of* α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)] **by** *fastforce*
also have ... = *src* $\{\!|Dom\ v|\!\}$
using *4* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *simp*
also have ... = *src* (*dom* $\{\!|v|\!\}$)
using *v* **by** *fastforce*
also have ... = $\{\!|Src\ v|\!\}$
using *v* **by** *auto*
finally show *?thesis* **by** *auto*
qed
show *trg* (α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)) = $\{\!|Trg\ t|\!\}$
proof −
have *trg* (α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)) = *trg* (($\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\}$) $\star$ $\{\!|Dom\ v|\!\}$)
using *2 trg-dom* [*of* α ($\{\!|t|\!\}$, $\{\!|u|\!\}$, $\{\!|v|\!\}$)] **by** *fastforce*
also have ... = *trg* $\{\!|Dom\ t|\!\}$
using *4* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *simp*
also have ... = *trg* (*dom* $\{\!|t|\!\}$)
using *t* **by** *fastforce*
also have ... = $\{\!|Trg\ t|\!\}$
using *t* **by** *auto*
finally show *?thesis* **by** *auto*
qed
qed
show *?thesis* **using** *2 3* **by** *simp*
qed
qed
show $\bigwedge t\ u\ v.$
(*Arr t* $\Longrightarrow$ «$\{\!|t|\!\}$: $\{\!|Src\ t|\!\} \rightarrow \{\!|Trg\ t|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|Dom\ t|\!\} \Rightarrow \{\!|Cod\ t|\!\}$») $\Longrightarrow$
(*Arr u* $\Longrightarrow$ «$\{\!|u|\!\}$: $\{\!|Src\ u|\!\} \rightarrow \{\!|Trg\ u|\!\}$» $\wedge$ «$\{\!|u|\!\}$: $\{\!|Dom\ u|\!\} \Rightarrow \{\!|Cod\ u|\!\}$») $\Longrightarrow$
(*Arr v* $\Longrightarrow$ «$\{\!|v|\!\}$: $\{\!|Src\ v|\!\} \rightarrow \{\!|Trg\ v|\!\}$» $\wedge$ «$\{\!|v|\!\}$: $\{\!|Dom\ v|\!\} \Rightarrow \{\!|Cod\ v|\!\}$») $\Longrightarrow$
Arr $\mathbf{a}^{-1}[t,\ u,\ v]$ $\Longrightarrow$
«$\{\!|\mathbf{a}^{-1}[t,\ u,\ v]|\!\}$: $\{\!|Src\ \mathbf{a}^{-1}[t,\ u,\ v]|\!\} \rightarrow \{\!|Trg\ \mathbf{a}^{-1}[t,\ u,\ v]|\!\}$» $\wedge$
«$\{\!|\mathbf{a}^{-1}[t,\ u,\ v]|\!\}$: $\{\!|Dom\ \mathbf{a}^{-1}[t,\ u,\ v]|\!\} \Rightarrow \{\!|Cod\ \mathbf{a}^{-1}[t,\ u,\ v]|\!\}$»
proof (*simp add*: *preserves-src preserves-trg*)
fix *t u v*
assume *t*: «$\{\!|t|\!\}$: $\{\!|Trg\ u|\!\} \rightarrow \{\!|Trg\ t|\!\}$» $\wedge$ «$\{\!|t|\!\}$: $\{\!|Dom\ t|\!\} \Rightarrow \{\!|Cod\ t|\!\}$»
assume *u*: «$\{\!|u|\!\}$: $\{\!|Trg\ v|\!\} \rightarrow \{\!|Trg\ u|\!\}$» $\wedge$ «$\{\!|u|\!\}$: $\{\!|Dom\ u|\!\} \Rightarrow \{\!|Cod\ u|\!\}$»
assume *v*: «$\{\!|v|\!\}$: $\{\!|Src\ v|\!\} \rightarrow \{\!|Trg\ v|\!\}$» $\wedge$ «$\{\!|v|\!\}$: $\{\!|Dom\ v|\!\} \Rightarrow \{\!|Cod\ v|\!\}$»

assume *tuv*: $Arr\ t \wedge Arr\ u \wedge Arr\ v \wedge Src\ t = Trg\ u \wedge Src\ u = Trg\ v$
show «$\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}) : \{\!|Src\ v|\!\} \rightarrow \{\!|Trg\ t|\!\}$» $\wedge$
«$\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}) :$
$\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\} \star \{\!|Dom\ v|\!\} \Rightarrow (\{\!|Cod\ t|\!\} \star \{\!|Cod\ u|\!\}) \star \{\!|Cod\ v|\!\}$»
proof −
have *1*: $VVV.in\text{-}hom\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$
$(\{\!|Dom\ t|\!\}, \{\!|Dom\ u|\!\}, \{\!|Dom\ v|\!\})\ (\{\!|Cod\ t|\!\}, \{\!|Cod\ u|\!\}, \{\!|Cod\ v|\!\})$
using *t u v tuv* $VVV.hom\text{-}char\ VVV.arr\text{-}char_{SbC}\ VV.arr\text{-}char_{SbC}\ VVV.dom\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$
apply (*elim conjE in-hhomE in-homE*)
apply (*intro* $VVV.in\text{-}homI$)
by *simp-all*
have *4*: $VVV.arr\ (\{\!|Dom\ t|\!\}, \{\!|Dom\ u|\!\}, \{\!|Dom\ v|\!\})$
using *1* $VVV.in\text{-}hom\text{-}char_{SbC}$ **by** *blast*
have *5*: $VVV.arr\ (\{\!|Cod\ t|\!\}, \{\!|Cod\ u|\!\}, \{\!|Cod\ v|\!\})$
using *1* $VVV.in\text{-}hom\text{-}char_{SbC}$ **by** *blast*
have *2*: «$\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}) :$
$\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\} \star \{\!|Dom\ v|\!\} \Rightarrow (\{\!|Cod\ t|\!\} \star \{\!|Cod\ u|\!\}) \star \{\!|Cod\ v|\!\}$»
using *1 4 5 HoHV-def HoVH-def* $\alpha'.map\text{-}def$
$\alpha'.preserves\text{-}hom\ [of\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})\ (\{\!|Dom\ t|\!\}, \{\!|Dom\ u|\!\}, \{\!|Dom\ v|\!\})$
$(\{\!|Cod\ t|\!\}, \{\!|Cod\ u|\!\}, \{\!|Cod\ v|\!\})]$
by *simp*
have *3*: «$\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}) : \{\!|Src\ v|\!\} \rightarrow \{\!|Trg\ t|\!\}$»
proof
show $arr\ (\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}))$
using *2* **by** *auto*
show $src\ (\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) = \{\!|Src\ v|\!\}$
proof −
have $src\ (\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) = src\ (\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\} \star \{\!|Dom\ v|\!\})$
using *2* $src\text{-}dom\ [of\ \alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})]$ **by** *fastforce*
also have ... $= src\ \{\!|Dom\ v|\!\}$
using *4* $VVV.arr\text{-}char_{SbC}\ VV.arr\text{-}char_{SbC}$ **by** *simp*
also have ... $= src\ (dom\ \{\!|v|\!\})$
using *v* **by** *fastforce*
also have ... $= \{\!|Src\ v|\!\}$
using *v* **by** *auto*
finally show *?thesis* **by** *auto*
qed
show $trg\ (\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) = \{\!|Trg\ t|\!\}$
proof −
have $trg\ (\alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) = trg\ (\{\!|Dom\ t|\!\} \star \{\!|Dom\ u|\!\} \star \{\!|Dom\ v|\!\})$
using *2* $trg\text{-}dom\ [of\ \alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})]$ **by** *fastforce*
also have ... $= trg\ \{\!|Dom\ t|\!\}$
using *4* $VVV.arr\text{-}char_{SbC}\ VV.arr\text{-}char_{SbC}\ hseqI'$ **by** *simp*
also have ... $= trg\ (dom\ \{\!|t|\!\})$
using *t* **by** *fastforce*
also have ... $= \{\!|Trg\ t|\!\}$
using *t* **by** *auto*
finally show *?thesis* **by** *auto*

```
      qed
    qed
    show ?thesis using 2 3 by simp
  qed
 qed
qed

lemma eval-in-hom [intro]:
assumes Arr t
shows «⦃t⦄ : ⦃Src t⦄ → ⦃Trg t⦄» and «⦃t⦄ : ⦃Dom t⦄ ⇒ ⦃Cod t⦄»
  using assms eval-in-hom' by simp-all


lemma eval-simps:
assumes Arr f
shows arr ⦃f⦄ and ⦃Src f⦄ = src ⦃f⦄ and ⦃Trg f⦄ = trg ⦃f⦄
and ⦃Dom f⦄ = dom ⦃f⦄ and ⦃Cod f⦄ = cod ⦃f⦄
  using assms eval-in-hom [of f] by auto

lemma eval-simps':
assumes Arr f
shows arr ⦃f⦄ and src ⦃f⦄ = ⦃Src f⦄ and trg ⦃f⦄ = ⦃Trg f⦄
and dom ⦃f⦄ = ⦃Dom f⦄ and cod ⦃f⦄ = ⦃Cod f⦄
  using assms eval-in-hom by auto

lemma obj-eval-Obj:
shows Obj t ⟹ obj ⦃t⦄
  using preserves-obj
  by (induct t) auto

lemma ide-eval-Ide:
shows Ide t ⟹ ide ⦃t⦄
  by (induct t, auto simp add: eval-simps')

lemma arr-eval-Arr [simp]:
assumes Arr t
shows arr ⦃t⦄
  using assms by (simp add: eval-simps')


lemma eval-Lunit [simp]:
assumes Arr t
shows ⦃𝗅[t]⦄ = 𝗅[⦃Cod t⦄] · (trg ⦃t⦄ ⋆ ⦃t⦄)
  using assms 𝗅.naturality2 𝗅-ide-simp by (simp add: eval-simps)

lemma eval-Lunit' [simp]:
assumes Arr t
shows ⦃𝗅⁻¹[t]⦄ = 𝗅⁻¹[⦃Cod t⦄] · ⦃t⦄
```

using *assms* $\mathfrak{l}'$*.naturality2* $\mathfrak{l}$*-ide-simp* **by** (*simp add*: *eval-simps*)

lemma *eval-Runit* [*simp*]:
assumes *Arr t*
shows $\{\!|\mathbf{r}[t]|\!\} = \mathrm{r}[\{\!|Cod\ t|\!\}] \cdot (\{\!|t|\!\} \star src\ \{\!|t|\!\})$
using *assms* $\mathfrak{r}$*.naturality2* $\mathfrak{r}$*-ide-simp* **by** (*simp add*: *eval-simps*)

lemma *eval-Runit'* [*simp*]:
assumes *Arr t*
shows $\{\!|\mathbf{r}^{-1}[t]|\!\} = \mathrm{r}^{-1}[\{\!|Cod\ t|\!\}] \cdot \{\!|t|\!\}$
using *assms* $\mathfrak{r}'$*.naturality2* $\mathfrak{r}$*-ide-simp* **by** (*simp add*: *eval-simps*)

lemma *eval-Assoc* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\{\!|\mathbf{a}[t, u, v]|\!\} = \alpha\ (cod\ \{\!|t|\!\}, cod\ \{\!|u|\!\}, cod\ \{\!|v|\!\}) \cdot ((\{\!|t|\!\} \star \{\!|u|\!\}) \star \{\!|v|\!\})$
proof −
have $\{\!|\mathbf{a}[t, u, v]|\!\} = \alpha\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$ **by** *simp*
also have ... $= \alpha\ (VVV.cod\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) \cdot HoHV\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$
using *assms* α*.naturality2* [*of* $(\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$] $VVV.arr\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ α*.extensionality* α*-def*
by *auto*
also have ... $= \alpha\ (cod\ \{\!|t|\!\}, cod\ \{\!|u|\!\}, cod\ \{\!|v|\!\}) \cdot ((\{\!|t|\!\} \star \{\!|u|\!\}) \star \{\!|v|\!\})$
unfolding *HoHV-def* α*-def*
using *assms* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ α*.extensionality*
by *auto*
finally show *?thesis* **by** *simp*
qed

lemma *eval-Assoc'* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\{\!|\mathbf{a}^{-1}[t, u, v]|\!\} = \mathrm{a}^{-1}[cod\ \{\!|t|\!\}, cod\ \{\!|u|\!\}, cod\ \{\!|v|\!\}] \cdot (\{\!|t|\!\} \star \{\!|u|\!\} \star \{\!|v|\!\})$
proof −
have $\{\!|\mathbf{a}^{-1}[t, u, v]|\!\} = \alpha'.map\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$ **by** *simp*
also have ... $= \alpha'.map\ (VVV.cod\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})) \cdot HoVH\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$
using *assms* α'*.naturality2* [*of* $(\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$] $VVV.arr\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ α'*.extensionality*
by *simp*
also have ... $= \alpha'.map\ (cod\ \{\!|t|\!\}, cod\ \{\!|u|\!\}, cod\ \{\!|v|\!\}) \cdot (\{\!|t|\!\} \star \{\!|u|\!\} \star \{\!|v|\!\})$
unfolding *HoVH-def*
using *assms* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VVV.cod\text{-}char_{SbC}$ α'*.extensionality*
apply *simp*
using *eval-simps'*(*2*) *eval-simps'*(*3*) **by** *presburger*
finally show *?thesis*
using a'*-def* **by** *simp*
qed

lemma *eval-Lunit-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{l}[t]|\!\} = \mathrm{l}[\{\!|t|\!\}]$

using *assms* $\mathfrak{l}$*-ide-simp ide-eval-Ide* **by** *simp*

lemma *eval-Lunit′-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{l}^{-1}[t]|\!\} = \mathrm{l}^{-1}[\{\!|t|\!\}]$
using *assms* $\mathfrak{l}$*-ide-simp ide-eval-Ide* **by** *simp*

lemma *eval-Runit-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{r}[t]|\!\} = \mathrm{r}[\{\!|t|\!\}]$
using *assms* $\mathfrak{r}$*-ide-simp ide-eval-Ide* **by** *simp*

lemma *eval-Runit′-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{r}^{-1}[t]|\!\} = \mathrm{r}^{-1}[\{\!|t|\!\}]$
using *assms* $\mathfrak{r}$*-ide-simp ide-eval-Ide* **by** *simp*

lemma *eval-Assoc-Ide* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Ide v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\{\!|\mathbf{a}[t, u, v]|\!\} = \alpha\ (\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\})$
using *assms* **by** *simp*

lemma *eval-Assoc′-Ide* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Ide v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\{\!|\mathbf{a}^{-1}[t, u, v]|\!\} = \mathrm{a}^{-1}[\{\!|t|\!\}, \{\!|u|\!\}, \{\!|v|\!\}]$
using *assms* a′*-def* **by** *simp*

lemma *iso-eval-Can*:
shows *Can t* $\Longrightarrow$ *iso* $\{\!|t|\!\}$
proof (*induct t*; *simp add*: *fsts.intros snds.intros*)
show $\bigwedge x.\ C.obj\ x \Longrightarrow iso\ (E\ x)$ **by** *auto*
show $\bigwedge t1\ t2.\ [\![\ iso\ \{\!|t1|\!\};\ iso\ \{\!|t2|\!\};\ Can\ t1 \wedge Can\ t2 \wedge Src\ t1 = Trg\ t2\ ]\!] \Longrightarrow$
$iso\ (\{\!|t1|\!\} \star \{\!|t2|\!\})$
using *Can-implies-Arr* **by** (*simp add*: *eval-simps′*)
show $\bigwedge t1\ t2.\ [\![\ iso\ \{\!|t1|\!\};\ iso\ \{\!|t2|\!\};\ Can\ t1 \wedge Can\ t2 \wedge Dom\ t1 = Cod\ t2\ ]\!] \Longrightarrow$
$iso\ (\{\!|t1|\!\} \cdot \{\!|t2|\!\})$
using *Can-implies-Arr isos-compose* **by** (*simp add*: *eval-simps′*)
show $\bigwedge t.\ [\![\ iso\ \{\!|t|\!\};\ Can\ t\ ]\!] \Longrightarrow iso\ (\mathfrak{l}\ \{\!|t|\!\})$
using $\mathfrak{l}$*.preserves-iso* **by** *auto*
show $\bigwedge t.\ [\![\ iso\ \{\!|t|\!\};\ Can\ t\ ]\!] \Longrightarrow iso\ (\mathfrak{l}'.map\ \{\!|t|\!\})$
using $\mathfrak{l}'$*.preserves-iso* **by** *simp*
show $\bigwedge t.\ [\![\ iso\ \{\!|t|\!\};\ Can\ t\ ]\!] \Longrightarrow iso\ (\mathfrak{r}\ \{\!|t|\!\})$
using $\mathfrak{r}$*.preserves-iso* **by** *auto*
show $\bigwedge t.\ [\![\ iso\ \{\!|t|\!\};\ Can\ t\ ]\!] \Longrightarrow iso\ (\mathfrak{r}'.map\ \{\!|t|\!\})$
using $\mathfrak{r}'$*.preserves-iso* **by** *simp*
fix *t1 t2 t3*
assume *t1*: *iso* $\{\!|t1|\!\}$ **and** *t2*: *iso* $\{\!|t2|\!\}$ **and** *t3*: *iso* $\{\!|t3|\!\}$
assume *1*: *Can t1* $\wedge$ *Can t2* $\wedge$ *Can t3* $\wedge$ *Src t1* = *Trg t2* $\wedge$ *Src t2* = *Trg t3*
have *2*: *VVV.iso* $(\{\!|t1|\!\}, \{\!|t2|\!\}, \{\!|t3|\!\})$

proof −
have *3*: *VxVxV.iso* (⦃*t1*⦄, ⦃*t2*⦄, ⦃*t3*⦄)
using *t1 t2 t3 Can-implies-Arr VxVxV.iso-char VxV.iso-char* **by** *simp*
moreover have *VVV.arr* (*VxVxV.inv* (⦃*t1*⦄, ⦃*t2*⦄, ⦃*t3*⦄))
proof −
have *VxVxV.inv* (⦃*t1*⦄, ⦃*t2*⦄, ⦃*t3*⦄) = (*inv* ⦃*t1*⦄, *inv* ⦃*t2*⦄, *inv* ⦃*t3*⦄)
using *t1 t2 t3 3* **by** *simp*
thus *?thesis*
using *t1 t2 t3 1 Can-implies-Arr VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by (*simp add*: *eval-simps′*)
qed
ultimately show *?thesis*
using *t1 t2 t3 1 Can-implies-Arr VVV.iso-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$
by (*auto simp add*: *eval-simps′*)
qed
show *iso* (α (⦃*t1*⦄, ⦃*t2*⦄, ⦃*t3*⦄))
using *2 α-def α.preserves-iso* **by** *auto*
show *iso* (*α′.map* (⦃*t1*⦄, ⦃*t2*⦄, ⦃*t3*⦄))
using *2 α′.preserves-iso* **by** *simp*
qed

lemma *eval-Inv-Can*:
shows *Can t* ⟹ ⦃*Inv t*⦄ = *inv* ⦃*t*⦄
proof (*induct t*)
show ⋀*x*. *Can* ⟨*x*⟩$_0$ ⟹ ⦃*Inv* ⟨*x*⟩$_0$⦄ = *inv* ⦃⟨*x*⟩$_0$⦄ **by** *auto*
show ⋀*x*. *Can* ⟨*x*⟩ ⟹ ⦃*Inv* ⟨*x*⟩⦄ = *inv* ⦃⟨*x*⟩⦄ **by** *simp*
show ⋀*t1 t2*. (*Can t1* ⟹ ⦃*Inv t1*⦄ = *inv* ⦃*t1*⦄) ⟹
(*Can t2* ⟹ ⦃*Inv t2*⦄ = *inv* ⦃*t2*⦄) ⟹
Can (*t1* ⋆ *t2*) ⟹ ⦃*Inv* (*t1* ⋆ *t2*)⦄ = *inv* ⦃*t1* ⋆ *t2*⦄
using *iso-eval-Can Can-implies-Arr*
by (*simp add*: *eval-simps′*)
show ⋀*t1 t2*. (*Can t1* ⟹ ⦃*Inv t1*⦄ = *inv* ⦃*t1*⦄) ⟹
(*Can t2* ⟹ ⦃*Inv t2*⦄ = *inv* ⦃*t2*⦄) ⟹
Can (*t1* · *t2*) ⟹ ⦃*Inv* (*t1* · *t2*)⦄ = *inv* ⦃*t1* · *t2*⦄
using *iso-eval-Can inv-comp Can-implies-Arr*
by (*simp add*: *eval-simps′*)
fix *t*
assume *I*: *Can t* ⟹ ⦃*Inv t*⦄ = *inv* ⦃*t*⦄
show *Can* $\mathfrak{l}[t]$ ⟹ ⦃*Inv* $\mathfrak{l}[t]$⦄ = *inv* ⦃$\mathfrak{l}[t]$⦄
proof −
assume *t*: *Can* $\mathfrak{l}[t]$
have ⦃*Inv* $\mathfrak{l}[t]$⦄ = ⦃$\mathfrak{l}^{-1}$[*Inv t*]⦄ **by** *simp*
also have ... = $\mathfrak{l}'$*.map* (*inv* ⦃*t*⦄)
using *t I* **by** *simp*
also have ... = $\mathfrak{l}'$*.map* (*cod* (*inv* ⦃*t*⦄)) · *inv* ⦃*t*⦄
using *t* $\mathfrak{l}'$*.naturality2 iso-inv-iso iso-eval-Can iso-is-arr*
by (*metis* (*no-types*, *lifting*) *Can.simps*(*5*) *map-simp*)
also have ... = *inv* (⦃*t*⦄ · $\mathfrak{l}$ (*dom* ⦃*t*⦄))
proof −

```
    have 1: iso ⦃t⦄ using t iso-eval-Can by simp
    moreover have iso (𝔩 (dom ⦃t⦄))
      using t 1 𝔩.components-are-iso ide-dom by blast
    moreover have seq ⦃t⦄ (𝔩 (dom ⦃t⦄))
      using t 1 iso-is-arr by auto
    ultimately show ?thesis
      using t 1 inv-comp by auto
  qed
  also have ... = inv ⦃𝗹[t]⦄
    using t iso-eval-Can 𝔩-ide-simp lunit-naturality Can-implies-Arr eval-Lunit
    by (auto simp add: eval-simps)
  finally show ?thesis by blast
qed
show Can 𝗹⁻¹[t] ⟹ ⦃Inv 𝗹⁻¹[t]⦄ = inv ⦃𝗹⁻¹[t]⦄
proof −
  assume t: Can (Lunit' t)
  have ⦃Inv (Lunit' t)⦄ = ⦃Lunit (Inv t)⦄ by simp
  also have ... = 𝔩 (inv ⦃t⦄)
    using t I by simp
  also have ... = inv ⦃t⦄ · 𝔩 (dom (inv ⦃t⦄))
    using t 𝔩.naturality1 iso-inv-iso iso-eval-Can iso-is-arr
    by (metis (no-types, lifting) Can.simps(6) map-simp)
  also have ... = inv (𝔩'.map (cod ⦃t⦄) · ⦃t⦄)
  proof −
    have 1: iso ⦃t⦄ using t iso-eval-Can by simp
    moreover have iso (𝔩'.map (cod ⦃t⦄))
      using t 1 𝔩'.components-are-iso ide-cod by blast
    moreover have seq (𝔩'.map (cod ⦃t⦄)) ⦃t⦄
      using t 1 iso-is-arr by auto
    ultimately show ?thesis
      using t 1 inv-comp by auto
  qed
  also have ... = inv ⦃𝗹⁻¹[t]⦄
    using t 𝔩'.naturality2 iso-eval-Can iso-is-arr by force
  finally show ?thesis by auto
qed
show Can 𝗿[t] ⟹ ⦃Inv 𝗿[t]⦄ = inv ⦃𝗿[t]⦄
proof −
  assume t: Can 𝗿[t]
  have ⦃Inv 𝗿[t]⦄ = ⦃𝗿⁻¹[Inv t]⦄ by simp
  also have ... = 𝔯'.map (inv ⦃t⦄)
    using t I by simp
  also have ... = 𝔯'.map (cod (inv ⦃t⦄)) · inv ⦃t⦄
    using t 𝔯'.naturality2 map-simp iso-inv-iso iso-eval-Can iso-is-arr
    by (metis (no-types, lifting) Can.simps(7))
  also have ... = inv (⦃t⦄ · 𝔯 (dom ⦃t⦄))
  proof −
    have 1: iso ⦃t⦄ using t iso-eval-Can by simp
    moreover have iso (𝔯 (dom ⦃t⦄))
```

using *t 1* $\mathfrak{r}$.*components-are-iso ide-dom* **by** *blast*
moreover have *seq* $\{\!|t|\!\}$ ($\mathfrak{r}$ (*dom* $\{\!|t|\!\}$))
using *t 1 iso-is-arr* **by** *simp*
ultimately show *?thesis*
using *t 1 inv-comp* **by** *auto*
qed
also have ... = *inv* $\{\!|\mathbf{r}[t]|\!\}$
using *t* $\mathfrak{r}$*-ide-simp iso-eval-Can runit-naturality Can-implies-Arr eval-Runit*
by (*auto simp add*: *eval-simps*)
finally show *?thesis* **by** *blast*
qed
show *Can* $\mathbf{r}^{-1}[t] \Longrightarrow \{\!|Inv\ \mathbf{r}^{-1}[t]|\!\} = inv\ \{\!|\mathbf{r}^{-1}[t]|\!\}$
proof −
assume *t*: *Can* $\mathbf{r}^{-1}[t]$
have $\{\!|Inv\ \mathbf{r}^{-1}[t]|\!\} = \{\!|\mathbf{r}[Inv\ t]|\!\}$
by *simp*
also have ... = $\mathfrak{r}$ (*inv* $\{\!|t|\!\}$)
using *t I* **by** *simp*
also have ... = *inv* $\{\!|t|\!\} \cdot \mathfrak{r}$ (*dom* (*inv* $\{\!|t|\!\}$))
using *t* $\mathfrak{r}$.*naturality1 map-simp iso-inv-iso iso-eval-Can iso-is-arr*
by (*metis* (*no-types*, *lifting*) *Can.simps*(*8*))
also have ... = *inv* ($\mathfrak{r}'$.*map* (*cod* $\{\!|t|\!\}$) $\cdot$ $\{\!|t|\!\}$)
proof −
have *1*: *iso* $\{\!|t|\!\}$ **using** *t iso-eval-Can* **by** *simp*
moreover have *iso* ($\mathfrak{r}'$.*map* (*cod* $\{\!|t|\!\}$))
using *t 1* $\mathfrak{r}'$.*components-are-iso ide-cod* **by** *blast*
moreover have *seq* ($\mathfrak{r}'$.*map* (*cod* $\{\!|t|\!\}$)) $\{\!|t|\!\}$
using *t 1 iso-is-arr* **by** *auto*
ultimately show *?thesis*
using *t 1 inv-comp* **by** *auto*
qed
also have ... = *inv* $\{\!|\mathbf{r}^{-1}[t]|\!\}$
using *t* $\mathfrak{r}'$.*naturality2 iso-eval-Can iso-is-arr* **by** *auto*
finally show *?thesis* **by** *auto*
qed
next
fix *t u v*
assume *I1*: *Can t* $\Longrightarrow \{\!|Inv\ t|\!\} = inv\ \{\!|t|\!\}$
assume *I2*: *Can u* $\Longrightarrow \{\!|Inv\ u|\!\} = inv\ \{\!|u|\!\}$
assume *I3*: *Can v* $\Longrightarrow \{\!|Inv\ v|\!\} = inv\ \{\!|v|\!\}$
show *Can* $\mathbf{a}[t, u, v] \Longrightarrow \{\!|Inv\ \mathbf{a}[t, u, v]|\!\} = inv\ \{\!|\mathbf{a}[t, u, v]|\!\}$
proof −
assume *Can* $\mathbf{a}[t, u, v]$
hence *tuv*: *Can t* $\wedge$ *Can u* $\wedge$ *Can v* $\wedge$ *Src t* = *Trg u* $\wedge$ *Src u* = *Trg v* **by** *simp*
have $\{\!|Inv\ \mathbf{a}[t, u, v]|\!\} = \{\!|\mathbf{a}^{-1}[Inv\ t, Inv\ u, Inv\ v]|\!\}$ **by** *simp*
also have ... = $\mathrm{a}^{-1}[dom\ \{\!|t|\!\}, dom\ \{\!|u|\!\}, dom\ \{\!|v|\!\}] \cdot (inv\ \{\!|t|\!\} \star inv\ \{\!|u|\!\} \star inv\ \{\!|v|\!\})$
using *tuv I1 I2 I3 eval-in-hom* α'.*map-ide-simp inv-in-hom iso-eval-Can assoc'-naturality*
Can-implies-Arr Src-Inv Trg-Inv eval-Assoc' Dom-Inv Can-Inv cod-inv
by *presburger*

also have ... = *inv* (($\llbracket t \rrbracket$ ⋆ $\llbracket u \rrbracket$ ⋆ $\llbracket v \rrbracket$) · α (*dom* $\llbracket t \rrbracket$, *dom* $\llbracket u \rrbracket$, *dom* $\llbracket v \rrbracket$))
using *tuv iso-eval-Can Can-implies-Arr eval-simps′*(*2*) *eval-simps′*(*3*) *α-def*
by (*simp add*: *inv-comp*)
also have ... = *inv* (α ($\llbracket t \rrbracket$, $\llbracket u \rrbracket$, $\llbracket v \rrbracket$))
using *tuv Can-implies-Arr α-def*
by (*metis assoc-naturality1 arr-eval-Arr eval-simps′*(*2*) *eval-simps′*(*3*) *fst-conv snd-conv*)
also have ... = *inv* $\llbracket \mathbf{a}[t, u, v] \rrbracket$ **by** *simp*
finally show *?thesis* **by** *blast*
qed
show *Can* $\mathbf{a}^{-1}[t, u, v] \Longrightarrow \llbracket$*Inv* $\mathbf{a}^{-1}[t, u, v] \rrbracket$ = *inv* $\llbracket \mathbf{a}^{-1}[t, u, v] \rrbracket$
proof −
assume *tuv*: *Can* $\mathbf{a}^{-1}[t, u, v]$
have *t*: *Can t* **using** *tuv* **by** *simp*
have *u*: *Can u* **using** *tuv* **by** *simp*
have *v*: *Can v* **using** *tuv* **by** *simp*
have $\llbracket$*Inv* $\mathbf{a}^{-1}[t, u, v] \rrbracket$ = $\llbracket \mathbf{a}[$*Inv t*, *Inv u*, *Inv v*$] \rrbracket$ **by** *simp*
also have ... = (*inv* $\llbracket t \rrbracket$ ⋆ *inv* $\llbracket u \rrbracket$ ⋆ *inv* $\llbracket v \rrbracket$) · α (*cod* $\llbracket t \rrbracket$, *cod* $\llbracket u \rrbracket$, *cod* $\llbracket v \rrbracket$)
using *α-def tuv I1 I2 I3 iso-eval-Can Can-implies-Arr eval-simps′*(*2*) *eval-simps′*(*3*)
apply *simp*
using *assoc-naturality1 arr-inv dom-inv src-inv trg-inv* **by** *presburger*
also have ... = *inv* (a^{-1}[*cod* $\llbracket t \rrbracket$, *cod* $\llbracket u \rrbracket$, *cod* $\llbracket v \rrbracket$] · ($\llbracket t \rrbracket$ ⋆ $\llbracket u \rrbracket$ ⋆ $\llbracket v \rrbracket$))
using *tuv inv-comp* [*of* $\llbracket t \rrbracket$ ⋆ $\llbracket u \rrbracket$ ⋆ $\llbracket v \rrbracket$ a^{-1}[*cod* $\llbracket t \rrbracket$, *cod* $\llbracket u \rrbracket$, *cod* $\llbracket v \rrbracket$]]
Can-implies-Arr iso-assoc α-def
by (*simp add*: *eval-simps′*(*2*) *eval-simps′*(*3*) *iso-eval-Can*)
also have *1*: ... = *inv* ((($\llbracket t \rrbracket$ ⋆ $\llbracket u \rrbracket$) ⋆ $\llbracket v \rrbracket$) · a^{-1}[*dom* $\llbracket t \rrbracket$, *dom* $\llbracket u \rrbracket$, *dom* $\llbracket v \rrbracket$])
using *tuv assoc′-naturality* [*of* $\llbracket t \rrbracket$ $\llbracket u \rrbracket$ $\llbracket v \rrbracket$] *Can-implies-Arr*
eval-simps′(*2*) *eval-simps′*(*3*)
by *simp*
also have ... = *inv* $\llbracket \mathbf{a}^{-1}[t, u, v] \rrbracket$
using *tuv 1 Can-implies-Arr eval-Assoc′* **by** *auto*
finally show *?thesis* **by** *blast*
qed
qed

lemma *eval-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *VSeq t u*
shows $\llbracket t \lfloor\cdot\rfloor u \rrbracket$ = $\llbracket t \rrbracket$ · $\llbracket u \rrbracket$
proof −
have $\bigwedge u.$ ⟦ *Nml t*; *Nml u*; *VSeq t u* ⟧ $\Longrightarrow$ $\llbracket t \lfloor\cdot\rfloor u \rrbracket$ = $\llbracket t \rrbracket$ · $\llbracket u \rrbracket$
proof (*induct t*, *simp-all add*: *eval-simps*)
fix *u a*
assume *u*: *Nml u*
assume *1*: *Arr u* ∧ $\langle a \rangle_0$ = *Cod u*
show $\llbracket u \rrbracket$ = *cod* $\llbracket u \rrbracket$ · $\llbracket u \rrbracket$
using *1 comp-cod-arr* **by** *simp*
next
fix *u f*
assume *u*: *Nml u*
assume *f*: *C.arr f*

```
    assume 1: Arr u ∧ ⟨C.dom f⟩ = Cod u
    show ⦃⟨f⟩ ⌊·⌋ u⦄ = E f · ⦃u⦄
      using f u 1 as-nat-trans.preserves-comp-2 by (cases u; simp)
    next
    fix u v w
    assume I1: ⋀u. ⟦ Nml v; Nml u; Arr u ∧ Dom v = Cod u ⟧ ⟹ ⦃v ⌊·⌋ u⦄ = ⦃v⦄ · ⦃u⦄
    assume I2: ⋀u. ⟦ Nml w; Nml u; Arr u ∧ Dom w = Cod u ⟧ ⟹ ⦃w ⌊·⌋ u⦄ = ⦃w⦄ · ⦃u⦄
    assume vw: Nml (v ⋆ w)
    have v: Nml v ∧ v = Prim (un-Prim v)
      using vw by (simp add: Nml-HcompD)
    have w: Nml w
      using vw by (simp add: Nml-HcompD)
    assume u: Nml u
    assume 1: Arr v ∧ Arr w ∧ Src v = Trg w ∧ Arr u ∧ Dom v ⋆ Dom w = Cod u
    show ⦃(v ⋆ w) ⌊·⌋ u⦄ = (⦃v⦄ ⋆ ⦃w⦄) · ⦃u⦄
      using u 1 HcompNml-in-Hom apply (cases u, simp-all)
    proof −
      fix x y
      assume 3: u = x ⋆ y
      have x: Nml x
        using u 1 3 Nml-HcompD by blast
      have y: Nml y
        using u x 1 3 Nml-HcompD by blast
      assume 4: Arr v ∧ Arr w ∧ Src v = Trg w ∧ Dom v = Cod x ∧ Dom w = Cod y
      have ⦃v ⌊·⌋ x⦄ ⋆ ⦃w ⌊·⌋ y⦄ = ⦃v ⌊·⌋ x ⋆ w ⌊·⌋ y⦄
        using v w x y 4 HcompNml-in-Hom by simp
      moreover have ... = ⦃v ⌊·⌋ x⦄ ⋆ ⦃w ⌊·⌋ y⦄ by simp
      moreover have ... = ⦃v⦄ · ⦃x⦄ ⋆ ⦃w⦄ · ⦃y⦄
        using v w x y 4 I1 [of x] I2 [of y] Nml-implies-Arr by simp
      moreover have ... = (⦃v⦄ ⋆ ⦃w⦄) · (⦃x⦄ ⋆ ⦃y⦄)
        using v w x y 4 Nml-implies-Arr interchange [of ⦃v⦄ ⦃x⦄ ⦃w⦄ ⦃y⦄]
        by (simp add: eval-simps′)
      ultimately have ⦃v ⌊·⌋ x⦄ ⋆ ⦃w ⌊·⌋ y⦄ = (⦃v⦄ ⋆ ⦃w⦄) · (⦃x⦄ ⋆ ⦃y⦄) by presburger
      moreover have arr ⦃v ⌊·⌋ x⦄ ∧ arr ⦃w ⌊·⌋ y⦄
        using v w x y 4 VcompNml-in-Hom by simp
      ultimately show ⦃v ⌊·⌋ x⦄ ⋆ ⦃w ⌊·⌋ y⦄ = (⦃v⦄ ⋆ ⦃w⦄) · (⦃x⦄ ⋆ ⦃y⦄)
        by simp
    qed
  qed
  thus ?thesis using assms by blast
qed

lemma eval-red-Hcomp:
assumes Ide a and Ide b
shows ⦃(a ⋆ b)↓⦄ = ⦃⌊a⌋ ⇓ ⌊b⌋⦄ · (⦃a↓⦄ ⋆ ⦃b↓⦄)
proof −
  have Nml (a ⋆ b) ⟹ ?thesis
  proof −
    assume 1: Nml (a ⋆ b)
```

```
      hence 2: Nml a ∧ Nml b ∧ Src a = Trg b
        using Nml-HcompD(3−4,7) by simp
      have ⦃(a ⋆ b)↓⦄ = ⦃a⦄ ⋆ ⦃b⦄
        using 1 Nml-HcompD
        by (metis eval.simps(3) red-Nml)
      also have ... = ⦃⌊a⌋ ⇓ ⌊b⌋⦄ · (⦃a↓⦄ ⋆ ⦃b↓⦄)
        using assms 1 2 ide-eval-Ide Nmlize-in-Hom red2-Nml Nmlize-Nml
        by (simp add: eval-simps′)
      finally show ?thesis by simp
    qed
    moreover have ¬ Nml (a ⋆ b) ⟹ ?thesis
      using assms Can-red2 by (simp add: Can-red(1) iso-eval-Can)
    ultimately show ?thesis by blast
  qed


  lemma eval-red2-Nml-Prim₀:
  assumes Ide t and Nml t and Src t = ⟨a⟩₀
  shows ⦃t ⇓ ⟨a⟩₀⦄ = r[⦃t⦄]
    using assms 𝔯-ide-simp
    apply (cases t)
          apply simp-all
  proof −
    show C.obj a ⟹ t = ⟨a⟩₀ ⟹ 𝔩 (E a) = r[E a]
      using unitor-coincidence obj-eval-Obj [of t] 𝔩-ide-simp by auto
    show ⋀b c. Ide b ∧ Ide c ∧ Src b = Trg c ⟹ 𝔯 (⦃b⦄ ⋆ ⦃c⦄) = r[⦃b⦄ ⋆ ⦃c⦄]
      using 𝔯-ide-simp by (simp add: eval-simps′ ide-eval-Ide)
  qed

end
```

Most of the time when we interpret the *evaluation-map* locale, we are evaluating terms formed from the arrows in a bicategory as arrows of the bicategory itself. The following locale streamlines that use case.

```
locale self-evaluation-map =
  bicategory
begin

  sublocale bicategorical-language V src trg ..

  sublocale evaluation-map V src trg V H a i src trg ‹λμ. if arr μ then μ else null›
    using src.extensionality trg.extensionality
    by (unfold-locales, auto)

  notation eval (‹⦃-⦄›)
  notation Nmlize (‹⌊-⌋›)

end
```

1.6.5 Coherence

We define an individual term to be *coherent* if it commutes, up to evaluation, with the reductions of its domain and codomain. We then formulate the coherence theorem as the statement "every formal arrow is coherent". Because reductions evaluate to isomorphisms, this implies the standard version of coherence, which says that "parallel canonical terms have equal evaluations".

```
context evaluation-map
begin

  abbreviation coherent
  where coherent t ≡ ⦃Cod t↓⦄ · ⦃t⦄ = ⦃⌊t⌋⦄ · ⦃Dom t↓⦄

  lemma Nml-implies-coherent:
  assumes Nml t
  shows coherent t
    using assms Nml-implies-Arr Ide-Dom Ide-Cod Nml-Dom Nml-Cod Nmlize-Nml red-Nml
    by (metis Dom-Cod VcompNml-Cod-Nml arr-eval-Arr comp-arr-dom eval-VcompNml
        eval-simps(4))

  lemma canonical-factorization:
  assumes Arr t
  shows coherent t ⟷ ⦃t⦄ = inv ⦃Cod t↓⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄
  proof
    assume 1: coherent t
    have inv ⦃Cod t↓⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄ = inv ⦃Cod t↓⦄ · ⦃Cod t↓⦄ · ⦃t⦄
      using 1 by simp
    also have ... = (inv ⦃Cod t↓⦄ · ⦃Cod t↓⦄) · ⦃t⦄
      using comp-assoc by simp
    also have ... = ⦃t⦄
      using assms red-in-Hom Ide-Cod Can-red iso-eval-Can comp-cod-arr
      by (simp add: comp-inv-arr′ eval-simps′(4) eval-simps′(5))
    finally show ⦃t⦄ = inv ⦃Cod t↓⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄
      by presburger
    next
    assume 1: ⦃t⦄ = inv ⦃Cod t↓⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄
    hence ⦃Cod t↓⦄ · ⦃t⦄ = ⦃Cod t↓⦄ · inv ⦃Cod t↓⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄ by simp
    also have ... = (⦃Cod t↓⦄ · inv ⦃Cod t↓⦄) · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄
      using comp-assoc by simp
    also have ... = ⦃⌊t⌋⦄ · ⦃Dom t↓⦄
    proof −
      have ⦃Cod t↓⦄ · inv ⦃Cod t↓⦄ = cod ⦃⌊t⌋⦄
        using assms red-in-Hom Ide-Cod Can-red iso-eval-Can
              inv-is-inverse Nmlize-in-Hom comp-arr-inv
        by (simp add: eval-simps′)
      thus ?thesis
        using assms red-in-Hom Ide-Cod Can-red iso-eval-Can
              Ide-Dom Nmlize-in-Hom comp-cod-arr
        by (auto simp add: eval-simps′)
```

```
  qed
  finally show coherent t by blast
qed

lemma coherent-iff-coherent-Inv:
assumes Can t
shows coherent t ⟷ coherent (Inv t)
proof
  have 1: ⋀t. Can t ⟹ coherent t ⟹ coherent (Inv t)
  proof −
    fix t
    assume Can t
    hence t: Can t ∧ Arr t ∧ Ide (Dom t) ∧ Ide (Cod t) ∧
             arr ⦃t⦄ ∧ iso ⦃t⦄ ∧ inverse-arrows ⦃t⦄ (inv ⦃t⦄) ∧
             Can ⌊t⌋ ∧ Arr ⌊t⌋ ∧ arr ⦃⌊t⌋⦄ ∧ iso ⦃⌊t⌋⦄ ∧ ⌊t⌋ ∈ VHom ⌊Dom t⌋ ⌊Cod t⌋ ∧
             inverse-arrows ⦃⌊t⌋⦄ (inv ⦃⌊t⌋⦄) ∧ Inv t ∈ VHom (Cod t) (Dom t)
      using assms Can-implies-Arr Ide-Dom Ide-Cod iso-eval-Can inv-is-inverse
            Nmlize-in-Hom Can-Nmlize-Can Inv-in-Hom
      by simp
    assume coh: coherent t
    have ⦃Cod (Inv t)↓⦄ · ⦃Inv t⦄ = (inv ⦃⌊t⌋⦄ · ⦃⌊t⌋⦄) · ⦃Cod (Inv t)↓⦄ · ⦃Inv t⦄
      using t comp-inv-arr red-in-Hom
            comp-cod-arr [of ⦃Cod (Inv t)↓⦄ · ⦃Inv t⦄ inv ⦃⌊t⌋⦄ · ⦃⌊t⌋⦄]
      by (auto simp add: eval-simps′)
    also have ... = inv ⦃⌊t⌋⦄ · ⦃⌊t⌋⦄ · ⦃Dom t↓⦄ · inv ⦃t⦄
      using t eval-Inv-Can comp-assoc by auto
    also have ... = inv ⦃⌊t⌋⦄ · (⦃⌊t⌋⦄ · ⦃Dom t↓⦄) · inv ⦃t⦄
      using comp-assoc by simp
    also have ... = inv ⦃⌊t⌋⦄ · (⦃Cod t↓⦄ · ⦃t⦄) · inv ⦃t⦄
      using t coh by simp
    also have ... = inv ⦃⌊t⌋⦄ · ⦃Cod t↓⦄ · ⦃t⦄ · inv ⦃t⦄
      using comp-assoc by simp
    also have ... = ⦃⌊Inv t⌋⦄ · ⦃Dom (Inv t)↓⦄
    proof −
      have ⦃Cod t↓⦄ · ⦃t⦄ · inv ⦃t⦄ = ⦃Dom (Inv t)↓⦄
        using t eval-Inv-Can red-in-Hom comp-arr-inv comp-arr-dom
        by (simp add: eval-simps′)
      thus ?thesis
        using t Nmlize-Inv eval-Inv-Can by simp
    qed
    finally show coherent (Inv t) by blast
  qed
  show coherent t ⟹ coherent (Inv t) using assms 1 by simp
  show coherent (Inv t) ⟹ coherent t
  proof −
    assume coherent (Inv t)
    hence coherent (Inv (Inv t))
      using assms 1 Can-Inv by blast
    thus ?thesis using assms by simp
```

qed
qed

The next two facts are trivially proved by the simplifier, so formal named facts are not really necessary, but we include them for logical completeness of the following development, which proves coherence by structural induction.

lemma *coherent-Prim*$_0$:
assumes $C.obj\ a$
shows $coherent\ \langle a\rangle_0$
by *simp*

lemma *coherent-Prim*:
assumes $Arr\ \langle f\rangle$
shows $coherent\ \langle f\rangle$
using *assms* **by** *simp*

lemma *coherent-Lunit-Ide*:
assumes $Ide\ t$
shows $coherent\ \mathbf{l}[t]$
proof −
have t: $Ide\ t \wedge Arr\ t \wedge Dom\ t = t \wedge Cod\ t = t \wedge$
$ide\ \{\!|t|\!\} \wedge ide\ \{\!|\lfloor t\rfloor|\!\} \wedge \{\!|t{\downarrow}|\!\} \in hom\ \{\!|t|\!\}\ \{\!|\lfloor t\rfloor|\!\}$
using *assms Ide-in-Hom Ide-Nmlize-Ide*
red-in-Hom eval-in-hom ide-eval-Ide
by *force*
have *1*: $Obj\ (Trg\ t)$ **using** t **by** *auto*
have $\{\!|Cod\ \mathbf{l}[t]{\downarrow}|\!\} \cdot \{\!|\mathbf{l}[t]|\!\} = \mathrm{l}[\{\!|\lfloor t\rfloor|\!\}] \cdot (\{\!|Trg\ t|\!\} \star \{\!|t{\downarrow}|\!\})$
using t ‹*-ide-simp lunit-naturality* [*of* $\{\!|t{\downarrow}|\!\}$] *red-in-Hom*
by (*simp add*: *eval-simps′*)
also have ... $= \{\!|\lfloor t\rfloor|\!\} \cdot \mathrm{l}[\{\!|\lfloor t\rfloor|\!\}] \cdot (\{\!|Trg\ t|\!\} \star \{\!|t{\downarrow}|\!\})$
using t *1 lunit-in-hom Nmlize-in-Hom ide-eval-Ide red-in-Hom comp-cod-arr*
Obj-implies-Ide
by (*auto simp add*: *eval-simps′*)
also have ... $= \{\!|\lfloor \mathbf{l}[t]\rfloor|\!\} \cdot \{\!|Dom\ \mathbf{l}[t]{\downarrow}|\!\}$
using *1* t *Nml-Trg* ‹*-ide-simp* **by** (*cases* $Trg\ t$; *simp*)
finally show *?thesis* **by** *simp*
qed

Unlike many of the other results, the next one was not quite so straightforward to adapt from `MonoidalCategory`.

lemma *coherent-Runit-Ide*:
assumes $Ide\ t$
shows $coherent\ \mathbf{r}[t]$
proof −
have t: $Ide\ t \wedge Arr\ t \wedge Dom\ t = t \wedge Cod\ t = t \wedge$
$ide\ \{\!|t|\!\} \wedge ide\ \{\!|\lfloor t\rfloor|\!\} \wedge \{\!|t{\downarrow}|\!\} \in hom\ \{\!|t|\!\}\ \{\!|\lfloor t\rfloor|\!\}$
using *assms Ide-in-Hom Ide-Nmlize-Ide*
red-in-Hom eval-in-hom ide-eval-Ide
by *force*

```
have ⦃Cod r[t]↓⦄ · ⦃r[t]⦄ = r[⦃⌊t⌋⦄] · (⦃t↓⦄ ⋆ ⦃Src t⦄)
  using t ɾ-ide-simp red-in-Hom runit-naturality [of ⦃t↓⦄]
  by (simp add: eval-simps′)
also have ... = ⦃⌊r[t]⌋⦄ · ⦃Dom r[t]↓⦄
proof −
  have ⦃⌊r[t]⌋⦄ · ⦃Dom r[t]↓⦄ = ⦃⌊t⌋⦄ · ⦃⌊t⌋ ⇓ ⌊Src t⌋⦄ · (⦃t↓⦄ ⋆ ⦃Src t↓⦄)
    using t by (cases t; simp; cases Src t; simp)
  also have ... = (⦃⌊t⌋⦄ · ⦃⌊t⌋ ⇓ ⌊Src t⌋⦄) · (⦃t↓⦄ ⋆ ⦃Src t↓⦄)
  proof −
    have ⦃⌊t⌋⦄ ∈ hom ⦃⌊t⌋⦄ ⦃⌊t⌋⦄
      using t Nmlize-in-Hom by auto
    moreover have ⦃⌊t⌋ ⇓ ⌊Src t⌋⦄ ∈ hom (⦃⌊t⌋⦄ ⋆ ⦃Src t⦄) ⦃⌊t⌋⦄
    proof −
      have ⦃⌊t⌋ ⇓ ⌊Src t⌋⦄ ∈ hom ⦃⌊t⌋ ⋆ ⌊Src t⌋⦄ ⦃⌊t⌋⦄
      proof −
        have Src ⌊t⌋ = Trg ⌊Src t⌋ ∧ ⌊⌊t⌋ ⋆ ⌊Src t⌋⌋ = ⌊t⌋
          using t Nmlize-Src Nml-Nmlize HcompNml-Nml-Src [of ⌊t⌋]
          by simp
        thus ?thesis
          using t Ide-Nmlize-Ide Nml-Nmlize Obj-Src red2-in-Hom(2) Obj-implies-Ide
          by (auto simp add: eval-simps′)
      qed
      thus ?thesis using t Nmlize-in-Hom Nmlize-Src by simp
    qed
    moreover have ⦃t↓⦄ ⋆ ⦃Src t↓⦄ ∈ hom (⦃t⦄ ⋆ ⦃Src t⦄) (⦃⌊t⌋⦄ ⋆ ⦃Src t⦄)
      using t red-in-Hom red-Src Obj-Src eval-simps′ Obj-implies-Ide by auto
    ultimately show ?thesis using comp-assoc by fastforce
  qed
  also have ... = r[⦃⌊t⌋⦄] · (⦃t↓⦄ ⋆ ⦃Src t↓⦄)
  proof −
    have ⦃⌊t⌋ ⇓ ⌊Src t⌋⦄ = r[⦃⌊t⌋⦄]
    proof −
      have Nml ⌊t⌋ using t Nml-Nmlize by blast
      moreover have is-Prim0 ⌊Src t⌋
        using t is-Prim0-Src Nmlize-Src by presburger
      ultimately show ?thesis
        apply (cases ⌊t⌋; simp; cases ⌊Src t⌋; simp)
        using t unitor-coincidence ɩ-ide-simp ɾ-ide-simp Nmlize-in-Hom
          apply simp-all
        using t is-Prim0-Src
        apply (cases ⌊t⌋; simp)
        using t Nmlize-Src unitor-coincidence preserves-obj by simp
    qed
    moreover have r[⦃⌊t⌋⦄] ∈ hom (⦃⌊t⌋⦄ ⋆ ⦃Src t⦄) ⦃⌊t⌋⦄
      using t Nmlize-in-Hom by (auto simp add: eval-simps′(2))
    ultimately show ?thesis
      using comp-cod-arr [of r[⦃⌊t⌋⦄]] by fastforce
  qed
  also have ... = r[⦃⌊t⌋⦄] · (⦃t↓⦄ ⋆ ⦃Src t⦄)
```

using *t red-Src* **by** *auto*
finally show *?thesis* **by** *argo*
qed
finally show *?thesis* **by** *blast*
qed

lemma *coherent-Lunit′-Ide*:
assumes *Ide a*
shows *coherent* $\mathbf{l}^{-1}[a]$
using *assms Ide-implies-Can coherent-Lunit-Ide*
coherent-iff-coherent-Inv [*of Lunit a*]
by *simp*

lemma *coherent-Runit′-Ide*:
assumes *Ide a*
shows *coherent* $\mathbf{r}^{-1}[a]$
using *assms Ide-implies-Can coherent-Runit-Ide*
coherent-iff-coherent-Inv [*of Runit a*]
by *simp*

lemma *red2-Nml-Src*:
assumes *Ide t* **and** *Nml t*
shows $\{\!|t \Downarrow Src\ t|\!\} = \mathrm{r}[\{\!|t|\!\}]$
using *assms eval-red2-Nml-Prim*$_0$ *is-Prim0-Src* [*of t*]
by (*cases Src t*; *simp*)

lemma *red2-Trg-Nml*:
assumes *Ide t* **and** *Nml t*
shows $\{\!|Trg\ t \Downarrow t|\!\} = \mathrm{l}[\{\!|t|\!\}]$
using *assms is-Prim0-Trg* [*of t*] *ι-ide-simp ide-eval-Ide*
by (*cases Trg t*; *simp*)

lemma *coherence-key-fact*:
assumes $Ide\ a \wedge Nml\ a$ **and** $Ide\ b \wedge Nml\ b$ **and** $Ide\ c \wedge Nml\ c$
and $Src\ a = Trg\ b$ **and** $Src\ b = Trg\ c$
shows $\{\!|(a \lfloor\star\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|a \Downarrow b|\!\} \star \{\!|c|\!\}) =$
$(\{\!|a \Downarrow (b \lfloor\star\rfloor\ c)|\!\} \cdot (\{\!|a|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
have $is\text{-}Prim_0\ b \Longrightarrow$ *?thesis*
proof −
assume *b*: $is\text{-}Prim_0\ b$
have $\{\!|a \Downarrow c|\!\} \cdot (\mathrm{r}[\{\!|a|\!\}] \star \{\!|c|\!\}) = (\{\!|a \Downarrow c|\!\} \cdot (\{\!|a|\!\} \star \mathrm{l}[\{\!|c|\!\}])) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|Trg\ c|\!\}, \{\!|c|\!\}]$
proof −
have $Src\ b = Trg\ b$
using *b* **by** (*cases b*; *simp*)
thus *?thesis*
using *assms triangle* [*of* $\{\!|c|\!\}$ $\{\!|a|\!\}$] *ide-eval-Ide comp-assoc*
by (*simp add*: *eval-simps′*)
qed

thus *?thesis*
using *assms b HcompNml-Nml-Src* [*of a*] *HcompNml-Trg-Nml red2-Nml-Src* [*of a*] *red2-Trg-Nml*
by (*cases b*, *simp-all*)
qed
moreover have $[\![\neg$ *is-Prim*$_0$ *b*; *is-Prim*$_0$ *c* $]\!] \Longrightarrow$ *?thesis*
proof −
assume *b*: $\neg$ *is-Prim*$_0$ *b*
assume *c*: *is-Prim*$_0$ *c*
have $\{\!|(a \lfloor\star\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|a \Downarrow b|\!\} \star \{\!|c|\!\}) = \{\!|(a \lfloor\star\rfloor\ b) \Downarrow Src\ b|\!\} \cdot (\{\!|a \Downarrow b|\!\} \star src\ \{\!|b|\!\})$
using *assms b c* **by** (*cases c*, *simp-all add*: *eval-simps′*)
also have ... $= \mathrm{r}[\{\!|a \lfloor\star\rfloor\ b|\!\}] \cdot (\{\!|a \Downarrow b|\!\} \star src\ \{\!|b|\!\})$
using *assms red2-Nml-Src* [*of a* $\lfloor\star\rfloor$ *b*] *Nml-HcompNml*(*1*) *Src-HcompNml Ide-HcompNml*
by *simp*
also have ... $= \{\!|a \Downarrow b|\!\} \cdot \mathrm{r}[\{\!|a|\!\} \star \{\!|b|\!\}]$
proof −
have «$\{\!|a \Downarrow b|\!\} : \{\!|a|\!\} \star \{\!|b|\!\} \Rightarrow \{\!|a \lfloor\star\rfloor\ b|\!\}$»
using *assms red2-in-Hom eval-in-hom* [*of a* $\Downarrow$ *b*] **by** *simp*
thus *?thesis*
using *assms runit-naturality* [*of* $\{\!|a \Downarrow b|\!\}$] *arr-dom in-homE src-dom src-hcomp hcomp-simps*(*1*)
by (*elim in-homE*, *metis*)
qed
also have ... $= (\{\!|a \Downarrow (b \lfloor\star\rfloor\ c)|\!\} \cdot (\{\!|a|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
have $(\{\!|a \Downarrow (b \lfloor\star\rfloor\ c)|\!\} \cdot (\{\!|a|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}] =$
$(\{\!|a \Downarrow b|\!\} \cdot (\{\!|a|\!\} \star \mathrm{r}[\{\!|b|\!\}])) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, src\ \{\!|b|\!\}]$
using *assms c red2-Nml-Src* [*of b*]
by (*cases c*, *simp-all add*: *eval-simps′*)
thus *?thesis*
using *assms runit-hcomp ide-eval-Ide comp-assoc*
by (*simp add*: *eval-simps′*)
qed
finally show *?thesis* **by** *blast*
qed
moreover have $[\![\neg$ *is-Prim*$_0$ *b*; $\neg$ *is-Prim*$_0$ *c* $]\!] \Longrightarrow$ *?thesis*
proof −
assume *b′*: $\neg$ *is-Prim*$_0$ *b*
hence *b*: $Ide\ b \wedge Nml\ b \wedge Arr\ b \wedge \neg$ *is-Prim*$_0$ $b \wedge$
$ide\ \{\!|b|\!\} \wedge arr\ \{\!|b|\!\} \wedge \lfloor b \rfloor = b \wedge b\downarrow = b \wedge Dom\ b = b \wedge Cod\ b = b$
using *assms Ide-Nmlize-Ide Ide-in-Hom ide-eval-Ide* **by** *simp*
assume *c′*: $\neg$ *is-Prim*$_0$ *c*
hence *c*: $Ide\ c \wedge Nml\ c \wedge Arr\ c \wedge \neg$ *is-Prim*$_0$ $c \wedge$
$ide\ \{\!|c|\!\} \wedge arr\ \{\!|c|\!\} \wedge \lfloor c \rfloor = c \wedge c\downarrow = c \wedge Dom\ c = c \wedge Cod\ c = c$
using *assms Ide-Nmlize-Ide Ide-in-Hom ide-eval-Ide* **by** *simp*
have $\bigwedge a.\ Ide\ a \wedge Nml\ a \wedge Src\ a = Trg\ b \Longrightarrow$
$\{\!|(a \lfloor\star\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|a \Downarrow b|\!\} \star \{\!|c|\!\})$
$= (\{\!|a \Downarrow (b \lfloor\star\rfloor\ c)|\!\} \cdot (\{\!|a|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −

fix a :: $'c$ *term*
show *Ide a* ∧ *Nml a* ∧ *Src a* = *Trg b* ⟹
$\{\!|(a \lfloor\star\rfloor b) \Downarrow c|\!\} \cdot (\{\!|a \Downarrow b|\!\} \star \{\!|c|\!\})$
$= (\{\!|a \Downarrow (b \lfloor\star\rfloor c)|\!\} \cdot (\{\!|a|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
apply (*induct a*)
using *b c HcompNml-in-Hom*
apply (*simp-all add*: *HcompNml-Nml-Src HcompNml-Trg-Nml*)
proof −
fix *f*
assume *f*: *C.ide f* ∧ *C.arr f* ∧ $\langle src_C\ f\rangle_0$ = *Trg b*
show $\{\!|(\langle f\rangle \star b) \Downarrow c|\!\} \cdot (\{\!|\langle f\rangle \Downarrow b|\!\} \star \{\!|c|\!\}) =$
$(\{\!|\langle f\rangle \Downarrow b \lfloor\star\rfloor c|\!\} \cdot (E\,f \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
have $\{\!|(\langle f\rangle \star b) \Downarrow c|\!\} \cdot (\{\!|\langle f\rangle \Downarrow b|\!\} \star \{\!|c|\!\}) =$
$((E\,f \star \{\!|b \lfloor\star\rfloor c|\!\}) \cdot (E\,f \star \{\!|b \Downarrow c|\!\}) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]) \cdot$
$((E\,f \star \{\!|b|\!\}) \star \{\!|c|\!\})$
proof −
have $((E\,f \star \{\!|b \lfloor\star\rfloor c|\!\}) \cdot (E\,f \star \{\!|b \Downarrow c|\!\}) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]) \cdot$
$((E\,f \star \{\!|b|\!\}) \star \{\!|c|\!\}) =$
$((E\,f \star \{\!|b \lfloor\star\rfloor c|\!\}) \cdot (E\,f \star \{\!|b \Downarrow c|\!\}) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]) \cdot$
$(\{\!|\langle f\rangle \Downarrow b|\!\} \star \{\!|c|\!\})$
using *f b red2-Nml* **by** *simp*
also have ... = $(\{\!|\langle f\rangle \Downarrow b \lfloor\star\rfloor c|\!\} \cdot (E\,f \star \{\!|b \Downarrow c|\!\}) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]) \cdot$
$(\{\!|\langle f\rangle \Downarrow b|\!\} \star \{\!|c|\!\})$
proof −
have $\{\!|\langle f\rangle \Downarrow b \lfloor\star\rfloor c|\!\} = E\,f \star \{\!|b \lfloor\star\rfloor c|\!\}$
using *assms*(*5*) *b c is-Hcomp-HcompNml red2-Nml Nml-HcompNml*(*1*)
is-Hcomp-def
by (*metis eval.simps*(*2*) *eval.simps*(*3*) *red2.simps*(*4*))
thus *?thesis* **by** *argo*
qed
also have ... = $\{\!|(\langle f\rangle \star b) \Downarrow c|\!\} \cdot (\{\!|\langle f\rangle \Downarrow b|\!\} \star \{\!|c|\!\})$
using *b c* α*-def* **by** (*cases c*, *simp-all*)
finally show *?thesis* **by** *argo*
qed
also have ... = $((E\,f \star \{\!|b \lfloor\star\rfloor c|\!\}) \cdot (E\,f \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
have *src* $(E\,f)$ = *trg* $\{\!|b|\!\}$
using *b f preserves-src*
by (*cases Trg b*, *auto simp add*: *eval-simps′*)
thus *?thesis*
using *assms b c f comp-arr-dom comp-assoc*
by (*auto simp add*: *eval-simps′*)
qed
also have ... = $(\{\!|\langle f\rangle \Downarrow (b \lfloor\star\rfloor c)|\!\} \cdot (E\,f \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[E\,f, \{\!|b|\!\}, \{\!|c|\!\}]$
using *assms f b c Ide-HcompNml Nml-HcompNml is-Hcomp-HcompNml* [*of b c*]
α*-def*
by (*cases b* $\lfloor\star\rfloor$ *c*) *simp-all*
finally show *?thesis* **by** *blast*

qed
next
fix x
assume x: $C.obj\ x \wedge \langle x \rangle_0 = Trg\ b$
show $\{\!|b \Downarrow c|\!\} \cdot (\{\!|Trg\ b \Downarrow b|\!\} \star \{\!|c|\!\}) =$
$(\{\!|Trg\ b \Downarrow b \lfloor\star\rfloor\ c|\!\} \cdot (\{\!|Trg\ b|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|Trg\ b|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
have *1*: $Trg\ (b \lfloor\star\rfloor\ c) = Trg\ b$
using *assms b c Trg-HcompNml* **by** *blast*
have *2*: $\{\!|Trg\ b \Downarrow b \lfloor\star\rfloor\ c|\!\} = \mathrm{l}[\{\!|b \lfloor\star\rfloor\ c|\!\}]$
using *assms b c 1 Nml-HcompNml red2-Trg-Nml* [*of b* $\lfloor\star\rfloor$ *c*] *Ide-HcompNml*
by *simp*
have $\{\!|b \Downarrow c|\!\} \cdot (\{\!|Trg\ b \Downarrow b|\!\} \star \{\!|c|\!\}) = \{\!|b \Downarrow c|\!\} \cdot (\mathrm{l}[\{\!|b|\!\}] \star \{\!|c|\!\})$
using *b c 1 2 HcompNml-Trg-Nml red2-Trg-Nml Trg-HcompNml* **by** *simp*
also have ... $= \{\!|b \Downarrow c|\!\} \cdot \mathrm{l}[\{\!|b|\!\} \star \{\!|c|\!\}] \cdot \mathrm{a}[\{\!|Trg\ b|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *assms b c lunit-hcomp* [*of* $\{\!|b|\!\}$ $\{\!|c|\!\}$]
by (*metis* (*no-types*, *lifting*) *eval-simps′*(*3*) *eval-simps*(*2*))
also have ... $= (\{\!|b \Downarrow c|\!\} \cdot \mathrm{l}[\{\!|b|\!\} \star \{\!|c|\!\}]) \cdot \mathrm{a}[\{\!|Trg\ b|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *comp-assoc* **by** *simp*
also have ... $= (\mathrm{l}[\{\!|b \lfloor\star\rfloor\ c|\!\}] \cdot (\{\!|Trg\ b|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|Trg\ b|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *assms b c lunit-naturality* [*of* $\{\!|b \Downarrow c|\!\}$] *red2-in-Hom*
by (*simp add*: *eval-simps′*)
also have ... $= (\{\!|Trg\ b \Downarrow b \lfloor\star\rfloor\ c|\!\} \cdot (\{\!|Trg\ b|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|Trg\ b|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *b c 1 2 HcompNml-Trg-Nml red2-Trg-Nml Trg-HcompNml comp-assoc*
by *simp*
finally show *?thesis*
by *blast*
qed
next
fix $d\ e$
assume I: $Nml\ e \Longrightarrow \{\!|(e \lfloor\star\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|e \Downarrow b|\!\} \star \{\!|c|\!\})$
$= (\{\!|e \Downarrow b \lfloor\star\rfloor\ c|\!\} \cdot (\{\!|e|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|e|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
assume *de*: $Ide\ d \wedge Ide\ e \wedge Src\ d = Trg\ e \wedge Nml\ (d \star e) \wedge Src\ e = Trg\ b$
show $\{\!|((d \star e) \lfloor\star\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|(d \star e) \Downarrow b|\!\} \star \{\!|c|\!\})$
$= (\{\!|(d \star e) \Downarrow (b \lfloor\star\rfloor\ c)|\!\} \cdot ((\{\!|d|\!\} \star \{\!|e|\!\}) \star \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|d|\!\} \star \{\!|e|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
proof −
let $?f = un\text{-}Prim\ d$
have $is\text{-}Prim\ d$
using *de Nml-HcompD*
by (*metis term.disc*(*12*))
hence $d = \langle ?f \rangle \wedge C.ide\ ?f$
using *de* **by** (*cases d*; *simp*)
hence *d*: $Ide\ d \wedge Arr\ d \wedge Dom\ d = d \wedge Cod\ d = d \wedge Nml\ d \wedge$
$d = \langle ?f \rangle \wedge C.ide\ ?f \wedge ide\ \{\!|d|\!\} \wedge arr\ \{\!|d|\!\}$
using *de ide-eval-Ide Nml-Nmlize*(*1*) *Ide-in-Hom Nml-HcompD* [*of d e*]
by *simp*
have $Nml\ e \wedge \neg\ is\text{-}Prim_0\ e$
using *de Nml-HcompD* **by** *metis*
hence *e*: $Ide\ e \wedge Arr\ e \wedge Dom\ e = e \wedge Cod\ e = e \wedge Nml\ e \wedge$

$\neg$ *is-Prim*$_0$ *e* $\wedge$ *ide* $\{\!|e|\!\}$ $\wedge$ *arr* $\{\!|e|\!\}$
using *de Ide-in-Hom ide-eval-Ide* **by** *simp*
have *1*: *is-Hcomp* (*e* $\lfloor\star\rfloor$ *b*) $\wedge$ *is-Hcomp* (*b* $\lfloor\star\rfloor$ *c*) $\wedge$ *is-Hcomp* (*e* $\lfloor\star\rfloor$ *b* $\lfloor\star\rfloor$ *c*)
using *assms b c e de is-Hcomp-HcompNml* [*of e b*] *Nml-HcompNml*
is-Hcomp-HcompNml [*of b c*] *is-Hcomp-HcompNml* [*of e b* $\lfloor\star\rfloor$ *c*]
by *auto*
have *eb*: *Src e* = *Trg b*
using *assms b c e de* **by** *argo*
have *bc*: *Src b* = *Trg c*
using *assms b c* **by** *simp*
have *4*: *is-Hcomp* ((*e* $\lfloor\star\rfloor$ *b*) $\lfloor\star\rfloor$ *c*)
using *assms b c e eb de 1 is-Hcomp-HcompNml* [*of e b*]
is-Hcomp-HcompNml [*of e* $\lfloor\star\rfloor$ *b c*] *is-Hcomp-HcompNml* [*of e b*]
Nml-HcompNml(*1*) [*of e b*] *Src-HcompNml*
by *auto*
have $\{\!|((d \star e) \lfloor\star\rfloor b) \Downarrow c|\!\}$ · ($\{\!|(d \star e) \Downarrow b|\!\}$ $\star$ $\{\!|c|\!\}$)
= (($\{\!|d|\!\}$ $\star$ $\{\!|(e \lfloor\star\rfloor b) \lfloor\star\rfloor c|\!\}$) · ($\{\!|d|\!\}$ $\star$ $\{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}$) ·
a[$\{\!|d|\!\}$, $\{\!|e \lfloor\star\rfloor b|\!\}$, $\{\!|c|\!\}$]) ·
(($\{\!|d|\!\}$ $\star$ $\{\!|e \lfloor\star\rfloor b|\!\}$) · ($\{\!|d|\!\}$ $\star$ $\{\!|e \Downarrow b|\!\}$) · a[$\{\!|d|\!\}$, $\{\!|e|\!\}$, $\{\!|b|\!\}$] $\star$ $\{\!|c|\!\}$)
proof −
have $\{\!|((d \star e) \lfloor\star\rfloor b) \Downarrow c|\!\}$
= ($\{\!|d|\!\}$ $\star$ $\{\!|(e \lfloor\star\rfloor b) \lfloor\star\rfloor c|\!\}$) · ($\{\!|d|\!\}$ $\star$ $\{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}$) ·
a[$\{\!|d|\!\}$, $\{\!|e \lfloor\star\rfloor b|\!\}$, $\{\!|c|\!\}$]
proof −
have ((*d* $\star$ *e*) $\lfloor\star\rfloor$ *b*) $\Downarrow$ *c* = (*d* $\star$ (*e* $\lfloor\star\rfloor$ *b*)) $\Downarrow$ *c*
using *b c d e de 1 HcompNml-Nml Nml-HcompNml HcompNml-assoc*
HcompNml-Prim
by (*metis term.distinct-disc*(*4*))
also have ... = (*d* $\Downarrow$ ((*e* $\lfloor\star\rfloor$ *b*) $\lfloor\star\rfloor$ *c*)) · (*d* $\star$ ((*e* $\lfloor\star\rfloor$ *b*) $\Downarrow$ *c*)) ·
a[*d*, *e* $\lfloor\star\rfloor$ *b*, *c*]
using *b c d e de 1 Nml-HcompNml Nmlize-Nml*
by (*cases c*) *simp-all*
also have ... = (*d* $\star$ ((*e* $\lfloor\star\rfloor$ *b*) $\lfloor\star\rfloor$ *c*)) · (*d* $\star$ ((*e* $\lfloor\star\rfloor$ *b*) $\Downarrow$ *c*)) ·
a[*d*, *e* $\lfloor\star\rfloor$ *b*, *c*]
using *d 4*
apply (*cases d*, *simp-all*)
by (*cases* (*e* $\lfloor\star\rfloor$ *b*) $\lfloor\star\rfloor$ *c*) *simp-all*
finally show *?thesis*
using *b c d e HcompNml-in-Hom red2-in-Hom*
Nml-HcompNml Ide-HcompNml α*-def*
by *simp*
qed
moreover have $\{\!|(d \star e) \Downarrow b|\!\}$
= ($\{\!|d|\!\}$ $\star$ $\{\!|e \lfloor\star\rfloor b|\!\}$) · ($\{\!|d|\!\}$ $\star$ $\{\!|e \Downarrow b|\!\}$) · a[$\{\!|d|\!\}$, $\{\!|e|\!\}$, $\{\!|b|\!\}$]
proof −
have (*d* $\star$ *e*) $\Downarrow$ *b* = (*d* $\Downarrow$ (*e* $\lfloor\star\rfloor$ *b*)) · (*d* $\star$ (*e* $\Downarrow$ *b*)) · **a**[*d*, *e*, *b*]
using *b c d e de 1 HcompNml-Prim Nmlize-Nml*
by (*cases b*, *simp-all*)
also have ... = (*d* $\star$ (*e* $\lfloor\star\rfloor$ *b*)) · (*d* $\star$ (*e* $\Downarrow$ *b*)) · **a**[*d*, *e*, *b*]

using *b c d e de 1 HcompNml-Nml Nml-HcompNml*
apply (*cases d*, *simp-all*)
by (*cases e* $\lfloor\star\rfloor$ *b*, *simp-all*)
finally show *?thesis*
using *b d e HcompNml-in-Hom red2-in-Hom* α*-def* **by** *simp*
qed
ultimately show *?thesis* **by** *argo*
qed
also have ... $= ((\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot \mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}]) \cdot$
$((\{\!|d|\!\} \star \{\!|e \Downarrow b|\!\}) \star \{\!|c|\!\}) \cdot (\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\})$
proof −
have $(\{\!|d|\!\} \star \{\!|e \lfloor\star\rfloor b|\!\}) \cdot (\{\!|d|\!\} \star \{\!|e \Downarrow b|\!\}) \cdot \mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\}$
$= ((\{\!|d|\!\} \star \{\!|e \Downarrow b|\!\}) \star \{\!|c|\!\}) \cdot (\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\})$
using *assms b c d e de eb HcompNml-in-Hom red2-in-Hom comp-cod-arr*
Ide-HcompNml Nml-HcompNml comp-assoc
interchange [*of* $\{\!|d|\!\} \star \{\!|e \Downarrow b|\!\}$ $\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}]$ $\{\!|c|\!\}$ $\{\!|c|\!\}$]
by (*auto simp add*: *eval-simps′*)
moreover have $(\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \lfloor\star\rfloor c|\!\}) \cdot (\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}] =$
$(\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot \mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}]$
proof −
have $(\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \lfloor\star\rfloor c|\!\}) \cdot (\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}] =$
$((\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \lfloor\star\rfloor c|\!\}) \cdot (\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\})) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}]$
using *comp-assoc* **by** *simp*
thus *?thesis*
using *assms b c d e de eb HcompNml-in-Hom red2-in-Hom*
Ide-HcompNml Nml-HcompNml comp-cod-arr
by (*simp add*: *eval-simps′*)
qed
ultimately show *?thesis* **by** *argo*
qed
also have ... $= (\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot (\{\!|d|\!\} \star (\{\!|e \Downarrow b|\!\} \star \{\!|c|\!\})) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\} \star \{\!|b|\!\}, \{\!|c|\!\}] \cdot (\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\})$
using *assms b c d e de HcompNml-in-Hom red2-in-Hom*
Ide-HcompNml Nml-HcompNml ide-eval-Ide
assoc-naturality [*of* $\{\!|d|\!\}$ $\{\!|e \Downarrow b|\!\}$ $\{\!|c|\!\}$]
comp-permute [*of* $\mathrm{a}[\{\!|d|\!\}, \{\!|e \lfloor\star\rfloor b|\!\}, \{\!|c|\!\}]$ $(\{\!|d|\!\} \star \{\!|e \Downarrow b|\!\}) \star \{\!|c|\!\}$
$\{\!|d|\!\} \star (\{\!|e \Downarrow b|\!\} \star \{\!|c|\!\})$ $\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\} \star \{\!|b|\!\}, \{\!|c|\!\}]$]
comp-assoc
by (*simp add*: *eval-simps′*)
also have ... $= ((\{\!|d|\!\} \star \{\!|(e \lfloor\star\rfloor b) \Downarrow c|\!\}) \cdot (\{\!|d|\!\} \star (\{\!|e \Downarrow b|\!\} \star \{\!|c|\!\}))) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\} \star \{\!|b|\!\}, \{\!|c|\!\}] \cdot (\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\})$
using *comp-assoc* **by** *simp*
also have ... $= (((\{\!|d|\!\} \star \{\!|e \Downarrow (b \lfloor\star\rfloor c)|\!\}) \cdot (\{\!|d|\!\} \star \{\!|e|\!\} \star \{\!|b \Downarrow c|\!\})) \cdot$
$(\{\!|d|\!\} \star \mathrm{a}[\{\!|e|\!\}, \{\!|b|\!\}, \{\!|c|\!\}])) \cdot$
$\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\} \star \{\!|b|\!\}, \{\!|c|\!\}] \cdot (\mathrm{a}[\{\!|d|\!\}, \{\!|e|\!\}, \{\!|b|\!\}] \star \{\!|c|\!\})$
using *assms b c d e de eb I HcompNml-in-Hom red2-in-Hom*

Ide-HcompNml Nml-HcompNml whisker-left [*of* ⦃d⦄]
interchange [*of* ⦃d⦄ ⦃d⦄ ⦃(e ⌊⋆⌋ b) ⇓ c⦄ ⦃e ⇓ b⦄ ⋆ ⦃c⦄]
by (*auto simp add*: *eval-simps′*)
also have ... = ((⦃d⦄ ⋆ ⦃e ⇓ (b ⌊⋆⌋ c)⦄) · (⦃d⦄ ⋆ ⦃e⦄ ⋆ ⦃b ⇓ c⦄)) ·
((⦃d⦄ ⋆ a[⦃e⦄, ⦃b⦄, ⦃c⦄]) ·
a[⦃d⦄, ⦃e⦄ ⋆ ⦃b⦄, ⦃c⦄] · (a[⦃d⦄, ⦃e⦄, ⦃b⦄] ⋆ ⦃c⦄))
using *comp-assoc* **by** *simp*
also have ... = ((⦃d⦄ ⋆ ⦃e ⇓ (b ⌊⋆⌋ c)⦄) · (⦃d⦄ ⋆ (⦃e⦄ ⋆ ⦃b ⇓ c⦄))) ·
a[⦃d⦄, ⦃e⦄, ⦃b⦄ ⋆ ⦃c⦄] · a[⦃d⦄ ⋆ ⦃e⦄, ⦃b⦄, ⦃c⦄]
using *assms b c d e de pentagon*
by (*simp add*: *eval-simps′*)
also have ... = (⦃d⦄ ⋆ ⦃e ⇓ (b ⌊⋆⌋ c)⦄) ·
((⦃d⦄ ⋆ (⦃e⦄ ⋆ ⦃b ⇓ c⦄)) · a[⦃d⦄, ⦃e⦄, ⦃b⦄ ⋆ ⦃c⦄]) ·
a[⦃d⦄ ⋆ ⦃e⦄, ⦃b⦄, ⦃c⦄]
using *comp-assoc* **by** *simp*
also have ... = ((⦃d⦄ ⋆ ⦃e ⌊⋆⌋ b ⌊⋆⌋ c⦄) · (⦃d⦄ ⋆ ⦃e ⇓ b ⌊⋆⌋ c⦄)) ·
(a[⦃d⦄, ⦃e⦄, ⦃b ⌊⋆⌋ c⦄] · ((⦃d⦄ ⋆ ⦃e⦄) ⋆ ⦃b ⇓ c⦄)) ·
a[⦃d⦄ ⋆ ⦃e⦄, ⦃b⦄, ⦃c⦄]
using *assms d e de HcompNml-in-Hom red2-in-Hom Ide-HcompNml Nml-HcompNml*
assoc-naturality [*of* ⦃d⦄ ⦃e⦄ ⦃b ⇓ c⦄] *comp-cod-arr* [*of* ⦃d⦄ ⋆ ⦃e ⇓ b ⌊⋆⌋ c⦄]
by (*simp add*: *eval-simps′*)
also have ... = ((⦃d⦄ ⋆ ⦃e ⌊⋆⌋ b ⌊⋆⌋ c⦄) · (⦃d⦄ ⋆ ⦃e ⇓ b ⌊⋆⌋ c⦄) ·
a[⦃d⦄, ⦃e⦄, ⦃b ⌊⋆⌋ c⦄]) ·
((⦃d⦄ ⋆ ⦃e⦄) ⋆ ⦃b ⇓ c⦄) · a[⦃d⦄ ⋆ ⦃e⦄, ⦃b⦄, ⦃c⦄]
using *comp-assoc* **by** *simp*
also have ... = ⦃(d ⋆ e) ⇓ (b ⌊⋆⌋ c)⦄ · ((⦃d⦄ ⋆ ⦃e⦄) ⋆ ⦃b ⇓ c⦄) ·
a[⦃d⦄ ⋆ ⦃e⦄, ⦃b⦄, ⦃c⦄]
proof −
have ⦃(d ⋆ e) ⇓ (b ⌊⋆⌋ c)⦄
= (⦃d⦄ ⋆ ⦃e ⌊⋆⌋ (b ⌊⋆⌋ c)⦄) · (⦃d⦄ ⋆ ⦃e ⇓ (b ⌊⋆⌋ c)⦄) ·
a[⦃d⦄, ⦃e⦄, ⦃b ⌊⋆⌋ c⦄]
proof −
have (d ⋆ e) ⇓ (b ⌊⋆⌋ c)
= (d ⇓ (e ⌊⋆⌋ ⌊b ⌊⋆⌋ c⌋)) · (d ⋆ (e ⇓ (b ⌊⋆⌋ c))) · **a**[d, e, b ⌊⋆⌋ c]
using *e 1* **by** (*cases b* ⌊⋆⌋ *c*) *auto*
also have ... = (d ⇓ (e ⌊⋆⌋ (b ⌊⋆⌋ c))) · (d ⋆ (e ⇓ (b ⌊⋆⌋ c))) ·
a[d, e, b ⌊⋆⌋ c]
using *assms Nml-HcompNml Nmlize-Nml* **by** *simp*
also have ... = (d ⋆ (e ⌊⋆⌋ (b ⌊⋆⌋ c))) · (d ⋆ (e ⇓ (b ⌊⋆⌋ c))) ·
a[d, e, b ⌊⋆⌋ c]
using *d 1*
apply (*cases e* ⌊⋆⌋ *b* ⌊⋆⌋ *c*, *simp-all*)
by (*cases d*, *simp-all*)
finally show *?thesis*
using *α-def* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = (⦃(d ⋆ e) ⇓ (b ⌊⋆⌋ c)⦄ · ((⦃d⦄ ⋆ ⦃e⦄) ⋆ ⦃b ⇓ c⦄)) ·

$\mathrm{a}[\{\!|d|\!\} \star \{\!|e|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *comp-assoc* **by** *simp*
finally show *?thesis* **by** *auto*
qed
qed
qed
thus *?thesis* **using** *assms*(*1*,*4*) **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *coherent-Assoc-Ide*:
assumes *Ide a* **and** *Ide b* **and** *Ide c* **and** *Src a* = *Trg b* **and** *Src b* = *Trg c*
shows *coherent* $\mathbf{a}[a, b, c]$
proof −
have *a*: $Ide\ a \wedge Arr\ a \wedge Dom\ a = a \wedge Cod\ a = a \wedge$
$ide\ \{\!|a|\!\} \wedge ide\ \{\!|\lfloor a \rfloor|\!\} \wedge \{\!|a{\downarrow}|\!\} \in hom\ \{\!|a|\!\}\ \{\!|\lfloor a \rfloor|\!\}$
using *assms Ide-in-Hom Ide-Nmlize-Ide ide-eval-Ide red-in-Hom eval-in-hom*(*2*)
by *force*
have *b*: $Ide\ b \wedge Arr\ b \wedge Dom\ b = b \wedge Cod\ b = b \wedge$
$ide\ \{\!|b|\!\} \wedge ide\ \{\!|\lfloor b \rfloor|\!\} \wedge \{\!|b{\downarrow}|\!\} \in hom\ \{\!|b|\!\}\ \{\!|\lfloor b \rfloor|\!\}$
using *assms Ide-in-Hom Ide-Nmlize-Ide ide-eval-Ide red-in-Hom*(*2*)
eval-in-hom(*2*) [*of* $b{\downarrow}$]
by *auto*
have *c*: $Ide\ c \wedge Arr\ c \wedge Dom\ c = c \wedge Cod\ c = c \wedge$
$ide\ \{\!|c|\!\} \wedge ide\ \{\!|\lfloor c \rfloor|\!\} \wedge \{\!|c{\downarrow}|\!\} \in hom\ \{\!|c|\!\}\ \{\!|\lfloor c \rfloor|\!\}$
using *assms Ide-in-Hom Ide-Nmlize-Ide red-in-Hom eval-in-hom*(*2*) [*of* $c{\downarrow}$]
ide-eval-Ide
by *auto*
have $\{\!|Cod\ \mathbf{a}[a, b, c]{\downarrow}|\!\} \cdot \{\!|\mathbf{a}[a, b, c]|\!\}$
$= (\{\!|\lfloor a \rfloor \Downarrow (\lfloor b \rfloor\ \lfloor\star\rfloor\ \lfloor c \rfloor)|\!\} \cdot (\{\!|\lfloor a \rfloor|\!\} \star \{\!|\lfloor b \rfloor \Downarrow \lfloor c \rfloor|\!\}) \cdot (\{\!|a{\downarrow}|\!\} \star \{\!|b{\downarrow}|\!\} \star \{\!|c{\downarrow}|\!\})) \cdot$
$\mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
using *assms a b c red-in-Hom red2-in-Hom Nml-Nmlize Ide-Nmlize-Ide*
α*-def eval-red-Hcomp interchange* [*of* $\{\!|\lfloor a \rfloor|\!\}\ \{\!|a{\downarrow}|\!\}$] *comp-cod-arr* [*of* $\{\!|a{\downarrow}|\!\}$]
by (*simp add*: *eval-simps′*)
also have ... $= ((\{\!|\lfloor a \rfloor \Downarrow (\lfloor b \rfloor\ \lfloor\star\rfloor\ \lfloor c \rfloor)|\!\} \cdot (\{\!|\lfloor a \rfloor|\!\} \star \{\!|\lfloor b \rfloor \Downarrow \lfloor c \rfloor|\!\})) \cdot \mathrm{a}[\{\!|\lfloor a \rfloor|\!\}, \{\!|\lfloor b \rfloor|\!\}, \{\!|\lfloor c \rfloor|\!\}])$
.
$((\{\!|a{\downarrow}|\!\} \star \{\!|b{\downarrow}|\!\}) \star \{\!|c{\downarrow}|\!\})$
using *assms red-in-Hom Ide-HcompNml assoc-naturality* [*of* $\{\!|a{\downarrow}|\!\}\ \{\!|b{\downarrow}|\!\}\ \{\!|c{\downarrow}|\!\}$] *comp-assoc*
by (*simp add*: *eval-simps′*)
also have ... $= (\{\!|(\lfloor a \rfloor\ \lfloor\star\rfloor\ \lfloor b \rfloor) \Downarrow \lfloor c \rfloor|\!\} \cdot (\{\!|\lfloor a \rfloor \Downarrow \lfloor b \rfloor|\!\} \star \{\!|\lfloor c \rfloor|\!\})) \cdot ((\{\!|a{\downarrow}|\!\} \star \{\!|b{\downarrow}|\!\}) \star \{\!|c{\downarrow}|\!\})$
using *assms Nml-Nmlize Ide-Nmlize-Ide coherence-key-fact* **by** *simp*
also have ... $= \{\!|\lfloor \mathbf{a}[a, b, c] \rfloor|\!\} \cdot \{\!|Dom\ \mathbf{a}[a, b, c]{\downarrow}|\!\}$
using *assms a b c red-in-Hom red2-in-Hom Ide-Nmlize-Ide*
Nml-Nmlize eval-red-Hcomp HcompNml-assoc
interchange [*of* $\{\!|\lfloor a \rfloor \Downarrow \lfloor b \rfloor|\!\}\ \{\!|a{\downarrow}|\!\} \star \{\!|b{\downarrow}|\!\}\ \{\!|\lfloor c \rfloor|\!\}\ \{\!|c{\downarrow}|\!\}$]
comp-cod-arr [*of* $\{\!|c{\downarrow}|\!\}\ \{\!|\lfloor c \rfloor|\!\}$]
apply (*simp add*: *eval-simps′*)
proof −

have *seq* $\{\!|(\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor) \Downarrow \lfloor c\rfloor|\!\}\ ((\{\!|\lfloor a\rfloor \Downarrow \lfloor b\rfloor|\!\} \star \{\!|\lfloor c\rfloor|\!\}) \cdot ((\{\!|a\downarrow|\!\} \star \{\!|b\downarrow|\!\}) \star \{\!|c\downarrow|\!\}))$
using *assms c red-in-Hom red2-in-Hom Nml-HcompNml Ide-Nmlize-Ide Ide-HcompNml Nml-Nmlize*
by (*simp-all add*: *eval-simps′*)
moreover have
$cod\ (\{\!|(\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor) \Downarrow \lfloor c\rfloor|\!\} \cdot (\{\!|\lfloor a\rfloor \Downarrow \lfloor b\rfloor|\!\} \star \{\!|\lfloor c\rfloor|\!\}) \cdot ((\{\!|a\downarrow|\!\} \star \{\!|b\downarrow|\!\}) \star \{\!|c\downarrow|\!\})) =$
$\{\!|\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor\ \lfloor\star\rfloor\ \lfloor c\rfloor|\!\}$
using *assms c red-in-Hom red2-in-Hom Nml-HcompNml Ide-Nmlize-Ide Ide-HcompNml Nml-Nmlize HcompNml-assoc*
by (*simp add*: *eval-simps′*)
ultimately
show $(\{\!|(\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor) \Downarrow \lfloor c\rfloor|\!\} \cdot (\{\!|\lfloor a\rfloor \Downarrow \lfloor b\rfloor|\!\} \star \{\!|\lfloor c\rfloor|\!\})) \cdot ((\{\!|a\downarrow|\!\} \star \{\!|b\downarrow|\!\}) \star \{\!|c\downarrow|\!\}) =$
$\{\!|\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor\ \lfloor\star\rfloor\ \lfloor c\rfloor|\!\} \cdot$
$\{\!|(\lfloor a\rfloor\ \lfloor\star\rfloor\ \lfloor b\rfloor) \Downarrow \lfloor c\rfloor|\!\} \cdot (\{\!|\lfloor a\rfloor \Downarrow \lfloor b\rfloor|\!\} \star \{\!|\lfloor c\rfloor|\!\}) \cdot ((\{\!|a\downarrow|\!\} \star \{\!|b\downarrow|\!\}) \star \{\!|c\downarrow|\!\})$
using *comp-cod-arr comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed

lemma *coherent-Assoc′-Ide*:
assumes *Ide a* **and** *Ide b* **and** *Ide c* **and** *Src a* = *Trg b* **and** *Src b* = *Trg c*
shows *coherent* $\mathbf{a}^{-1}[a,\ b,\ c]$
using *assms Ide-implies-Can coherent-Assoc-Ide Inv-Ide coherent-iff-coherent-Inv Can.simps(10) Inv.simps(10)*
by *presburger*

lemma *eval-red2-naturality*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows $\{\!|Cod\ t \Downarrow Cod\ u|\!\} \cdot (\{\!|t|\!\} \star \{\!|u|\!\}) = \{\!|t\ \lfloor\star\rfloor\ u|\!\} \cdot \{\!|Dom\ t \Downarrow Dom\ u|\!\}$
proof −
have $*$: $\bigwedge t\ u.\ Nml\ (t \star u) \Longrightarrow arr\ \{\!|t|\!\} \wedge arr\ \{\!|u|\!\}$
using *Nml-implies-Arr Nml-HcompD*
by (*metis eval-simps′(1)*)
have $is\text{-}Prim_0\ t \Longrightarrow$ *?thesis*
using *assms Nml-implies-Arr is-Prim0-Trg* ‹.*naturality* [*of* $\{\!|u|\!\}$]
by (*cases t*) (*simp-all add*: *eval-simps′*, *cases Trg t*, *simp-all*)
moreover have $\neg\ is\text{-}Prim_0\ t \wedge is\text{-}Prim_0\ u \Longrightarrow$ *?thesis*
using *assms Nml-implies-Arr eval-red2-Nml-Prim*$_0$ *runit-naturality* [*of* $\{\!|t|\!\}$]
by (*cases u*) (*simp-all add*: *eval-simps′*)
moreover have $\neg\ is\text{-}Prim_0\ t \wedge \neg\ is\text{-}Prim_0\ u \Longrightarrow$ *?thesis*
using *assms* $*$ *Nml-implies-Arr*
apply (*induct t*, *simp-all*)
proof −
fix *f*
assume *f*: *C.arr f*
assume $\neg\ is\text{-}Prim_0\ u$
hence *u*: $\neg\ is\text{-}Prim_0\ u \wedge$
$Nml\ u \wedge Nml\ (Dom\ u) \wedge Nml\ (Cod\ u) \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge$
$arr\ \{\!|u|\!\} \wedge arr\ \{\!|Dom\ u|\!\} \wedge arr\ \{\!|Cod\ u|\!\} \wedge ide\ \{\!|Dom\ u|\!\} \wedge ide\ \{\!|Cod\ u|\!\}$

using *assms*(*2*) *Nml-implies-Arr ide-eval-Ide* **by** *simp*
hence *1*: $\neg$ *is-Prim*$_0$ (*Dom u*) $\wedge$ $\neg$ *is-Prim*$_0$ (*Cod u*)
using *u* **by** (*cases u*) *simp-all*
assume $\langle src_C\ f\rangle_0 = Trg\ u$
hence $\{\!|\langle src_C\ f\rangle_0|\!\} = \{\!|Trg\ u|\!\}$ **by** *simp*
hence *fu*: *src* (*E f*) = *trg* $\{\!|u|\!\}$
using *f u preserves-src Nml-implies-Arr*
by (*simp add*: *eval-simps′*)
show $\{\!|\langle C.cod\ f\rangle \Downarrow Cod\ u|\!\} \cdot (E\ f \star \{\!|u|\!\}) = (E\ f \star \{\!|u|\!\}) \cdot \{\!|\langle C.dom\ f\rangle \Downarrow Dom\ u|\!\}$
proof −
have $\{\!|\langle C.cod\ f\rangle \Downarrow Cod\ u|\!\} = E\ (C.cod\ f) \star cod\ \{\!|u|\!\}$
using *f u 1 Nml-implies-Arr*
by (*cases Cod u*, *simp-all add*: *eval-simps′*)
moreover have $\{\!|\langle C.dom\ f\rangle \Downarrow Dom\ u|\!\} = E\ (C.dom\ f) \star dom\ \{\!|u|\!\}$
using *f u 1 Nml-implies-Arr*
by (*cases Dom u*, *simp-all add*: *eval-simps′*)
moreover have «$E\ f \star \{\!|u|\!\} : E\ (C.dom\ f) \star \{\!|Dom\ u|\!\} \Rightarrow E\ (C.cod\ f) \star \{\!|Cod\ u|\!\}$»
using *f u fu Nml-implies-Arr*
apply (*intro hcomp-in-vhom*)
apply *auto*
by (*metis C.src-dom eval-simps*(*4*) *preserves-src trg-dom u*)
ultimately show *?thesis*
using *f u comp-arr-dom comp-cod-arr*
by (*simp add*: *fu*)
qed
next
fix *v w*
assume *I2*: $[\![\ \neg$ *is-Prim*$_0$ *w*; *Nml w* $]\!] \Longrightarrow$
$\{\!|Cod\ w \Downarrow Cod\ u|\!\} \cdot (\{\!|w|\!\} \star \{\!|u|\!\}) = \{\!|w\ \lfloor\star\rfloor\ u|\!\} \cdot \{\!|Dom\ w \Downarrow Dom\ u|\!\}$
assume $\neg$ *is-Prim*$_0$ *u*
hence *u*: $\neg$ *is-Prim*$_0$ *u* $\wedge$ *Arr u* $\wedge$ *Arr* (*Dom u*) $\wedge$ *Arr* (*Cod u*) $\wedge$
Nml u $\wedge$ *Nml* (*Dom u*) $\wedge$ *Nml* (*Cod u*) $\wedge$ *Ide* (*Dom u*) $\wedge$ *Ide* (*Cod u*) $\wedge$
arr $\{\!|u|\!\}$ $\wedge$ *arr* $\{\!|Dom\ u|\!\}$ $\wedge$ *arr* $\{\!|Cod\ u|\!\}$ $\wedge$ *ide* $\{\!|Dom\ u|\!\}$ $\wedge$ *ide* $\{\!|Cod\ u|\!\}$
using *assms*(*2*) *Nml-implies-Arr ide-eval-Ide* **by** *simp*
assume *vw*: *Nml* ($v \star w$)
assume *wu*: *Src w* = *Trg u*
let *?f* = *un-Prim v*
have $v = \langle ?f\rangle \wedge C.arr\ ?f$
using *vw* **by** (*metis Nml-HcompD*(*1*) *Nml-HcompD*(*2*))
hence *Arr v* $\wedge$ $v = \langle un\text{-}Prim\ v\rangle \wedge C.arr\ ?f \wedge Nml\ v$ **by** (*cases v*; *simp*)
hence *v*: $v = \langle ?f\rangle \wedge C.arr\ ?f \wedge Arr\ v \wedge Arr\ (Dom\ v) \wedge Arr\ (Cod\ v) \wedge Nml\ v \wedge$
Nml (*Dom v*) $\wedge$ *Nml* (*Cod v*) $\wedge$
arr $\{\!|v|\!\}$ $\wedge$ *arr* $\{\!|Dom\ v|\!\}$ $\wedge$ *arr* $\{\!|Cod\ v|\!\}$ $\wedge$ *ide* $\{\!|Dom\ v|\!\}$ $\wedge$ *ide* $\{\!|Cod\ v|\!\}$
using *vw* ∗ **by** (*cases v*, *simp-all*)
have *Nml w* $\wedge$ $\neg$ *is-Prim*$_0$ *w*
using *vw v* **by** (*metis Nml.simps*(*3*))
hence *w*: $\neg$ *is-Prim*$_0$ *w* $\wedge$ *Arr w* $\wedge$ *Arr* (*Dom w*) $\wedge$ *Arr* (*Cod w*) $\wedge$
Nml w $\wedge$ *Nml* (*Dom w*) $\wedge$ *Nml* (*Cod w*) $\wedge$
Ide (*Dom w*) $\wedge$ *Ide* (*Cod w*) $\wedge$

$arr\ \{\!|w|\!\} \wedge arr\ \{\!|Dom\ w|\!\} \wedge arr\ \{\!|Cod\ w|\!\} \wedge ide\ \{\!|Dom\ w|\!\} \wedge ide\ \{\!|Cod\ w|\!\}$
using *vw* ∗ *Nml-implies-Arr ide-eval-Ide* **by** *simp*
have u': $\neg$ *is-Prim*$_0$ (*Dom u*) $\wedge$ $\neg$ *is-Prim*$_0$ (*Cod u*)
using *u* **by** (*cases u*, *simp-all*)
have w': $\neg$ *is-Prim*$_0$ (*Dom w*) $\wedge$ $\neg$ *is-Prim*$_0$ (*Cod w*)
using *w* **by** (*cases w*, *simp-all*)
have *vw'*: *Src v* = *Trg w*
using *vw Nml-HcompD*(*7*) **by** *simp*
have *X*: *Nml* (*Dom v* ⋆ (*Dom w* ⌊⋆⌋ *Dom u*))
using *u u' v w w' wu vw is-Hcomp-HcompNml Nml-HcompNml Nml-Dom*
by (*metis Dom.simps*(*3*) *Nml.simps*(*3*) *term.distinct-disc*(*3*))
have *Y*: *Nml* (*Cod v* ⋆ (*Cod w* ⌊⋆⌋ *Cod u*))
using *u u' w w' wu vw is-Hcomp-HcompNml Nml-HcompNml Src-Cod Trg-Cod*
by (*metis Cod.simps*(*3*) *Nml.simps*(*3*) *Nml-Cod term.distinct-disc*(*3*) *v*)
show $\{\!|$(*Cod v* ⋆ *Cod w*) ⇓ *Cod u*$|\!\}$ · (($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
= $\{\!|$(*v* ⋆ *w*) ⌊⋆⌋ *u*$|\!\}$ · $\{\!|$(*Dom v* ⋆ *Dom w*) ⇓ *Dom u*$|\!\}$
proof −
have $\{\!|$(*Cod v* ⋆ *Cod w*) ⇓ *Cod u*$|\!\}$ · (($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
= ($\{\!|$*Cod v* ⇓ (*Cod w* ⌊⋆⌋ *Cod u*)$|\!\}$ · ($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) ·
a[$\{\!|$*Cod v*$|\!\}$, $\{\!|$*Cod w*$|\!\}$, $\{\!|$*Cod u*$|\!\}$]) · (($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
proof −
have (*Cod v* ⋆ *Cod w*) ⇓ *Cod u*
= (*Cod v* ⇓ (*Cod w* ⌊⋆⌋ ⌊*Cod u*⌋)) · (*Cod v* ⋆ *Cod w* ⇓ *Cod u*) ·
a[*Cod v*, *Cod w*, *Cod u*]
using *u v w* **by** (*cases u*) *simp-all*
hence $\{\!|$(*Cod v* ⋆ *Cod w*) ⇓ *Cod u*$|\!\}$
= $\{\!|$*Cod v* ⇓ (*Cod w* ⌊⋆⌋ *Cod u*)$|\!\}$ · ($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) ·
a[$\{\!|$*Cod v*$|\!\}$, $\{\!|$*Cod w*$|\!\}$, $\{\!|$*Cod u*$|\!\}$]
using *u v w* *α-def* **by** *simp*
thus *?thesis* **by** *presburger*
qed
also have ... = (($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⌊⋆⌋ *Cod u*$|\!\}$) · ($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) ·
a[$\{\!|$*Cod v*$|\!\}$, $\{\!|$*Cod w*$|\!\}$, $\{\!|$*Cod u*$|\!\}$]) · (($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
using *u v w Y red2-Nml* **by** *simp*
also have ... = (($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) · a[$\{\!|$*Cod v*$|\!\}$, $\{\!|$*Cod w*$|\!\}$, $\{\!|$*Cod u*$|\!\}$] ·
(($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
using *u v w vw' wu comp-cod-arr red2-in-Hom HcompNml-in-Hom comp-reduce*
by (*simp add*: *eval-simps'*)
also have ... = ($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) · a[$\{\!|$*Cod v*$|\!\}$, $\{\!|$*Cod w*$|\!\}$, $\{\!|$*Cod u*$|\!\}$] ·
(($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$) ⋆ $\{\!|u|\!\}$)
using *comp-assoc* **by** *simp*
also have ... = ($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) · ($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$ ⋆ $\{\!|u|\!\}$) ·
a[$\{\!|$*Dom v*$|\!\}$, $\{\!|$*Dom w*$|\!\}$, $\{\!|$*Dom u*$|\!\}$]
using *u v w vw' wu assoc-naturality* [*of* $\{\!|v|\!\}$ $\{\!|w|\!\}$ $\{\!|u|\!\}$]
by (*simp add*: *eval-simps'*)
also have ... = (($\{\!|$*Cod v*$|\!\}$ ⋆ $\{\!|$*Cod w* ⇓ *Cod u*$|\!\}$) · ($\{\!|v|\!\}$ ⋆ $\{\!|w|\!\}$ ⋆ $\{\!|u|\!\}$)) ·
a[$\{\!|$*Dom v*$|\!\}$, $\{\!|$*Dom w*$|\!\}$, $\{\!|$*Dom u*$|\!\}$]
using *comp-assoc* **by** *simp*
also have

... = (⦃v⦄ ⋆ ⦃w ⌊⋆⌋ u⦄ · ⦃Dom w ⇓ Dom u⦄) · a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *v w u vw′ wu I2 red2-in-Hom HcompNml-in-Hom comp-cod-arr*
interchange [*of* ⦃Cod v⦄ ⦃v⦄ ⦃Cod w ⇓ Cod u⦄ ⦃w⦄ ⋆ ⦃u⦄]
by (*simp add*: *eval-simps′*)
also have ... = ((⦃v⦄ ⋆ ⦃w ⌊⋆⌋ u⦄) · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄)) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *v w u vw′ wu red2-in-Hom HcompNml-in-Hom comp-arr-dom*
interchange [*of* ⦃v⦄ ⦃Dom v⦄ ⦃w ⌊⋆⌋ u⦄ ⦃Dom w ⇓ Dom u⦄]
by (*simp add*: *eval-simps′*)
also have ... = (⦃v⦄ ⋆ ⦃w ⌊⋆⌋ u⦄) · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *comp-assoc* **by** *simp*
also have ... = ⦃v ⌊⋆⌋ w ⌊⋆⌋ u⦄ · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *u u′ v w vw′ wu is-Hcomp-HcompNml HcompNml-Prim* [*of w* ⌊⋆⌋ *u ?f*]
by *force*
also have ... = ⦃v ⌊⋆⌋ w ⌊⋆⌋ u⦄ · ⦃Dom v ⌊⋆⌋ Dom w ⌊⋆⌋ Dom u⦄ ·
(⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) · a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
proof −
have ⦃v ⌊⋆⌋ w ⌊⋆⌋ u⦄ · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄] =
(⦃v ⌊⋆⌋ w ⌊⋆⌋ u⦄ · ⦃Dom v ⌊⋆⌋ Dom w ⌊⋆⌋ Dom u⦄) ·
(⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) · a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *u v w vw′ wu comp-arr-dom Nml-HcompNml HcompNml-in-Hom*
by (*simp add*: *eval-simps′*)
also have ... = ⦃v ⌊⋆⌋ w ⌊⋆⌋ u⦄ · ⦃Dom v ⌊⋆⌋ Dom w ⌊⋆⌋ Dom u⦄ ·
(⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) · a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *comp-assoc* **by** *simp*
finally show *?thesis* **by** *blast*
qed
also have ... = ⦃(v ⌊⋆⌋ w) ⌊⋆⌋ u⦄ · ⦃(Dom v ⋆ Dom w) ⇓ Dom u⦄
proof −
have (Dom v ⋆ Dom w) ⇓ Dom u
= (Dom v ⇓ (Dom w ⌊⋆⌋ ⌊Dom u⌋)) · (Dom v ⋆ (Dom w ⇓ Dom u)) ·
a[Dom v, Dom w, Dom u]
using *u u′ v w vw′ wu* **by** (*cases u*, *simp-all*)
hence
⦃(Dom v ⋆ Dom w) ⇓ Dom u⦄
= ⦃Dom v ⇓ (Dom w ⌊⋆⌋ Dom u)⦄ · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *u v w α-def* **by** *simp*
also have
... = ⦃Dom v ⌊⋆⌋ Dom w ⌊⋆⌋ Dom u⦄ · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]
using *X HcompNml-Nml red2-Nml* **by** *presburger*
finally have
⦃(Dom v ⋆ Dom w) ⇓ Dom u⦄
= ⦃Dom v ⌊⋆⌋ Dom w ⌊⋆⌋ Dom u⦄ · (⦃Dom v⦄ ⋆ ⦃Dom w ⇓ Dom u⦄) ·
a[⦃Dom v⦄, ⦃Dom w⦄, ⦃Dom u⦄]

by *blast*
thus *?thesis*
using *assms v w vw′ wu HcompNml-assoc* **by** *presburger*
qed
finally show *?thesis*
using *vw HcompNml-Nml* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *coherent-Hcomp*:
assumes $Arr\ t$ **and** $Arr\ u$ **and** $Src\ t = Trg\ u$ **and** $coherent\ t$ **and** $coherent\ u$
shows $coherent\ (t \star u)$
proof −
have t: $Arr\ t \wedge Ide\ (Dom\ t) \wedge Ide\ (Cod\ t) \wedge Ide\ \lfloor Dom\ t \rfloor \wedge Ide\ \lfloor Cod\ t \rfloor \wedge$
$arr\ \{\!|t|\!\} \wedge arr\ \{\!|Dom\ t|\!\} \wedge ide\ \{\!|Dom\ t|\!\} \wedge arr\ \{\!|Cod\ t|\!\} \wedge ide\ \{\!|Cod\ t|\!\}$
using *assms Ide-Nmlize-Ide ide-eval-Ide* **by** *auto*
have u: $Arr\ u \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge Ide\ \lfloor Dom\ u \rfloor \wedge Ide\ \lfloor Cod\ u \rfloor \wedge$
$arr\ \{\!|u|\!\} \wedge arr\ \{\!|Dom\ u|\!\} \wedge ide\ \{\!|Dom\ u|\!\} \wedge arr\ \{\!|Cod\ u|\!\} \wedge ide\ \{\!|Cod\ u|\!\}$
using *assms Ide-Nmlize-Ide ide-eval-Ide* **by** *auto*
have $\{\!|Cod\ (t \star u)\downarrow|\!\} \cdot (\{\!|t|\!\} \star \{\!|u|\!\})$
$= (\{\!|\lfloor Cod\ t \rfloor \Downarrow \lfloor Cod\ u \rfloor|\!\} \cdot (\{\!|Cod\ t\downarrow|\!\} \star \{\!|Cod\ u\downarrow|\!\})) \cdot (\{\!|t|\!\} \star \{\!|u|\!\})$
using *t u eval-red-Hcomp* **by** *simp*
also have ... $= \{\!|\lfloor Cod\ t \rfloor \Downarrow \lfloor Cod\ u \rfloor|\!\} \cdot (\{\!|Cod\ t\downarrow|\!\} \star \{\!|Cod\ u\downarrow|\!\}) \cdot (\{\!|t|\!\} \star \{\!|u|\!\})$
using *comp-assoc* **by** *simp*
also have ... $= \{\!|\lfloor Cod\ t \rfloor \Downarrow \lfloor Cod\ u \rfloor|\!\} \cdot (\{\!|\lfloor t \rfloor|\!\} \star \{\!|\lfloor u \rfloor|\!\}) \cdot (\{\!|Dom\ t\downarrow|\!\} \star \{\!|Dom\ u\downarrow|\!\})$
using *assms t u Nmlize-in-Hom red-in-Hom*
interchange [*of* $\{\!|Cod\ t\downarrow|\!\}$ $\{\!|t|\!\}$ $\{\!|Cod\ u\downarrow|\!\}$ $\{\!|u|\!\}$]
interchange [*of* $\{\!|\lfloor t \rfloor|\!\}$ $\{\!|Dom\ t\downarrow|\!\}$ $\{\!|\lfloor u \rfloor|\!\}$ $\{\!|Dom\ u\downarrow|\!\}$]
by (*simp add*: *eval-simps′*)
also have ... $= (\{\!|\lfloor Cod\ t \rfloor \Downarrow \lfloor Cod\ u \rfloor|\!\} \cdot (\{\!|\lfloor t \rfloor|\!\} \star \{\!|\lfloor u \rfloor|\!\})) \cdot (\{\!|Dom\ t\downarrow|\!\} \star \{\!|Dom\ u\downarrow|\!\})$
using *comp-assoc* **by** *simp*
also have ... $= (\{\!|\lfloor t \rfloor\ \lfloor \star \rfloor\ \lfloor u \rfloor|\!\} \cdot \{\!|\lfloor Dom\ t \rfloor \Downarrow \lfloor Dom\ u \rfloor|\!\}) \cdot (\{\!|Dom\ t\downarrow|\!\} \star \{\!|Dom\ u\downarrow|\!\})$
using *assms t u Nml-Nmlize Nmlize-in-Hom*
eval-red2-naturality [*of Nmlize t Nmlize u*]
by *simp*
also have ... $= \{\!|\lfloor t \rfloor\ \lfloor \star \rfloor\ \lfloor u \rfloor|\!\} \cdot \{\!|\lfloor Dom\ t \rfloor \Downarrow \lfloor Dom\ u \rfloor|\!\} \cdot (\{\!|Dom\ t\downarrow|\!\} \star \{\!|Dom\ u\downarrow|\!\})$
using *comp-assoc* **by** *simp*
also have ... $= \{\!|\lfloor t \rfloor\ \lfloor \star \rfloor\ \lfloor u \rfloor|\!\} \cdot \{\!|(Dom\ t \star Dom\ u)\downarrow|\!\}$
using *t u eval-red-Hcomp* **by** *simp*
finally have $\{\!|Cod\ (t \star u)\downarrow|\!\} \cdot (\{\!|t|\!\} \star \{\!|u|\!\}) = \{\!|\lfloor t \rfloor\ \lfloor \star \rfloor\ \lfloor u \rfloor|\!\} \cdot \{\!|(Dom\ t \star Dom\ u)\downarrow|\!\}$
by *blast*
thus *?thesis* **using** *t u* **by** *simp*
qed

lemma *coherent-Vcomp*:
assumes $Arr\ t$ **and** $Arr\ u$ **and** $Dom\ t = Cod\ u$
and $coherent\ t$ **and** $coherent\ u$

shows *coherent* $(t \cdot u)$
proof −
  have *t*: $Arr\ t \wedge Ide\ (Dom\ t) \wedge Ide\ (Cod\ t) \wedge Ide\ \lfloor Dom\ t \rfloor \wedge Ide\ \lfloor Cod\ t \rfloor \wedge$
    $arr\ \{\!|t|\!\} \wedge arr\ \{\!|Dom\ t|\!\} \wedge ide\ \{\!|Dom\ t|\!\} \wedge arr\ \{\!|Cod\ t|\!\} \wedge ide\ \{\!|Cod\ t|\!\}$
    using *assms Ide-Nmlize-Ide ide-eval-Ide* **by** *auto*
  have *u*: $Arr\ u \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge Ide\ \lfloor Dom\ u \rfloor \wedge Ide\ \lfloor Cod\ u \rfloor \wedge$
    $arr\ \{\!|u|\!\} \wedge arr\ \{\!|Dom\ u|\!\} \wedge ide\ \{\!|Dom\ u|\!\} \wedge arr\ \{\!|Cod\ u|\!\} \wedge ide\ \{\!|Cod\ u|\!\}$
    using *assms Ide-Nmlize-Ide ide-eval-Ide* **by** *auto*
  have $\{\!|Cod\ (t \cdot u)\downarrow|\!\} \cdot \{\!|t \cdot u|\!\} = \{\!|Cod\ t\downarrow|\!\} \cdot \{\!|t|\!\} \cdot \{\!|u|\!\}$
    using *t u* **by** *simp*
  also have ... $= (\{\!|Cod\ t\downarrow|\!\} \cdot \{\!|t|\!\}) \cdot \{\!|u|\!\}$
  proof −
    have *seq* $\{\!|Cod\ t\downarrow|\!\}\ \{\!|t|\!\}$
      using *assms t red-in-Hom*
      by (*intro seqI*, *auto simp add*: *eval-simps′*)
    moreover have *seq* $\{\!|t|\!\}\ \{\!|u|\!\}$
      using *assms t u* **by** (*auto simp add*: *eval-simps′*)
    ultimately show *?thesis*
      using *comp-assoc* **by** *auto*
  qed
  also have ... $= \{\!|\lfloor t \cdot u \rfloor|\!\} \cdot \{\!|Dom\ (t \cdot u)\downarrow|\!\}$
    using *t u assms red-in-Hom Nml-Nmlize comp-assoc*
    by (*simp add*: *eval-simps′ Nml-implies-Arr eval-VcompNml*)
  finally show *coherent* $(t \cdot u)$ **by** *blast*
qed

The main result: "Every formal arrow is coherent."

theorem *coherence*:
assumes *Arr t*
shows *coherent t*
proof −
  have $Arr\ t \Longrightarrow coherent\ t$
  proof (*induct t*)
    show $\bigwedge a.\ Arr\ \langle a \rangle_0 \Longrightarrow coherent\ \langle a \rangle_0$ **by** *simp*
    show $\bigwedge \mu.\ Arr\ \langle \mu \rangle \Longrightarrow coherent\ \langle \mu \rangle$ **by** *simp*
    fix *u v*
    show $[\![\ Arr\ u \Longrightarrow coherent\ u;\ Arr\ v \Longrightarrow coherent\ v\ ]\!] \Longrightarrow Arr\ (u \star v)$
      $\Longrightarrow coherent\ (u \star v)$
      using *coherent-Hcomp* **by** *simp*
    show $[\![\ Arr\ u \Longrightarrow coherent\ u;\ Arr\ v \Longrightarrow coherent\ v\ ]\!] \Longrightarrow Arr\ (u \cdot v)$
      $\Longrightarrow coherent\ (u \cdot v)$
      using *coherent-Vcomp* **by** *simp*
    next
    fix *t*
    assume *I*: $Arr\ t \Longrightarrow coherent\ t$
    show *Lunit*: $Arr\ \mathfrak{l}[t] \Longrightarrow coherent\ \mathfrak{l}[t]$
      using *I Ide-Dom coherent-Lunit-Ide Ide-in-Hom*
        coherent-Vcomp [*of t* $\mathfrak{l}[Dom\ t]$] *Nmlize-Vcomp-Arr-Dom*
        eval-in-hom $\mathfrak{l}$.*naturality1* [*of* $\{\!|t|\!\}$]

by *force*
show *Runit*: *Arr* $\mathbf{r}[t] \Longrightarrow$ *coherent* $\mathbf{r}[t]$
using *I Ide-Dom coherent-Runit-Ide Ide-in-Hom ide-eval-Ide*
coherent-Vcomp [*of t* $\mathbf{r}$[*Dom t*]] *Nmlize-Vcomp-Arr-Dom* $\mathfrak{r}$*-ide-simp*
eval-in-hom $\mathfrak{r}$*.naturality1* [*of* $\{\!|t|\!\}$]
by *force*
show *Arr* $\mathbf{l}^{-1}[t] \Longrightarrow$ *coherent* $\mathbf{l}^{-1}[t]$
proof −
assume *Arr* $\mathbf{l}^{-1}[t]$
hence *t*: *Arr t* **by** *simp*
have *coherent* ($\mathbf{l}^{-1}$[*Cod t*] · *t*)
using *t I Ide-Cod coherent-Lunit′-Ide Ide-in-Hom coherent-Vcomp* [*of* $\mathbf{l}^{-1}$[*Cod t*] *t*]
Arr.simps(*6*) *Dom.simps*(*6*) *Dom-Cod Ide-implies-Arr*
by *presburger*
moreover have $\{\!|\mathbf{l}^{-1}[t]|\!\} = \{\!|\mathbf{l}^{-1}[Cod\ t] \cdot t|\!\}$
using *t* $\mathfrak{l}'$*.naturality2* [*of* $\{\!|t|\!\}$]
by (*simp add*: *eval-simps*(*5*))
ultimately show *?thesis*
using *t Nmlize-Vcomp-Cod-Arr* **by** *simp*
qed
show *Arr* $\mathbf{r}^{-1}[t] \Longrightarrow$ *coherent* $\mathbf{r}^{-1}[t]$
proof −
assume *Arr* $\mathbf{r}^{-1}[t]$
hence *t*: *Arr t* **by** *simp*
have *coherent* ($\mathbf{r}^{-1}$[*Cod t*] · *t*)
using *t I Ide-Cod coherent-Runit′-Ide Ide-in-Hom coherent-Vcomp* [*of* $\mathbf{r}^{-1}$[*Cod t*] *t*]
Arr.simps(*8*) *Dom.simps*(*8*) *Dom-Cod Ide-implies-Arr*
by *presburger*
moreover have $\{\!|\mathbf{r}^{-1}[t]|\!\} = \{\!|\mathbf{r}^{-1}[Cod\ t] \cdot t|\!\}$
using *t* $\mathfrak{r}'$*.naturality2* [*of* $\{\!|t|\!\}$]
by (*simp add*: *eval-simps*(*5*))
ultimately show *?thesis*
using *t Nmlize-Vcomp-Cod-Arr* **by** *simp*
qed
next
fix *t u v*
assume *I1*: *Arr t* $\Longrightarrow$ *coherent t*
assume *I2*: *Arr u* $\Longrightarrow$ *coherent u*
assume *I3*: *Arr v* $\Longrightarrow$ *coherent v*
show *Arr* $\mathbf{a}[t, u, v] \Longrightarrow$ *coherent* $\mathbf{a}[t, u, v]$
proof −
assume *tuv*: *Arr* $\mathbf{a}[t, u, v]$
have *t*: *Arr t* **using** *tuv* **by** *simp*
have *u*: *Arr u* **using** *tuv* **by** *simp*
have *v*: *Arr v* **using** *tuv* **by** *simp*
have *tu*: *Src t* = *Trg u* **using** *tuv* **by** *simp*
have *uv*: *Src u* = *Trg v* **using** *tuv* **by** *simp*
have *coherent* (($t \star u \star v$) · $\mathbf{a}$[*Dom t*, *Dom u*, *Dom v*])
proof −

```
    have Arr (t ⋆ u ⋆ v) ∧ coherent (t ⋆ u ⋆ v)
      using t u v tu uv tuv I1 I2 I3 coherent-Hcomp Arr.simps(3) Trg.simps(3)
      by presburger
    moreover have Arr a[Dom t, Dom u, Dom v]
      using t u v tu uv Ide-Dom by simp
    moreover have coherent a[Dom t, Dom u, Dom v]
      using t u v tu uv Src-Dom Trg-Dom Ide-Dom coherent-Assoc-Ide by metis
    moreover have Dom (t ⋆ u ⋆ v) = Cod a[Dom t, Dom u, Dom v]
      using t u v by simp
    ultimately show ?thesis
      using t u v coherent-Vcomp by blast
  qed
  moreover have VPar a[t, u, v] ((t ⋆ u ⋆ v) · a[Dom t, Dom u, Dom v])
    using t u v tu uv Ide-Dom by simp
  moreover have ⌊a[t, u, v]⌋ = ⌊(t ⋆ u ⋆  v) · a[Dom t, Dom u, Dom v]⌋
  proof −
    have (⌊t⌋ ⌊⋆⌋ ⌊u⌋) ⌊⋆⌋ ⌊v⌋
          = (⌊t⌋ ⌊⋆⌋ ⌊u⌋ ⌊⋆⌋ ⌊v⌋) ⌊·⌋ ((⌊Dom t⌋ ⌊⋆⌋ ⌊Dom u⌋) ⌊⋆⌋ ⌊Dom v⌋)
    proof −
      have 1: Nml ⌊t⌋ ∧ Nml ⌊u⌋ ∧ Nml ⌊v⌋ ∧
              Dom ⌊t⌋ = ⌊Dom t⌋ ∧ Dom ⌊u⌋ = ⌊Dom u⌋ ∧ Dom ⌊v⌋ = ⌊Dom v⌋
        using t u v Nml-Nmlize by blast
      moreover have Nml (⌊t⌋ ⌊⋆⌋ ⌊u⌋)
        using 1 t u tu Nmlize-Src Nmlize-Trg Nml-HcompNml(1)
        by presburger
      moreover have ⋀t. Arr t ⟹ ⌊t⌋ ⌊·⌋ ⌊Dom t⌋ = ⌊t⌋
        using t Nmlize-Vcomp-Arr-Dom by simp
      moreover have Dom ⌊a[t, u, v]⌋ = ⌊Dom a[t, u, v]⌋
        using Nml-Nmlize tuv by blast
      ultimately show ?thesis
        using t u v tu uv tuv 1 HcompNml-assoc Nml-HcompNml
              Nml-Nmlize VcompNml-Nml-Dom [of (⌊t⌋ ⌊⋆⌋ ⌊u⌋) ⌊⋆⌋ ⌊v⌋]
        by auto
    qed
    thus ?thesis
      using t u v Nmlize-Vcomp-Arr-Dom VcompNml-HcompNml Nml-Nmlize
      by simp
  qed
  moreover have ⦃a[t, u, v]⦄ = ⦃(t ⋆ u ⋆ v) · a[Dom t, Dom u, Dom v]⦄
    using t u v tu uv Ide-Dom comp-cod-arr ide-eval-Ide α-def
    apply (simp add: eval-simps′)
    using assoc-naturality1 arr-eval-Arr eval-simps′(2−4) by presburger
  ultimately show coherent a[t, u, v] by argo
qed
show Arr a⁻¹[t, u, v] ⟹ coherent a⁻¹[t, u, v]
proof −
  assume tuv: Arr a⁻¹[t, u, v]
  have t: Arr t using tuv by simp
  have u: Arr u using tuv by simp
```

```
      have v: Arr v using tuv by simp
      have tu: Src t = Trg u using tuv by simp
      have uv: Src u = Trg v using tuv by simp
      have coherent (((t ⋆ u) ⋆ v) · a⁻¹[Dom t, Dom u, Dom v])
      proof −
        have Arr ((t ⋆ u) ⋆ v) ∧ coherent ((t ⋆ u) ⋆ v)
          using t u v tu uv tuv I1 I2 I3 coherent-Hcomp Arr.simps(3) Src.simps(3)
          by presburger
        moreover have Arr a⁻¹[Dom t, Dom u, Dom v]
          using t u v tu uv Ide-Dom by simp
        moreover have coherent a⁻¹[Dom t, Dom u, Dom v]
          using t u v tu uv Src-Dom Trg-Dom Ide-Dom coherent-Assoc′-Ide
          by metis
        moreover have Dom ((t ⋆ u) ⋆ v) = Cod a⁻¹[Dom t, Dom u, Dom v]
          using t u v by simp
        ultimately show ?thesis
          using t u v coherent-Vcomp by metis
      qed
      moreover have VPar a⁻¹[t, u, v] (((t ⋆ u) ⋆ v) · a⁻¹[Dom t, Dom u, Dom v])
        using t u v tu uv Ide-Dom by simp
      moreover have ⌊a⁻¹[t, u, v]⌋ = ⌊((t ⋆ u) ⋆ v) · a⁻¹[Dom t, Dom u, Dom v]⌋
        using t u v tu uv Nmlize-Vcomp-Arr-Dom VcompNml-HcompNml Nml-Nmlize
              HcompNml-assoc Nml-HcompNml HcompNml-in-Hom
              VcompNml-Nml-Dom [of (⌊t⌋ ⌊⋆⌋ ⌊u⌋) ⌊⋆⌋ ⌊v⌋]
        using Nmlize.simps(3−4,10) by presburger
      moreover have ⦃a⁻¹[t, u, v]⦄ = ⦃((t ⋆ u) ⋆ v) · a⁻¹[Dom t, Dom u, Dom v]⦄
      proof −
        have 1: VVV.arr (⦃t⦄, ⦃u⦄, ⦃v⦄)
          using tuv α′.preserves-reflects-arr arr-eval-Arr eval.simps(10)
          by (metis (no-types, lifting))
        have ⦃a⁻¹[t, u, v]⦄ = ((⦃t⦄ ⋆ ⦃u⦄) ⋆ ⦃v⦄) · a⁻¹[dom ⦃t⦄, dom ⦃u⦄, dom ⦃v⦄]
        proof −
          have VVV.arr (⦃t⦄, ⦃u⦄, ⦃v⦄)
            using tuv α′.preserves-reflects-arr arr-eval-Arr eval.simps(10)
            by (metis (no-types, lifting))
          thus ?thesis
            using t u v α′.naturality1 [of (⦃t⦄, ⦃u⦄, ⦃v⦄)] HoHV-def a′-def
                  VVV.dom-simp
            by simp
        qed
        also have ... = ⦃((t ⋆ u) ⋆ v) · a⁻¹[Dom t, Dom u, Dom v]⦄
          by (simp add: eval-simps′(4) t u v a′-def)
        finally show ?thesis by blast
      qed
      ultimately show coherent a⁻¹[t, u, v] by argo
    qed
  qed
  thus ?thesis using assms by blast
qed
```

corollary *eval-eqI*:
assumes *VPar t u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows $\{\!|t|\!\} = \{\!|u|\!\}$
using *assms coherence canonical-factorization* **by** *simp*

The following allows us to prove that two 1-cells in a bicategory are isomorphic simply by expressing them as the evaluations of terms having the same normalization. The benefits are: (1) we do not have to explicitly exhibit the isomorphism, which is canonical and is obtained by evaluating the reductions of the terms to their normalizations, and (2) the normalizations can be computed automatically by the simplifier.

lemma *canonically-isomorphicI*:
assumes $f = \{\!|t|\!\}$ **and** $g = \{\!|u|\!\}$ **and** *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows $f \cong g$
proof −
have $f \cong \{\!|t|\!\}$
using *assms isomorphic-reflexive ide-eval-Ide* **by** *blast*
also have ... $\cong \{\!|\lfloor t \rfloor|\!\}$
proof −
have «$\{\!|t\downarrow|\!\} : \{\!|t|\!\} \Rightarrow \{\!|\lfloor t \rfloor|\!\}$» $\wedge$ *iso* $\{\!|t\downarrow|\!\}$
using *assms*(*1*,*3*) *Can-red iso-eval-Can red-in-Hom*(*2*) *eval-in-hom*(*2*) **by** *fastforce*
thus *?thesis*
using *isomorphic-def* **by** *blast*
qed
also have ... $\cong \{\!|\lfloor u \rfloor|\!\}$
using *assms isomorphic-reflexive*
by (*simp add*: *Ide-Nmlize-Ide ide-eval-Ide*)
also have ... $\cong \{\!|u|\!\}$
proof −
have «$\{\!|u\downarrow|\!\} : \{\!|u|\!\} \Rightarrow \{\!|\lfloor u \rfloor|\!\}$» $\wedge$ *iso* $\{\!|u\downarrow|\!\}$
using *assms*(*2*,*4*) *Can-red iso-eval-Can red-in-Hom*(*2*) *eval-in-hom*(*2*) **by** *fastforce*
thus *?thesis*
using *isomorphic-def isomorphic-symmetric* **by** *blast*
qed
also have ... $\cong g$
using *assms isomorphic-reflexive ide-eval-Ide* **by** *blast*
finally show *?thesis* **by** *simp*
qed

end

end

1.7 Canonical Isomorphisms

In this section we develop some technology for working with canonical isomorphisms in a bicategory, which permits them to be specified simply by giving syntactic terms that evaluate to the domain and codomain, rather than often-cumbersome formulas expressed

in terms of unitors and associators.

theory *CanonicalIsos*
imports *Coherence*
begin

context *bicategory*
begin

interpretation *bicategorical-language* **..**
interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
notation *E.eval* (‹⦃-⦄›)

The next definition defines *can u t*, which denotes the unique canonical isomorphism from ⦃*t*⦄ to ⦃*u*⦄. The ordering of the arguments of *can* has been chosen to be the opposite of what was used for *hom*. Having the arguments to *can* this way makes it easier to see at a glance when canonical isomorphisms are composable. It could probably be argued that *hom* should have been defined this way as well, but that choice is somewhat well-entrenched by now and the argument for *can* is stronger, as it denotes an arrow and therefore appears in expressions composed with other arrows, rather than just as a hypothesis or conclusion.

definition *can*
where *can u t* ≡ ⦃*Inv* (*u*↓) · *t*↓⦄

1.7.1 Basic Properties

The following develop basic properties of *can*.

lemma *can-in-hom* [*intro*]:
assumes *Ide t* **and** *Ide u* **and** ⌊*t*⌋ = ⌊*u*⌋
shows «*can u t* : ⦃*t*⦄ ⇒ ⦃*u*⦄»
proof −
let *?v* = *Inv* (*u*↓) · *t*↓
have *1*: *Can ?v* ∧ *Dom ?v* = *t* ∧ *Cod ?v* = *u*
using *assms red-in-Hom Can-red Inv-in-Hom Can-Inv*(*1*) **by** *simp*
show «*can u t* : ⦃*t*⦄ ⇒ ⦃*u*⦄»
unfolding *can-def* **using** *1 E.eval-in-hom Can-implies-Arr*
by (*metis* (*no-types*, *lifting*))
qed

lemma *can-simps* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** ⌊*t*⌋ = ⌊*u*⌋
shows *arr* (*can u t*) **and** *dom* (*can u t*) = ⦃*t*⦄ **and** *cod* (*can u t*) = ⦃*u*⦄
using *assms can-in-hom* **by** *auto*

lemma *inverse-arrows-can*:
assumes *Ide t* **and** *Ide u* **and** ⌊*t*⌋ = ⌊*u*⌋
shows *iso* (*can u t*) **and** *inverse-arrows* (*can u t*) (*can t u*)
proof −
let *?v* = *Inv* (*u*↓) · *t*↓

```
have 1: Can ?v ∧ Dom ?v = t ∧ Cod ?v = u
  using assms red-in-Hom Can-red Inv-in-Hom Can-Inv(1) by simp
show iso (can u t)
  unfolding can-def using 1 E.iso-eval-Can by blast
show inverse-arrows (can u t) (can t u)
proof (unfold can-def)
  show inverse-arrows ⦃Inv (u↓) · t↓⦄ ⦃Inv (t↓) · u↓⦄
  proof
    show ide (⦃Inv (u↓) · t↓⦄ · ⦃Inv (t↓) · u↓⦄)
    proof −
      have ⦃Inv (u↓) · t↓⦄ · ⦃Inv (t↓) · u↓⦄ = ⦃(Inv (u↓) · t↓) · (Inv (t↓) · u↓)⦄
        by simp
      also have ... = ⦃u⦄
      proof (intro E.eval-eqI)
        show 2: VPar ((Inv (u↓) · t↓) · Inv (t↓) · u↓) u
          using assms 1 red-in-Hom Inv-in-Hom Ide-implies-Can Can-Inv Can-implies-Arr
                Can-red(1)
          by (simp add: Dom-Ide Cod-Ide)
        show ⌊(Inv (u↓) · t↓) · Inv (t↓) · u↓⌋ = ⌊u⌋
        proof −
          have 3: Can (Inv (t↓) · u↓)
            using Arr.simps(4) Can.simps(4) Can-Inv(1) Can-red(1) 2 assms(1) assms(2)
            by presburger
          have VSeq (Inv (u↓) · t↓) (Inv (t↓) · u↓)
            using 2 Arr.simps(4) by blast
          hence Can (Inv (u↓) · t↓) ∧ Can (Inv (t↓) · u↓) ∧
                  Dom (Inv (u↓) · t↓) = Cod (Inv (t↓) · u↓)
            using 3 1 by metis
          thus ?thesis
            by (metis (no-types) 2 Can.simps(4) Nmlize-Dom Dom-Ide Ide-Nmlize-Can
                assms(2))
        qed
      qed
      finally have ⦃Inv (u↓) · t↓⦄ · ⦃Inv (t↓) · u↓⦄ = ⦃u⦄
        by blast
      moreover have ide ⦃u⦄
        using assms E.ide-eval-Ide by simp
      ultimately show ?thesis by simp
    qed
    show ide (⦃Inv (t↓) · u↓⦄ · ⦃Inv (u↓) · t↓⦄)
    proof −
      have ⦃Inv (t↓) · u↓⦄ · ⦃Inv (u↓) · t↓⦄ = ⦃(Inv (t↓) · u↓) · (Inv (u↓) · t↓)⦄
        by simp
      also have ... = ⦃t⦄
      proof (intro E.eval-eqI)
        show 2: VPar ((Inv (t↓) · u↓) · Inv (u↓) · t↓) t
          using assms 1 red-in-Hom Inv-in-Hom Ide-implies-Can Can-Inv Can-implies-Arr
                Can-red(1)
          by (simp add: Dom-Ide Cod-Ide)
```

```
        show ⌊(Inv (t↓) · u↓) · Inv (u↓) · t↓⌋ = ⌊t⌋
          using assms 1 2
          by (metis (full-types) Arr.simps(4) Can.simps(4) Can-Inv(1) Can-red(1)
              Nml-Nmlize(4) Dom-Ide Ide-Nmlize-Can)
      qed
      finally have ⦃Inv (t↓) · u↓⦄ · ⦃Inv (u↓) · t↓⦄ = ⦃t⦄
        by blast
      moreover have ide ⦃t⦄
        using assms E.ide-eval-Ide by simp
      ultimately show ?thesis by simp
    qed
  qed
 qed
qed

lemma inv-can [simp]:
assumes Ide t and Ide u and ⌊t⌋ = ⌊u⌋
shows inv (can u t) = can t u
  using assms inverse-arrows-can by (simp add: inverse-unique)

lemma vcomp-can [simp]:
assumes Ide t and Ide u and Ide v and ⌊t⌋ = ⌊u⌋ and ⌊u⌋ = ⌊v⌋
shows can v u · can u t = can v t
proof (unfold can-def)
  have ⦃Inv (v↓) · u↓⦄ · ⦃Inv (u↓) · t↓⦄ = ⦃(Inv (v↓) · u↓) · (Inv (u↓) · t↓)⦄
    using assms by simp
  also have ... = ⦃Inv (v↓) · t↓⦄
  proof (intro E.eval-eqI)
    show VPar ((Inv (v↓) · u↓) · Inv (u↓) · t↓) (Inv (v↓) · t↓)
      using assms red-in-Hom Inv-in-Hom Ide-implies-Can
      by (simp add: Can-red(1))
    show ⌊(Inv (v↓) · u↓) · Inv (u↓) · t↓⌋ = ⌊Inv (v↓) · t↓⌋
      using assms Can-red(1) Nml-Nmlize(1) Nmlize-Inv Ide-Nmlize-Can
          Ide-implies-Can ‹VPar ((Inv (v↓) · u↓) · Inv (u↓) · t↓) (Inv (v↓) · t↓)›
      apply simp
      by (metis red-simps(4) Nmlize-red Dom-Cod VcompNml-Nml-Dom)
  qed
  finally show ⦃Inv (v↓) · u↓⦄ · ⦃Inv (u↓) · t↓⦄ = ⦃Inv (v↓) · t↓⦄
    by blast
qed

lemma hcomp-can [simp]:
assumes Ide t and Ide u and Ide v and Ide w and ⌊t⌋ = ⌊u⌋ and ⌊v⌋ = ⌊w⌋
and Src t = Trg v and Src u = Trg w
shows can u t ⋆ can w v = can (u ⋆ w) (t ⋆ v)
proof (unfold can-def)
  have ⦃Inv (u↓) · t↓⦄ ⋆ ⦃Inv (w↓) · v↓⦄ = ⦃(Inv (u↓) · t↓) ⋆ (Inv (w↓) · v↓)⦄
    using assms by simp
  also have ... = ⦃Inv ((u ⋆ w)↓) · (t ⋆ v)↓⦄
```

```
  proof (intro E.eval-eqI)
    show VPar (Inv (u⇓) · t⇓ ⋆ Inv (w⇓) · v⇓) (Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓)
    proof −
      have Arr (Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓)
      proof −
        have Ide (u ⋆ w)
          using assms by simp
        hence Can ((u ⋆ w)⇓)
          using assms Can-red by blast
        thus ?thesis
          using assms Can.simps(4) Can-Inv(1) Dom-Inv Can-implies-Arr Can-red(1)
                red-simps(4) Nmlize.simps(3) Ide.simps(3)
          by presburger
      qed
      moreover have Arr (Inv (u⇓) · t⇓ ⋆ Inv (w⇓) · v⇓)
        using assms red-in-Hom Inv-in-Hom Ide-implies-Can
        by (simp add: Can-red(1))
      moreover have Dom (Inv (u⇓) · t⇓ ⋆ Inv (w⇓) · v⇓) =
                    Dom (Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓)
        using assms red-in-Hom Inv-in-Hom Ide-implies-Can
        by (metis (no-types, lifting) Nml-HcompD(3−4) Dom.simps(3−4) red.simps(3)
            red-Nml)
      moreover have Cod (Inv (u⇓) · t⇓ ⋆ Inv (w⇓) · v⇓) =
                    Cod (Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓)
        using assms red-in-Hom Inv-in-Hom Ide-implies-Can red-Nml
        by (simp add: Can-red(1) Cod-Ide)
      ultimately show ?thesis by simp
    qed
    show ⌊Inv (u⇓) · t⇓ ⋆ Inv (w⇓) · v⇓⌋ = ⌊Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓⌋
      using assms Inv-in-Hom Ide-implies-Can Nmlize-Inv Ide-Nmlize-Can Can-red
            red2-Nml
      apply auto
      using VcompNml-HcompNml [of u w u w]
        apply (metis red-simps(4) Nml-HcompD(3−4) Nmlize-Nml red-simps(3) red-Nml)
        apply (metis Nml-HcompD(3−4) Nmlize.simps(3) Nmlize-Nml
          red-simps(3) Ide.simps(3) VcompNml-Nml-Dom red-Nml)
       apply (metis Can-red2(1) red-simps(4) Nml-HcompD(3−4) Nmlize.simps(3)
          Nmlize-Nml VcompNml-Cod-Nml red-Nml)
      using red2-Nml Nmlize-red2 Can-red2(1) Nmlize-Hcomp Dom-Ide Ide-implies-Arr
        VcompNml-Nml-Dom Nml-Nmlize(1) Nml-Nmlize(2) Nml-Nmlize(3)
        Nmlize.simps(3)
      by metis
  qed
  finally show ⦃Inv (u⇓) · t⇓⦄ ⋆ ⦃Inv (w⇓) · v⇓⦄ = ⦃Inv ((u ⋆ w)⇓) · (t ⋆ v)⇓⦄
    by blast
qed
```

1.7.2 Introduction Rules

To make the *can* notation useful, we need a way to introduce it. This is a bit tedious, because in general there can multiple *can* notations for the same isomorphism, and we have to use the right ones in the right contexts, otherwise we won't be able to compose them properly. Thankfully, we don't need the inverse versions of the theorems below, as they are easily provable from the non-inverse versions using *inv-can*.

lemma *canI-unitor-0*:
assumes *ide f*
shows $\mathrm{l}[f] = can\ \langle f\rangle\ (\langle trg\ f\rangle_0 \star \langle f\rangle)$
and $\mathrm{r}[f] = can\ \langle f\rangle\ (\langle f\rangle \star \langle src\ f\rangle_0)$
proof −
  have $can\ \langle f\rangle\ (\langle trg\ f\rangle_0 \star \langle f\rangle) = \llbracket \mathbf{l}[\langle f\rangle]\rrbracket$
    unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
  thus *1*: $\mathrm{l}[f] = can\ \langle f\rangle\ (\langle trg\ f\rangle_0 \star \langle f\rangle)$
    using *assms* **by** (*simp add*: $\mathfrak{l}$-*ide-simp*)
  have $can\ \langle f\rangle\ (\langle f\rangle \star \langle src\ f\rangle_0) = \llbracket \mathbf{r}[\langle f\rangle]\rrbracket$
    unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
  thus $\mathrm{r}[f] = can\ \langle f\rangle\ (\langle f\rangle \star \langle src\ f\rangle_0)$
    using *assms* **by** (*simp add*: $\mathfrak{r}$-*ide-simp*)
qed

lemma *canI-unitor-1*:
assumes *obj a*
shows $\mathrm{l}[a] = can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0)$
and $\mathrm{r}[a] = can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0)$
proof −
  have $can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0) = \llbracket \mathbf{l}[\langle a\rangle_0]\rrbracket$
    unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
  thus *1*: $\mathrm{l}[a] = can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0)$
    using *assms* **by** (*auto simp add*: $\mathfrak{l}$-*ide-simp*)
  have $can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0) = \llbracket \mathbf{r}[\langle a\rangle_0]\rrbracket$
    unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
  thus $\mathrm{r}[a] = can\ \langle a\rangle_0\ (\langle a\rangle_0 \star \langle a\rangle_0)$
    using *assms* **by** (*auto simp add*: $\mathfrak{r}$-*ide-simp*)
qed

lemma *canI-associator-0*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** $src\ f = trg\ g$ **and** $src\ g = trg\ h$
shows $\mathrm{a}[f, g, h] = can\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle)$
proof −
  have $can\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle) = \llbracket \mathbf{a}[\langle f\rangle, \langle g\rangle, \langle h\rangle]\rrbracket$
    unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
  thus $\mathrm{a}[f, g, h] = can\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle)$
    using *assms* **by** (*simp add*: α-*def*)
qed

lemma *canI-associator-1*:
assumes *ide f* **and** *ide g* **and** $src\ f = trg\ g$

shows a[*trg f*, *f*, *g*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*g*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*g*⟩)
and a[*f*, *src f*, *g*] = *can* (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$ ⋆ ⟨*g*⟩) ((⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ⋆ ⟨*g*⟩)
and a[*f*, *g*, *src g*] = *can* (⟨*f*⟩ ⋆ ⟨*g*⟩ ⋆ ⟨*src g*⟩$_0$) ((⟨*f*⟩ ⋆ ⟨*g*⟩) ⋆ ⟨*src g*⟩$_0$)
proof −
  show a[*trg f*, *f*, *g*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*g*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*g*⟩)
  proof −
    have *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*g*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*g*⟩) = ⦃**a**[⟨*trg f*⟩$_0$, ⟨*f*⟩, ⟨*g*⟩]⦄
      unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
    thus *?thesis*
      using *assms α-def* **by** *simp*
  qed
  show a[*f*, *src f*, *g*] = *can* (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$ ⋆ ⟨*g*⟩) ((⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ⋆ ⟨*g*⟩)
  proof −
    have *can* (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$ ⋆ ⟨*g*⟩) ((⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ⋆ ⟨*g*⟩) = ⦃**a**[⟨*f*⟩, ⟨*src f*⟩$_0$, ⟨*g*⟩]⦄
      unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
    thus *?thesis*
      using *assms α-def* **by** *simp*
  qed
  show a[*f*, *g*, *src g*] = *can* (⟨*f*⟩ ⋆ ⟨*g*⟩ ⋆ ⟨*src g*⟩$_0$) ((⟨*f*⟩ ⋆ ⟨*g*⟩) ⋆ ⟨*src g*⟩$_0$)
  proof −
    have *can* (⟨*f*⟩ ⋆ ⟨*g*⟩ ⋆ ⟨*src g*⟩$_0$) ((⟨*f*⟩ ⋆ ⟨*g*⟩) ⋆ ⟨*src g*⟩$_0$) = ⦃**a**[⟨*f*⟩, ⟨*g*⟩, ⟨*src g*⟩$_0$]⦄
      unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
    thus *?thesis*
      using *assms α-def* **by** *simp*
  qed
qed

lemma *canI-associator-2*:
assumes *ide f*
shows a[*trg f*, *trg f*, *f*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$) ⋆ ⟨*f*⟩)
and a[*trg f*, *f*, *src f*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*src f*⟩$_0$)
and a[*f*, *src f*, *src f*] = *can* (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$ ⋆ ⟨*src f*⟩$_0$) ((⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ⋆ ⟨*src f*⟩$_0$)
proof −
  show a[*trg f*, *trg f*, *f*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$) ⋆ ⟨*f*⟩)
  proof −
    have *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ((⟨*trg f*⟩$_0$ ⋆ ⟨*trg f*⟩$_0$) ⋆ ⟨*f*⟩) = ⦃**a**[⟨*trg f*⟩$_0$, ⟨*trg f*⟩$_0$, ⟨*f*⟩]⦄
      unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
    thus *?thesis*
      using *assms α-def* **by** *simp*
  qed
  show a[*trg f*, *f*, *src f*] = *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*src f*⟩$_0$)
  proof −
    have *can* (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ((⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩) ⋆ ⟨*src f*⟩$_0$) = ⦃**a**[⟨*trg f*⟩$_0$, ⟨*f*⟩, ⟨*src f*⟩$_0$]⦄
      unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
    thus *?thesis*
      using *assms α-def* **by** *simp*
  qed
  show a[*f*, *src f*, *src f*] = *can* (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$ ⋆ ⟨*src f*⟩$_0$) ((⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$) ⋆ ⟨*src f*⟩$_0$)
  proof −

have *can* $(\langle f\rangle \star \langle src\ f\rangle_0 \star \langle src\ f\rangle_0)\ ((\langle f\rangle \star \langle src\ f\rangle_0) \star \langle src\ f\rangle_0) =$
$\{\!|\mathbf{a}[\langle f\rangle, \langle src\ f\rangle_0, \langle src\ f\rangle_0]|\!\}$
unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
thus *?thesis*
using *assms* α*-def* **by** *simp*
qed
qed

lemma *canI-associator-3*:
assumes *obj a*
shows $\mathrm{a}[a, a, a] = can\ (\langle a\rangle_0 \star \langle a\rangle_0 \star \langle a\rangle_0)\ ((\langle a\rangle_0 \star \langle a\rangle_0) \star \langle a\rangle_0)$
proof −
have $can\ (\langle a\rangle_0 \star \langle a\rangle_0 \star \langle a\rangle_0)\ ((\langle a\rangle_0 \star \langle a\rangle_0) \star \langle a\rangle_0) = \{\!|\mathbf{a}[\langle a\rangle_0, \langle a\rangle_0, \langle a\rangle_0]|\!\}$
unfolding *can-def* **using** *assms* **by** (*intro E.eval-eqI*, *simp-all*)
thus *?thesis*
using *assms* α*-def* **by** *auto*
qed

lemma *canI-associator-hcomp*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *ide k*
and *src f* = *trg g* **and** *src g* = *trg h* **and** *src h* = *trg k*
shows $\mathrm{a}[f \star g, h, k] = can\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)\ ((((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle) \star \langle k\rangle)$
and $\mathrm{a}[f, g \star h, k] = can\ (\langle f\rangle \star (\langle g\rangle \star \langle h\rangle) \star \langle k\rangle)\ ((\langle f\rangle \star (\langle g\rangle \star \langle h\rangle)) \star \langle k\rangle)$
and $\mathrm{a}[f, g, h \star k] = can\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle \star \langle k\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)$
proof −
show $\mathrm{a}[f \star g, h, k] = can\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)\ ((((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle) \star \langle k\rangle)$
proof −
have $can\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)\ ((((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle) \star \langle k\rangle) =$
$(((f \star g) \star h \star k) \cdot (\mathrm{a}^{-1}[f, g, h \star k] \cdot (f \star g \star h \star k)) \cdot (f \star g \star h \star k)) \cdot$
$((f \star g \star h \star k) \cdot (f \star (g \star h \star k) \cdot (g \star h \star k) \cdot \mathrm{a}[g, h, k]) \cdot \mathrm{a}[f, g \star h, k]) \cdot$
$(((f \star g \star h) \cdot (f \star g \star h) \cdot \mathrm{a}[f, g, h]) \cdot ((f \star g) \star h) \star k)$
unfolding *can-def* **using** *assms* α*-def* a′*-def* α'*.map-ide-simp* **by** *simp*
also have ... $= \mathrm{a}^{-1}[f, g, h \star k] \cdot (f \star \mathrm{a}[g, h, k]) \cdot \mathrm{a}[f, g \star h, k] \cdot (\mathrm{a}[f, g, h] \star k)$
using *assms comp-arr-dom comp-cod-arr comp-assoc* **by** *simp*
also have ... $= \mathrm{a}[f \star g, h, k]$
using *assms pentagon* [*of f g h k*] *invert-side-of-triangle*(*1*) α*-def* α'*.map-ide-simp*
assoc-simps(*1*,*4*−*5*) *ideD*(*1*) *iso-assoc preserves-ide seqI*
by *simp*
finally show *?thesis* **by** *simp*
qed
show $\mathrm{a}[f, g \star h, k] = can\ (\langle f\rangle \star (\langle g\rangle \star \langle h\rangle) \star \langle k\rangle)\ ((\langle f\rangle \star (\langle g\rangle \star \langle h\rangle)) \star \langle k\rangle)$
proof −
have $can\ (\langle f\rangle \star (\langle g\rangle \star \langle h\rangle) \star \langle k\rangle)\ ((\langle f\rangle \star (\langle g\rangle \star \langle h\rangle)) \star \langle k\rangle) =$
$((f \star ((g \star h) \star k) \cdot (\mathrm{a}^{-1}[g, h, k] \cdot (g \star h \star k)) \cdot (g \star h \star k)) \cdot (f \star g \star h \star k)) \cdot$
$((f \star g \star h \star k) \cdot (f \star (g \star h \star k) \cdot (g \star h \star k) \cdot \mathrm{a}[g, h, k]) \cdot \mathrm{a}[f, g \star h, k]) \cdot$
$((f \star g \star h) \star k)$
unfolding *can-def* **using** *assms* α*-def* α'*.map-ide-simp* a′*-def* **by** *simp*
also have ... $= ((f \star \mathrm{a}^{-1}[g, h, k]) \cdot (f \star \mathrm{a}[g, h, k])) \cdot \mathrm{a}[f, g \star h, k]$
using *assms comp-arr-dom comp-cod-arr comp-assoc* **by** *simp*

```
    also have ... = a[f, g ⋆ h, k]
      using assms comp-cod-arr whisker-left [of f a⁻¹[g, h, k] a[g, h, k]]
            comp-assoc-assoc'
      by simp
    finally show ?thesis by simp
  qed
  show a[f, g, h ⋆ k] = can (⟨f⟩ ⋆ ⟨g⟩ ⋆ ⟨h⟩ ⋆ ⟨k⟩) ((⟨f⟩ ⋆ ⟨g⟩) ⋆ ⟨h⟩ ⋆ ⟨k⟩)
  proof −
    have can (⟨f⟩ ⋆ ⟨g⟩ ⋆ ⟨h⟩ ⋆ ⟨k⟩) ((⟨f⟩ ⋆ ⟨g⟩) ⋆ ⟨h⟩ ⋆ ⟨k⟩) =
          (f ⋆ g ⋆ h ⋆ k) · ((f ⋆ g ⋆ h ⋆ k) · (f ⋆ g ⋆ h ⋆ k) · a[f, g, h ⋆ k]) · ((f ⋆ g) ⋆ h ⋆ k)
      unfolding can-def using assms α-def α'.map-ide-simp by simp
    also have ... = a[f, g, h ⋆ k]
      using assms comp-arr-dom comp-cod-arr by simp
    finally show ?thesis by simp
  qed
qed
```

1.7.3 Rules for Eliminating 'can'

The following rules are used for replacing *can* in an expression by terms expressed using unit and associativity isomorphisms. They are not really expressed in the form of elimination rules, so the names are perhaps a bit misleading. They are typically applied as simplifications.

```
lemma canE-unitor:
assumes Ide f
shows can f (f ⋆ Src f) = r[⦃f⦄]
and can f (Trg f ⋆ f) = l[⦃f⦄]
and can (f ⋆ Src f) f = r⁻¹[⦃f⦄]
and can (Trg f ⋆ f) f = l⁻¹[⦃f⦄]
proof −
  show 1: can f (f ⋆ Src f) = r[⦃f⦄]
  proof −
    have f: ¬Nml (f ⋆ Src f)
      using assms Nml-HcompD(5) is-Prim0-Src by blast
    have can f (f ⋆ Src f) = ⦃Inv (f↓) · (⌊f⌋ ⇓ ⌊Src f⌋) · (f↓ ⋆ Src f↓)⦄
      using assms f can-def by simp
    also have ... = ⦃r[f]⦄
    proof (intro E.eval-eqI)
      show VPar (Inv (f↓) · (⌊f⌋ ⇓ ⌊Src f⌋) · (f↓ ⋆ Src f↓)) r[f]
     using assms Nmlize-in-Hom red-in-Hom red2-in-Hom Inv-in-Hom Can-red Can-implies-Arr
             Nml-Nmlize(1) Ide-implies-Can Nml-Src Nml-implies-Arr
             HcompNml-Nml-Src Ide-Cod Obj-implies-Ide
        apply (simp add: Dom-Ide Cod-Ide)
        apply (intro conjI)
      proof −
        assume f: Ide f
        have 1: Nml (Src f)
        proof −
          have Ide (Src f)
```

```
        using f Obj-implies-Ide by simp
      thus ?thesis
        using f Obj-Src Nml-Nmlize(1) Nmlize-Src(2) Ide-implies-Arr
        by metis
    qed
    show Arr (⌊f⌋ ⇓ ⌊Src f⌋)
      using f 1 Can-red2 Ide-Nmlize-Ide Nml-Nmlize Obj-implies-Ide by simp
    show Dom (⌊f⌋ ⇓ ⌊Src f⌋) = ⌊f⌋ ⋆ ⌊Src f⌋
      using f 1 Nml-Nmlize red2-in-Hom Ide-Nmlize-Ide Obj-implies-Ide by auto
    show ⌊f⌋ = Cod (⌊f⌋ ⇓ ⌊Src f⌋)
    proof −
      have Src ⌊f⌋ = Trg ⌊Src f⌋
        using f Nml-Nmlize Obj-implies-Ide by simp
      moreover have ⌊⌊f⌋ ⋆ ⌊Src f⌋⌋ = ⌊f⌋
        using f 1 Nml-Nmlize Nmlize-Src HcompNml-Nml-Src Nml-Src
        by (auto simp add: HcompNml-Nml-Obj)
      thus ?thesis
        using f 1 Obj-Src red2-in-Hom [of ⌊f⌋ ⌊Src f⌋] HcompNml-Nml-Src
              Nml-Nmlize Ide-Nmlize-Ide Obj-implies-Ide
        by auto
    qed
  qed
  show ⌊Inv (f↓) · (⌊f⌋ ⇓ ⌊Src f⌋) · (f↓ ⋆ Src f↓)⌋ = ⌊r[f]⌋
    using assms f HcompNml-Nml-Src Nml-Nmlize Can-red Nmlize-Hcomp
          Nmlize-Inv Nmlize-Src(1) Nmlize-red Nmlize-red2
          Ide-Nmlize-Can VcompNml-Nml-Ide red-Src
    apply (simp add: HcompNml-Nml-Obj)
  proof −
    assume f: Ide f
    have ⌊⌊f⌋ ⇓ Src f⌋ = ⌊f⌋
    proof −
      have Obj (Src f)
        using f Obj-Src by simp
      thus ?thesis
        using f apply (cases Src f)
        by (simp-all add: Nml-Nmlize(1) Nml-Nmlize(2) Ide-Nmlize-Ide)
    qed
    thus ⌊f⌋ ⌊·⌋ ⌊⌊f⌋ ⇓ Src f⌋ ⌊·⌋ ⌊f⌋ = ⌊f⌋
      by (metis Cod-Inv Can-red(1) Cod.simps(4) Nmlize.simps(4)
          Nmlize.simps(7) Nmlize-Vcomp-Cod-Arr red-simps(3)
          ‹VPar (Inv (f↓) · (⌊f⌋ ⇓ ⌊Src f⌋) · (f↓ ⋆ Src f↓)) r[f]› f)
  qed
qed
also have ... = r[⦃f⦄]
  using assms E.eval-Runit-Ide by blast
finally show ?thesis by simp
qed
show 2: can f (Trg f ⋆ f) = l[⦃f⦄]
proof −
```

```
      have f: ¬Nml (Trg f ⋆ f)
        using assms by (metis Nml.simps(4) Nml-HcompD(6))
      have can f (Trg f ⋆ f) = ⦃Inv (f↓) · (⌊Trg f⌋ ⇓ ⌊f⌋) · (Trg f↓ ⋆ f↓)⦄
        using assms f can-def by simp
      also have ... = ⦃𝗅[f]⦄
      proof (intro E.eval-eqI)
        show VPar (Inv (f↓) · (⌊Trg f⌋ ⇓ ⌊f⌋) · (Trg f↓ ⋆ f↓)) 𝗅[f]
       using assms Nmlize-in-Hom red-in-Hom red2-in-Hom Inv-in-Hom Can-red Can-implies-Arr
              Nml-Nmlize(1) Ide-implies-Can Nml-Trg Nml-implies-Arr
              HcompNml-Trg-Nml Ide-Cod Nmlize-Trg(1) Obj-implies-Ide
          apply (simp add: Dom-Ide Cod-Ide)
          apply (intro conjI)
        proof −
          assume f: Ide f
          have 1: Nml (Trg f)
          proof −
            have Ide (Trg f)
              using f Obj-implies-Ide by simp
            thus ?thesis
              using f Obj-Trg Nml-Nmlize(1) Nmlize-Trg(2) Ide-implies-Arr
              by metis
          qed
          show Arr (Trg f ⇓ ⌊f⌋)
            using f 1 Can-red2 Ide-Nmlize-Ide Nml-Nmlize(1,3) Obj-implies-Ide by simp
          show Dom (Trg f ⇓ ⌊f⌋) = Trg f ⋆ ⌊f⌋
            using f Obj-Trg 1 Nml-Nmlize(1,3) red2-in-Hom Ide-Nmlize-Ide Obj-implies-Ide by
auto
          show ⌊f⌋ = Cod (Trg f ⇓ ⌊f⌋)
          proof −
            have Src (Trg f) = Trg ⌊f⌋
              using f Nml-Nmlize(3) by simp
            thus ?thesis
           using f 1 Obj-Trg HcompNml-Trg-Nml Nml-Nmlize(1) Ide-Nmlize-Ide Obj-implies-Ide
              by auto
          qed
        qed
        show ⌊Inv (f↓) · (⌊Trg f⌋ ⇓ ⌊f⌋) · (Trg f↓ ⋆ f↓)⌋ = ⌊𝗅[f]⌋
          using assms f HcompNml-Nml-Src Nml-Nmlize Can-red Nmlize-Hcomp
                Nmlize-Inv Nmlize-Trg(1) Nmlize-red Nmlize-red2
                Ide-Nmlize-Can VcompNml-Nml-Ide red-Trg
          apply (simp add: HcompNml-Obj-Nml)
        proof −
          assume f: Ide f
          have ⌊Trg f ⇓ ⌊f⌋⌋ = ⌊f⌋
          proof −
            have Obj (Trg f)
              using f Obj-Trg by simp
            thus ?thesis
              using f apply (cases Trg f)
```

```
        by (simp-all add: Nml-Nmlize(1) Nml-Nmlize(2) Ide-Nmlize-Ide)
      qed
      thus ⌊f⌋ ⌊·⌋ ⌊Trg f⇓ ⌊f⌋⌋ ⌊·⌋ ⌊f⌋ = ⌊f⌋
        by (metis Cod-Inv Can-red(1) Cod.simps(4) Nmlize.simps(4)
            Nmlize.simps(5) Nmlize-Vcomp-Cod-Arr red-simps(3)
            ‹VPar (Inv (f↓) · (⌊Trg f⌋ ⇓ ⌊f⌋) · (Trg f↓ ⋆ f↓)) l[f]› f)
    qed
  qed
  also have ... = l[⦃f⦄]
    using assms E.eval-Lunit-Ide by blast
  finally show ?thesis by simp
qed
show can (f ⋆ Src f) f = r⁻¹[⦃f⦄]
  using assms 1 inv-can inv-inv
  by (metis (no-types, lifting) Nml-Nmlize(1) Nmlize.simps(3)
      Nmlize-Src(1) HcompNml-Nml-Src Ide.simps(3) Ide-implies-Arr
      Obj-Src Obj-implies-Ide Trg-Src)
show can (Trg f ⋆ f) f = l⁻¹[⦃f⦄]
  using assms 2 inv-can inv-inv
  by (metis (no-types, lifting) Nml-Nmlize(1) Nmlize.simps(3)
      Nmlize-Trg(1) HcompNml-Trg-Nml Ide.simps(3) Ide-implies-Arr
      Obj-Trg Obj-implies-Ide Src-Trg)
qed

lemma canE-associator:
assumes Ide f and Ide g and Ide h and Src f = Trg g and Src g = Trg h
shows can (f ⋆ g ⋆ h) ((f ⋆ g) ⋆ h) = a[⦃f⦄, ⦃g⦄, ⦃h⦄]
and can ((f ⋆ g) ⋆ h) (f ⋆ g ⋆ h) = a⁻¹[⦃f⦄, ⦃g⦄, ⦃h⦄]
proof −
  show can (f ⋆ g ⋆ h) ((f ⋆ g) ⋆ h) = a[⦃f⦄, ⦃g⦄, ⦃h⦄]
  proof −
    have can (f ⋆ g ⋆ h) ((f ⋆ g) ⋆ h) = ⦃Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓⦄
      using can-def by simp
    also have ... = ⦃a[f, g, h]⦄
    proof (intro E.eval-eqI)
      have 1: Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓ ∈ VHom ((f ⋆ g) ⋆ h) (f ⋆ g ⋆ h)
        using assms Inv-in-Hom [of (f ⋆ g ⋆ h)↓] Can-red [of f ⋆ g ⋆ h]
              red-in-Hom [of f ⋆ g ⋆ h] red-in-Hom [of (f ⋆ g) ⋆ h]
              Nmlize-Hcomp-Hcomp Nmlize-Hcomp-Hcomp'
              Ide-implies-Arr Nml-HcompNml Nmlize-Nml Ide-HcompNml
        by auto
      show par: VPar (Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓) a[f, g, h]
        using assms 1 Inv-in-Hom red-in-Hom Ide-in-Hom by simp
      show ⌊Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓⌋ = ⌊a[f, g, h]⌋
      proof −
        have ⌊Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓⌋ = Dom ⌊Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓⌋
        proof −
          have Can (Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓)
```

```
        using assms Nmlize-Inv Can-Inv
              Arr.simps(10) Arr.simps(4) Can.simps(4) Can-red(1) Ide.simps(3)
              Src.simps(3) Trg.simps(3) par
        by presburger
      hence Ide ⌊Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓⌋
        using Ide-Nmlize-Can by blast
      thus ?thesis
        using Ide-in-Hom Dom-Ide by presburger
    qed
    also have 6: ... = ⌊(f ⋆ g) ⋆ h⌋
      using 1 Nmlize-Dom [of Inv ((f ⋆ g ⋆ h)↓) · ((f ⋆ g) ⋆ h)↓]
      by (metis (mono-tags, lifting) mem-Collect-eq)
    also have 5: ... = Dom ⌊a[f, g, h]⌋
      using assms 6 par Nmlize-Dom Nml-Nmlize(4) by metis
    also have ... = ⌊a[f, g, h]⌋
      using assms 5 Ide-in-Hom by auto
    finally show ?thesis by simp
  qed
qed
also have ... = a[⦃f⦄, ⦃g⦄, ⦃h⦄]
  using assms E.eval-Assoc-Ide α-def by fastforce
finally show ?thesis by simp
qed
show can ((f ⋆ g) ⋆ h) (f ⋆ g ⋆ h) = a⁻¹[⦃f⦄, ⦃g⦄, ⦃h⦄]
proof −
  have can ((f ⋆ g) ⋆ h) (f ⋆ g ⋆ h) = ⦃Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓⦄
    using can-def by simp
  also have ... = ⦃a⁻¹[f, g, h]⦄
  proof (intro E.eval-eqI)
    have 1: Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓ ∈ VHom (f ⋆ g ⋆ h) ((f ⋆ g) ⋆ h)
      using assms Inv-in-Hom [of ((f ⋆ g) ⋆ h)↓] Can-red [of (f ⋆ g) ⋆ h]
            red-in-Hom [of (f ⋆ g) ⋆ h] red-in-Hom [of f ⋆ g ⋆ h]
            Nmlize-Hcomp-Hcomp Nmlize-Hcomp-Hcomp′
            Ide-implies-Arr Nml-HcompNml Nmlize-Nml Ide-HcompNml
      by auto
    show par: VPar (Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓) a⁻¹[f, g, h]
      using assms 1 Inv-in-Hom red-in-Hom Ide-in-Hom by simp
    show ⌊Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓⌋ = ⌊a⁻¹[f, g, h]⌋
    proof −
     have ⌊Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓⌋ = Dom ⌊Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓⌋
      proof −
        have Can (Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓)
          using assms Nmlize-Inv Can-Inv
                Arr.simps(10) Arr.simps(4) Can.simps(4) Can-red(1) Ide.simps(3)
                Src.simps(3) Trg.simps(3) par
          by presburger
        hence Ide ⌊Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓⌋
          using Ide-Nmlize-Can by blast
        thus ?thesis
```

```
          using Ide-in-Hom Dom-Ide by presburger
        qed
        also have 6: ... = ⌊f ⋆ g ⋆ h⌋
          using 1 Nmlize-Dom [of Inv (((f ⋆ g) ⋆ h)↓) · (f ⋆ g ⋆ h)↓]
          by (metis (mono-tags, lifting) mem-Collect-eq)
        also have 5: ... = Dom ⌊a⁻¹[f, g, h]⌋
          using assms 6 par Nmlize-Dom Nml-Nmlize(4) by metis
        also have ... = ⌊a⁻¹[f, g, h]⌋
          using assms 5 Ide-in-Hom by auto
        finally show ?thesis by simp
      qed
    qed
    also have ... = a⁻¹[⦃f⦄, ⦃g⦄, ⦃h⦄]
      using assms E.eval-Assoc'-Ide by fastforce
    finally show ?thesis by simp
  qed
qed

lemma can-Ide-self:
assumes Ide t
shows can t t = ⦃t⦄
proof (unfold can-def)
  show ⦃Inv (t↓) · t↓⦄ = ⦃t⦄
  proof (intro E.eval-eqI)
    show VPar (Inv (t↓) · t↓) t
    using assms red-in-Hom Inv-in-Hom Ide-implies-Can Can-Inv Can-red(1) Ide-in-Hom(2)
      by auto
    show ⌊Inv (t↓) · t↓⌋ = ⌊t⌋
      using assms red-in-Hom Inv-in-Hom Ide-implies-Can Cod-Inv
      by (metis (mono-tags, lifting) Can-red(1) Nml-Nmlize(1) Nmlize.simps(4)
          Nmlize-Inv Ide-Nmlize-Ide Nmlize-red Inv-Ide VcompNml-Ide-Nml
          ‹VPar (Inv (t↓) · t↓) t›)
  qed
qed
```

1.7.4 Rules for Whiskering

```
lemma whisker-can-right-0:
assumes Ide t and Ide u and ⌊t⌋ = ⌊u⌋ and ide f and Src t = ⟨trg f⟩₀
shows can u t ⋆ f = can (u ⋆ ⟨f⟩) (t ⋆ ⟨f⟩)
proof −
  have f = can ⟨f⟩ ⟨f⟩
    using assms can-Ide-self by simp
  thus ?thesis
    using assms Ide-implies-Arr hcomp-can
    by (metis Nml-Nmlize(2) Ide.simps(2) Trg.simps(2))
qed

lemma whisker-can-right-1:
```

assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** *obj a* **and** *Src t* $= \langle a \rangle_0$
shows *can u t* $\star$ *a* = *can* $(u \star \langle a \rangle_0)$ $(t \star \langle a \rangle_0)$
proof −
  have a = *can* $\langle a \rangle_0$ $\langle a \rangle_0$
    using *assms can-Ide-self* **by** *auto*
  thus *?thesis*
    using *assms Ide-implies-Arr hcomp-can*
    by (*metis Nml-Nmlize*(*2*) *Ide.simps*(*1*) *Trg.simps*(*1*))
qed

lemma *whisker-can-left-0*:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** *ide g* **and** *Trg t* $= \langle src\ g \rangle_0$
shows $g \star$ *can u t* = *can* $(\langle g \rangle \star u)$ $(\langle g \rangle \star t)$
proof −
  have g = *can* $\langle g \rangle$ $\langle g \rangle$
    using *assms can-Ide-self* **by** *simp*
  thus *?thesis*
    using *assms Ide-implies-Arr hcomp-can*
    by (*metis Nml-Nmlize*(*3*) *Ide.simps*(*2*) *Src.simps*(*2*))
qed

lemma *whisker-can-left-1*:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** *obj b* **and** *Trg t* $= \langle b \rangle_0$
shows $b \star$ *can u t* = *can* $(\langle b \rangle_0 \star u)$ $(\langle b \rangle_0 \star t)$
proof −
  have b = *can* $\langle b \rangle_0$ $\langle b \rangle_0$
    using *assms can-Ide-self* **by** *auto*
  thus *?thesis*
    using *assms Ide-implies-Arr hcomp-can*
    by (*metis Nml-Nmlize*(*3*) *Ide.simps*(*1*) *Src.simps*(*1*))
qed

end

end

1.8 Sub-Bicategories

In this section we give a construction of a sub-bicategory in terms of a predicate on the arrows of an ambient bicategory that has certain closure properties with respect to that bicategory. While the construction given here is likely to be of general use, it is not the most general sub-bicategory construction that one could imagine, because it requires that the sub-bicategory actually contain the unit and associativity isomorphisms of the ambient bicategory. Our main motivation for including this construction here is to apply it to exploit the fact that the sub-bicategory of endo-arrows of a fixed object is a monoidal category, which will enable us to transfer to bicategories a result about unit isomorphisms in monoidal categories.

theory *Subbicategory*
imports *Bicategory*
begin

1.8.1 Construction

locale *subbicategory* =
  B: *bicategory* V H a_B i src_B trg_B +
  subcategory V *Arr*
for V :: $'a$ *comp* (**infixr** ‹$\cdot_B$› *55*)
and H :: $'a$ *comp* (**infixr** ‹$\star_B$› *55*)
and a_B :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a_B[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and src_B :: $'a \Rightarrow 'a$
and trg_B :: $'a \Rightarrow 'a$
and *Arr* :: $'a \Rightarrow$ *bool* +
assumes *src-closed*: $Arr\ f \Longrightarrow Arr\ (src_B\ f)$
and *trg-closed*: $Arr\ f \Longrightarrow Arr\ (trg_B\ f)$
and *hcomp-closed*: $[\![\ Arr\ f;\ Arr\ g;\ trg_B\ f = src_B\ g\ ]\!] \Longrightarrow Arr\ (g \star_B f)$
and *assoc-closed*: $[\![\ Arr\ f \wedge B.ide\ f;\ Arr\ g \wedge B.ide\ g;\ Arr\ h \wedge B.ide\ h;$
$src_B\ f = trg_B\ g;\ src_B\ g = trg_B\ h\ ]\!] \Longrightarrow Arr\ (\mathrm{a}_B\ f\ g\ h)$
and *assoc'-closed*: $[\![\ Arr\ f \wedge B.ide\ f;\ Arr\ g \wedge B.ide\ g;\ Arr\ h \wedge B.ide\ h;$
$src_B\ f = trg_B\ g;\ src_B\ g = trg_B\ h\ ]\!] \Longrightarrow Arr\ (B.inv\ (\mathrm{a}_B\ f\ g\ h))$
and *lunit-closed*: $[\![\ Arr\ f;\ B.ide\ f\ ]\!] \Longrightarrow Arr\ (B.\mathfrak{l}\ f)$
and *lunit'-closed*: $[\![\ Arr\ f;\ B.ide\ f\ ]\!] \Longrightarrow Arr\ (B.inv\ (B.\mathfrak{l}\ f))$
and *runit-closed*: $[\![\ Arr\ f;\ B.ide\ f\ ]\!] \Longrightarrow Arr\ (B.\mathfrak{r}\ f)$
and *runit'-closed*: $[\![\ Arr\ f;\ B.ide\ f\ ]\!] \Longrightarrow Arr\ (B.inv\ (B.\mathfrak{r}\ f))$
begin

notation *B.in-hom* (‹«- : - $\Rightarrow_B$ -»›)

notation *comp* (**infixr** ‹·› *55*)

definition *hcomp* (**infixr** ‹$\star$› *53*)
where $g \star f = (if\ arr\ f \wedge arr\ g \wedge trg_B\ f = src_B\ g\ then\ g \star_B f\ else\ null)$

definition *src*
where $src\ \mu = (if\ arr\ \mu\ then\ src_B\ \mu\ else\ null)$

definition *trg*
where $trg\ \mu = (if\ arr\ \mu\ then\ trg_B\ \mu\ else\ null)$

interpretation *src*: *endofunctor* ‹(·)› *src*
  using *src-def null-char inclusion* $arr\text{-}char_{SbC}$ *src-closed trg-closed dom-closed cod-closed*
    dom-simp cod-simp
  apply *unfold-locales*
    apply *auto*[*4*]
  by (*metis B.src.as-nat-trans.preserves-comp-2 comp-char* $seq\text{-}char_{SbC}$)

interpretation *trg*: *endofunctor* ‹(·)› *trg*
using *trg-def null-char inclusion arr-char*$_{SbC}$ *src-closed trg-closed dom-closed cod-closed dom-simp cod-simp*
apply *unfold-locales*
apply *auto*[*4*]
by (*metis B.trg.as-nat-trans.preserves-comp-2 comp-char seq-char*$_{SbC}$)

interpretation *horizontal-homs* ‹(·)› *src trg*
using *src-def trg-def src.preserves-arr trg.preserves-arr null-char ide-char*$_{SbC}$ *arr-char*$_{SbC}$ *inclusion*
by (*unfold-locales*, *simp-all*)

interpretation *functor VV.comp* ‹(·)› ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$›
using *hcomp-def VV.arr-char*$_{SbC}$ *src-def trg-def arr-char*$_{SbC}$ *hcomp-closed dom-char*$_{SbC}$ *cod-char*$_{SbC}$ *VV.dom-char*$_{SbC}$ *VV.cod-char*$_{SbC}$
apply *unfold-locales*
apply *auto*[*2*]
proof −
fix *f*
assume *f*: *VV.arr f*
show *dom* (*fst f* $\star$ *snd f*) = *fst* (*VV.dom f*) $\star$ *snd* (*VV.dom f*)
proof −
have *dom* (*fst f* $\star$ *snd f*) = *B.dom* (*fst f*) $\star_B$ *B.dom* (*snd f*)
proof −
have *dom* (*fst f* $\star$ *snd f*) = *B.dom* (*fst f* $\star$ *snd f*)
using *f dom-char*$_{SbC}$
by (*simp add*: *arr-char*$_{SbC}$ *hcomp-closed hcomp-def*)
also have ... = *B.dom* (*fst f*) $\star_B$ *B.dom* (*snd f*)
using *f*
by (*metis* (*no-types*, *lifting*) *B.hcomp-simps*(*3*) *B.hseqI′ VV.arrE arrE hcomp-def inclusion src-def trg-def*)
finally show *?thesis* **by** *blast*
qed
also have ... = *fst* (*VV.dom f*) $\star$ *snd* (*VV.dom f*)
using *f VV.arr-char*$_{SbC}$ *VV.dom-char*$_{SbC}$ *arr-char*$_{SbC}$ *hcomp-def B.seq-if-composable dom-closed*
apply *simp*
by (*metis* (*no-types*, *lifting*) *dom-char*$_{SbC}$)
finally show *?thesis* **by** *simp*
qed
show *cod* (*fst f* $\star$ *snd f*) = *fst* (*VV.cod f*) $\star$ *snd* (*VV.cod f*)
proof −
have *cod* (*fst f* $\star$ *snd f*) = *B.cod* (*fst f*) $\star_B$ *B.cod* (*snd f*)
using *f VV.arr-char*$_{SbC}$ *arr-char*$_{SbC}$ *cod-char*$_{SbC}$ *hcomp-def src-def trg-def src-closed trg-closed hcomp-closed inclusion B.hseq-char arrE*
by *auto*
also have ... = *fst* (*VV.cod f*) $\star$ *snd* (*VV.cod f*)
using *f VV.arr-char*$_{SbC}$ *VV.cod-char*$_{SbC}$ *arr-char*$_{SbC}$ *hcomp-def B.seq-if-composable*

cod-closed
apply *simp*
by (*metis* (*no-types*, *lifting*) *cod-char*$_{SbC}$)
finally show *?thesis* **by** *simp*
qed
next
fix *f g*
assume *fg*: *VV.seq g f*
show *fst* (*VV.comp g f*) $\star$ *snd* (*VV.comp g f*) = (*fst g* $\star$ *snd g*) $\cdot$ (*fst f* $\star$ *snd f*)
proof −
have *fst* (*VV.comp g f*) $\star$ *snd* (*VV.comp g f*) = *fst g* $\cdot$ *fst f* $\star$ *snd g* $\cdot$ *snd f*
using *fg VV.seq-char*$_{SbC}$ *VV.comp-char VxV.comp-char VxV.not-Arr-Null*
by (*metis* (*no-types*, *lifting*) *VxV.seqE*$_{PC}$ *prod.sel*(*1*−*2*))
also have ... = (*fst g* $\cdot_B$ *fst f*) $\star_B$ (*snd g* $\cdot_B$ *snd f*)
using *fg comp-char hcomp-def VV.seq-char*$_{SbC}$ *inclusion arr-char*$_{SbC}$ *seq-char*$_{SbC}$ *B.hseq-char*
by (*metis* (*no-types*, *lifting*) *B.hseq-char′ VxV.seq-char null-char*)
also have *1*: ... = (*fst g* $\star_B$ *snd g*) $\cdot_B$ (*fst f* $\star_B$ *snd f*)
proof −
have *src*$_B$ (*fst g*) = *trg*$_B$ (*snd g*)
by (*metis* (*no-types*, *lifting*) *VV.arrE VV.seq-char*$_{SbC}$ *fg src-def trg-def*)
thus *?thesis*
using *fg VV.seq-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char*$_{SbC}$ *seq-char*$_{SbC}$ *inclusion B.interchange*
by (*meson VxV.seqE*$_{PC}$)
qed
also have ... = (*fst g* $\star$ *snd g*) $\cdot$ (*fst f* $\star$ *snd f*)
using *fg comp-char hcomp-def VV.seq-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char*$_{SbC}$ *seq-char*$_{SbC}$ *inclusion*
B.hseq-char′ hcomp-closed src-def trg-def
by (*metis* (*no-types*, *lifting*) *1*)
finally show *?thesis* **by** *auto*
qed
qed

interpretation *horizontal-composition* ‹(·)› ‹($\star$)› *src trg*
using *arr-char*$_{SbC}$ *src-def trg-def src-closed trg-closed*
apply (*unfold-locales*)
using *hcomp-def inclusion not-arr-null* **by** *auto*

abbreviation a
where a μ ν τ ≡ *if VVV.arr* (μ, ν, τ) *then* a$_B$ μ ν τ *else null*

abbreviation (*input*) α_{SB}
where α_{SB} $\mu\nu\tau$ ≡ a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))

lemma *assoc-closed′*:
assumes *VVV.arr* $\mu\nu\tau$
shows *Arr* (α_{SB} $\mu\nu\tau$)

proof −
have *1*: $B.VVV.arr\ \mu\nu\tau$
using *assms* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$
src-def trg-def inclusion
by *auto*
show $Arr\ (\alpha_{SB}\ \mu\nu\tau)$
proof −
have $Arr\ (\alpha_{SB}\ \mu\nu\tau) =$
$Arr\ ((fst\ \mu\nu\tau \star_B fst\ (snd\ \mu\nu\tau)) \star_B snd\ (snd\ \mu\nu\tau)) \cdot_B \alpha_{SB}\ (B.VVV.dom\ \mu\nu\tau))$
using *assms 1 B.α-def B.assoc-naturality1* [*of* $fst\ \mu\nu\tau$ $fst\ (snd\ \mu\nu\tau)$ $snd\ (snd\ \mu\nu\tau)$]
$VV.arr\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VVV.dom\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$
apply *simp*
by (*metis* (*no-types*, *lifting*) $arr\text{-}char_{SbC}$ $dom\text{-}char_{SbC}$ *dom-closed src.preserves-dom trg.preserves-dom*)
also have ...
proof (*intro comp-closed*)
show $Arr\ (fst\ \mu\nu\tau \star_B fst\ (snd\ \mu\nu\tau) \star_B snd\ (snd\ \mu\nu\tau))$
using *assms 1* $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *hcomp-closed*
by (*metis* (*no-types*, *lifting*) *B.H.preserves-reflects-arr B.trg-hcomp* $VV.arr\text{-}char_{SbC}$ $VVV.arrE\ arr\text{-}char_{SbC}$)
show $B.cod\ (\mathrm{a}\ (fst\ (B.VVV.dom\ \mu\nu\tau))\ (fst\ (snd\ (B.VVV.dom\ \mu\nu\tau)))$
$(snd\ (snd\ (B.VVV.dom\ \mu\nu\tau)))) =$
$B.dom\ (fst\ \mu\nu\tau \star_B fst\ (snd\ \mu\nu\tau) \star_B snd\ (snd\ \mu\nu\tau))$
using *assms 1* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *B.VxVxV.dom-char B.VVV.dom-simp B.VVV.cod-simp*
apply *simp*
by (*metis* (*no-types*, *lifting*) $B.VV.arr\text{-}char_{SbC}$ *B.VVV.arrE B.α.preserves-reflects-arr B.assoc-naturality1 B.seqE arr-dom* $dom\text{-}char_{SbC}$ *src-dom trg-dom*)
show $Arr\ (\mathrm{a}\ (fst\ (B.VVV.dom\ \mu\nu\tau))\ (fst\ (snd\ (B.VVV.dom\ \mu\nu\tau)))$
$(snd\ (snd\ (B.VVV.dom\ \mu\nu\tau))))$
proof −
have $VVV.arr\ (B.VVV.dom\ \mu\nu\tau)$
using *1* $B.VVV.dom\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
apply *simp*
by (*metis* (*no-types*, *lifting*) *VVV.arrE arr-dom assms dom-simp src-dom trg-dom*)
moreover have $Arr\ (\mathrm{a}_B\ (B.dom\ (fst\ \mu\nu\tau))\ (B.dom\ (fst\ (snd\ \mu\nu\tau)))$
$(B.dom\ (snd\ (snd\ \mu\nu\tau))))$
proof −
have $B.VVV.ide\ (B.VVV.dom\ \mu\nu\tau)$
using *1 B.VVV.ide-dom* **by** *blast*
thus *?thesis*
using *assms B.α-def* $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *B.VVV.ide-char B.VV.ide-char*
dom-closed assoc-closed
by (*metis* (*no-types*, *lifting*) *1 B.ide-dom B.src-dom B.trg-dom* $VV.arr\text{-}char_{SbC}$ *VVV.arrE* $arr\text{-}char_{SbC}$)

```
      qed
      ultimately show ?thesis
        using 1 B.VVV.ide-dom assoc-closed B.VVV.dom-char_SbC
        apply simp
        by (metis (no-types, lifting) B.VV.arr-char_SbC B.VVV.arrE B.VVV.inclusion
            B.VxV.dom-char B.VxVxV.arrE B.VxVxV.dom-char prod.sel(1) prod.sel(2))
    qed
  qed
  finally show ?thesis by blast
 qed
qed

lemma lunit-closed':
assumes Arr f
shows Arr (B.𝔩 f)
proof −
 have 1: arr f ∧ arr (B.𝔩 (B.dom f))
   using assms arr-char_SbC lunit-closed dom-closed B.ide-dom inclusion by simp
 moreover have B.dom f = B.cod (B.𝔩 (B.dom f))
   using 1 arr-char_SbC B.𝔩.preserves-cod inclusion by simp
 moreover have B.𝔩 f = f · B.𝔩 (B.dom f)
   using assms 1 B.𝔩.naturality1 inclusion comp-char arr-char_SbC by simp
 ultimately show ?thesis
   using arr-char_SbC comp-closed cod-char_SbC seqI dom-simp by auto
qed

lemma runit-closed':
assumes Arr f
shows Arr (B.𝔯 f)
proof −
 have 1: arr f ∧ arr (B.𝔯 (B.dom f))
   using assms arr-char_SbC runit-closed dom-closed B.ide-dom inclusion
   by simp
 moreover have B.dom f = B.cod (B.𝔯 (B.dom f))
   using 1 arr-char_SbC B.𝔩.preserves-cod inclusion by simp
 moreover have B.𝔯 f = f · B.𝔯 (B.dom f)
   using assms 1 B.𝔯.naturality1 inclusion comp-char arr-char_SbC by simp
 ultimately show ?thesis
   using arr-char_SbC comp-closed cod-char_SbC seqI dom-simp by auto
qed

interpretation natural-isomorphism VVV.comp ‹(·)› HoHV HoVH α_SB
proof
 fix μντ
 show ¬ VVV.arr μντ ⟹ α_SB μντ = null
   by simp
 assume μντ: VVV.arr μντ
 have 1: B.VVV.arr μντ
   using μντ VVV.arr-char_SbC VV.arr-char_SbC B.VVV.arr-char_SbC B.VV.arr-char_SbC
```

$arr\text{-}char_{SbC}$
$src\text{-}def\ trg\text{-}def\ inclusion$
by *auto*
show $arr\ (\alpha_{SB}\ \mu\nu\tau)$
using $\mu\nu\tau$ *assoc-closed'* **by** *blast*
have *3*: $Arr\ (fst\ \mu\nu\tau) \wedge Arr\ (fst\ (snd\ \mu\nu\tau)) \wedge Arr\ (snd\ (snd\ \mu\nu\tau)) \wedge$
$src_B\ (fst\ \mu\nu\tau) = trg_B\ (fst\ (snd\ \mu\nu\tau)) \wedge$
$src_B\ (fst\ (snd\ \mu\nu\tau)) = trg_B\ (snd\ (snd\ \mu\nu\tau))$
using $\mu\nu\tau$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *src-def trg-def* $arr\text{-}char_{SbC}$ **by** *auto*
show $HoVH\ \mu\nu\tau \cdot \alpha_{SB}\ (VVV.dom\ \mu\nu\tau) = \alpha_{SB}\ \mu\nu\tau$
proof −
have $\alpha_{SB}\ \mu\nu\tau = (fst\ \mu\nu\tau \star_B fst\ (snd\ \mu\nu\tau) \star_B snd\ (snd\ \mu\nu\tau)) \cdot_B$
$\mathrm{a}_B\ (B.dom\ (fst\ \mu\nu\tau))\ (B.dom\ (fst\ (snd\ \mu\nu\tau)))\ (B.dom\ (snd\ (snd\ \mu\nu\tau)))$
using *3 inclusion B.assoc-naturality1* [*of* $fst\ \mu\nu\tau\ fst\ (snd\ \mu\nu\tau)\ snd\ (snd\ \mu\nu\tau)$]
by (*simp add*: $\mu\nu\tau$)
also have ... $= (fst\ \mu\nu\tau \star fst\ (snd\ \mu\nu\tau) \star snd\ (snd\ \mu\nu\tau)) \cdot$
$\mathrm{a}_B\ (dom\ (fst\ \mu\nu\tau))\ (dom\ (fst\ (snd\ \mu\nu\tau)))\ (dom\ (snd\ (snd\ \mu\nu\tau)))$
using *1 3* $\mu\nu\tau$ *hcomp-closed assoc-closed dom-closed hcomp-def comp-def inclusion comp-char* $dom\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
apply *simp*
using *B.hcomp-simps(2−3)* $arr\text{-}char_{SbC}$ **by** *presburger*
also have ... $= HoVH\ \mu\nu\tau \cdot \alpha_{SB}\ (VVV.dom\ \mu\nu\tau)$
using $\mu\nu\tau$ *B.α-def HoVH-def* $VVV.dom\text{-}char_{SbC}$ $VV.dom\text{-}char_{SbC}$ *VxVxV.dom-char*
apply *simp*
by (*metis* (*no-types*, *lifting*) $VV.arr\text{-}char_{SbC}$ *VVV.arrE VVV.arr-dom VxV.dom-char dom-simp*)
finally show *?thesis* **by** *argo*
qed
show $\alpha_{SB}\ (VVV.cod\ \mu\nu\tau) \cdot HoHV\ \mu\nu\tau = \alpha_{SB}\ \mu\nu\tau$
proof −
have $\alpha_{SB}\ \mu\nu\tau =$
$\mathrm{a}_B\ (B.cod\ (fst\ \mu\nu\tau))\ (B.cod\ (fst\ (snd\ \mu\nu\tau)))\ (B.cod\ (snd\ (snd\ \mu\nu\tau))) \cdot_B$
$(fst\ \mu\nu\tau \star_B fst\ (snd\ \mu\nu\tau)) \star_B snd\ (snd\ \mu\nu\tau)$
using *3 inclusion B.assoc-naturality2* [*of* $fst\ \mu\nu\tau\ fst\ (snd\ \mu\nu\tau)\ snd\ (snd\ \mu\nu\tau)$]
by (*simp add*: $\mu\nu\tau$)
also have ... $= \mathrm{a}_B\ (cod\ (fst\ \mu\nu\tau))\ (cod\ (fst\ (snd\ \mu\nu\tau)))\ (cod\ (snd\ (snd\ \mu\nu\tau))) \cdot$
$((fst\ \mu\nu\tau \star fst\ (snd\ \mu\nu\tau)) \star snd\ (snd\ \mu\nu\tau))$
by (*simp add*: *3* $arrI_{SbC}$ *assoc-closed* $cod\text{-}char_{SbC}$ *cod-closed hcomp-closed hcomp-def inclusion comp-def*)
also have ... $= \alpha_{SB}\ (VVV.cod\ \mu\nu\tau) \cdot HoHV\ \mu\nu\tau$
using $\mu\nu\tau$ *B.α-def HoHV-def* $VVV.cod\text{-}char_{SbC}$ $VV.cod\text{-}char_{SbC}$ *VxVxV.cod-char* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *arr-cod src-cod trg-cod*
by *simp*
finally show *?thesis* **by** *argo*
qed
next
fix *fgh*
assume *fgh*: *VVV.ide fgh*
show *iso* (α_{SB} *fgh*)

```
  proof −
    have 1: B.arr (fst (snd fgh)) ∧ B.arr (snd (snd fgh)) ∧
              src_B (fst (snd fgh)) = trg_B (snd (snd fgh)) ∧
              src_B (fst fgh) = trg_B (fst (snd fgh))
      using fgh VVV.ide-char VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def
            arr-char_SbC inclusion
      by auto
    have 2: B.ide (fst fgh) ∧ B.ide (fst (snd fgh)) ∧ B.ide (snd (snd fgh))
      using fgh VVV.ide-char_SbC ide-char_SbC by blast
    have α_SB fgh = a_B (fst fgh) (fst (snd fgh)) (snd (snd fgh))
      using fgh B.α-def by simp
    moreover have B.VVV.ide fgh
    using fgh 1 2 VVV.ide-char B.VVV.ide-char_SbC VVV.arr-char_SbC B.VVV.arr-char_SbC
            src-def trg-def inclusion arr-char_SbC B.VV.arr-char_SbC
      by simp
    moreover have Arr (a_B (fst fgh) (fst (snd fgh)) (snd (snd fgh)))
      using fgh 1 VVV.ide-char VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def
            arr-char_SbC assoc-closed' B.α-def
      by simp
    moreover have Arr (B.inv (a_B (fst fgh) (fst (snd fgh)) (snd (snd fgh))))
      using fgh 1 VVV.ide-char VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def
            arr-char_SbC assoc'-closed
      by (simp add: VVV.arr-char_SbC 2 B.VVV.ide-char calculation(2))
    ultimately show ?thesis
      using fgh iso-char_SbC B.α.components-are-iso by auto
  qed
qed

interpretation L: endofunctor ‹(·)› L
  using endofunctor-L by auto
interpretation R: endofunctor ‹(·)› R
  using endofunctor-R by auto

interpretation L: faithful-functor ‹(·)› ‹(·)› L
proof
  fix f f'
  assume par: par f f'
  assume eq: L f = L f'
  have B.par f f'
    using par inclusion arr-char_SbC dom-simp cod-simp by fastforce
  moreover have B.L f = B.L f'
  proof −
    have ∀ a. Arr a ⟶ B.arr a
      by (simp add: inclusion)
    moreover have 1: ∀ a. arr a ⟶ (if arr a then hseq (trg a) a else arr null)
      using L.preserves-arr by presburger
    moreover have Arr f ∧ Arr (trg f) ∧ trg_B f = src_B (trg f)
      by (simp add: ‹B.par f f'› arrE par trg-closed trg-def)
    ultimately show ?thesis
```

```
      by (metis ‹B.par f f'› eq hcomp-def hseq-char' par trg-def)
  qed
  ultimately show f = f'
    using B.L.is-faithful by blast
qed
interpretation L: full-functor ‹(·)› ‹(·)› L
proof
  fix f f' ν
  assume f: ide f and f': ide f' and ν: «ν : L f ⇒ L f'»
  have 1: L f = trg_B f ⋆_B f ∧ L f' = trg_B f' ⋆_B f'
    using f f' hcomp-def trg-def arr-char_SbC ide-char_SbC trg-closed by simp
  have 2: «ν : trg_B f ⋆_B f ⇒_B trg_B f' ⋆_B f'»
    using 1 f f' ν hcomp-def trg-def src-def inclusion
          dom-char_SbC cod-char_SbC hseq-char' arr-char_SbC ide-char_SbC trg-closed null-char
    by (simp add: arr-char_SbC in-hom-char_SbC)
  show ∃μ. «μ : f ⇒ f'» ∧ L μ = ν
  proof −
    let ?μ = B.ι f' ·_B ν ·_B B.inv (B.ι f)
    have μ: «?μ : f ⇒ f'» ∧ «?μ : f ⇒_B f'»
    proof −
      have «?μ : f ⇒_B f'»
        using f f' ν 2 B.ι-ide-simp lunit'-closed lunit-closed' ide-char_SbC by auto
      thus ?thesis
        using f f' ν in-hom-char_SbC arr-char_SbC comp-closed ide-char_SbC
              lunit'-closed lunit-closed
        by (metis (no-types, lifting) B.arrI B.seqE in-homE)
    qed
    have μ-eq: ?μ = B.ι f' · ν · B.inv (B.ι f)
    proof −
      have ?μ = B.ι f' · ν ·_B B.inv (B.ι f)
        by (metis (no-types, lifting) B.arrI B.seqE μ ν arrE comp-closed f f'
            ide-char_SbC in-hom-char_SbC comp-def lunit'-closed lunit-closed)
      also have ... = B.ι f' · ν · B.inv (B.ι f)
        using f f' ν μ arr-char_SbC inclusion comp-char comp-closed ide-char_SbC
              lunit'-closed lunit-closed
        by (metis (no-types, lifting) B.seqE in-homE)
      finally show ?thesis by simp
    qed
    have L ?μ = ν
    proof −
      have L ?μ = trg_B ?μ ⋆_B ?μ
        using μ μ-eq hcomp-def trg-def inclusion arr-char_SbC trg-closed by auto
      also have ... = (trg_B ?μ ⋆_B ?μ) ·_B (B.inv (B.ι f) ·_B B.ι f)
      proof −
        have B.inv (B.ι f) ·_B B.ι f = trg_B f ⋆_B f
          using f ide-char_SbC B.comp-inv-arr B.inv-is-inverse by auto
        moreover have B.dom (trg_B ?μ ⋆_B ?μ) = trg_B f ⋆_B f
        proof −
          have B.dom (trg_B ?μ) = trg_B f
```

```
          using f μ B.vconn-implies-hpar(2) by force
        moreover have B.dom ?μ = f
          using μ by blast
        ultimately show ?thesis
          using B.hcomp-simps [of trg_B ?μ ?μ]
          by (metis (no-types, lifting) B.hseqI' B.ideD(1) B.src-trg
              B.trg.preserves-reflects-arr B.trg-dom f ide-char_SbC)
      qed
      ultimately show ?thesis
        using μ μ-eq B.comp-arr-dom in-hom-char_SbC by auto
    qed
    also have ... = ((trg_B ?μ ⋆_B ?μ) ·_B B.inv (B.𝔦 f)) ·_B B.𝔦 f
      using B.comp-assoc by simp
    also have ... = (B.inv (B.𝔦 f') ·_B ?μ) ·_B B.𝔦 f
      using μ μ-eq B.𝔦'.naturality [of ?μ] by auto
    also have ... = (B.inv (B.𝔦 f') ·_B B.𝔦 f') ·_B ν ·_B (B.inv (B.𝔦 f) ·_B B.𝔦 f)
      using μ μ-eq arr-char_SbC arrI comp-simp B.comp-assoc by metis
    also have ... = ν
      using f f' ν 2 B.comp-arr-dom B.comp-cod-arr ide-char_SbC
            B.𝔦.components-are-iso B.𝔦-ide-simp B.comp-inv-arr'
      by auto
    finally show ?thesis by blast
  qed
  thus ?thesis
    using μ by auto
 qed
qed

interpretation R: faithful-functor ‹(·)› ‹(·)› R
proof
 fix f f'
 assume par: par f f'
 assume eq: R f = R f'
 have B.par f f'
   using par inclusion arr-char_SbC dom-simp cod-simp by fastforce
 moreover have B.R f = B.R f'
 proof −
   have ∀ a. Arr a ⟶ B.arr a
     by (simp add: inclusion)
   moreover have 1: ∀ a. arr a ⟶ (if arr a then hseq a (src a) else arr null)
     using R.preserves-arr by presburger
   moreover have arr f ∧ arr (src f) ∧ trg_B (src f) = src_B f
     by (meson 1 hcomp-def hseq-char' par)
   ultimately show ?thesis
     by (metis ‹B.par f f'› eq hcomp-def hseq-char' src-def)
 qed
 ultimately show f = f'
   using B.R.is-faithful by blast
qed
```

interpretation R: *full-functor* ‹(·)› ‹(·)› R
proof
  fix $f\ f'\ \nu$
  assume f: *ide* f **and** f': *ide* f' **and** ν: «$\nu : R\ f \Rightarrow R\ f'$»
  have *1*: $R\ f = f \star_B src_B\ f \wedge R\ f' = f' \star_B src_B\ f'$
    using $f\ f'$ *hcomp-def src-def* $arr\text{-}char_{SbC}$ $ide\text{-}char_{SbC}$ *src-closed* **by** *simp*
  have *2*: «$\nu : f \star_B src_B\ f \Rightarrow_B f' \star_B src_B\ f'$»
    using *1* $f\ f'\ \nu$ *hcomp-def trg-def src-def inclusion*
      $dom\text{-}char_{SbC}$ $cod\text{-}char_{SbC}$ *hseq-char'* $arr\text{-}char_{SbC}$ $ide\text{-}char_{SbC}$ *trg-closed null-char*
    by (*simp add*: $arr\text{-}char_{SbC}$ $in\text{-}hom\text{-}char_{SbC}$)
  show $\exists \mu.$ «$\mu : f \Rightarrow f'$» $\wedge R\ \mu = \nu$
  proof −
    let $?\mu = B.\mathfrak{r}\ f' \cdot_B \nu \cdot_B B.inv\ (B.\mathfrak{r}\ f)$
    have μ: «$?\mu : f \Rightarrow f'$» $\wedge$ «$?\mu : f \Rightarrow_B f'$»
    proof −
      have «$?\mu : f \Rightarrow_B f'$»
        using $f\ f'\ \nu$ *2* $B.\mathfrak{r}$*-ide-simp runit'-closed runit-closed'* $ide\text{-}char_{SbC}$ **by** *auto*
      thus *?thesis*
        by (*metis* (*no-types*, *lifting*) *B.arrI B.seqE* ν *arrE* $arrI_{SbC}$ *comp-closed* $f\ f'$
          $ide\text{-}char_{SbC}$ $in\text{-}hom\text{-}char_{SbC}$ *runit'-closed runit-closed'*)
    qed
    have μ*-eq*: $?\mu = B.\mathfrak{r}\ f' \cdot \nu \cdot B.inv\ (B.\mathfrak{r}\ f)$
    proof −
      have $?\mu = B.\mathfrak{r}\ f' \cdot \nu \cdot_B B.inv\ (B.\mathfrak{r}\ f)$
        using $f\ f'\ \nu\ \mu$ $arr\text{-}char_{SbC}$ *inclusion comp-char comp-closed* $ide\text{-}char_{SbC}$
          runit'-closed runit-closed
        by (*metis* (*no-types*, *lifting*) *B.seqE in-homE*)
      also have ... $= B.\mathfrak{r}\ f' \cdot \nu \cdot B.inv\ (B.\mathfrak{r}\ f)$
        using $f\ f'\ \nu\ \mu$ $arr\text{-}char_{SbC}$ *inclusion comp-char comp-closed* $ide\text{-}char_{SbC}$
          runit'-closed runit-closed
        by (*metis* (*no-types*, *lifting*) *B.arrI B.comp-in-homE* $in\text{-}hom\text{-}char_{SbC}$)
      finally show *?thesis* **by** *simp*
    qed
    have $R\ ?\mu = \nu$
    proof −
      have $R\ ?\mu = ?\mu \star_B src_B\ ?\mu$
        using μ μ*-eq hcomp-def src-def inclusion* $arr\text{-}char_{SbC}$ *src-closed* **by** *auto*
      also have ... $= (?\mu \star_B src_B\ ?\mu) \cdot_B (B.inv\ (B.\mathfrak{r}\ f) \cdot_B B.\mathfrak{r}\ f)$
      proof −
        have $B.inv\ (B.\mathfrak{r}\ f) \cdot_B B.\mathfrak{r}\ f = f \star_B src_B\ f$
          using f $ide\text{-}char_{SbC}$ *B.comp-inv-arr B.inv-is-inverse* **by** *auto*
        moreover have $B.dom\ (?\mu \star_B src_B\ ?\mu) = f \star_B src_B\ f$
          using f μ μ*-eq ide-char* $arr\text{-}char_{SbC}$ *B.src-dom* [*of* $?\mu$]
          by (*metis* (*no-types*, *lifting*) *B.R.as-nat-trans.preserves-comp-2 B.R.preserves-seq*
            B.dom-src B.hcomp-simps(*3*) *B.in-homE*)
        ultimately show *?thesis*
          using μ μ*-eq B.comp-arr-dom* $in\text{-}hom\text{-}char_{SbC}$ **by** *auto*
      qed
      also have ... $= ((?\mu \star_B src_B\ ?\mu) \cdot_B B.inv\ (B.\mathfrak{r}\ f)) \cdot_B B.\mathfrak{r}\ f$

```
      using B.comp-assoc by simp
    also have ... = (B.inv (B.𝔯 f') ·B ?μ) ·B B.𝔯 f
      using μ μ-eq B.𝔯'.naturality [of ?μ] by auto
    also have ... = (B.inv (B.𝔯 f') ·B B.𝔯 f') ·B ν ·B (B.inv (B.𝔯 f) ·B B.𝔯 f)
      using μ μ-eq arr-char_SbC arrI comp-simp B.comp-assoc by metis
    also have ... = ν
      using f f' ν 2 B.comp-arr-dom B.comp-cod-arr ide-char_SbC
            B.𝔩.components-are-iso B.𝔩-ide-simp B.comp-inv-arr'
      by auto
    finally show ?thesis by blast
  qed
  thus ?thesis
    using μ by blast
 qed
qed

interpretation bicategory ‹(·)› ‹(⋆)› a i src trg
proof
 show ⋀a. obj a ⟹ «i[a] : a ⋆ a ⇒ a»
 proof (intro in-homI)
  fix a
  assume a: obj a
  have 1: Arr (i a)
    using a obj-def src-def trg-def in-hom-char_SbC B.unit-in-hom
          arr-char_SbC hcomp-def B.obj-def ide-char_SbC objE hcomp-closed
   by (metis (no-types, lifting) B.𝔩-ide-simp B.unitor-coincidence(1) inclusion lunit-closed)
  show 2: arr i[a]
    using 1 arr-char_SbC by simp
  show dom i[a] = a ⋆ a
    using a 2 dom-char_SbC
    by (metis (full-types) B.objI-trg B.unit-simps(4) R.preserves-reflects-arr
        hcomp-def hseq-char' inclusion objE obj-simps(1)
        subcategory.arrE subcategory-axioms trg-def)
  show cod i[a] = a
    using a 2 cod-char_SbC
    by (metis B.obj-def' B.unit-simps(5) inclusion objE obj-simps(1)
        subcategory.arrE subcategory-axioms trg-def)
 qed
 show ⋀a. obj a ⟹ iso (i a)
 proof −
  fix a
  assume a: obj a
  have 1: trg_B a = src_B a
    using a obj-def src-def trg-def B.obj-def arr-char_SbC
    by (metis horizontal-homs.objE horizontal-homs-axioms)
  have 2: Arr (i a)
    using a 1 obj-def src-def trg-def in-hom-char_SbC B.unit-in-hom
          arr-char_SbC hcomp-def B.obj-def ide-char_SbC objE hcomp-closed
   by (metis (no-types, lifting) B.𝔩-ide-simp B.unitor-coincidence(1) inclusion lunit-closed)
```

have *iso* (B.ɪ a)
using *a 2 obj-def B.iso-unit iso-char*$_{SbC}$ *arr-char*$_{SbC}$ *lunit-closed lunit′-closed B.iso-lunit*
apply *simp*
by (*metis* (*no-types*, *lifting*) *B.*ɪ*.components-are-iso B.ide-src inclusion src-def*)
thus *iso* (i a)
using *a 2 obj-def B.iso-unit iso-char*$_{SbC}$ *arr-char*$_{SbC}$ *B.unitor-coincidence*
apply *simp*
by (*metis* (*no-types*, *lifting*) *B.*ɪ*-ide-simp B.ide-src B.obj-src inclusion src-def*)
qed
show $\bigwedge f\ g\ h\ k.$ ⟦ *ide f*; *ide g*; *ide h*; *ide k*;
src f = *trg g*; *src g* = *trg h*; *src h* = *trg k* ⟧ ⟹
(f ⋆ a g h k) · a f (g ⋆ h) k · (a f g h ⋆ k) =
a f g (h ⋆ k) · a (f ⋆ g) h k
using *B.pentagon VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *hcomp-def assoc-closed arr-char*$_{SbC}$ *comp-char*
hcomp-closed comp-closed ide-char$_{SbC}$ *inclusion src-def trg-def*
by *simp*
qed

proposition *is-bicategory*:
shows *bicategory* (·) (⋆) a i *src trg*
..

lemma *obj-char*:
shows *obj a* ⟷ *arr a* ∧ *B.obj a*
proof
assume *a*: *obj a*
show *arr a* ∧ *B.obj a*
using *a obj-def B.obj-def src-def arr-char*$_{SbC}$ *inclusion* **by** *metis*
next
assume *a*: *arr a* ∧ *B.obj a*
have *src a* = *a*
using *a src-def* **by** *auto*
thus *obj a*
using *a obj-def* **by** *simp*
qed

lemma *hcomp-char*:
shows *hcomp* = (λ*f g*. *if arr f* ∧ *arr g* ∧ *src f* = *trg g then f* ⋆$_B$ *g else null*)
using *hcomp-def src-def trg-def* **by** *metis*

lemma *assoc-simp*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows a f g h = a$_B$ f g h
using *assms VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *auto*

lemma *assoc′-simp*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows a′ f g h = B.a′ f g h

```
proof −
  have a′ f g h = B.inv (a_B f g h)
    using assms inv-char_SbC by fastforce
  also have ... = B.a′ f g h
    using assms ide-char_SbC src-def trg-def
          B.VVV.ide-char_SbC B.VVV.arr-char_SbC B.VV.arr-char_SbC
    by force
  finally show ?thesis by blast
qed

lemma lunit-simp:
assumes ide f
shows lunit f = B.lunit f
proof −
  have B.lunit f = lunit f
  proof (intro lunit-eqI)
    show ide f by fact
    show 1: «B.lunit f : trg f ⋆ f ⇒ f»
    proof
      show 2: arr (B.lunit f)
        using assms arr-char_SbC lunit-closed
        by (simp add: arr-char_SbC B.ℓ-ide-simp ide-char_SbC)
      show dom (B.lunit f) = trg f ⋆ f
        using assms 2 dom-char_SbC hcomp-char ide-char_SbC src-trg trg.preserves-arr trg-def
        by auto
      show cod (B.lunit f) = f
        using assms 2 in-hom-char_SbC
        by (simp add: cod-simp ide-char_SbC)
    qed
    show trg f ⋆ B.lunit f = (i[trg f] ⋆ f) · a′ (trg f) (trg f) f
    proof −
      have trg f ⋆ B.lunit f = trg_B f ⋆_B B.lunit f
        using assms 1 arr-char_SbC hcomp-char
        by (metis (no-types, lifting) ideD(1) src-trg trg.preserves-reflects-arr
            trg-def vconn-implies-hpar(2,4))
      also have ... = (i[trg f] ⋆_B f) ·_B B.a′ (trg f) (trg f) f
        using assms ide-char_SbC B.lunit-char(2) trg-def by simp
      also have ... = (i[trg f] ⋆_B f) ·_B a′ (trg f) (trg f) f
        using assms assoc′-simp [of trg f trg f f] by simp
      also have ... = (i[trg f] ⋆ f) ·_B a′ (trg f) (trg f) f
        using assms hcomp-char by simp
      also have ... = (i[trg f] ⋆ f) · a′ (trg f) (trg f) f
        using assms seq-char_SbC [of i[trg f] ⋆ f a′ (trg f) (trg f) f]
              comp-char [of i[trg f] ⋆ f a′ (trg f) (trg f) f]
        by simp
      finally show ?thesis by blast
    qed
  qed
  thus ?thesis by simp
```

```
qed

lemma lunit′-simp:
assumes ide f
shows lunit′ f = B.lunit′ f
  using assms inv-char_SbC [of lunit f] lunit-simp by fastforce

lemma runit-simp:
assumes ide f
shows runit f = B.runit f
proof −
  have B.runit f = runit f
  proof (intro runit-eqI)
    show ide f by fact
    show 1: «B.runit f : f ⋆ src f ⇒ f»
    proof
      show 2: arr (B.runit f)
        using assms arr-char_SbC runit-closed
        by (simp add: arr-char_SbC B.𝔯-ide-simp ide-char_SbC)
      show dom (B.runit f) = f ⋆ src f
        using assms 2 dom-char_SbC hcomp-char
        by (metis (no-types, lifting) B.runit-simps(4) ide-char_SbC src.preserves-reflects-arr
            src-def trg-src)
      show cod (B.runit f) = f
        using assms 2 in-hom-char_SbC
        by (simp add: cod-simp ide-char_SbC)
    qed
    show B.runit f ⋆ src f = (f ⋆ i[src f]) · a f (src f) (src f)
    proof −
      have B.runit f ⋆ src f = B.runit f ⋆_B src_B f
        using assms 1 arr-char_SbC hcomp-char
        by (metis (no-types, lifting) ideD(1) src.preserves-reflects-arr src-def
            trg-src vconn-implies-hpar(1,3))
      also have ... = (f ⋆_B i[src f]) ·_B a_B f (src f) (src f)
        using assms ide-char_SbC B.runit-char(2) src-def by simp
      also have ... = (f ⋆_B i[src f]) ·_B a f (src f) (src f)
        using assms assoc-simp by simp
      also have ... = (f ⋆ i[src f]) ·_B a f (src f) (src f)
        using assms 1 hcomp-char by simp
      also have ... = (f ⋆ i[src f]) · a f (src f) (src f)
      proof −
        have B.seq (f ⋆ i[src f]) (a f (src f) (src f))
          using assms seq-char_SbC [of f ⋆ i[src f] a f (src f) (src f)] by simp
        thus ?thesis
          using assms comp-char [of f ⋆ i[src f] a f (src f) (src f)] by simp
      qed
      finally show ?thesis by blast
    qed
  qed
```

thus *?thesis* **by** *simp*
qed

lemma *runit′-simp*:
assumes *ide f*
shows *runit′ f* = *B.runit′ f*
using *assms inv-char*$_{SbC}$ [*of runit f*] *runit-simp* **by** *fastforce*

lemma *comp-eqI* [*intro*]:
assumes *seq f g* **and** $f = f'$ **and** $g = g'$
shows $f \cdot g = f' \cdot_B g'$
using *assms comp-char ext ext not-arr-null* **by** *auto*

lemma *comp-eqI′* [*intro*]:
assumes *seq f g* **and** $f = f'$ **and** $g = g'$
shows $f \cdot_B g = f' \cdot g'$
using *assms comp-char ext ext not-arr-null* **by** *auto*

lemma *hcomp-eqI* [*intro*]:
assumes *hseq f g* **and** $f = f'$ **and** $g = g'$
shows $f \star g = f' \star_B g'$
using *assms hcomp-char not-arr-null* **by** *auto*

lemma *hcomp-eqI′* [*intro*]:
assumes *hseq f g* **and** $f = f'$ **and** $g = g'$
shows $f \star_B g = f' \star g'$
using *assms hcomp-char not-arr-null* **by** *auto*

lemma *arr-compI*:
assumes *seq f g*
shows *arr* $(f \cdot_B g)$
using *assms seq-char*$_{SbC}$ *dom-char*$_{SbC}$ *cod-char*$_{SbC}$
by (*metis* (*no-types*, *lifting*) *comp-simp*)

lemma *arr-hcompI*:
assumes *hseq f g*
shows *arr* $(f \star_B g)$
using *assms hseq-char src-def trg-def hcomp-eqI′* **by** *auto*

end

sublocale *subbicategory* ⊆ *bicategory* ‹(·)› ‹(⋆)› a i *src trg*
using *is-bicategory* **by** *auto*

1.8.2 The Sub-bicategory of Endo-arrows of an Object

We now consider the sub-bicategory consisting of all arrows having the same object a both as their source and their target and we show that the resulting structure is a monoidal category. We actually prove a slightly more general result, in which the unit

of the monoidal category is taken to be an arbitrary isomorphism «ω : $w \star_B w \Rightarrow w$» with w isomorphic to a, rather than the particular choice «i[a] : $a \star_B a \Rightarrow a$» made by the ambient bicategory.

locale *subbicategory-at-object* =
  B: *bicategory V H* a_B i src_B trg_B +
  subbicategory V H a_B i src_B trg_B ‹$\lambda\mu$. *B.arr* $\mu \wedge src_B\ \mu = a \wedge trg_B\ \mu = a$›
for *V* :: *'a comp* (**infixr** ‹$\cdot_B$› *55*)
and *H* :: *'a comp* (**infixr** ‹$\star_B$› *55*)
and a_B :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a_B[-, -, -]›)
and i :: *'a* ⇒ *'a* (‹i[-]›)
and src_B :: *'a* ⇒ *'a*
and trg_B :: *'a* ⇒ *'a*
and *a* :: *'a*
and *w* :: *'a*
and ω :: *'a* +
assumes *obj-a*: *B.obj a*
and *isomorphic-a-w*: *B.isomorphic a w*
and *ω-in-vhom*: «ω : $w \star_B w \Rightarrow w$»
and *ω-is-iso*: *B.iso* ω
begin

  notation *hcomp* (**infixr** ‹$\star$› *53*)

  lemma *arr-simps*:
  assumes *arr* μ
  shows *src* $\mu = a$ **and** *trg* $\mu = a$
    apply (*metis* (*no-types*, *lifting*) *arrE assms src-def*)
    by (*metis* (*no-types*, *lifting*) *arrE assms trg-def*)

  lemma *ω-simps* [*simp*]:
  shows *arr* ω
  and *src* $\omega = a$ **and** *trg* $\omega = a$
  and *dom* $\omega = w \star_B w$ **and** *cod* $\omega = w$
    using *isomorphic-a-w ω-in-vhom in-hom-char$_{SbC}$ arr-simps* **by** *auto*

  lemma *ide-w*:
  shows *B.ide w*
    using *isomorphic-a-w B.isomorphic-def* **by** *auto*

  lemma *w-simps* [*simp*]:
  shows *ide w* **and** *B.ide w*
  and *src w* = *a* **and** *trg w* = *a* **and** $src_B\ w = a$ **and** $trg_B\ w = a$
  and *dom w* = *w* **and** *cod w* = *w*
  proof −
    show *w*: *ide w*
      using *ω-in-vhom ide-cod* **by** *blast*
    show *B.ide w*
      using *w ide-char$_{SbC}$* **by** *simp*
    obtain φ **where** φ: «φ : $a \Rightarrow_B w$» $\wedge$ *B.iso* φ

```
    using isomorphic-a-w B.isomorphic-def by auto
  show src_B w = a
    using obj-a w φ B.src-cod by force
  show trg_B w = a
    using obj-a w φ B.src-cod by force
  show src w = a
    using ‹src_B w = a› w ide-w src-def by simp
  show trg w = a
    using ‹src_B w = a› w ide-w trg-def
    by (simp add: ‹trg_B w = a›)
  show dom w = w
    using w by simp
  show cod w = w
    using w by simp
qed

lemma VxV-arr-eq-VV-arr:
shows VxV.arr f ⟷ VV.arr f
  using inclusion VxV.arr-char VV.arr-char_SbC arr-char_SbC src-def trg-def
  by auto

lemma VxV-comp-eq-VV-comp:
shows VxV.comp = VV.comp
proof −
  have ⋀f g. VxV.comp f g = VV.comp f g
  proof −
    fix f g
    show VxV.comp f g = VV.comp f g
      unfolding VV.comp-def
      using VxV.comp-char arr-simps(1) arr-simps(2)
      apply (cases seq (fst f) (fst g), cases seq (snd f) (snd g))
      by (elim seqE) auto
  qed
  thus ?thesis by blast
qed

lemma VxVxV-arr-eq-VVV-arr:
shows VxVxV.arr f ⟷ VVV.arr f
  using VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def inclusion arr-char_SbC
  by auto

lemma VxVxV-comp-eq-VVV-comp:
shows VxVxV.comp = VVV.comp
proof −
  have ⋀f g. VxVxV.comp f g = VVV.comp f g
  proof −
    fix f g
    show VxVxV.comp f g = VVV.comp f g
    proof (cases VxVxV.seq f g)
```

assume *1*: $\neg$ *VxVxV.seq f g*
have *VxVxV.comp f g* = *VxVxV.null*
using *1 VxVxV.ext* **by** *blast*
also have ... = (*null*, *null*, *null*)
using *VxVxV.null-char VxV.null-char* **by** *simp*
also have ... = *VVV.null*
using *VVV.null-char VV.null-char* **by** *simp*
also have ... = *VVV.comp f g*
proof −
have $\neg$ *VVV.seq f g*
using *1* $VVV.seq\text{-}char_{SbC}$ **by** *blast*
thus *?thesis*
by (*metis* (*no-types*, *lifting*) *VVV.ext*)
qed
finally show *?thesis* **by** *simp*
next
assume *1*: *VxVxV.seq f g*
have *2*: *B.arr* (*fst f*) $\wedge$ *B.arr* (*fst* (*snd f*)) $\wedge$ *B.arr* (*snd* (*snd f*)) $\wedge$
src_B (*fst f*) = *a* $\wedge$ src_B (*fst* (*snd f*)) = *a* $\wedge$ src_B (*snd* (*snd f*)) = *a* $\wedge$
trg_B (*fst f*) = *a* $\wedge$ trg_B (*fst* (*snd f*)) = *a* $\wedge$ trg_B (*snd* (*snd f*)) = *a*
using *1 VxVxV.seq-char VxV.seq-char* $arr\text{-}char_{SbC}$ **by** *blast*
have *3*: *B.arr* (*fst g*) $\wedge$ *B.arr* (*fst* (*snd g*)) $\wedge$ *B.arr* (*snd* (*snd g*)) $\wedge$
src_B (*fst g*) = *a* $\wedge$ src_B (*fst* (*snd g*)) = *a* $\wedge$ src_B (*snd* (*snd g*)) = *a* $\wedge$
trg_B (*fst g*) = *a* $\wedge$ trg_B (*fst* (*snd g*)) = *a* $\wedge$ trg_B (*snd* (*snd g*)) = *a*
using *1 VxVxV.seq-char VxV.seq-char* $arr\text{-}char_{SbC}$ **by** *blast*
have *4*: *B.seq* (*fst f*) (*fst g*) $\wedge$ *B.seq* (*fst* (*snd f*)) (*fst* (*snd g*)) $\wedge$
B.seq (*snd* (*snd f*)) (*snd* (*snd g*))
using *1 VxVxV.seq-char VxV.seq-char* $seq\text{-}char_{SbC}$ **by** *blast*
have *5*: *VxVxV.comp f g* =
(*fst f* · *fst g*, *fst* (*snd f*) · *fst* (*snd g*), *snd* (*snd f*) · *snd* (*snd g*))
using *1 2 3 4* $VxVxV.seqE_{PC}$ *VxVxV.comp-char VxV.comp-char* $seq\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*))
also have ... = *VVV.comp f g*
using *1 VVV.comp-char* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
apply *simp*
using *2 3 5* $arrI_{SbC}$ *arr-simps*(*1*) *arr-simps*(*2*) **by** *presburger*
finally show *?thesis* **by** *blast*
qed
qed
thus *?thesis* **by** *blast*
qed

interpretation *H*: *functor VxV.comp* ‹(·)› ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$›
using *H.functor-axioms hcomp-def VxV-comp-eq-VV-comp* **by** *simp*

interpretation *H*: *binary-endofunctor* ‹(·)› ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$› **..**

lemma *HoHV-eq-ToTC*:

shows *HoHV* = *H.ToTC*
using *HoHV-def H.ToTC-def VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *src-def trg-def inclusion arr-char*$_{SbC}$
by *auto*

lemma *HoVH-eq-ToCT*:
shows *HoVH* = *H.ToCT*
using *HoVH-def H.ToCT-def VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *src-def trg-def inclusion arr-char*$_{SbC}$
by *auto*

interpretation *ToTC*: *functor VxVxV.comp* ‹(·)› *H.ToTC*
using *HoHV-eq-ToTC VxVxV-comp-eq-VVV-comp HoHV.functor-axioms* **by** *simp*
interpretation *ToCT*: *functor VxVxV.comp* ‹(·)› *H.ToCT*
using *HoVH-eq-ToCT VxVxV-comp-eq-VVV-comp HoVH.functor-axioms* **by** *simp*

interpretation α: *natural-isomorphism VxVxV.comp* ‹(·)› *H.ToTC H.ToCT* α
unfolding α*-def*
using α*.natural-isomorphism-axioms HoHV-eq-ToTC HoVH-eq-ToCT* α*-def VxVxV-comp-eq-VVV-comp*
by *simp*

interpretation *L*: *endofunctor* ‹(·)› ‹λf. *fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)›
proof
fix *f*
show ¬ *arr f* ⟹ *fst* (*w*, *f*) ⋆ *snd* (*w*, *f*) = *null*
using *arr-char*$_{SbC}$ *hcomp-def* **by** *auto*
assume *f*: *arr f*
show *hseq* (*fst* (*w*, *f*)) (*snd* (*w*, *f*))
using *f hseq-char arr-char*$_{SbC}$ *src-def trg-def* ω*-in-vhom cod-char*$_{SbC}$ **by** *simp*
show *dom* (*fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)) = *fst* (*w*, *dom f*) ⋆ *snd* (*w*, *dom f*)
using *f arr-char*$_{SbC}$ *hcomp-def dom-simp* **by** *simp*
show *cod* (*fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)) = *fst* (*w*, *cod f*) ⋆ *snd* (*w*, *cod f*)
using *f arr-char*$_{SbC}$ *hcomp-def cod-simp* **by** *simp*
next
fix *f g*
assume *fg*: *seq g f*
show *fst* (*w*, *g* · *f*) ⋆ *snd* (*w*, *g* · *f*) = (*fst* (*w*, *g*) ⋆ *snd* (*w*, *g*)) · (*fst* (*w*, *f*) ⋆ *snd* (*w*, *f*))
by (*simp add*: *fg whisker-left*)
qed

interpretation *L′*: *equivalence-functor* ‹(·)› ‹(·)› ‹λf. *fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)›
proof −
obtain φ **where** φ: «φ : *w* $\Rightarrow_B$ *a*» ∧ *B.iso* φ
using *isomorphic-a-w B.isomorphic-symmetric* **by** *force*
have «φ : *w* ⇒ *a*»
using φ *in-hom-char*$_{SbC}$
by (*metis* (*no-types*, *lifting*) *B.in-homE B.src-cod B.src-src B.trg-cod B.trg-trg* ω*-in-vhom arr-char*$_{SbC}$ *arr-cod cod-simp*)

hence φ: «$\varphi : w \Rightarrow_B a$» $\wedge$ *B.iso* φ $\wedge$ «$\varphi : w \Rightarrow a$» $\wedge$ *iso* φ
  using φ *iso-char*$_{SbC}$ *arr-char*$_{SbC}$ **by** *auto*
interpret l: *natural-isomorphism* ‹$(\cdot)$› ‹$(\cdot)$›
    ‹$\lambda f.$ *fst* $(w, f) \star$ *snd* (w, f)› *map* ‹$\lambda f.$ ⦇ $f \cdot (\varphi \star$ *dom* $f)$›
proof
  fix μ
  show $\neg$ *arr* $\mu \Longrightarrow$ ⦇ $\mu \cdot (\varphi \star$ *dom* $\mu) =$ *null*
    using φ *arr-char*$_{SbC}$ *dom-char*$_{SbC}$ *ext*
    apply *simp*
    using *null-is-zero*(*2*) *hcomp-def* **by** *fastforce*
  assume μ: *arr* μ
  have *0*: *in-hhom* (*dom* μ) *a a*
    using μ *arr-char*$_{SbC}$ *src-dom trg-dom src-def trg-def dom-simp* **by** *simp*
  have *1*: *in-hhom* φ *a a*
    using φ *arr-char*$_{SbC}$ *src-dom trg-dom src-def trg-def* **by** *auto*
  have *2*: *hseq* φ (*B.dom* μ)
    using μ *0 1 dom-simp* **by** (*intro hseqI*) *auto*
  have *3*: *seq* (⦇ μ) ($\varphi \star$ *dom* μ)
  proof (*intro seqI'*)
    show «$\varphi \star$ *dom* $\mu : w \star$ *dom* $\mu \Rightarrow a \star$ *dom* μ»
      by (*metis* (*no-types*, *lifting*) *0* μ φ *hcomp-in-vhom ide-dom ide-in-hom*(*2*)
        in-hhom-def w-simps(*3*))
    show «⦇ $\mu : a \star$ *dom* $\mu \Rightarrow$ *cod* μ»
      using μ *2* ⦇*.*preserves-hom* [*of* μ *dom* μ *cod* μ] *arr-simps*(*2*) *arr-cod* **by** *fastforce*
  qed
  show *arr* (⦇ $\mu \cdot (\varphi \star$ *dom* $\mu)$)
  using *3* **by** *blast*
  show *map* $\mu \cdot$ ⦇ (*dom* μ) $\cdot$ ($\varphi \star$ *dom* (*dom* μ)) = ⦇ $\mu \cdot (\varphi \star$ *dom* μ)
  proof −
    have *map* $\mu \cdot$ ⦇ (*dom* μ) $\cdot$ ($\varphi \star$ *dom* (*dom* μ)) = (*map* $\mu \cdot$ ⦇ (*dom* μ)) $\cdot$ ($\varphi \star$ *dom* μ)
      using μ *comp-assoc* **by** *simp*
    also have ... = ⦇ $\mu \cdot (\varphi \star$ *dom* $\mu)$
      using μ φ ⦇*.naturality1* **by** *auto*
    finally show *?thesis* **by** *blast*
  qed
  show (⦇ (*cod* μ) $\cdot$ ($\varphi \star$ *dom* (*cod* μ))) $\cdot$ (*fst* $(w, \mu) \star$ *snd* (w, μ)) = ⦇ $\mu \cdot (\varphi \star$ *dom* $\mu)$
  proof −
    have (⦇ (*cod* μ) $\cdot$ ($\varphi \star$ *dom* (*cod* μ))) $\cdot$ (*fst* $(w, \mu) \star$ *snd* (w, μ)) =
        (⦇ (*cod* μ) $\cdot$ ($\varphi \star$ *B.cod* μ)) $\cdot$ ($w \star \mu$)
      using μ φ *dom-char*$_{SbC}$ *arr-char*$_{SbC}$ ω*-in-vhom cod-simp* **by** *simp*
    also have ... = ⦇ (*cod* μ) $\cdot$ ($\varphi \cdot w \star$ *B.cod* $\mu \cdot \mu$)
    proof −
      have *seq* φ w
        using φ ω*-in-vhom w-simps*(*1*) **by** *blast*
      moreover have *2*: *seq* (*B.cod* μ) μ
        using μ *seq-char*$_{SbC}$ *cod-simp* **by** (*simp add*: *comp-cod-arr*)
      moreover have *src* φ = *trg* (*B.cod* μ)
        using μ φ *2*
        by (*metis* (*no-types*, *lifting*) *arr-simps*(*2*) *seqE vconn-implies-hpar*(*1*) *w-simps*(*3*))

```
    ultimately show ?thesis
      using interchange comp-assoc by simp
  qed
  also have ... = ι (cod μ) · (φ ⋆ μ)
    using μ φ ω-in-vhom comp-arr-dom comp-cod-arr cod-simp
    apply (elim conjE in-homE) by auto
  also have ... = (ι (cod μ) · (cod φ ⋆ μ)) · (φ ⋆ dom μ)
  proof −
    have 1: seq (cod φ) φ
      using φ arr-cod-iff-arr dom-cod iso-is-arr seqI by presburger
    moreover have 2: seq μ (dom μ)
      using μ by (simp add: comp-arr-dom)
    moreover have src (cod φ) = trg μ
      using μ φ arr-cod arr-simps(1−2) iso-is-arr by auto
    ultimately show ?thesis
      using 1 2 interchange [of cod φ φ μ dom μ] comp-arr-dom comp-cod-arr
            comp-assoc by fastforce
  qed
  also have ... = ι μ · (φ ⋆ dom μ)
  proof −
    have L μ = cod φ ⋆ μ
      using μ φ arr-simps(2) in-homE by auto
    hence ι (cod μ) · (cod φ ⋆ μ) = ι μ
      using μ ι.naturality2 [of μ] by simp
    thus ?thesis by simp
  qed
  finally show ?thesis by simp
qed
next
show ⋀f. ide f ⟹ iso (ι f · (φ ⋆ dom f))
proof −
  fix f
  assume f: ide f
  have iso (ι f)
    using f iso-lunit by simp
  moreover have iso (φ ⋆ dom f)
    using φ f src-def trg-def ide-char_SbC arr-char_SbC
    apply (intro iso-hcomp, simp-all)
    by (metis (no-types, lifting) in-homE)
  moreover have seq (ι f) (φ ⋆ dom f)
  proof (intro seqI′)
    show «ι f : a ⋆ f ⇒ f»
      using f lunit-in-hom(2) ι-ide-simp ide-char_SbC arr-char_SbC trg-def by simp
    show «φ ⋆ dom f : w ⋆ f ⇒ a ⋆ f»
      using φ f ide-char_SbC arr-char_SbC hcomp-def src-def trg-def obj-a ide-in-hom
            in-hom-char_SbC
      by (intro hcomp-in-vhom, auto)
  qed
  ultimately show iso (ι f · (φ ⋆ dom f))
```

using *isos-compose* **by** *simp*
qed
qed
show *equivalence-functor* $(\cdot)$ $(\cdot)$ $(\lambda f.\ fst\ (w,\ f) \star snd\ (w,\ f))$
using l.*natural-isomorphism-axioms* *L.isomorphic-to-identity-is-equivalence* **by** *simp*
qed
interpretation *L*: *equivalence-functor* ‹$(\cdot)$› ‹$(\cdot)$› ‹$\lambda f.\ fst\ (cod\ \omega,\ f) \star snd\ (cod\ \omega,\ f)$›
proof −
have $(\lambda f.\ fst\ (cod\ \omega,\ f) \star snd\ (cod\ \omega,\ f)) = (\lambda f.\ fst\ (w,\ f) \star snd\ (w,\ f))$
using *ω-in-vhom* **by** *simp*
thus *equivalence-functor* $(\cdot)$ $(\cdot)$ $(\lambda f.\ fst\ (cod\ \omega,\ f) \star snd\ (cod\ \omega,\ f))$
using *L′.equivalence-functor-axioms* **by** *simp*
qed

interpretation *R*: *endofunctor* ‹$(\cdot)$› ‹$\lambda f.\ fst\ (f,\ w) \star snd\ (f,\ w)$›
proof
fix *f*
show $\neg\ arr\ f \Longrightarrow fst\ (f,\ w) \star snd\ (f,\ w) = null$
using *arr-char*$_{SbC}$ *hcomp-def* **by** *auto*
assume *f*: *arr f*
show *hseq* (*fst* (*f*, *w*)) (*snd* (*f*, *w*))
using *f hseq-char arr-char*$_{SbC}$ *src-def trg-def ω-in-vhom cod-char*$_{SbC}$ *isomorphic-a-w*
B.isomorphic-def in-hom-char$_{SbC}$
by *simp*
show *dom* (*fst* (*f*, *w*) ⋆ *snd* (*f*, *w*)) = *fst* (*dom f*, *w*) ⋆ *snd* (*dom f*, *w*)
using *f arr-char*$_{SbC}$ *dom-char*$_{SbC}$ *cod-char*$_{SbC}$ *hcomp-def ω-in-vhom* **by** *simp*
show *cod* (*fst* (*f*, *w*) ⋆ *snd* (*f*, *w*)) = *fst* (*cod f*, *w*) ⋆ *snd* (*cod f*, *w*)
using *f arr-char*$_{SbC}$ *dom-char*$_{SbC}$ *cod-char*$_{SbC}$ *hcomp-def ω-in-vhom* **by** *simp*
next
fix *f g*
assume *fg*: *seq g f*
have *1*: $a \cdot_B a = a$
using *obj-a* **by** *auto*
show *fst* (*g* · *f*, *w*) ⋆ *snd* (*g* · *f*, *w*) = (*fst* (*g*, *w*) ⋆ *snd* (*g*, *w*)) · (*fst* (*f*, *w*) ⋆ *snd* (*f*, *w*))
by (*simp add*: *fg whisker-right*)
qed

interpretation *R′*: *equivalence-functor* ‹$(\cdot)$› ‹$(\cdot)$› ‹$\lambda f.\ fst\ (f,\ w) \star snd\ (f,\ w)$›
proof −
obtain φ **where** φ: «$\varphi : w \Rightarrow_B a$» $\wedge$ *B.iso* φ
using *isomorphic-a-w B.isomorphic-symmetric* **by** *force*
have «$\varphi : w \Rightarrow a$»
using φ *in-hom-char*$_{SbC}$
by (*metis* (*no-types*, *lifting*) *B.in-homE B.src-cod B.src-src B.trg-cod B.trg-trg*
ω-in-vhom arr-char$_{SbC}$ *arr-cod cod-simp*)
hence φ: «$\varphi : w \Rightarrow_B a$» $\wedge$ *B.iso* φ $\wedge$ «$\varphi : w \Rightarrow a$» $\wedge$ *iso* φ
using φ *iso-char*$_{SbC}$ *arr-char*$_{SbC}$ **by** *auto*
interpret r: *natural-isomorphism comp comp*
‹$\lambda f.\ fst\ (f,\ w) \star snd\ (f,\ w)$› *map* ‹$\lambda f.\ \mathfrak{r}\ f \cdot (dom\ f \star \varphi)$›

```
proof
  fix μ
  show ¬ arr μ ⟹ 𝔯 μ · (dom μ ⋆ φ) = null
    using φ arr-char_SbC dom-char_SbC ext
    apply simp
    using null-is-zero(2) hcomp-def by fastforce
  assume μ: arr μ
  have 0: in-hhom (dom μ) a a
    using μ arr-char_SbC src-dom trg-dom src-def trg-def dom-simp by simp
  have 1: in-hhom φ a a
    using φ arr-char_SbC src-dom trg-dom src-def trg-def by auto
  have 2: hseq (B.dom μ) φ
    using μ 0 1 dom-simp hseqI by auto
  have 3: seq (𝔯 μ) (dom μ ⋆ φ)
  proof (intro seqI')
    show «dom μ ⋆ φ : dom μ ⋆ w ⇒ dom μ ⋆ a»
      by (metis (no-types, lifting) 0 1 μ φ hcomp-in-vhom hseqI hseq-char
          ide-dom ide-in-hom(2) vconn-implies-hpar(2))
    show «𝔯 μ : dom μ ⋆ a ⇒ cod μ»
      using μ 2 𝔯.preserves-hom [of μ dom μ cod μ] arr-simps(2) arr-cod
            dom-simp cod-simp
      by fastforce
  qed
  show arr (𝔯 μ · (dom μ ⋆ φ))
    using 3 by blast
  show map μ · 𝔯 (dom μ) · (dom (dom μ) ⋆ φ) = 𝔯 μ · (dom μ ⋆ φ)
  proof −
    have map μ · 𝔯 (dom μ) · (dom (dom μ) ⋆ φ) =
          (map μ · 𝔯 (dom μ)) · (dom (dom μ) ⋆ φ)
      using comp-assoc by simp
    also have ... = (map μ · 𝔯 (dom μ)) · (dom μ ⋆ φ)
      using μ dom-dom by simp
    also have ... = 𝔯 μ · (dom μ ⋆ φ)
      using μ φ 𝔯.naturality1 by auto
    finally show ?thesis by blast
  qed
  show (𝔯 (cod μ) · (dom (cod μ) ⋆ φ)) · (fst (μ, w) ⋆ snd (μ, w)) = 𝔯 μ · (dom μ ⋆ φ)
  proof −
    have (𝔯 (cod μ) · (dom (cod μ) ⋆ φ)) · (fst (μ, w) ⋆ snd (μ, w)) =
          (𝔯 (cod μ) · (B.cod μ ⋆ φ)) · (μ ⋆ w)
      using μ φ dom-char_SbC arr-char_SbC ω-in-vhom cod-simp by simp
    also have ... = 𝔯 (cod μ) · (B.cod μ · μ ⋆ φ · w)
    proof −
      have 2: seq φ w
        using φ ω-in-vhom w-simps(1) by blast
      moreover have seq (B.cod μ) μ
        using μ seq-char_SbC cod-simp by (simp add: comp-cod-arr)
      moreover have src (B.cod μ) = trg φ
        using μ φ 2
```

```
      using arr-simps(1) calculation(2) seq-char_SbC vconn-implies-hpar(2) by force
    ultimately show ?thesis
      using interchange comp-assoc by simp
  qed
  also have ... = 𝔯 (cod μ) · (μ ⋆ φ)
    using μ φ ω-in-vhom comp-arr-dom comp-cod-arr cod-simp
    apply (elim conjE in-homE) by auto
  also have ... = (𝔯 (cod μ) · (μ ⋆ cod φ)) · (dom μ ⋆ φ)
  proof −
    have (μ ⋆ cod φ) · (dom μ ⋆ φ) = μ ⋆ φ
    proof −
      have seq μ (dom μ)
        using μ by (simp add: comp-arr-dom)
      moreover have seq (cod φ) φ
        using φ iso-is-arr arr-cod dom-cod by auto
      moreover have src μ = trg (cod φ)
        using μ φ 2
        by (metis (no-types, lifting) arr-simps(1) arr-simps(2) calculation(2) seqE)
      ultimately show ?thesis
        using μ φ iso-is-arr comp-arr-dom comp-cod-arr
              interchange [of μ dom μ cod φ φ]
        by simp
    qed
    thus ?thesis
      using comp-assoc by simp
  qed
  also have ... = 𝔯 μ · (dom μ ⋆ φ)
  proof −
    have μ ⋆ cod φ = R μ
      using μ φ arr-simps(1) in-homE by auto
    hence 𝔯 (cod μ) · (μ ⋆ cod φ) = 𝔯 μ
      using μ φ 𝔯.naturality2 by simp
    thus ?thesis by simp
  qed
  finally show ?thesis by simp
qed
next
show ⋀f. ide f ⟹ iso (𝔯 f · (dom f ⋆ φ))
proof −
  fix f
  assume f: ide f
  have 1: iso (𝔯 f)
    using f iso-lunit by simp
  moreover have 2: iso (dom f ⋆ φ)
    using φ f src-def trg-def ide-char_SbC arr-char_SbC
    apply (intro iso-hcomp, simp-all)
    by (metis (no-types, lifting) in-homE)
  moreover have seq (𝔯 f) (dom f ⋆ φ)
  proof (intro seqI′)
```

```
      show «𝔯 f : f ⋆ a ⇒ f»
        using f runit-in-hom(2) 𝔯-ide-simp ide-char_SbC arr-char_SbC src-def by simp
      show «dom f ⋆ φ : f ⋆ w ⇒ f ⋆ a»
        using φ f ide-char_SbC arr-char_SbC hcomp-def src-def trg-def obj-a ide-in-hom
              in-hom-char_SbC
        by (intro hcomp-in-vhom, auto)
    qed
    ultimately show iso (𝔯 f · (dom f ⋆ φ))
      using isos-compose by blast
  qed
qed
show equivalence-functor (·) (·) (λf. fst (f, w) ⋆ snd (f, w))
  using 𝔯.natural-isomorphism-axioms R.isomorphic-to-identity-is-equivalence by simp
qed
interpretation R: equivalence-functor ‹(·)› ‹(·)› ‹λf. fst (f, cod ω) ⋆ snd (f, cod ω)›
proof −
  have (λf. fst (f, cod ω) ⋆ snd (f, cod ω)) = (λf. fst (f, w) ⋆ snd (f, w))
    using ω-in-vhom by simp
  thus equivalence-functor (·) (·) (λf. fst (f, cod ω) ⋆ snd (f, cod ω))
    using R′.equivalence-functor-axioms by simp
qed

interpretation M: monoidal-category ‹(·)› ‹λμν. fst μν ⋆ snd μν› α ω
proof
  show «ω : fst (cod ω, cod ω) ⋆ snd (cod ω, cod ω) ⇒ cod ω»
    using ω-in-vhom hcomp-def arr-char_SbC by auto
  show iso ω
    using ω-is-iso iso-char_SbC arr-char_SbC inv-char_SbC ω-in-vhom by auto
  show ⋀f g h k. ⟦ ide f; ide g; ide h; ide k ⟧ ⟹
                 (fst (f, α (g, h, k)) ⋆ snd (f, α (g, h, k))) ·
                   α (f, hcomp (fst (g, h)) (snd (g, h)), k) ·
                   (fst (α (f, g, h), k) ⋆ snd (α (f, g, h), k)) =
                 α (f, g, fst (h, k) ⋆ snd (h, k)) · α (fst (f, g) ⋆ snd (f, g), h, k)
  proof −
    fix f g h k
    assume f: ide f and g: ide g and h: ide h and k: ide k
    have 1: VVV.arr (f, g, h) ∧ VVV.arr (g, h, k)
     using f g h k VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def ide-char_SbC arr-char_SbC
      by simp
    have 2: VVV.arr (f, g ⋆ h, k)
     using f g h k src-def trg-def ide-char_SbC arr-char_SbC VxVxV-comp-eq-VVV-comp hseqI′
      by auto
    have 3: VVV.arr (f, g, h ⋆ k)
    using f g h k 1 VVV.arr-char_SbC VV.arr-char_SbC src-def trg-def ide-char_SbC arr-char_SbC
          VxV.arrI VxVxV.arrI VxVxV-comp-eq-VVV-comp H.preserves-reflects-arr hseqI′
      by auto
    have 4: VVV.arr (f ⋆ g, h, k)
    using f g h k src-def trg-def ide-char_SbC arr-char_SbC hseq-char VxVxV-comp-eq-VVV-comp
      by auto
```

have (*fst* (f, α (g, h, k)) $\star$ *snd* (f, α (g, h, k))) $\cdot$
α (f, *fst* (g, h) $\star$ *snd* (g, h), k) $\cdot$
(*fst* (α (f, g, h), k) $\star$ *snd* (α (f, g, h), k)) =
($f \star \mathrm{a}_B[g, h, k]$) $\cdot$ $\mathrm{a}_B[f, g \star h, k]$ $\cdot$ ($\mathrm{a}_B[f, g, h] \star k$)
unfolding *α-def* **by** (*simp add*: *1 2*)
also have ... = ($f \star_B \mathrm{a}_B\ g\ h\ k$) $\cdot$ $\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot$ ($\mathrm{a}_B\ f\ g\ h \star_B k$)
unfolding *hcomp-def*
using *f g h k src-def trg-def* $arr\text{-}char_{SbC}$
using *assoc-closed* $ide\text{-}char_{SbC}$ **by** *auto*
also have ... = ($f \star_B \mathrm{a}_B\ g\ h\ k$) $\cdot_B$ $\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot_B$ ($\mathrm{a}_B\ f\ g\ h \star_B k$)
proof −
have *arr* ($f \star_B \mathrm{a}_B\ g\ h\ k$)
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k* **by** *simp*
moreover have *arr* ($\mathrm{a}_B\ f$ ($g \star_B h$) k)
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k* **by** *simp*
moreover have *arr* ($\mathrm{a}_B\ f\ g\ h \star_B k$)
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k* **by** *simp*
moreover have *arr* ($\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot_B$ ($\mathrm{a}_B\ f\ g\ h \star_B k$))
unfolding $arr\text{-}char_{SbC}$
apply (*intro conjI*)
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k B.HoHV-def B.HoVH-def*
apply (*intro B.seqI*)
apply *simp-all*
proof −
have *1*: *B.arr* ($\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot_B$ $\mathrm{a}_B\ f\ g\ h \star_B k$)
using *f g h k* $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *B.HoHV-def B.HoVH-def*
apply (*intro B.seqI*)
by *auto*
show src_B ($\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot_B$ $\mathrm{a}_B\ f\ g\ h \star_B k$) = a
using *1 f g h k* $arr\text{-}char_{SbC}$ *B.src-vcomp B.vseq-implies-hpar*(*1*) **by** *fastforce*
show trg_B ($\mathrm{a}_B\ f$ ($g \star_B h$) k $\cdot_B$ $\mathrm{a}_B\ f\ g\ h \star_B k$) = a
using *1* $arr\text{-}char_{SbC}$ *calculation*(*2*−*3*) **by** *auto*
qed
ultimately show *?thesis*
using *B.ext comp-char* **by** (*metis* (*no-types*, *lifting*))
qed
also have ... = $\mathrm{a}_B\ f\ g$ ($h \star_B k$) $\cdot_B$ a_B ($f \star_B g$) h k
using *f g h k src-def trg-def* $arr\text{-}char_{SbC}$ $ide\text{-}char_{SbC}$ *B.pentagon*
using *4 α-def hcomp-def* **by** *auto*
also have ... = $\mathrm{a}_B\ f\ g$ ($h \star_B k$) $\cdot$ a_B ($f \star_B g$) h k
proof −
have *arr* (a_B ($f \star_B g$) h k)
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k* **by** *simp*
moreover have *arr* ($\mathrm{a}_B\ f\ g$ ($h \star_B k$))
using $ide\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ *assoc-closed f g h hcomp-closed k* **by** *fastforce*
ultimately show *?thesis*
using *B.ext comp-char* **by** *auto*
qed

also have ... $= \mathrm{a}_B[f,\ g,\ fst\ (h,\ k) \star snd\ (h,\ k)] \cdot \mathrm{a}_B[fst\ (f,\ g) \star snd\ (f,\ g),\ h,\ k]$
unfolding *hcomp-def*
using *f g h k src-def trg-def* $arr\text{-}char_{SbC}$ $ide\text{-}char_{SbC}$ **by** *simp*
also have ... $= \alpha\ (f,\ g,\ fst\ (h,\ k) \star snd\ (h,\ k)) \cdot \alpha\ (fst\ (f,\ g) \star snd\ (f,\ g),\ h,\ k)$
unfolding *α-def* **using** *1 2 3 4* **by** *simp*
finally show $(fst\ (f,\ \alpha\ (g,\ h,\ k)) \star snd\ (f,\ \alpha\ (g,\ h,\ k))) \cdot$
$\alpha\ (f,\ fst\ (g,\ h) \star snd\ (g,\ h),\ k) \cdot$
$(fst\ (\alpha\ (f,\ g,\ h),\ k) \star snd\ (\alpha\ (f,\ g,\ h),\ k)) =$
$\alpha\ (f,\ g,\ fst\ (h,\ k) \star snd\ (h,\ k)) \cdot \alpha\ (fst\ (f,\ g) \star snd\ (f,\ g),\ h,\ k)$
by *simp*
qed
qed

proposition *is-monoidal-category*:
shows *monoidal-category* $(\cdot)$ $(\lambda\mu\nu.\ fst\ \mu\nu \star snd\ \mu\nu)$ $\alpha\ \omega$
..

end

In a bicategory, the "objects" are essentially arbitrarily chosen representatives of their isomorphism classes. Choosing any other representatives results in an equivalent structure. Each object a is additionally equipped with an arbitrarily chosen unit isomorphism «$\iota : a \star a \Rightarrow a$». For any $(a,\ \iota)$ and $(a',\ \iota')$, where a and a' are isomorphic to the same object, there exists a unique isomorphism «ψ: $a \Rightarrow a'$» that is compatible with the chosen unit isomorphisms ι and ι'. We have already proved this property for monoidal categories, which are bicategories with just one "object". Here we use that already-proven property to establish its generalization to arbitary bicategories, by exploiting the fact that if a is an object in a bicategory, then the sub-bicategory consisting of all μ such that $src\ \mu = a = trg\ \mu$, is a monoidal category.

At some point it would potentially be nicer to transfer the proof for monoidal categories to obtain a direct, "native" proof of this fact for bicategories.

lemma (**in** *bicategory*) *unit-unique-upto-unique-iso*:
assumes *obj a*
and *isomorphic a w*
and «$\omega : w \star w \Rightarrow w$»
and *iso* ω
shows $\exists!\psi.$ «$\psi : a \Rightarrow w$» $\wedge$ *iso* $\psi \wedge \psi \cdot \mathrm{i}[a] = \omega \cdot (\psi \star \psi)$
proof −
have *ω-in-hhom*: «$\omega : a \rightarrow a$»
using *assms*
apply (*intro in-hhomI*)
apply *auto*
apply (*metis src-cod in-homE isomorphic-implies-hpar*(*3*) *objE*)
by (*metis trg-cod in-homE isomorphic-implies-hpar*(*4*) *objE*)
interpret *S*: *subbicategory V H* a i *src trg* ‹$\lambda\mu.\ arr\ \mu \wedge src\ \mu = a \wedge trg\ \mu = a$›
using *assms iso-unit in-homE isoE isomorphicE VVV*.$arr\text{-}char_{SbC}$ *VV*.$arr\text{-}char_{SbC}$
apply *unfold-locales*
apply *auto*[*7*]

proof
  fix $f\ g\ h$
  assume f: $(arr\ f \wedge src\ f = a \wedge trg\ f = a) \wedge ide\ f$
  and g: $(arr\ g \wedge src\ g = a \wedge trg\ g = a) \wedge ide\ g$
  and h: $(arr\ h \wedge src\ h = a \wedge trg\ h = a) \wedge ide\ h$
  and fg: $src\ f = trg\ g$ **and** gh: $src\ g = trg\ h$
  show $arr\ (\mathrm{a}[f,\ g,\ h])$
    using *assms f g h fg gh* **by** *auto*
  show $src\ (\mathrm{a}[f,\ g,\ h]) = a \wedge trg\ (\mathrm{a}[f,\ g,\ h]) = a$
    using *assms f g h fg gh* **by** *auto*
  show $arr\ (inv\ (\mathrm{a}[f,\ g,\ h])) \wedge src\ (inv\ (\mathrm{a}[f,\ g,\ h])) = a \wedge trg\ (inv\ (\mathrm{a}[f,\ g,\ h])) = a$
    using *assms f g h fg gh* α*.preserves-hom src-dom trg-dom* **by** *simp*
  next
  fix f
  assume f: $arr\ f \wedge src\ f = a \wedge trg\ f = a$
  assume *ide-left*: $ide\ f$
  show $arr\ (\mathfrak{l}\ f) \wedge src\ (\mathfrak{l}\ f) = a \wedge trg\ (\mathfrak{l}\ f) = a$
    using *f assms(1)* $\mathfrak{l}$*.preserves-hom src-cod* [*of* $\mathfrak{l}\ f$] *trg-cod* [*of* $\mathfrak{l}\ f$] **by** *simp*
  show $arr\ (inv\ (\mathfrak{l}\ f)) \wedge src\ (inv\ (\mathfrak{l}\ f)) = a \wedge trg\ (inv\ (\mathfrak{l}\ f)) = a$
    using *f ide-left assms(1)* $\mathfrak{l}'$*.preserves-hom src-dom* [*of* $\mathfrak{l}'$*.map f*] *trg-dom* [*of* $\mathfrak{l}'$*.map f*]
    by *simp*
  show $arr\ (\mathfrak{r}\ f) \wedge src\ (\mathfrak{r}\ f) = a \wedge trg\ (\mathfrak{r}\ f) = a$
    using *f assms(1)* $\mathfrak{r}$*.preserves-hom src-cod* [*of* $\mathfrak{r}\ f$] *trg-cod* [*of* $\mathfrak{r}\ f$] **by** *simp*
  show $arr\ (inv\ (\mathfrak{r}\ f)) \wedge src\ (inv\ (\mathfrak{r}\ f)) = a \wedge trg\ (inv\ (\mathfrak{r}\ f)) = a$
    using *f ide-left assms(1)* $\mathfrak{r}'$*.preserves-hom src-dom* [*of* $\mathfrak{r}'$*.map f*] *trg-dom* [*of* $\mathfrak{r}'$*.map f*]
    by *simp*
qed
interpret S: *subbicategory-at-object V H* a i *src trg a a* ‹i[a]›
proof
  show *obj a* **by** *fact*
  show *isomorphic a a*
    using *assms(1) isomorphic-reflexive* **by** *blast*
  show *S.in-hom* i[a] $(a \star a)$ a
    using S*.arr-char*$_{SbC}$ S*.in-hom-char*$_{SbC}$ *assms(1)* **by** *fastforce*
  show *iso* i[a]
    using *assms iso-unit* **by** *simp*
qed
interpret S_ω: *subbicategory-at-object V H* a i *src trg a w* ω
proof
  show *obj a* **by** *fact*
  show *iso* ω **by** *fact*
  show *isomorphic a w*
    using *assms* **by** *simp*
  show *S.in-hom* ω $(w \star w)$ w
    using *assms* S*.arr-char*$_{SbC}$ S*.dom-char*$_{SbC}$ S*.cod-char*$_{SbC}$ ω*-in-hhom*
    by (*intro S.in-homI*, *auto*)
qed
interpret M: *monoidal-category S.comp* ‹$\lambda\mu\nu$. *S.hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› $S.\alpha$ ‹i[a]›
  using S*.is-monoidal-category* **by** *simp*

interpret M_ω: *monoidal-category S.comp* ‹$\lambda\mu\nu$. *S.hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› *S.*α ω
  using S_ω*.is-monoidal-category* **by** *simp*
interpret *M*: *monoidal-category-with-alternate-unit*
    S.comp ‹$\lambda\mu\nu$. *S.hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› *S.*α ‹i[*a*]› ω **..**
have *1*: M_ω*.unity* = *w*
  using *assms S.cod-char*$_{SbC}$ *S.arr-char*$_{SbC}$
  by (*metis* (*no-types*, *lifting*) *S.in-homE* S_ω*.*ω*-in-vhom*)
have *2*: *M.unity* = *a*
  using *assms S.cod-char*$_{SbC}$ *S.arr-char*$_{SbC}$ **by** *simp*
have $\exists!\psi$. *S.in-hom* ψ *a w* $\wedge$ *S.iso* ψ $\wedge$ *S.comp* ψ i[*a*] = *S.comp* ω (*M.tensor* ψ ψ)
  using *assms 1 2 M.unit-unique-upto-unique-iso S.cod-char*$_{SbC}$
  by *simp*
show $\exists!\psi$. «ψ : *a* $\Rightarrow$ *w*» $\wedge$ *iso* ψ $\wedge$ ψ $\cdot$ i[*a*] = ω $\cdot$ (ψ $\star$ ψ)
proof −
  have *1*: $\bigwedge\psi$. *S.in-hom* ψ *a w* $\longleftrightarrow$ «ψ : *a* $\Rightarrow$ *w*»
    using *assms S.in-hom-char*$_{SbC}$ *S.arr-char*$_{SbC}$
    by (*metis* (*no-types*, *lifting*) *S.ideD*(*1*) *S.w-simps*(*1*) S_ω*.w-simps*(*1*) *in-homE*
      src-dom trg-dom)
  moreover have $\bigwedge\psi$. *S.in-hom* ψ *a w* $\Longrightarrow$ *S.iso* ψ $\longleftrightarrow$ *iso* ψ
    using *assms S.in-hom-char*$_{SbC}$ *S.arr-char*$_{SbC}$ *S.iso-char*$_{SbC}$ **by** *auto*
  moreover have $\bigwedge\psi$. *S.in-hom* ψ *a w* $\Longrightarrow$ *M.tensor* ψ ψ = ψ $\star$ ψ
    using *assms S.in-hom-char*$_{SbC}$ *S.arr-char*$_{SbC}$ *S.hcomp-def* **by** *simp*
  moreover have $\bigwedge\psi$. *S.in-hom* ψ *a w* $\Longrightarrow$ *S.comp* ψ i[*a*] = ψ $\cdot$ i[*a*]
    using *assms S.in-hom-char*$_{SbC}$ *S.comp-char* **by** *auto*
  moreover have $\bigwedge\psi$. *S.in-hom* ψ *a w* $\Longrightarrow$ *S.comp* ω (*M.tensor* ψ ψ) = ω $\cdot$ (ψ $\star$ ψ)
    using *assms S.in-hom-char*$_{SbC}$ *S.arr-char*$_{SbC}$ *S.hcomp-def S.comp-char S.dom-char*$_{SbC}$
S.cod-char$_{SbC}$
    by (*metis* (*no-types*, *lifting*) M_ω*.arr-tensor* S_ω*.*ω*-simps*(*1*) *calculation*(*3*) *ext*)
  ultimately show *?thesis*
    by (*metis* (*no-types*, *lifting*) *M.unit-unique-upto-unique-iso S.*ω*-in-vhom S.in-homE*
      S_ω*.*ω*-in-vhom*)
qed
qed

end

1.9 Internal Equivalences

theory *InternalEquivalence*
imports *Bicategory*
begin

An *internal equivalence* in a bicategory consists of antiparallel 1-cells f and g together with invertible 2-cells «η : *src f* $\Rightarrow$ g $\star$ f» and «ε : f $\star$ g $\Rightarrow$ *src g*». Objects in a bicategory are said to be *equivalent* if they are connected by an internal equivalence. In this section we formalize the definition of internal equivalence and the related notions "equivalence map" and "equivalent objects", and we establish some basic facts about these notions.

1.9.1 Definition of Equivalence

The following locale is defined to prove some basic facts about an equivalence (or an adjunction) in a bicategory that are "syntactic" in the sense that they depend only on the configuration (source, target, domain, codomain) of the arrows involved and not on further properties such as the triangle identities (for adjunctions) or assumptions about invertibility (for equivalences). Proofs about adjunctions and equivalences become more automatic once we have introduction and simplification rules in place about this syntax.

locale *adjunction-data-in-bicategory* =
  bicategory +
fixes f :: $'a$
and g :: $'a$
and η :: $'a$
and ε :: $'a$
assumes *ide-left* [*simp*]: *ide f*
and *ide-right* [*simp*]: *ide g*
and *unit-in-vhom*: «η : *src f* $\Rightarrow$ $g \star f$»
and *counit-in-vhom*: «ε : $f \star g$ $\Rightarrow$ *src g*»
begin

lemma *antipar* :
shows *trg g* = *src f* **and** *src g* = *trg f*
  apply (*metis counit-in-vhom hseqE ideD*(*1*) *ide-right src.preserves-reflects-arr*
      vconn-implies-hpar(*3*))
  by (*metis arrI not-arr-null seq-if-composable src.preserves-reflects-arr*
      unit-in-vhom vconn-implies-hpar(*1*) *vconn-implies-hpar*(*3*))

lemma *counit-in-hom* [*intro*]:
shows «ε : *trg f* $\rightarrow$ *trg f*» **and** «ε : $f \star g$ $\Rightarrow$ *trg f*»
  using *counit-in-vhom vconn-implies-hpar antipar* **by** *auto*

lemma *unit-in-hom* [*intro*]:
shows «η : *src f* $\rightarrow$ *src f*» **and** «η : *src f* $\Rightarrow$ $g \star f$»
  using *unit-in-vhom vconn-implies-hpar antipar* **by** *auto*

lemma *unit-simps* [*simp*]:
shows *arr* η **and** *dom* η = *src f* **and** *cod* η = $g \star f$
and *src* η = *src f* **and** *trg* η = *src f*
  using *unit-in-hom antipar* **by** *auto*

lemma *counit-simps* [*simp*]:
shows *arr* ε **and** *dom* ε = $f \star g$ **and** *cod* ε = *trg f*
and *src* ε = *trg f* **and** *trg* ε = *trg f*
  using *counit-in-hom antipar* **by** *auto*

The expressions found in the triangle identities for an adjunction come up relatively frequently, so it is useful to have established some basic facts about them, even if the triangle identities themselves have not actually been introduced as assumptions in the

current context.

lemma *triangle-in-hom*:
shows «$(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) : f \star \mathit{src}\ f \Rightarrow \mathit{trg}\ f \star f$»
and «$(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) : \mathit{trg}\ g \star g \Rightarrow g \star \mathit{src}\ g$»
and «$\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] : f \Rightarrow f$»
and «$\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] : g \Rightarrow g$»
  using *antipar* **by** *auto*

lemma *triangle-equiv-form*:
shows $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \longleftrightarrow$
    $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
and $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \longleftrightarrow$
    $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] = g$
proof −
  show $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \longleftrightarrow$
    $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
  proof
    assume *1*: $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
    have $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] =$
      $\mathrm{l}[f] \cdot ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)) \cdot \mathrm{r}^{-1}[f]$
      using *comp-assoc* **by** *simp*
    also have ... $= \mathrm{l}[f] \cdot (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]) \cdot \mathrm{r}^{-1}[f]$
      using *1* **by** *simp*
    also have ... $= f$
      using *comp-assoc comp-arr-inv′ comp-inv-arr′ iso-lunit iso-runit*
        comp-arr-dom comp-cod-arr
      by *simp*
    finally show $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
      by *simp*
    next
    assume *2*: $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
    have $\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] = \mathrm{l}^{-1}[f] \cdot f \cdot \mathrm{r}[f]$
      using *comp-cod-arr* **by** *simp*
    also have ... $= (\mathrm{l}^{-1}[f] \cdot \mathrm{l}[f]) \cdot ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)) \cdot (\mathrm{r}^{-1}[f] \cdot \mathrm{r}[f])$
      using *2 comp-assoc* **by** (*metis* (*no-types*, *lifting*))
    also have ... $= (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)$
      using *comp-arr-inv′ comp-inv-arr′ iso-lunit iso-runit comp-arr-dom comp-cod-arr*
        hseqI′ antipar
      by (*metis ide-left in-homE lunit-simps*(*4*) *runit-simps*(*4*) *triangle-in-hom*(*1*))
    finally show $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
      by *simp*
  qed
  show $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \longleftrightarrow$
    $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] = g$
  proof
    assume *1*: $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
    have $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] =$
      $\mathrm{r}[g] \cdot ((g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)) \cdot \mathrm{l}^{-1}[g]$
      using *comp-assoc* **by** *simp*

```
      also have ... = r[g] · (r⁻¹[g] · l[g]) · l⁻¹[g]
        using 1 by simp
      also have ... = g
        using comp-assoc comp-arr-inv′ comp-inv-arr′ iso-lunit iso-runit
              comp-arr-dom comp-cod-arr
        by simp
      finally show r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g] = g
        by simp
      next
      assume 2: r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g] = g
      have r⁻¹[g] · l[g] = r⁻¹[g] · g · l[g]
        using comp-cod-arr by simp
      also have ... = r⁻¹[g] · (r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g]) · l[g]
        using 2 by simp
      also have ... = (r⁻¹[g] · r[g]) · ((g ⋆ ε) · a[g, f, g] · (η ⋆ g)) · (l⁻¹[g] · l[g])
        using comp-assoc by simp
      also have ... = (g ⋆ ε) · a[g, f, g] · (η ⋆ g)
        using comp-arr-inv′ comp-inv-arr′ iso-lunit iso-runit comp-arr-dom comp-cod-arr
              hseqI′ antipar
        by (metis ide-right in-homE lunit-simps(4) runit-simps(4) triangle-in-hom(2))
      finally show (g ⋆ ε) · a[g, f, g] · (η ⋆ g) = r⁻¹[g] · l[g]
        by simp
    qed
  qed

end

locale equivalence-in-bicategory =
  adjunction-data-in-bicategory +
assumes unit-is-iso [simp]: iso η
and counit-is-iso [simp]: iso ε
begin

  lemma dual-equivalence:
  shows equivalence-in-bicategory V H a i src trg g f (inv ε) (inv η)
    using antipar by unfold-locales auto

end

abbreviation (in bicategory) internal-equivalence
  where internal-equivalence f g φ ψ ≡ equivalence-in-bicategory V H a i src trg f g φ ψ
```

1.9.2 Quasi-Inverses and Equivalence Maps

Antiparallel 1-cells f and g are *quasi-inverses* if they can be extended to an internal equivalence. We will use the term *equivalence map* to refer to a 1-cell that has a quasi-inverse.

```
context bicategory
begin
```

definition *quasi-inverses*
where *quasi-inverses f g* $\equiv \exists \varphi\ \psi.$ *internal-equivalence f g* $\varphi\ \psi$

lemma *quasi-inversesI*:
assumes *ide f* **and** *ide g*
and *src f* $\cong$ *g* $\star$ *f* **and** *f* $\star$ *g* $\cong$ *trg f*
shows *quasi-inverses f g*
proof (*unfold quasi-inverses-def*)
  have *1*: *src g* = *trg f*
    using *assms ideD*(*1*) *isomorphic-implies-ide*(*2*) **by** *blast*
  obtain φ **where** φ: «φ : *src f* $\Rightarrow$ *g* $\star$ *f*» $\wedge$ *iso* φ
    using *assms isomorphic-def* **by** *auto*
  obtain ψ **where** ψ: «ψ : *f* $\star$ *g* $\Rightarrow$ *trg f*» $\wedge$ *iso* ψ
    using *assms isomorphic-def* **by** *auto*
 have *equivalence-in-bicategory V H* a i *src trg f g* $\varphi\ \psi$
    using *assms 1* $\varphi\ \psi$ **by** *unfold-locales auto*
  thus $\exists \varphi\ \psi.$ *internal-equivalence f g* $\varphi\ \psi$ **by** *auto*
qed

lemma *quasi-inversesE*:
assumes *quasi-inverses f g*
and $[\![$*ide f*; *ide g*; *src f* $\cong$ *g* $\star$ *f*; *f* $\star$ *g* $\cong$ *trg f*$]\!] \Longrightarrow T$
shows *T*
proof −
  obtain $\varphi\ \psi$ **where** $\varphi\psi$: *internal-equivalence f g* $\varphi\ \psi$
    using *assms quasi-inverses-def* **by** *auto*
  interpret $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f g* $\varphi\ \psi$
    using $\varphi\psi$ **by** *simp*
  have *ide f* $\wedge$ *ide g*
    by *simp*
  moreover have *src f* $\cong$ *g* $\star$ *f*
    using *isomorphic-def* $\varphi\psi$*.unit-in-hom* **by** *auto*
  moreover have *f* $\star$ *g* $\cong$ *trg f*
    using *isomorphic-def* $\varphi\psi$*.counit-in-hom* **by** *auto*
  ultimately show *T*
    using *assms* **by** *blast*
qed

lemma *quasi-inverse-unique*:
assumes *quasi-inverses f g* **and** *quasi-inverses f g′*
shows *isomorphic g g′*
proof −
  obtain $\varphi\ \psi$ **where** $\varphi\psi$: *internal-equivalence f g* $\varphi\ \psi$
    using *assms quasi-inverses-def* **by** *auto*
  interpret $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f g* $\varphi\ \psi$
    using $\varphi\psi$ **by** *simp*
  obtain $\varphi'\ \psi'$ **where** $\varphi'\psi'$: *internal-equivalence f g′* $\varphi'\ \psi'$
    using *assms quasi-inverses-def* **by** *auto*

interpret $\varphi'\psi'$: *equivalence-in-bicategory V H* a i *src trg f g′ φ′ ψ′*
using $\varphi'\psi'$ **by** *simp*
have «$\mathrm{r}[g'] \cdot (g' \star \psi) \cdot \mathrm{a}[g', f, g] \cdot (\varphi' \star g) \cdot \mathrm{l}^{-1}[g] : g \Rightarrow g'$»
using *φψ.unit-in-hom φψ.unit-is-iso φψ.antipar φ′ψ′.antipar*
by (*intro comp-in-homI′ hseqI′*) *auto*
moreover have *iso* ($\mathrm{r}[g'] \cdot (g' \star \psi) \cdot \mathrm{a}[g', f, g] \cdot (\varphi' \star g) \cdot \mathrm{l}^{-1}[g]$)
using *φψ.unit-in-hom φψ.unit-is-iso φψ.antipar φ′ψ′.antipar*
by (*intro isos-compose*) *auto*
ultimately show *?thesis*
using *isomorphic-def* **by** *auto*
qed

lemma *quasi-inverses-symmetric*:
assumes *quasi-inverses f g*
shows *quasi-inverses g f*
using *assms quasi-inverses-def equivalence-in-bicategory.dual-equivalence* **by** *metis*

definition *equivalence-map*
where *equivalence-map f* $\equiv \exists\, g\ \eta\ \varepsilon.$ *equivalence-in-bicategory V H* a i *src trg f g η ε*

lemma *equivalence-mapI*:
assumes *quasi-inverses f g*
shows *equivalence-map f*
using *assms quasi-inverses-def equivalence-map-def* **by** *auto*

lemma *equivalence-mapE*:
assumes *equivalence-map f*
obtains *g* **where** *quasi-inverses f g*
using *assms equivalence-map-def quasi-inverses-def* **by** *auto*

lemma *equivalence-map-is-ide*:
assumes *equivalence-map f*
shows *ide f*
using *assms adjunction-data-in-bicategory.ide-left equivalence-in-bicategory-def equivalence-map-def*
by *fastforce*

lemma *obj-is-equivalence-map*:
assumes *obj a*
shows *equivalence-map a*
using *assms*
by (*metis equivalence-mapI isomorphic-symmetric objE obj-self-composable*(*2*) *quasi-inversesI*)

lemma *equivalence-respects-iso*:
assumes *equivalence-in-bicategory V H* a i *src trg f g η ε*
and «$\varphi : f \Rightarrow f'$» **and** *iso φ* **and** «$\psi : g \Rightarrow g'$» **and** *iso ψ*
shows *internal-equivalence f′ g′* $((g' \star \varphi) \cdot (\psi \star f) \cdot \eta)$ $(\varepsilon \cdot (\mathit{inv}\ \varphi \star g) \cdot (f' \star \mathit{inv}\ \psi))$
proof −
interpret *E*: *equivalence-in-bicategory V H* a i *src trg f g η ε*

```
    using assms by auto
  show ?thesis
  proof
    show f′: ide f′ using assms by auto
    show g′: ide g′ using assms by auto
    show 1: «(g′ ⋆ φ) · (ψ ⋆ f) · η : src f′ ⇒ g′ ⋆ f′»
      using assms f′ g′ E.unit-in-hom E.antipar(2) vconn-implies-hpar(3)
     apply (intro comp-in-homI)
       apply auto
      by (intro hcomp-in-vhom) auto
    show iso ((g′ ⋆ φ) · (ψ ⋆ f) · η)
      using assms 1 g′ vconn-implies-hpar(3) E.antipar(2) iso-hcomp
      by (intro isos-compose) auto
    show 1: «ε · (inv φ ⋆ g) · (f′ ⋆ inv ψ) : f′ ⋆ g′ ⇒ src g′»
      using assms f′ ide-in-hom(2) vconn-implies-hpar(3−4) E.antipar(1−2)
      by (intro comp-in-homI) auto
    show iso (ε · (inv φ ⋆ g) · (f′ ⋆ inv ψ))
      using assms 1 isos-compose
      by (metis E.counit-is-iso E.ide-right arrI f′ hseqE ide-is-iso iso-hcomp
          iso-inv-iso seqE)
  qed
qed

lemma equivalence-map-preserved-by-iso:
assumes equivalence-map f and f ≅ f′
shows equivalence-map f′
proof −
  obtain g η ε where E: equivalence-in-bicategory V H a i src trg f g η ε
    using assms equivalence-map-def by auto
  interpret E: equivalence-in-bicategory V H a i src trg f g η ε
    using E by auto
  obtain φ where φ: «φ : f ⇒ f′» ∧ iso φ
    using assms isomorphic-def isomorphic-symmetric by blast
  have equivalence-in-bicategory V H a i src trg f′ g
          ((g ⋆ φ) · (g ⋆ f) · η) (ε · (inv φ ⋆ g) · (f′ ⋆ inv g))
    using E φ equivalence-respects-iso [of f g η ε φ f′ g g] by fastforce
  thus ?thesis
    using equivalence-map-def by auto
qed

lemma equivalence-preserved-by-iso-right:
assumes equivalence-in-bicategory V H a i src trg f g η ε
and «φ : g ⇒ g′» and iso φ
shows equivalence-in-bicategory V H a i src trg f g′ ((φ ⋆ f) · η) (ε · (f ⋆ inv φ))
proof
  interpret E: equivalence-in-bicategory V H a i src trg f g η ε
    using assms by auto
  show ide f by simp
  show ide g′
```

using *assms*(*2*) *isomorphic-def* **by** *auto*
show «$(\varphi \star f) \cdot \eta$: *src* $f \Rightarrow g' \star f$»
using *assms E.antipar*(*2*) *E.ide-left* **by** *blast*
show «$\varepsilon \cdot (f \star$ *inv* $\varphi)$: $f \star g' \Rightarrow$ *src* g'»
using *assms vconn-implies-hpar*(*3*−*4*) *E.counit-in-vhom E.antipar*(*1*) *ide-in-hom*(*2*)
by (*intro comp-in-homI*, *auto*)
show *iso* $((\varphi \star f) \cdot \eta)$
using *assms E.antipar isos-compose* **by** *auto*
show *iso* $(\varepsilon \cdot (f \star$ *inv* $\varphi))$
using *assms E.antipar isos-compose* **by** *auto*
qed

lemma *equivalence-preserved-by-iso-left*:
assumes *equivalence-in-bicategory V H* a i *src trg f g* η ε
and «$\varphi : f \Rightarrow f'$» **and** *iso* φ
shows *equivalence-in-bicategory V H* a i *src trg* f' g $((g \star \varphi) \cdot \eta)$ $(\varepsilon \cdot ($*inv* $\varphi \star g))$
proof
interpret *E*: *equivalence-in-bicategory V H* a i *src trg f g* η ε
using *assms* **by** *auto*
show *ide* f'
using *assms* **by** *auto*
show *ide g*
by *simp*
show «$(g \star \varphi) \cdot \eta$: *src* $f' \Rightarrow g \star f'$»
using *assms E.unit-in-hom E.antipar* **by** *auto*
show «$\varepsilon \cdot ($*inv* $\varphi \star g)$: $f' \star g \Rightarrow$ *src g*»
using *assms E.counit-in-hom E.antipar ide-in-hom*(*2*) *vconn-implies-hpar*(*3*) **by** *auto*
show *iso* $((g \star \varphi) \cdot \eta)$
using *assms E.antipar isos-compose* **by** *auto*
show *iso* $(\varepsilon \cdot ($*inv* $\varphi \star g))$
using *assms E.antipar isos-compose* **by** *auto*
qed

definition *some-quasi-inverse*
where *some-quasi-inverse f* = (*SOME g. quasi-inverses f g*)

notation *some-quasi-inverse* (‹-$^{\sim}$› [*1000*] *1000*)

lemma *quasi-inverses-some-quasi-inverse*:
assumes *equivalence-map f*
shows *quasi-inverses f* $f^{\sim}$
and *quasi-inverses* $f^{\sim}$ *f*
using *assms some-quasi-inverse-def quasi-inverses-def equivalence-map-def*
someI-ex [*of* λg. *quasi-inverses f g*] *quasi-inverses-symmetric*
by *auto*

lemma *quasi-inverse-antipar*:
assumes *equivalence-map f*
shows *src* $f^{\sim}$ = *trg f* **and** *trg* $f^{\sim}$ = *src f*

proof −
  obtain φ ψ **where** $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f* $f^{\sim}$ φ ψ
    using *assms equivalence-map-def quasi-inverses-some-quasi-inverse quasi-inverses-def*
    by *blast*
  interpret $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f* ‹$f^{\sim}$› φ ψ
    using $\varphi\psi$ **by** *simp*
  show *src* $f^{\sim}$ = *trg f*
    using $\varphi\psi$.*antipar* **by** *simp*
  show *trg* $f^{\sim}$ = *src f*
    using $\varphi\psi$.*antipar* **by** *simp*
qed

lemma *quasi-inverse-in-hom* [*intro*]:
assumes *equivalence-map f*
shows «$f^{\sim}$: *trg f* → *src f*»
and «$f^{\sim}$: $f^{\sim}$ ⇒ $f^{\sim}$»
  using *assms equivalence-mapE*
  apply (*intro in-homI in-hhomI*)
    apply (*metis equivalence-map-is-ide ideD*(*1*) *not-arr-null quasi-inverse-antipar*(*2*)
        src.preserves-ide trg.extensionality)
    apply (*simp-all add*: *quasi-inverse-antipar*)
  using *assms quasi-inversesE quasi-inverses-some-quasi-inverse*(*2*) **by** *blast*

lemma *quasi-inverse-simps* [*simp*]:
assumes *equivalence-map f*
shows *equivalence-map* $f^{\sim}$ **and** *ide* $f^{\sim}$
and *src* $f^{\sim}$ = *trg f* **and** *trg* $f^{\sim}$ = *src f*
and *dom* $f^{\sim}$ = $f^{\sim}$ **and** *cod* $f^{\sim}$ = $f^{\sim}$
  using *assms equivalence-mapE quasi-inverse-in-hom quasi-inverses-some-quasi-inverse*
    equivalence-map-is-ide
    apply *auto*
  by (*meson equivalence-mapI*)

lemma *quasi-inverse-quasi-inverse*:
assumes *equivalence-map f*
shows $(f^{\sim})^{\sim}$ ≅ *f*
proof −
  have *quasi-inverses* $f^{\sim}$ $(f^{\sim})^{\sim}$
    using *assms quasi-inverses-some-quasi-inverse* **by** *simp*
  moreover have *quasi-inverses* $f^{\sim}$ *f*
    using *assms quasi-inverses-some-quasi-inverse quasi-inverses-symmetric* **by** *simp*
  ultimately show *?thesis*
    using *quasi-inverse-unique* **by** *simp*
qed

lemma *comp-quasi-inverse*:
assumes *equivalence-map f*
shows $f^{\sim}$ ⋆ *f* ≅ *src f* **and** *f* ⋆ $f^{\sim}$ ≅ *trg f*
proof −

obtain φ ψ **where** $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f* $f^{\sim}$ φ ψ
  using *assms equivalence-map-def quasi-inverses-some-quasi-inverse*
    quasi-inverses-def
  by *blast*
interpret $\varphi\psi$: *equivalence-in-bicategory V H* a i *src trg f* ‹$f^{\sim}$› φ ψ
  using $\varphi\psi$ **by** *simp*
show $f^{\sim} \star f \cong$ *src f*
  using *quasi-inverses-some-quasi-inverse quasi-inverses-def*
    φψ.unit-in-hom φψ.unit-is-iso isomorphic-def
  by *blast*
show $f \star f^{\sim} \cong$ *trg f*
  using *quasi-inverses-some-quasi-inverse quasi-inverses-def*
    φψ.counit-in-hom φψ.counit-is-iso isomorphic-def
  by *blast*
qed

lemma *quasi-inverse-transpose*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** $f \star g \cong h$
shows *equivalence-map g* $\Longrightarrow f \cong h \star g^{\sim}$
and *equivalence-map f* $\Longrightarrow g \cong f^{\sim} \star h$
proof −
  have *1*: *src f* = *trg g*
    using *assms equivalence-map-is-ide* **by** *fastforce*
  show *equivalence-map g* $\Longrightarrow f \cong h \star g^{\sim}$
  proof −
    assume *g*: *equivalence-map g*
    have *2*: *ide* $g^{\sim}$
      using *g* **by** *simp*
    have $f \cong f \star$ *src f*
      using *assms isomorphic-unit-right isomorphic-symmetric* **by** *blast*
    also have ... $\cong f \star$ *trg g*
      using *assms 1 isomorphic-reflexive* **by** *auto*
    also have ... $\cong f \star g \star g^{\sim}$
      using *assms g 1 comp-quasi-inverse*(*2*) *isomorphic-symmetric hcomp-ide-isomorphic*
      by *simp*
    also have ... $\cong (f \star g) \star g^{\sim}$
      using *assms g 1 2 assoc′-in-hom* [*of f g* $g^{\sim}$] *iso-assoc′ isomorphic-def* **by** *auto*
    also have ... $\cong h \star g^{\sim}$
      using *assms g 1 2*
      by (*simp add*: *hcomp-isomorphic-ide*)
    finally show *?thesis* **by** *blast*
  qed
  show *equivalence-map f* $\Longrightarrow g \cong f^{\sim} \star h$
  proof −
    assume *f*: *equivalence-map f*
    have *2*: *ide* $f^{\sim}$
      using *f* **by** *simp*
    have $g \cong$ *src f* $\star g$
      using *assms 1 isomorphic-unit-left isomorphic-symmetric* **by** *metis*

also have ... $\cong (f^{\sim} \star f) \star g$
using *assms f 1 comp-quasi-inverse(1)* [*of f*] *isomorphic-symmetric hcomp-isomorphic-ide*
by *simp*
also have ... $\cong f^{\sim} \star f \star g$
using *assms f 1 assoc-in-hom* [*of* $f^{\sim}$ *f g*] *iso-assoc isomorphic-def* **by** *auto*
also have ... $\cong f^{\sim} \star h$
using *assms f 1 equivalence-map-is-ide quasi-inverses-some-quasi-inverse hcomp-ide-isomorphic*
by *simp*
finally show *?thesis* **by** *blast*
qed
qed

end

1.9.3 Composing Equivalences

locale *composite-equivalence-in-bicategory* =
bicategory V H a i *src trg* +
fg: *equivalence-in-bicategory V H* a i *src trg f g* ζ ξ +
hk: *equivalence-in-bicategory V H* a i *src trg h k* σ τ
for $V :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹·› *55*)
and $H :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a $:: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i $:: 'a \Rightarrow 'a$ (‹i[-]›)
and $src :: 'a \Rightarrow 'a$
and $trg :: 'a \Rightarrow 'a$
and $f :: 'a$
and $g :: 'a$
and $\zeta :: 'a$
and $\xi :: 'a$
and $h :: 'a$
and $k :: 'a$
and $\sigma :: 'a$
and $\tau :: 'a$ +
assumes *composable*: $src\ h = trg\ f$
begin

abbreviation η
where $\eta \equiv \mathrm{a}^{-1}[g,\ k,\ h \star f] \cdot (g \star \mathrm{a}[k,\ h,\ f]) \cdot (g \star \sigma \star f) \cdot (g \star \mathrm{l}^{-1}[f]) \cdot \zeta$

abbreviation ε
where $\varepsilon \equiv \tau \cdot (h \star \mathrm{l}[k]) \cdot (h \star \xi \star k) \cdot (h \star \mathrm{a}^{-1}[f,\ g,\ k]) \cdot \mathrm{a}[h,\ f,\ g \star k]$

interpretation *adjunction-data-in-bicategory V H* a i *src trg* ‹$h \star f$› ‹$g \star k$› η ε
proof
show *ide* $(h \star f)$
using *composable* **by** *simp*

```
    show ide (g ⋆ k)
      using fg.antipar hk.antipar composable by simp
    show «η : src (h ⋆ f) ⇒ (g ⋆ k) ⋆ h ⋆ f»
      using fg.antipar hk.antipar composable by fastforce
    show «ε : (h ⋆ f) ⋆ g ⋆ k ⇒ src (g ⋆ k)»
      using fg.antipar hk.antipar composable by fastforce
  qed

  interpretation equivalence-in-bicategory V H a i src trg ‹h ⋆ f› ‹g ⋆ k› η ε
  proof
    show iso η
      using fg.antipar hk.antipar composable fg.unit-is-iso hk.unit-is-iso iso-hcomp
            iso-lunit' hseq-char
      by (intro isos-compose, auto)
    show iso ε
      using fg.antipar hk.antipar composable fg.counit-is-iso hk.counit-is-iso iso-hcomp
            iso-lunit hseq-char
      by (intro isos-compose, auto)
  qed

  lemma is-equivalence:
  shows equivalence-in-bicategory V H a i src trg (h ⋆ f) (g ⋆ k) η ε
    ..

  sublocale equivalence-in-bicategory V H a i src trg ‹h ⋆ f› ‹g ⋆ k› η ε
    using is-equivalence by simp

end

context bicategory
begin

  lemma equivalence-maps-compose:
  assumes equivalence-map f and equivalence-map f' and src f' = trg f
  shows equivalence-map (f' ⋆ f)
  proof −
    obtain g φ ψ where fg: equivalence-in-bicategory V H a i src trg f g φ ψ
      using assms(1) equivalence-map-def by auto
    interpret fg: equivalence-in-bicategory V H a i src trg f g φ ψ
      using fg by auto
    obtain g' φ' ψ' where f'g': equivalence-in-bicategory V H a i src trg f' g' φ' ψ'
      using assms(2) equivalence-map-def by auto
    interpret f'g': equivalence-in-bicategory V H a i src trg f' g' φ' ψ'
      using f'g' by auto
    interpret composite-equivalence-in-bicategory V H a i src trg f g φ ψ f' g' φ' ψ'
      using assms(3) by (unfold-locales, auto)
    show ?thesis
      using equivalence-map-def equivalence-in-bicategory-axioms by auto
  qed
```

lemma *quasi-inverse-hcomp′*:
assumes *equivalence-map f* **and** *equivalence-map f′* **and** *equivalence-map* $(f \star f')$
and *quasi-inverses f g* **and** *quasi-inverses f′ g′*
shows *quasi-inverses* $(f \star f')$ $(g' \star g)$
proof −
  obtain φ ψ **where** *g*: *internal-equivalence f g* φ ψ
    using *assms quasi-inverses-def* **by** *auto*
  interpret *g*: *equivalence-in-bicategory V H* a i *src trg f g* φ ψ
    using *g* **by** *simp*
  obtain φ' ψ' **where** *g′*: *internal-equivalence f′ g′* φ' ψ'
    using *assms quasi-inverses-def* **by** *auto*
  interpret *g′*: *equivalence-in-bicategory V H* a i *src trg f′ g′* φ' ψ'
    using *g′* **by** *simp*
  interpret *gg′*: *composite-equivalence-in-bicategory V H* a i *src trg f′ g′* φ' ψ' *f g* φ ψ
    using *assms equivalence-map-is-ide* [*of* $f \star f'$]
    apply *unfold-locales*
    using *ideD*(*1*) **by** *fastforce*
  show *?thesis*
    unfolding *quasi-inverses-def*
    using *gg′.equivalence-in-bicategory-axioms* **by** *auto*
qed

lemma *quasi-inverse-hcomp*:
assumes *equivalence-map f* **and** *equivalence-map f′* **and** *equivalence-map* $(f \star f')$
shows $(f \star f')^{\sim} \cong f'^{\sim} \star f^{\sim}$
  using *assms quasi-inverses-some-quasi-inverse quasi-inverse-hcomp′ quasi-inverse-unique*
  by *metis*

lemma *quasi-inverse-respects-isomorphic*:
assumes *equivalence-map f* **and** *equivalence-map f′* **and** $f \cong f'$
shows $f^{\sim} \cong f'^{\sim}$
proof −
  have *hpar*: $src\ f = src\ f' \wedge trg\ f = trg\ f'$
    using *assms isomorphic-implies-hpar* **by** *simp*
  have $f^{\sim} \cong f^{\sim} \star trg\ f$
    using *isomorphic-unit-right*
    by (*metis assms*(*1*) *isomorphic-symmetric quasi-inverse-simps*(*2*−*3*))
  also have ... $\cong f^{\sim} \star f' \star f'^{\sim}$
  proof −
    have $trg\ f \cong f' \star f'^{\sim}$
      using *assms quasi-inverse-hcomp*
      by (*simp add*: *comp-quasi-inverse*(*2*) *hpar isomorphic-symmetric*)
    thus *?thesis*
      using *assms hpar hcomp-ide-isomorphic isomorphic-implies-hpar*(*2*) **by** *auto*
  qed
  also have ... $\cong (f^{\sim} \star f') \star f'^{\sim}$
  using *assms hcomp-assoc-isomorphic hpar isomorphic-implies-ide*(*2*) *isomorphic-symmetric*
    by *auto*

also have ... $\cong (f^{\sim} \star f) \star f'^{\sim}$
proof −
have $f^{\sim} \star f' \cong f^{\sim} \star f$
using *assms isomorphic-symmetric hcomp-ide-isomorphic isomorphic-implies-hpar*(*1*)
by *auto*
thus *?thesis*
using *assms hcomp-isomorphic-ide isomorphic-implies-hpar*(*1*) **by** *auto*
qed
also have ... $\cong$ *src f* $\star f'^{\sim}$
proof −
have $f^{\sim} \star f \cong$ *src f*
using *assms comp-quasi-inverse* **by** *simp*
thus *?thesis*
using *assms hcomp-isomorphic-ide isomorphic-implies-hpar* **by** *simp*
qed
also have ... $\cong f'^{\sim}$
using *assms isomorphic-unit-left*
by (*metis hpar quasi-inverse-simps*(*2*,*4*))
finally show *?thesis* **by** *blast*
qed

end

1.9.4 Equivalent Objects

context *bicategory*
begin

definition *equivalent-objects*
where *equivalent-objects a b* $\equiv \exists f.$ «$f : a \to b$» $\wedge$ *equivalence-map f*

lemma *equivalent-objects-reflexive*:
assumes *obj a*
shows *equivalent-objects a a*
using *assms*
by (*metis equivalent-objects-def ide-in-hom*(*1*) *objE obj-is-equivalence-map*)

lemma *equivalent-objects-symmetric*:
assumes *equivalent-objects a b*
shows *equivalent-objects b a*
using *assms*
by (*metis equivalent-objects-def in-hhomE quasi-inverse-in-hom*(*1*) *quasi-inverse-simps*(*1*))

lemma *equivalent-objects-transitive* [*trans*]:
assumes *equivalent-objects a b* **and** *equivalent-objects b c*
shows *equivalent-objects a c*
proof −
obtain *f* **where** *f*: «$f : a \to b$» $\wedge$ *equivalence-map f*
using *assms equivalent-objects-def* **by** *auto*

obtain g **where** g: «$g : b \to c$» $\wedge$ *equivalence-map* g
using *assms equivalent-objects-def* **by** *auto*
have «$g \star f : a \to c$» $\wedge$ *equivalence-map* $(g \star f)$
using *f g equivalence-maps-compose* **by** *auto*
thus *?thesis*
using *equivalent-objects-def* **by** *auto*
qed

end

1.9.5 Transporting Arrows along Equivalences

We show in this section that transporting the arrows of one hom-category to another along connecting equivalence maps yields an equivalence of categories. This is useful, because it seems otherwise hard to establish that the transporting functor is full.

locale *two-equivalences-in-bicategory* =
bicategory V H a i *src trg* +
e_0: *equivalence-in-bicategory* V H a i *src trg* e_0 d_0 η_0 ε_0 +
e_1: *equivalence-in-bicategory* V H a i *src trg* e_1 d_1 η_1 ε_1
for V :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and e_0 :: $'a$
and d_0 :: $'a$
and η_0 :: $'a$
and ε_0 :: $'a$
and e_1 :: $'a$
and d_1 :: $'a$
and η_1 :: $'a$
and ε_1 :: $'a$
begin

interpretation *hom*: *subcategory* V ‹$\lambda\mu$. «μ : *src* $e_0 \to$ *src* e_1»›
using *hhom-is-subcategory* **by** *simp*

no-notation *in-hom* (‹«- : - → -»›)

interpretation *hom′*: *subcategory* V ‹$\lambda\mu$. «μ : *trg* $e_0 \to$ *trg* e_1»›
using *hhom-is-subcategory* **by** *simp*

no-notation *in-hom* (‹«- : - → -»›)

abbreviation (*input*) F
where $F \equiv \lambda\mu.\ e_1 \star \mu \star d_0$

interpretation F: *functor hom.comp hom′.comp F*
proof
  show *1*: $\bigwedge f.$ *hom.arr f* $\Longrightarrow$ *hom′.arr* $(e_1 \star f \star d_0)$
    using $hom.arr\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ *in-hhom-def* e_0*.antipar*(*1*−*2*) **by** *simp*
  show $\bigwedge f.$ $\neg$ *hom.arr f* $\Longrightarrow$ $e_1 \star f \star d_0 =$ *hom′.null*
    using *1* $hom.arr\text{-}char_{SbC}$ *hom′.null-char in-hhom-def*
    by (*metis* e_0*.antipar*(*1*) *hseqE hseq-char′ hcomp-simps*(*2*))
  show $\bigwedge f.$ *hom.arr f* $\Longrightarrow$ *hom′.dom* $(e_1 \star f \star d_0) = e_1 \star$ *hom.dom f* $\star\ d_0$
    using $hom.arr\text{-}char_{SbC}$ $hom.dom\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ $hom'.dom\text{-}char_{SbC}$
    by (*metis 1 hcomp-simps*(*3*) e_0*.ide-right* e_1*.ide-left hom′.inclusion hseq-char ide-char*)
  show $\bigwedge f.$ *hom.arr f* $\Longrightarrow$ *hom′.cod* $(e_1 \star f \star d_0) = e_1 \star$ *hom.cod f* $\star\ d_0$
    using $hom.arr\text{-}char_{SbC}$ $hom.cod\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ $hom'.cod\text{-}char_{SbC}$
    by (*metis 1 hcomp-simps*(*4*) e_0*.ide-right* e_1*.ide-left hom′.inclusion hseq-char ide-char*)
  show $\bigwedge g\ f.$ *hom.seq g f* $\Longrightarrow$
      $e_1 \star$ *hom.comp g f* $\star\ d_0 =$ *hom′.comp* $(e_1 \star g \star d_0)$ $(e_1 \star f \star d_0)$
  using *1* $hom.seq\text{-}char_{SbC}$ $hom.arr\text{-}char_{SbC}$ *hom.comp-char* $hom'.arr\text{-}char_{SbC}$ *hom′.comp-char*
      whisker-left [*of* e_1] *whisker-right* [*of* d_0]
    apply *auto*
     apply (*metis hseq-char seqE src-vcomp*)
    by (*metis hseq-char hseq-char′*)
qed

abbreviation (*input*) G
where $G \equiv \lambda\mu'.\ d_1 \star \mu' \star e_0$

interpretation G: *functor hom′.comp hom.comp G*
proof
  show *1*: $\bigwedge f.$ *hom′.arr f* $\Longrightarrow$ *hom.arr* $(d_1 \star f \star e_0)$
    using $hom.arr\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ *in-hhom-def* e_1*.antipar*(*1*) e_1*.antipar*(*2*)
    by *simp*
  show $\bigwedge f.$ $\neg$ *hom′.arr f* $\Longrightarrow$ $d_1 \star f \star e_0 =$ *hom.null*
    using *1* $hom.arr\text{-}char_{SbC}$ *hom′.null-char in-hhom-def*
    by (*metis* e_1*.antipar*(*2*) $hom'.arrI_{SbC}$ *hom.null-char hseqE hseq-char′ hcomp-simps*(*2*))
  show $\bigwedge f.$ *hom′.arr f* $\Longrightarrow$ *hom.dom* $(d_1 \star f \star e_0) = d_1 \star$ *hom′.dom f* $\star\ e_0$
    using *1* $hom.arr\text{-}char_{SbC}$ $hom.dom\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ $hom'.dom\text{-}char_{SbC}$
    by (*metis hcomp-simps*(*3*) e_0*.ide-left* e_1*.ide-right hom.inclusion hseq-char ide-char*)
  show $\bigwedge f.$ *hom′.arr f* $\Longrightarrow$ *hom.cod* $(d_1 \star f \star e_0) = d_1 \star$ *hom′.cod f* $\star\ e_0$
    using *1* $hom.arr\text{-}char_{SbC}$ $hom.cod\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ $hom'.cod\text{-}char_{SbC}$
    by (*metis hcomp-simps*(*4*) e_0*.ide-left* e_1*.ide-right hom.inclusion hseq-char ide-char*)
  show $\bigwedge g\ f.$ *hom′.seq g f* $\Longrightarrow$
      $d_1 \star$ *hom′.comp g f* $\star\ e_0 =$ *hom.comp* $(d_1 \star g \star e_0)$ $(d_1 \star f \star e_0)$
  using *1* $hom'.seq\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ *hom′.comp-char* $hom.arr\text{-}char_{SbC}$ *hom.comp-char*
      whisker-left [*of* d_1] *whisker-right* [*of* e_0]
    apply *auto*
     apply (*metis hseq-char seqE src-vcomp*)
    by (*metis hseq-char hseq-char′*)
qed

interpretation GF: *composite-functor hom.comp hom′.comp hom.comp F G* **..**

interpretation *FG*: *composite-functor hom′.comp hom.comp hom′.comp G F* **..**

abbreviation (*input*) φ_0
where $\varphi_0\ f \equiv (d_1 \star \mathrm{a}^{-1}[e_1,\ f \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,\ (f \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[f,\ d_0,\ e_0]) \cdot (\eta_1 \star f \star \eta_0) \cdot \mathrm{l}^{-1}[f \star src\ e_0] \cdot \mathrm{r}^{-1}[f]$

lemma φ_0*-in-hom*:
assumes «f : *src* $e_0 \rightarrow$ *src* e_1» **and** *ide f*
shows «$\varphi_0\ f$: *src* $e_0 \rightarrow$ *src* e_1»
and «$\varphi_0\ f : f \Rightarrow d_1 \star (e_1 \star f \star d_0) \star e_0$»
proof −
  show «$\varphi_0\ f : f \Rightarrow d_1 \star (e_1 \star f \star d_0) \star e_0$»
    using *assms* e_0*.antipar* e_1*.antipar* **by** *fastforce*
  thus «$\varphi_0\ f$: *src* $e_0 \rightarrow$ *src* e_1»
    using *assms src-dom* [*of* $\varphi_0\ f$] *trg-dom* [*of* $\varphi_0\ f$]
    by (*metis arrI dom-comp in-hhom-def runit′-simps*(*4*) *seqE*)
qed

lemma *iso-*φ_0:
assumes «f : *src* $e_0 \rightarrow$ *src* e_1» **and** *ide f*
shows *iso* ($\varphi_0\ f$)
  using *assms iso-lunit′ iso-runit′* e_0*.antipar* e_1*.antipar*
  by (*intro isos-compose*) *auto*

interpretation φ: *transformation-by-components hom.comp hom.comp hom.map* ‹*G o F*› φ_0
proof
  fix *f*
  assume *f*: *hom.ide f*
  show *hom.in-hom* ($\varphi_0\ f$) (*hom.map f*) (*GF.map f*)
  proof (*unfold hom.in-hom-char*$_{SbC}$, *intro conjI*)
    show *hom.arr* (*hom.map f*)
      using *f* **by** *simp*
    show *hom.arr* (*GF.map f*)
      using *f* **by** *simp*
    show *hom.arr* ($\varphi_0\ f$)
      using *f hom.ide-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ φ_0*-in-hom* **by** *simp*
    show «$\varphi_0\ f$: *hom.map f* ⇒ *GF.map f*»
    using *f hom.ide-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ *hom′.ide-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$ *F.preserves-arr*
      φ_0*-in-hom*
      by *auto*
  qed
  next
  fix μ
  assume μ: *hom.arr* μ
  show *hom.comp* (φ_0 (*hom.cod* μ)) (*hom.map* μ) =
    hom.comp (*GF.map* μ) (φ_0 (*hom.dom* μ))
  proof −
    have *hom.comp* (φ_0 (*hom.cod* μ)) (*hom.map* μ) = φ_0 (*cod* μ) $\cdot$ μ
    proof −

have *hom.map* $\mu = \mu$
using μ **by** *simp*
moreover have φ_0 (*hom.cod* μ) = φ_0 (*cod* μ)
using μ *hom.arr-char*$_{SbC}$ *hom.cod-char*$_{SbC}$ **by** *simp*
moreover have *hom.arr* (φ_0 (*cod* μ))
using μ *hom.arr-char*$_{SbC}$ φ_0*-in-hom* [*of cod* μ]
by (*metis hom.arr-cod hom.cod-simp ide-cod in-hhom-def*)
moreover have *seq* (φ_0 (*cod* μ)) μ
proof
show *1*: «μ : *dom* $\mu \Rightarrow$ *cod* μ»
using μ *hom.arr-char*$_{SbC}$ **by** *auto*
show «φ_0 (*cod* μ) : *cod* $\mu \Rightarrow d_1 \star (e_1 \star$ *cod* $\mu \star d_0) \star e_0$»
using μ *hom.arr-char*$_{SbC}$ φ_0*-in-hom*
by (*metis 1 arrI hom.arr-cod hom.cod-simp ide-cod*)
qed
ultimately show *?thesis*
using μ *hom.comp-char* **by** *simp*
qed
also have ... = $(d_1 \star \mathrm{a}^{-1}[e_1,$ *cod* $\mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,$ (*cod* $\mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[$*cod* $\mu,\ d_0,\ e_0]) \cdot (\eta_1 \star$ *cod* $\mu \star \eta_0) \cdot$
$\mathrm{l}^{-1}[$*cod* $\mu \star$ *src* $e_0] \cdot \mathrm{r}^{-1}[$*cod* $\mu] \cdot \mu$
using μ *hom.arr-char*$_{SbC}$ *comp-assoc* **by** *auto*
also have ... = $((d_1 \star \mathrm{a}^{-1}[e_1,$ *cod* $\mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,$ (*cod* $\mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[$*cod* $\mu,\ d_0,\ e_0]) \cdot (\eta_1 \star$ *cod* $\mu \star \eta_0) \cdot$
$\mathrm{l}^{-1}[$*cod* $\mu \star$ *src* $e_0] \cdot (\mu \star$ *src* $e_0)) \cdot \mathrm{r}^{-1}[$*dom* $\mu]$
using μ *hom.arr-char*$_{SbC}$ *runit′-naturality* [*of* μ] *comp-assoc* **by** *auto*
also have ... = $((d_1 \star \mathrm{a}^{-1}[e_1,$ *cod* $\mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,$ (*cod* $\mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[$*cod* $\mu,\ d_0,\ e_0]) \cdot (\eta_1 \star$ *cod* $\mu \star \eta_0) \cdot$
(*src* $e_1 \star \mu \star$ *src* $e_0) \cdot \mathrm{l}^{-1}[$*dom* $\mu \star$ *src* $e_0]) \cdot \mathrm{r}^{-1}[$*dom* $\mu]$
using μ *hom.arr-char*$_{SbC}$ *lunit′-naturality* [*of* $\mu \star$ *src* e_0] **by** *force*
also have ... = $((d_1 \star \mathrm{a}^{-1}[e_1,$ *cod* $\mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,$ (*cod* $\mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[$*cod* $\mu,\ d_0,\ e_0]) \cdot (\eta_1 \star$ *cod* $\mu \star \eta_0) \cdot$
(*src* $e_1 \star \mu \star$ *src* $e_0)) \cdot \mathrm{l}^{-1}[$*dom* $\mu \star$ *src* $e_0] \cdot \mathrm{r}^{-1}[$*dom* $\mu]$
using *comp-assoc* **by** *simp*
also have ... = $((d_1 \star \mathrm{a}^{-1}[e_1,$ *cod* $\mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,$ (*cod* $\mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star \mathrm{a}^{-1}[$*cod* $\mu,\ d_0,\ e_0]) \cdot ((d_1 \star e_1) \star \mu \star d_0 \star e_0)) \cdot$
$(\eta_1 \star$ *dom* $\mu \star \eta_0) \cdot \mathrm{l}^{-1}[$*dom* $\mu \star$ *src* $e_0] \cdot \mathrm{r}^{-1}[$*dom* $\mu]$
proof −
have $(\eta_1 \star$ *cod* $\mu \star \eta_0) \cdot$ (*src* $e_1 \star \mu \star$ *src* $e_0) = \eta_1 \star \mu \star \eta_0$
using μ *hom.arr-char*$_{SbC}$ *comp-arr-dom comp-cod-arr*
interchange [*of* η_1 *src* e_1 *cod* $\mu \star \eta_0$ $\mu \star$ *src* e_0]
interchange [*of cod* μ μ η_0 *src* e_0]
by (*metis* e_0*.unit-in-hom*(*1*) e_0*.unit-simps*(*2*) e_1*.unit-simps*(*1*) e_1*.unit-simps*(*2*)
hseqI′ in-hhom-def)
also have ... = $((d_1 \star e_1) \star \mu \star d_0 \star e_0) \cdot (\eta_1 \star$ *dom* $\mu \star \eta_0)$
proof −
have $\eta_1 \star \mu \star \eta_0 = (d_1 \star e_1) \cdot \eta_1 \star \mu \cdot$ *dom* $\mu \star (d_0 \star e_0) \cdot \eta_0$
using μ *hom.arr-char*$_{SbC}$ *comp-arr-dom comp-cod-arr* **by** *auto*
also have ... = $(d_1 \star e_1) \cdot \eta_1 \star (\mu \star d_0 \star e_0) \cdot$ (*dom* $\mu \star \eta_0)$

using μ *hom.arr-char*$_{SbC}$ *comp-cod-arr*
interchange [*of* μ *dom* μ $d_0 \star e_0$ η_0]
by *auto*
also have ... $= ((d_1 \star e_1) \star \mu \star d_0 \star e_0) \cdot (\eta_1 \star dom\ \mu \star \eta_0)$
using μ *hom.arr-char*$_{SbC}$ *comp-arr-dom comp-cod-arr*
interchange [*of* $d_1 \star e_1$ η_1 $\mu \star d_0 \star e_0$ *dom* $\mu \star \eta_0$]
interchange [*of* μ *dom* μ $d_0 \star e_0$ η_0]
by (*metis* e_0*.unit-in-hom*(*1*) e_0*.unit-simps*(*1*) e_0*.unit-simps*(*3*) e_1*.unit-simps*(*1*) e_1*.unit-simps*(*3*) *hom.inclusion hseqI*)
finally show *?thesis* **by** *simp*
qed
finally have $(\eta_1 \star cod\ \mu \star \eta_0) \cdot (src\ e_1 \star \mu \star src\ e_0) =$
$((d_1 \star e_1) \star \mu \star d_0 \star e_0) \cdot (\eta_1 \star dom\ \mu \star \eta_0)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((d_1 \star \mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,\ (cod\ \mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star (\mu \star d_0) \star e_0) \cdot ((d_1 \star e_1) \star \mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0])) \cdot$
$(\eta_1 \star dom\ \mu \star \eta_0) \cdot \mathrm{l}^{-1}[dom\ \mu \star src\ e_0] \cdot \mathrm{r}^{-1}[dom\ \mu]$
using μ *hom.arr-char*$_{SbC}$ e_0*.antipar* e_1*.antipar assoc′-naturality* [*of* μ d_0 e_0]
whisker-left [*of* $d_1 \star e_1$ $\mathrm{a}^{-1}[cod\ \mu,\ d_0,\ e_0]$ $\mu \star d_0 \star e_0$]
whisker-left [*of* $d_1 \star e_1$ $(\mu \star d_0) \star e_0$ $\mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0]$]
by *auto*
also have ... $= ((d_1 \star \mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0]) \cdot \mathrm{a}[d_1,\ e_1,\ (cod\ \mu \star d_0) \star e_0] \cdot$
$((d_1 \star e_1) \star (\mu \star d_0) \star e_0)) \cdot ((d_1 \star e_1) \star \mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0]) \cdot$
$(\eta_1 \star dom\ \mu \star \eta_0) \cdot \mathrm{l}^{-1}[dom\ \mu \star src\ e_0] \cdot \mathrm{r}^{-1}[dom\ \mu]$
using *comp-assoc* **by** *simp*
also have ... $= ((d_1 \star \mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0]) \cdot (d_1 \star e_1 \star (\mu \star d_0) \star e_0) \cdot$
$\mathrm{a}[d_1,\ e_1,\ (dom\ \mu \star d_0) \star e_0]) \cdot ((d_1 \star e_1) \star \mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0]) \cdot$
$(\eta_1 \star dom\ \mu \star \eta_0) \cdot \mathrm{l}^{-1}[dom\ \mu \star src\ e_0] \cdot \mathrm{r}^{-1}[dom\ \mu]$
using μ *hom.arr-char*$_{SbC}$ e_0*.antipar* e_1*.antipar*
assoc-naturality [*of* d_1 e_1 $(\mu \star d_0) \star e_0$] *hseqI*
by *auto*
also have ... $= ((d_1 \star \mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0]) \cdot (d_1 \star e_1 \star (\mu \star d_0) \star e_0)) \cdot$
$\mathrm{a}[d_1,\ e_1,\ (dom\ \mu \star d_0) \star e_0] \cdot ((d_1 \star e_1) \star \mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0]) \cdot$
$(\eta_1 \star dom\ \mu \star \eta_0) \cdot \mathrm{l}^{-1}[dom\ \mu \star src\ e_0] \cdot \mathrm{r}^{-1}[dom\ \mu]$
using *comp-assoc* **by** *simp*
also have ... $= ((d_1 \star (e_1 \star \mu \star d_0) \star e_0) \cdot (d_1 \star \mathrm{a}^{-1}[e_1,\ dom\ \mu \star d_0,\ e_0])) \cdot$
$\mathrm{a}[d_1,\ e_1,\ (dom\ \mu \star d_0) \star e_0] \cdot ((d_1 \star e_1) \star \mathrm{a}^{-1}[dom\ \mu,\ d_0,\ e_0]) \cdot$
$(\eta_1 \star dom\ \mu \star \eta_0) \cdot \mathrm{l}^{-1}[dom\ \mu \star src\ e_0] \cdot \mathrm{r}^{-1}[dom\ \mu]$
using μ *hom.arr-char*$_{SbC}$ e_0*.antipar* e_1*.antipar*
assoc′-naturality [*of* e_1 $\mu \star d_0$ e_0]
whisker-left [*of* d_1 $\mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0]$ $e_1 \star (\mu \star d_0) \star e_0$]
whisker-left [*of* d_1 $(e_1 \star \mu \star d_0) \star e_0$ $\mathrm{a}^{-1}[e_1,\ dom\ \mu \star d_0,\ e_0]$]
by *auto*
also have ... $=$ *hom.comp* (*GF.map* μ) (φ_0 (*hom.dom* μ))
proof −
have *hom.arr* (*GF.map* μ)

using μ **by** *blast*
moreover have *hom.arr* (φ_0 (*hom.dom* μ))
using μ *hom.arr-char*$_{SbC}$ *hom.in-hom-char*$_{SbC}$ φ_0*-in-hom*(*1*) *hom.dom-closed hom.dom-simp*
hom.inclusion ide-dom
by *metis*
moreover have *seq* (*GF.map* μ) (φ_0 (*hom.dom* μ))
proof
show «φ_0 (*hom.dom* μ) : *dom* $\mu \Rightarrow d_1 \star (e_1 \star$ *dom* $\mu \star d_0) \star e_0$»
using μ *hom.arr-char*$_{SbC}$ *hom.dom-char*$_{SbC}$ *hom.in-hom-char*$_{SbC}$ φ_0*-in-hom hom.dom-closed*
hom.dom-simp hom.inclusion ide-dom
by *metis*
show «*GF.map* μ : $d_1 \star (e_1 \star$ *dom* $\mu \star d_0) \star e_0 \Rightarrow d_1 \star (e_1 \star$ *cod* $\mu \star d_0) \star e_0$»
using μ *hom.arr-char*$_{SbC}$ *hom'.arr-char*$_{SbC}$ *F.preserves-arr*
apply *simp*
apply (*intro hcomp-in-vhom*)
by (*auto simp add*: e_0*.antipar* e_1*.antipar in-hhom-def*)
qed
ultimately show *?thesis*
using μ *hom.comp-char comp-assoc hom.dom-simp* **by** *auto*
qed
finally show *?thesis* **by** *blast*
qed
qed

lemma *transformation-by-components-*φ_0:
shows *transformation-by-components hom.comp hom.comp hom.map* (*G o F*) φ_0
..

interpretation φ: *natural-isomorphism hom.comp hom.comp hom.map* ‹*G o F*› *φ.map*
proof
fix *f*
assume *hom.ide f*
hence *f*: *ide f* $\wedge$ «*f* : *src* $e_0 \rightarrow$ *src* e_1»
using *hom.ide-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ **by** *simp*
show *hom.iso* (φ*.map f*)
proof (*unfold hom.iso-char*$_{SbC}$ *hom.arr-char*$_{SbC}$, *intro conjI*)
show *iso* (φ*.map f*)
using *f iso-*φ_0 *φ.map-simp-ide hom.ide-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ **by** *simp*
moreover show «φ*.map f* : *src* $e_0 \rightarrow$ *src* e_1»
using *f hom.ide-char hom.arr-char*$_{SbC}$ **by** *blast*
ultimately show «*inv* (φ*.map f*) : *src* $e_0 \rightarrow$ *src* e_1»
by *auto*
qed
qed

lemma *natural-isomorphism-*φ:
shows *natural-isomorphism hom.comp hom.comp hom.map* (*G o F*) *φ.map*

..

definition φ
where $\varphi \equiv \varphi.map$

lemma φ-*ide-simp*:
assumes «f : $src\ e_0 \rightarrow src\ e_1$» **and** $ide\ f$
shows $\varphi\ f = \varphi_0\ f$
  unfolding φ-*def*
  using *assms* $hom.ide\text{-}char_{SbC}$ $hom.arr\text{-}char_{SbC}$ **by** *simp*

lemma φ-*components-are-iso*:
assumes «f : $src\ e_0 \rightarrow src\ e_1$» **and** $ide\ f$
shows $iso\ (\varphi\ f)$
using *assms* φ-*def* $\varphi.components\text{-}are\text{-}iso$ $hom.ide\text{-}char_{SbC}$ $hom.arr\text{-}char_{SbC}$ $hom.iso\text{-}char_{SbC}$
  by *simp*

lemma φ-*eq*:
shows $\varphi = (\lambda\mu.\ if$ «μ : $src\ e_0 \rightarrow src\ e_1$» $then\ \varphi_0\ (cod\ \mu) \cdot \mu\ else\ null)$
proof
  fix μ
  have $\neg$ «μ : $src\ e_0 \rightarrow src\ e_1$» $\Longrightarrow \varphi.map\ \mu = null$
    using $hom.comp\text{-}char$ $hom.null\text{-}char$ $hom.arr\text{-}char_{SbC}$
    by (*simp add*: $\varphi.extensionality$)
  moreover have «μ : $src\ e_0 \rightarrow src\ e_1$» $\Longrightarrow \varphi.map\ \mu = \varphi_0\ (cod\ \mu) \cdot \mu$
    unfolding $\varphi.map\text{-}def$
    apply *auto*
    using $hom.comp\text{-}char$ $hom.arr\text{-}char_{SbC}$ $hom.dom\text{-}simp$ $hom.cod\text{-}simp$
    apply *simp*
    by (*metis* (*no-types*, *lifting*) φ_0-*in-hom*(*1*) $hom.cod\text{-}closed$ $hom.inclusion$ $ide\text{-}cod$ $local.ext$)
  ultimately show $\varphi\ \mu = (if$ «μ : $src\ e_0 \rightarrow src\ e_1$» $then\ \varphi_0\ (cod\ \mu) \cdot \mu\ else\ null)$
    unfolding φ-*def* **by** *auto*
qed

lemma φ-*in-hom* [*intro*]:
assumes «μ : $src\ e_0 \rightarrow src\ e_1$»
shows «$\varphi\ \mu$: $src\ e_0 \rightarrow src\ e_1$»
and «$\varphi\ \mu$: $dom\ \mu \Rightarrow d_1 \star (e_1 \star cod\ \mu \star d_0) \star e_0$»
proof −
  show «$\varphi\ \mu$: $src\ e_0 \rightarrow src\ e_1$»
    using *assms* φ-*eq* φ-*def* $hom.arr\text{-}char_{SbC}$ $\varphi.preserves\text{-}reflects\text{-}arr$ **by** *presburger*
  show «$\varphi\ \mu$: $dom\ \mu \Rightarrow d_1 \star (e_1 \star cod\ \mu \star d_0) \star e_0$»
    unfolding φ-*eq*
    using *assms* **apply** *simp*
    apply (*intro comp-in-homI*)
        apply *auto*
  proof −
    show «$\mathrm{r}^{-1}[cod\ \mu]$: $cod\ \mu \Rightarrow cod\ \mu \star src\ e_0$»
      using *assms* **by** *auto*

show «$\mathrm{l}^{-1}[cod\ \mu \star src\ e_0] : cod\ \mu \star src\ e_0 \Rightarrow src\ e_1 \star cod\ \mu \star src\ e_0$»
using *assms* **by** *auto*
show «$\eta_1 \star cod\ \mu \star \eta_0 : src\ e_1 \star cod\ \mu \star src\ e_0 \Rightarrow (d_1 \star e_1) \star cod\ \mu \star (d_0 \star e_0)$»
using *assms* e_0.*unit-in-hom*(*2*) e_1.*unit-in-hom*(*2*)
apply (*intro hcomp-in-vhom*)
apply *auto*
by *fastforce*
show «$(d_1 \star e_1) \star \mathrm{a}^{-1}[cod\ \mu,\ d_0,\ e_0] :$
$(d_1 \star e_1) \star cod\ \mu \star d_0 \star e_0 \Rightarrow (d_1 \star e_1) \star (cod\ \mu \star d_0) \star e_0$»
using *assms assoc′-in-hom* e_0.*antipar*(*1−2*) e_1.*antipar*(*2*)
apply (*intro hcomp-in-vhom*) **by** *auto*
show «$\mathrm{a}[d_1,\ e_1,\ (cod\ \mu \star d_0) \star e_0] :$
$(d_1 \star e_1) \star (cod\ \mu \star d_0) \star e_0 \Rightarrow d_1 \star e_1 \star (cod\ \mu \star d_0) \star e_0$»
using *assms assoc-in-hom* e_0.*antipar*(*1−2*) e_1.*antipar*(*2*) **by** *auto*
show «$d_1 \star \mathrm{a}^{-1}[e_1,\ cod\ \mu \star d_0,\ e_0] :$
$d_1 \star e_1 \star (cod\ \mu \star d_0) \star e_0 \Rightarrow d_1 \star (e_1 \star cod\ \mu \star d_0) \star e_0$»
using *assms assoc′-in-hom* [*of* $d_1\ e_1 \star cod\ \mu \star d_0\ e_0$]
e_0.*antipar*(*1−2*) e_1.*antipar*(*1−2*)
apply (*intro hcomp-in-vhom*)
apply *auto*
by *fastforce*
qed
qed

lemma φ-*simps* [*simp*]:
assumes «$\mu : src\ e_0 \to src\ e_1$»
shows $arr\ (\varphi\ \mu)$
and $src\ (\varphi\ \mu) = src\ e_0$ **and** $trg\ (\varphi\ \mu) = src\ e_1$
and $dom\ (\varphi\ \mu) = dom\ \mu$ **and** $cod\ (\varphi\ \mu) = d_1 \star (e_1 \star cod\ \mu \star d_0) \star e_0$
using *assms* φ-*in-hom* **by** *auto*

interpretation d_0: *equivalence-in-bicategory* $V\ H$ a i $src\ trg\ d_0\ e_0$ ‹*inv* ε_0› ‹*inv* η_0›
using e_0.*dual-equivalence* **by** *simp*
interpretation d_1: *equivalence-in-bicategory* $V\ H$ a i $src\ trg\ d_1\ e_1$ ‹*inv* ε_1› ‹*inv* η_1›
using e_1.*dual-equivalence* **by** *simp*
interpretation d_0e_0: *two-equivalences-in-bicategory* $V\ H$ a i $src\ trg$
$d_0\ e_0$ ‹*inv* ε_0› ‹*inv* η_0› $d_1\ e_1$ ‹*inv* ε_1› ‹*inv* η_1›
..

interpretation ψ: *inverse-transformation hom′.comp hom′.comp hom′.map* ‹*F o G*› $d_0e_0.\varphi$
proof −
interpret ψ': *natural-isomorphism hom′.comp hom′.comp hom′.map* ‹*F o G*› $d_0e_0.\varphi$
using d_0e_0.*natural-isomorphism*-φ e_0.*antipar* e_1.*antipar* $d_0e_0.\varphi$-*eq* $d_0e_0.\varphi$-*def* **by** *metis*
show *inverse-transformation hom′.comp hom′.comp hom′.map* (*F o G*) $d_0e_0.\varphi$
..
qed

definition ψ
where $\psi \equiv \psi.map$

lemma *ψ-ide-simp*:
assumes «f': *trg* $e_0 \to$ *trg* e_1» **and** *ide* f'
shows $\psi\ f' = \mathrm{r}[f'] \cdot \mathrm{l}[f' \star \mathit{trg}\ e_0] \cdot (\varepsilon_1 \star f' \star \varepsilon_0) \cdot ((e_1 \star d_1) \star \mathrm{a}[f',\ e_0,\ d_0]) \cdot$
$\mathrm{a}^{-1}[e_1,\ d_1,\ (f' \star e_0) \star d_0] \cdot (e_1 \star \mathrm{a}[d_1,\ f' \star e_0,\ d_0])$
proof −
have *hom′.ide* f'
using *assms hom′.ide-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$ **by** *simp*
hence *ψ.map* f' = *hom′.inv* ($d_0 e_0.\varphi\ f'$)
using *assms* **by** *simp*
also have ... = *inv* ($d_0 e_0.\varphi\ f'$)
using *hom′.inv-char*$_{SbC}$ ‹*hom′.ide* f'› **by** *simp*
also have ... = *inv* ($d_0 e_0.\varphi_0\ f'$)
using *assms* e_0*.antipar* e_1*.antipar* $d_0 e_0$*.φ-eq* $d_0 e_0$*.φ-ide-simp* [*of* f'] **by** *simp*
also have ... = ((((*inv* $\mathrm{r}^{-1}[f']$ · *inv* $\mathrm{l}^{-1}[f' \star \mathit{trg}\ e_0]$) · *inv* (*inv* $\varepsilon_1 \star f' \star$ *inv* ε_0)) ·
inv $((e_1 \star d_1) \star \mathrm{a}^{-1}[f',\ e_0,\ d_0])$) · *inv* $\mathrm{a}[e_1,\ d_1,\ (f' \star e_0) \star d_0]$) ·
inv $(e_1 \star \mathrm{a}^{-1}[d_1,\ f' \star e_0,\ d_0])$
proof −

There has to be a better way to do this.

have *1*: $\bigwedge A\ B\ C\ D\ E\ F.$
⟦ *iso* A; *iso* B; *iso* C; *iso* D; *iso* E; *iso* F;
arr (((((($A \cdot B$) · C) · D) · E) · F) ⟧ ⟹
inv (((((($A \cdot B$) · C) · D) · E) · F) =
inv F · *inv* E · *inv* D · *inv* C · *inv* B · *inv* A
using *inv-comp isos-compose seqE* **by** *metis*
have *arr* ($d_0 e_0.\varphi_0\ f'$)
using *assms* e_0*.antipar*(*2*) e_1*.antipar*(*2*) $d_0 e_0$*.iso-φ*$_0$ [*of* f'] *iso-is-arr* **by** *simp*
moreover have *iso* $\mathrm{r}^{-1}[f']$
using *assms iso-runit′* **by** *simp*
moreover have *iso* $\mathrm{l}^{-1}[f' \star \mathit{trg}\ e_0]$
using *assms iso-lunit′* **by** *auto*
moreover have *iso* (*inv* $\varepsilon_1 \star f' \star$ *inv* ε_0)
using *assms* e_0*.antipar*(*2*) e_1*.antipar*(*2*) *in-hhom-def* **by** *simp*
moreover have *iso* $((e_1 \star d_1) \star \mathrm{a}^{-1}[f',\ e_0,\ d_0])$
using *assms* e_0*.antipar* e_1*.antipar*(*1*) e_1*.antipar*(*2*) *in-hhom-def iso-hcomp*
by (*metis calculation*(*1*) e_0*.ide-left* e_0*.ide-right* e_1*.ide-left* e_1*.ide-right hseqE*
ide-is-iso iso-assoc′ seqE)
moreover have *iso* $\mathrm{a}[e_1,\ d_1,\ (f' \star e_0) \star d_0]$
using *assms* e_0*.antipar* e_1*.antipar* **by** *auto*
moreover have *iso* $(e_1 \star \mathrm{a}^{-1}[d_1,\ f' \star e_0,\ d_0])$
using *assms* e_0*.antipar* e_1*.antipar*
by (*metis calculation*(*1*) e_0*.ide-left* e_0*.ide-right* e_1*.ide-left* e_1*.ide-right*
iso-hcomp ide-hcomp hseqE ideD(*1*) *ide-is-iso in-hhomE iso-assoc′*
seqE hcomp-simps(*1*−*2*))
ultimately show *?thesis*
using *1* [*of* $e_1 \star \mathrm{a}^{-1}[d_1,\ f' \star e_0,\ d_0]$ $\mathrm{a}[e_1,\ d_1,\ (f' \star e_0) \star d_0]$
$(e_1 \star d_1) \star \mathrm{a}^{-1}[f',\ e_0,\ d_0]$ *inv* $\varepsilon_1 \star f' \star$ *inv* ε_0 $\mathrm{l}^{-1}[f' \star \mathit{trg}\ e_0]$ $\mathrm{r}^{-1}[f']$]
comp-assoc

by (*metis* e_0.*antipar*(*2*))
qed
also have ... $= inv\ \mathrm{r}^{-1}[f'] \cdot inv\ \mathrm{l}^{-1}[f' \star trg\ e_0] \cdot inv\ (inv\ \varepsilon_1 \star f' \star inv\ \varepsilon_0) \cdot$
$inv\ ((e_1 \star d_1) \star \mathrm{a}^{-1}[f',\ e_0,\ d_0]) \cdot inv\ \mathrm{a}[e_1,\ d_1,\ (f' \star e_0) \star d_0] \cdot$
$inv\ (e_1 \star \mathrm{a}^{-1}[d_1,\ f' \star e_0,\ d_0])$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{r}[f'] \cdot \mathrm{l}[f' \star trg\ e_0] \cdot (\varepsilon_1 \star f' \star \varepsilon_0) \cdot ((e_1 \star d_1) \star \mathrm{a}[f',\ e_0,\ d_0]) \cdot$
$\mathrm{a}^{-1}[e_1,\ d_1,\ (f' \star e_0) \star d_0] \cdot (e_1 \star \mathrm{a}[d_1,\ f' \star e_0,\ d_0])$
proof −
have $inv\ \mathrm{r}^{-1}[f'] = \mathrm{r}[f']$
using *assms inv-inv iso-runit* **by** *blast*
moreover have $inv\ \mathrm{l}^{-1}[f' \star trg\ e_0] = \mathrm{l}[f' \star trg\ e_0]$
using *assms iso-lunit* **by** *auto*
moreover have $inv\ (inv\ \varepsilon_1 \star f' \star inv\ \varepsilon_0) = \varepsilon_1 \star f' \star \varepsilon_0$
proof −
have $src\ (inv\ \varepsilon_1) = trg\ f'$
using *assms*(*1*) e_1.*antipar*(*2*) **by** *auto*
moreover have $src\ f' = trg\ (inv\ \varepsilon_0)$
using *assms*(*1*) e_0.*antipar*(*2*) **by** *auto*
ultimately show *?thesis*
using *assms*(*2*) e_0.*counit-is-iso* e_1.*counit-is-iso* **by** *simp*
qed
ultimately show *?thesis*
using *assms* e_0.*antipar* e_1.*antipar* **by** *auto*
qed
finally show *?thesis*
using ψ-*def* **by** *simp*
qed

lemma ψ-*components-are-iso*:
assumes «$f' : trg\ e_0 \to trg\ e_1$» **and** $ide\ f'$
shows $iso\ (\psi\ f')$
using *assms* ψ-*def* ψ.*components-are-iso* $hom'.ide\text{-}char_{SbC}$ $hom'.arr\text{-}char_{SbC}$ $hom'.iso\text{-}char_{SbC}$
by *simp*

lemma ψ-*eq*:
shows $\psi = (\lambda\mu'.\ if$ «μ': $trg\ e_0 \to trg\ e_1$» $then$
$\mu' \cdot \mathrm{r}[dom\ \mu'] \cdot \mathrm{l}[dom\ \mu' \star trg\ e_0] \cdot (\varepsilon_1 \star dom\ \mu' \star \varepsilon_0) \cdot$
$((e_1 \star d_1) \star \mathrm{a}[dom\ \mu',\ e_0,\ d_0]) \cdot \mathrm{a}^{-1}[e_1,\ d_1,\ (dom\ \mu' \star e_0) \star d_0] \cdot$
$(e_1 \star \mathrm{a}[d_1,\ dom\ \mu' \star e_0,\ d_0])$
$else\ null)$
proof
fix μ'
have $\neg$ «μ': $trg\ e_0 \to trg\ e_1$» $\Longrightarrow \psi.map\ \mu' = null$
using ψ.*extensionality* $hom'.arr\text{-}char_{SbC}$ $hom'.null\text{-}char$ **by** *simp*
moreover have «μ': $trg\ e_0 \to trg\ e_1$» $\Longrightarrow$
$\psi.map\ \mu' = \mu' \cdot \mathrm{r}[dom\ \mu'] \cdot \mathrm{l}[dom\ \mu' \star trg\ e_0] \cdot (\varepsilon_1 \star dom\ \mu' \star \varepsilon_0) \cdot$
$((e_1 \star d_1) \star \mathrm{a}[dom\ \mu',\ e_0,\ d_0]) \cdot \mathrm{a}^{-1}[e_1,\ d_1,\ (dom\ \mu' \star e_0) \star d_0] \cdot$
$(e_1 \star \mathrm{a}[d_1,\ dom\ \mu' \star e_0,\ d_0])$

```
  proof −
    assume μ′: «μ′: trg e₀ → trg e₁»
    have «ψ.map (dom μ′) : trg e₀ → trg e₁»
      using μ′ hom′.arr-char_SbC hom′.dom-closed by auto
    moreover have seq μ′ (ψ.map (dom μ′))
    proof −
      have hom′.seq μ′ (ψ.map (dom μ′))
        using μ′ ψ.preserves-cod hom′.arrI_SbC hom′.dom-simp hom′.cod-simp
        apply (intro hom′.seqI) by auto
      thus ?thesis
        using hom′.seq-char_SbC by blast
    qed
    ultimately show ?thesis
      using μ′ ψ.naturality1 [of μ′] hom′.comp-char hom′.arr-char_SbC hom′.dom-closed
            ψ-ide-simp [of dom μ′] hom′.dom-simp hom′.cod-simp
      apply auto
      by (metis ψ-def hom′.inclusion ide-dom)
  qed
  ultimately show ψ μ′ = (if «μ′ : trg e₀ → trg e₁» then
                            μ′ · r[dom μ′] · l[dom μ′ ⋆ trg e₀] · (ε₁ ⋆ dom μ′ ⋆ ε₀) ·
                              ((e₁ ⋆ d₁) ⋆ a[dom μ′, e₀, d₀]) ·
                              a⁻¹[e₁, d₁, (dom μ′ ⋆ e₀) ⋆ d₀] ·
                              (e₁ ⋆ a[d₁, dom μ′ ⋆ e₀, d₀])
                          else null)
    unfolding ψ-def by argo
qed

lemma ψ-in-hom [intro]:
assumes «μ′ : trg e₀ → trg e₁»
shows «ψ μ′ : trg e₀ → trg e₁»
and «ψ μ′ : e₁ ⋆ (d₁ ⋆ dom μ′ ⋆ e₀) ⋆ d₀ ⇒ cod μ′»
proof −
  have 0: ψ μ′ = ψ.map μ′
    using ψ-def by auto
  hence 1: hom′.arr (ψ μ′)
    using assms hom′.arr-char_SbC ψ.preserves-reflects-arr by auto
  show «ψ μ′ : trg e₀ → trg e₁»
    using 1 hom′.arr-char_SbC by blast
  show «ψ μ′ : e₁ ⋆ (d₁ ⋆ dom μ′ ⋆ e₀) ⋆ d₀ ⇒ cod μ′»
    using assms 0 1 ψ.preserves-hom hom′.in-hom-char_SbC hom′.arr-char_SbC
          e₀.antipar e₁.antipar ψ.preserves-dom ψ.preserves-cod hom′.dom-char_SbC
    apply (intro in-homI)
      apply auto[1]
  proof −
    show dom (ψ μ′) = e₁ ⋆ (d₁ ⋆ dom μ′ ⋆ e₀) ⋆ d₀
    proof −
      have hom′.dom (ψ μ′) = FG.map (hom′.dom μ′)
        using assms 0 ψ.preserves-dom hom′.arr-char_SbC
        by (metis (no-types, lifting))
```

thus *?thesis*
using *assms 0 1 hom.arr-char$_{SbC}$ hom′.arr-char$_{SbC}$ hom′.dom-char$_{SbC}$ G.preserves-arr hom′.dom-closed*
by *auto*
qed
show *cod* (ψ μ') = *cod* μ'
proof −
have *hom′.cod* (ψ μ') = *cod* μ'
using *assms 0 ψ.preserves-cod hom′.arr-char$_{SbC}$ hom′.cod-simp* **by** *auto*
thus *?thesis*
using *assms 0 1 hom′.arr-char$_{SbC}$ hom′.cod-char$_{SbC}$ G.preserves-arr hom′.cod-closed*
by *auto*
qed
qed
qed

lemma *ψ-simps* [*simp*]:
assumes «μ' : *trg* e_0 → *trg* e_1»
shows *arr* (ψ μ')
and *src* (ψ μ') = *trg* e_0 **and** *trg* (ψ μ') = *trg* e_1
and *dom* (ψ μ') = e_1 ⋆ (d_1 ⋆ *dom* μ' ⋆ e_0) ⋆ d_0 **and** *cod* (ψ μ') = *cod* μ'
using *assms ψ-in-hom* **by** *auto*

interpretation *equivalence-of-categories hom′.comp hom.comp F G φ ψ*
proof −
interpret φ: *natural-isomorphism hom.comp hom.comp hom.map ‹G o F› φ*
using *φ.natural-isomorphism-axioms φ-def* **by** *simp*
interpret ψ: *natural-isomorphism hom′.comp hom′.comp ‹F o G› hom′.map ψ*
using *ψ.natural-isomorphism-axioms ψ-def* **by** *simp*
show *equivalence-of-categories hom′.comp hom.comp F G φ ψ*
..
qed

lemma *induces-equivalence-of-hom-categories*:
shows *equivalence-of-categories hom′.comp hom.comp F G φ ψ*
..

lemma *equivalence-functor-F*:
shows *equivalence-functor hom.comp hom′.comp F*
proof −
interpret φ': *inverse-transformation hom.comp hom.comp hom.map ‹G o F› φ* **..**
interpret ψ': *inverse-transformation hom′.comp hom′.comp ‹F o G› hom′.map ψ* **..**
interpret *E*: *equivalence-of-categories hom.comp hom′.comp G F ψ'.map φ'.map* **..**
show *?thesis*
using *E.equivalence-functor-axioms* **by** *simp*
qed

lemma *equivalence-functor-G*:
shows *equivalence-functor hom′.comp hom.comp G*

using *equivalence-functor-axioms* **by** *simp*

end

context *bicategory*
begin

We now use the just-established equivalence of hom-categories to prove some cancellation laws for equivalence maps. It is relatively straightforward to prove these results directly, without using the just-established equivalence, but the proofs are somewhat longer that way.

lemma *equivalence-cancel-left*:
assumes *equivalence-map e*
and *par* μ μ' **and** *src e* = *trg* μ **and** $e \star \mu = e \star \mu'$
shows $\mu = \mu'$
proof −
  obtain *d* η ε **where** $d\eta\varepsilon$: *equivalence-in-bicategory V H* a i *src trg e d* η ε
    using *assms equivalence-map-def* **by** *auto*
  interpret *e*: *equivalence-in-bicategory V H* a i *src trg e d* η ε
    using $d\eta\varepsilon$ **by** *simp*
  interpret *i*: *equivalence-in-bicategory V H* a i *src trg*
        ‹*src* μ› ‹*src* μ› ‹*inv* i[*src* μ]› ‹i[*src* μ]›
    using *assms iso-unit obj-src*
    by *unfold-locales simp-all*
  interpret *two-equivalences-in-bicategory V H* a i *src trg*
        ‹*src* μ› ‹*src* μ› ‹*inv* i[*src* μ]› ‹i[*src* μ]› *e d* η ε
    ..
  interpret *hom*: *subcategory V* ‹$\lambda\mu'$. *in-hhom* μ' (*src* (*src* μ)) (*src e*)›
    using *hhom-is-subcategory* **by** *blast*
  interpret *hom'*: *subcategory V* ‹$\lambda\mu'$. *in-hhom* μ' (*trg* (*src* μ)) (*trg e*)›
    using *hhom-is-subcategory* **by** *blast*
  interpret *F*: *equivalence-functor hom.comp hom'.comp* ‹$\lambda\mu'$. $e \star \mu' \star$ *src* μ›
    using *equivalence-functor-F* **by** *simp*
  have $F\ \mu = F\ \mu'$
    using *assms* $hom.arr\text{-}char_{SbC}$
    apply *simp*
    by (*metis e.ide-left hcomp-reassoc*(*2*) *i.ide-right ideD*(*1*) *src-dom trg-dom trg-src*)
  moreover have *hom.par* μ μ'
    using *assms* $hom.arr\text{-}char_{SbC}$ $hom.dom\text{-}char_{SbC}$ $hom.cod\text{-}char_{SbC}$
    by (*metis* (*no-types*, *lifting*) *in-hhomI src-dom src-src trg-dom*)
  ultimately show $\mu = \mu'$
    using *F.is-faithful* **by** *blast*
qed

lemma *equivalence-cancel-right*:
assumes *equivalence-map e*
and *par* μ μ' **and** *src* μ = *trg e* **and** $\mu \star e = \mu' \star e$
shows $\mu = \mu'$
proof −

obtain $d\ \eta\ \varepsilon$ **where** $d\eta\varepsilon$: *equivalence-in-bicategory* $V\ H$ a i *src trg* $e\ d\ \eta\ \varepsilon$
using *assms equivalence-map-def* **by** *auto*
interpret e: *equivalence-in-bicategory* $V\ H$ a i *src trg* $e\ d\ \eta\ \varepsilon$
using $d\eta\varepsilon$ **by** *simp*
interpret i: *equivalence-in-bicategory* $V\ H$ a i *src trg*
‹*trg* μ› ‹*trg* μ› ‹*inv* i[*trg* μ]› ‹i[*trg* μ]›
using *assms iso-unit obj-src*
by *unfold-locales simp-all*
interpret *two-equivalences-in-bicategory* $V\ H$ a i *src trg*
$e\ d\ \eta\ \varepsilon$ ‹*trg* μ› ‹*trg* μ› ‹*inv* i[*trg* μ]› ‹i[*trg* μ]›
..
interpret *hom*: *subcategory* V ‹$\lambda\mu'$. *in-hhom* μ' (*trg* e) (*trg* (*trg* μ))›
using *hhom-is-subcategory* **by** *blast*
interpret *hom'*: *subcategory* V ‹$\lambda\mu'$. *in-hhom* μ' (*src* e) (*src* (*trg* μ))›
using *hhom-is-subcategory* **by** *blast*
interpret G: *equivalence-functor hom.comp hom'.comp* ‹$\lambda\mu'$. *trg* $\mu \star \mu' \star e$›
using *equivalence-functor-G* **by** *simp*
have $G\ \mu = G\ \mu'$
using *assms* $hom.arr\text{-}char_{SbC}$ **by** *simp*
moreover have *hom.par* $\mu\ \mu'$
using *assms* $hom.arr\text{-}char_{SbC}$ $hom.dom\text{-}char_{SbC}$ $hom.cod\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *in-hhomI src-dom trg-dom trg-trg*)
ultimately show $\mu = \mu'$
using *G.is-faithful* **by** *blast*
qed

lemma *equivalence-isomorphic-cancel-left*:
assumes *equivalence-map* e **and** *ide* f **and** *ide* f'
and *src* f = *src* f' **and** *src* e = *trg* f **and** $e \star f \cong e \star f'$
shows $f \cong f'$
proof −
have ef': *src* e = *trg* f'
using *assms R.as-nat-iso.components-are-iso iso-is-arr isomorphic-implies-hpar*(*2*)
by *blast*
obtain $d\ \eta\ \varepsilon$ **where** e: *equivalence-in-bicategory* $V\ H$ a i *src trg* $e\ d\ \eta\ \varepsilon$
using *assms equivalence-map-def* **by** *auto*
interpret e: *equivalence-in-bicategory* $V\ H$ a i *src trg* $e\ d\ \eta\ \varepsilon$
using e **by** *simp*
interpret i: *equivalence-in-bicategory* $V\ H$ a i *src trg*
‹*src* f› ‹*src* f› ‹*inv* i[*src* f]› ‹i[*src* f]›
using *assms iso-unit obj-src*
by *unfold-locales auto*
interpret *two-equivalences-in-bicategory* $V\ H$ a i *src trg*
‹*src* f› ‹*src* f› ‹*inv* i[*src* f]› ‹i[*src* f]› $e\ d\ \eta\ \varepsilon$
..
interpret *hom*: *subcategory* V ‹$\lambda\mu'$. *in-hhom* μ' (*src* (*src* f)) (*src* e)›
using *hhom-is-subcategory* **by** *blast*
interpret *hom'*: *subcategory* V ‹$\lambda\mu'$. *in-hhom* μ' (*trg* (*src* f)) (*trg* e)›
using *hhom-is-subcategory* **by** *blast*

interpret F: *equivalence-functor hom.comp hom′.comp* ‹$\lambda\mu'.\ e \star \mu' \star src\ f$›
using *equivalence-functor-F* **by** *simp*
have *1*: $F\ f \cong F\ f'$
proof −
have $F\ f \cong (e \star f) \star src\ f$
using *assms hcomp-assoc-isomorphic equivalence-map-is-ide isomorphic-symmetric*
by *auto*
also have ... $\cong (e \star f') \star src\ f'$
using *assms ef′* **by** (*simp add*: *hcomp-isomorphic-ide*)
also have ... $\cong F\ f'$
using *assms ef′ hcomp-assoc-isomorphic equivalence-map-is-ide* **by** *auto*
finally show *?thesis* **by** *blast*
qed
show $f \cong f'$
proof −
obtain ψ **where** ψ: «$\psi : F\ f \Rightarrow F\ f'$» $\wedge$ *iso* ψ
using *1 isomorphic-def* **by** *auto*
have *2*: *hom′.iso* ψ
using *assms* ψ *hom′.iso-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$ *vconn-implies-hpar*(*1*,*2*)
by *auto*
have *3*: *hom′.in-hom* ψ $(F\ f)$ $(F\ f')$
using *assms 2* ψ *ef′ hom′.in-hom-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$
by (*metis F.preserves-reflects-arr hom′.iso-is-arr hom.arr-char*$_{SbC}$ *i.antipar*(*1*)
ideD(*1*) *ide-in-hom*(*1*) *trg-src*)
obtain φ **where** φ: *hom.in-hom* $\varphi\ f\ f' \wedge F\ \varphi = \psi$
using *assms 3* ψ *F.is-full F.preserves-reflects-arr hom′.in-hom-char*$_{SbC}$ *hom.ide-char*$_{SbC}$
by *fastforce*
have *hom.iso* φ
using *2* φ *F.reflects-iso* **by** *auto*
thus *?thesis*
using φ *isomorphic-def hom.in-hom-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ *hom.iso-char*$_{SbC}$ **by** *auto*
qed
qed

lemma *equivalence-isomorphic-cancel-right*:
assumes *equivalence-map e* **and** *ide f* **and** *ide f′*
and *trg f* = *trg f′* **and** *src f* = *trg e* **and** $f \star e \cong f' \star e$
shows $f \cong f'$
proof −
have *f′e*: *src f′* = *trg e*
using *assms R.as-nat-iso.components-are-iso iso-is-arr isomorphic-implies-hpar*(*2*)
by *blast*
obtain *d* η ε **where** *dηε*: *equivalence-in-bicategory V H* a i *src trg e d* η ε
using *assms equivalence-map-def* **by** *auto*
interpret *e*: *equivalence-in-bicategory V H* a i *src trg e d* η ε
using *dηε* **by** *simp*
interpret *i*: *equivalence-in-bicategory V H* a i *src trg*
‹*trg f*› ‹*trg f*› ‹*inv* i[*trg f*]› ‹i[*trg f*]›

using *assms iso-unit obj-src*
by *unfold-locales auto*
interpret *two-equivalences-in-bicategory V H* a i *src trg*
e d η ε ‹*trg f*› ‹*trg f*› ‹*inv* i[*trg f*]› ‹i[*trg f*]›
..
interpret *hom*: *subcategory V* ‹$\lambda\mu'$. *in-hhom* μ' (*trg e*) (*trg* (*trg f*))›
using *hhom-is-subcategory* **by** *blast*
interpret *hom′*: *subcategory V* ‹$\lambda\mu'$. *in-hhom* μ' (*src e*) (*src* (*trg f*))›
using *hhom-is-subcategory* **by** *blast*
interpret *G*: *equivalence-functor hom.comp hom′.comp* ‹$\lambda\mu'$. *trg f* $\star$ μ' $\star$ *e*›
using *equivalence-functor-G* **by** *simp*
have *1*: $G\ f \cong G\ f'$
using *assms hcomp-isomorphic-ide hcomp-ide-isomorphic* **by** *simp*
show $f \cong f'$
proof −
obtain ψ **where** ψ: «$\psi : G\ f \Rightarrow G\ f'$» $\wedge$ *iso* ψ
using *1 isomorphic-def* **by** *auto*
have *2*: *hom′.iso* ψ
using *assms* ψ *hom′.iso-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$ *vconn-implies-hpar*(*1*−*2*) **by** *auto*
have *3*: *hom′.in-hom* ψ ($G\ f$) ($G\ f'$)
using *assms 2* ψ *f′e hom′.in-hom-char*$_{SbC}$ *hom′.arr-char*$_{SbC}$
by (*metis G.preserves-arr hom′.iso-is-arr hom.ideI*$_{SbC}$ *hom.ide-char ideD*(*1*)
ide-in-hom(*1*) *trg-trg*)
obtain φ **where** φ: *hom.in-hom* φ $f\ f'$ $\wedge$ $G\ \varphi = \psi$
using *assms 3* ψ *G.is-full G.preserves-reflects-arr hom′.in-hom-char*$_{SbC}$ *hom.ide-char*$_{SbC}$
by *fastforce*
have *hom.iso* φ
using *2* φ *G.reflects-iso* **by** *auto*
thus *?thesis*
using φ *isomorphic-def hom.in-hom-char*$_{SbC}$ *hom.arr-char*$_{SbC}$ *hom.iso-char*$_{SbC}$ **by** *auto*
qed
qed

end

end

1.10 Pseudofunctors

theory *Pseudofunctor*
imports *MonoidalCategory.MonoidalFunctor Bicategory Subbicategory InternalEquivalence Coherence*
begin

The traditional definition of a pseudofunctor $F : C \to D$ between bicategories C and D is in terms of two maps: an "object map" F_o that takes objects of C to objects of D and an "arrow map" F_a that assigns to each pair of objects a and b of C a functor $F_a\ a\ b$ from the hom-category $hom_C\ a\ b$ to the hom-category $hom_D\ (F_o\ a)\ (F_o\ b)$. In addition,

there is assigned to each object a of C an invertible 2-cell «$\Psi\ a : F_o\ a \Rightarrow_D (F_a\ a\ a)\ a$», and to each pair $(f,\ g)$ of composable 1-cells of C there is assigned an invertible 2-cell «$\Phi\ (f,\ g) : F\ g \star F\ f \Rightarrow F\ (g \star f)$», all subject to naturality and coherence conditions.

In keeping with the "object-free" style in which we have been working, we do not wish to adopt a definition of pseudofunctor that distinguishes between objects and other arrows. Instead, we would like to understand a pseudofunctor as an ordinary functor between (vertical) categories that weakly preserves horizontal composition in a suitable sense. So, we take as a starting point that a pseudofunctor $F : C \to D$ is a functor from C to D, when these are regarded as ordinary categories with respect to vertical composition. Next, F should preserve source and target, but only "weakly" (up to isomorphism, rather than "on the nose"). Weak preservation of horizontal composition is expressed by specifying, for each horizontally composable pair of vertical identities $(f,\ g)$ of C, a "compositor" «$\Phi\ (f,\ g) : F\ g \star F\ f \Rightarrow F\ (g \star f)$» in D, such that the $\Phi\ (f,\ g)$ are the components of a natural isomorphism. Associators must also be weakly preserved by F; this is expressed by a coherence condition that relates an associator $\mathrm{a}_C[f,\ g,\ h]$ in C, its image $F\ \mathrm{a}_C[f,\ g,\ h]$, the associator $\mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$ in D and compositors involving f, g, and h. As regards the weak preservation of unitors, just as for monoidal functors, which are in fact pseudofunctors between one-object bicategories, it is only necessary to assume that $F\ \mathrm{i}_C[a]$ and $\mathrm{i}_D[F\ a]$ are isomorphic in D for each object a of C, for there is then a canonical way to obtain, for each a, an isomorphism «$\Psi\ a : src\ (F\ a) \to F\ a$» that satisfies the usual coherence conditions relating the unitors and the associators. Note that the map $a \mapsto src\ (F\ a)$ amounts to the traditional "object map" F_o, so that this becomes a derived notion, rather than a primitive one.

1.10.1 Weak Arrows of Homs

We begin with a locale that defines a functor between "horizontal homs" that preserves source and target up to isomorphism.

locale *weak-arrow-of-homs* =
  C: *horizontal-homs* $C\ src_C\ trg_C$ +
  D: *horizontal-homs* $D\ src_D\ trg_D$ +
  functor $C\ D\ F$
for C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$ +
assumes *weakly-preserves-src*: $\bigwedge\mu.\ C.arr\ \mu \Longrightarrow D.isomorphic\ (F\ (src_C\ \mu))\ (src_D\ (F\ \mu))$
and *weakly-preserves-trg*: $\bigwedge\mu.\ C.arr\ \mu \Longrightarrow D.isomorphic\ (F\ (trg_C\ \mu))\ (trg_D\ (F\ \mu))$
begin

  lemma *isomorphic-src*:
  assumes $C.obj\ a$
  shows $D.isomorphic\ (src_D\ (F\ a))\ (F\ a)$

using *assms weakly-preserves-src* [*of a*] *D.isomorphic-symmetric* **by** *auto*

lemma *isomorphic-trg*:
assumes $C.obj\ a$
shows $D.isomorphic\ (trg_D\ (F\ a))\ (F\ a)$
using *assms weakly-preserves-trg* [*of a*] *D.isomorphic-symmetric* **by** *auto*

abbreviation (*input*) $hseq_C$
where $hseq_C\ \mu\ \nu \equiv C.arr\ \mu \wedge C.arr\ \nu \wedge src_C\ \mu = trg_C\ \nu$

abbreviation (*input*) $hseq_D$
where $hseq_D\ \mu\ \nu \equiv D.arr\ \mu \wedge D.arr\ \nu \wedge src_D\ \mu = trg_D\ \nu$

lemma *preserves-hseq*:
assumes $hseq_C\ \mu\ \nu$
shows $hseq_D\ (F\ \mu)\ (F\ \nu)$
by (*metis D.isomorphic-def D.src-src D.src-trg D.vconn-implies-hpar*(*3*)
assms preserves-reflects-arr weakly-preserves-src weakly-preserves-trg)

Though F does not preserve objects "on the nose", we can recover from it the usual "object map", which does. It is slightly confusing at first to get used to the idea that applying the object map of a weak arrow of homs to an object does not give the same thing as applying the underlying functor, but rather only something isomorphic to it.

The following defines the object map associated with F.

definition map_0
where $map_0\ a \equiv src_D\ (F\ a)$

lemma map_0*-simps* [*simp*]:
assumes $C.obj\ a$
shows $D.obj\ (map_0\ a)$
and $src_D\ (map_0\ a) = map_0\ a$ **and** $trg_D\ (map_0\ a) = map_0\ a$
and $D.dom\ (map_0\ a) = map_0\ a$ **and** $D.cod\ (map_0\ a) = map_0\ a$
using *assms* map_0*-def* **by** *auto*

lemma *preserves-src* [*simp*]:
assumes $C.arr\ \mu$
shows $src_D\ (F\ \mu) = map_0\ (src_C\ \mu)$
using *assms*
by (*metis C.src.preserves-arr C.src-src C.trg-src* map_0*-def preserves-hseq*)

lemma *preserves-trg* [*simp*]:
assumes $C.arr\ \mu$
shows $trg_D\ (F\ \mu) = map_0\ (trg_C\ \mu)$
using *assms* map_0*-def preserves-hseq C.src-trg C.trg.preserves-arr* **by** *presburger*

lemma *preserves-hhom* [*intro*]:
assumes $C.arr\ \mu$
shows $D.in\text{-}hhom\ (F\ \mu)\ (map_0\ (src_C\ \mu))\ (map_0\ (trg_C\ \mu))$
using *assms* **by** *simp*

We define here the lifting of F to a functor FF: $CC \to DD$. We need this to define the domains and codomains of the compositors.

```
definition FF
where FF ≡ λμν. if C.VV.arr μν then (F (fst μν), F (snd μν)) else D.VV.null

sublocale FF: functor C.VV.comp D.VV.comp FF
proof −
  have 1: ⋀μν. C.VV.arr μν ⟹ D.VV.arr (FF μν)
    unfolding FF-def using C.VV.arr-char_SbC D.VV.arr-char_SbC preserves-hseq by simp
  show functor C.VV.comp D.VV.comp FF
  proof
    fix μν
    show ¬ C.VV.arr μν ⟹ FF μν = D.VV.null
      using FF-def by simp
    show C.VV.arr μν ⟹ D.VV.arr (FF μν)
      using 1 by simp
    assume μν: C.VV.arr μν
    show D.VV.dom (FF μν) = FF (C.VV.dom μν)
   using μν 1 FF-def C.VV.arr-char_SbC D.VV.arr-char_SbC C.VV.dom-simp D.VV.dom-simp
      by simp
    show D.VV.cod (FF μν) = FF (C.VV.cod μν)
   using μν 1 FF-def C.VV.arr-char_SbC D.VV.arr-char_SbC C.VV.cod-simp D.VV.cod-simp
      by simp
    next
    fix μν ττπ
    assume 2: C.VV.seq μν ττπ
    show FF (C.VV.comp μν ττπ) = D.VV.comp (FF μν) (FF ττπ)
    proof −
      have FF (C.VV.comp μν ττπ) = (F (fst μν) ·_D F (fst ττπ), F (snd μν) ·_D F (snd ττπ))
        using 1 2 FF-def C.VV.comp-char C.VxV.comp-char C.VV.arr-char_SbC
        by (metis (no-types, lifting) C.VV.seq-char_SbC C.VxV.seqE_PC fst-conv
            as-nat-trans.preserves-comp-2 snd-conv)
      also have ... = D.VV.comp (FF μν) (FF ττπ)
     using 1 2 FF-def D.VV.comp-char D.VxV.comp-char C.VV.arr-char_SbC D.VV.arr-char_SbC
              C.VV.seq-char_SbC C.VxV.seqE_PC preserves-seq
        by simp meson
      finally show ?thesis by simp
    qed
  qed
qed

lemma functor-FF:
shows functor C.VV.comp D.VV.comp FF
  ..

end
```

1.10.2 Definition of Pseudofunctors

I don't much like the term "pseudofunctor", which is suggestive of something that is "not really" a functor. In the development here we can see that a pseudofunctor is really a *bona fide* functor with respect to vertical composition, which happens to have in addition a weak preservation property with respect to horizontal composition. This weak preservation of horizontal composition is captured by extra structure, the "compositors", which are the components of a natural transformation. So "pseudofunctor" is really a misnomer; it's an actual functor that has been equipped with additional structure relating to horizontal composition. I would use the term "bifunctor" for such a thing, but it seems to not be generally accepted and also tends to conflict with the usage of that term to refer to an ordinary functor of two arguments; which I have called a "binary functor". Sadly, there seem to be no other plausible choices of terminology, other than simply "functor" (recommended on n-Lab https://ncatlab.org/nlab/show/pseudofunctor), but that is not workable here because we need a name that does not clash with that used for an ordinary functor between categories.

locale *pseudofunctor* =
  C: *bicategory* V_C H_C a_C i_C src_C trg_C +
  D: *bicategory* V_D H_D a_D i_D src_D trg_D +
  weak-arrow-of-homs V_C src_C trg_C V_D src_D trg_D F +
  FoH_C: *composite-functor* $C.VV.comp$ V_C V_D ‹$\lambda\mu\nu.$ H_C (*fst* $\mu\nu$) (*snd* $\mu\nu$)› F +
  $H_D oFF$: *composite-functor* $C.VV.comp$ $D.VV.comp$ V_D
    FF ‹$\lambda\mu\nu.$ H_D (*fst* $\mu\nu$) (*snd* $\mu\nu$)› +
  Φ: *natural-isomorphism* $C.VV.comp$ V_D $H_D oFF.map$ $FoH_C.map$ Φ
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ :: $'c * 'c \Rightarrow 'd$ +
assumes *assoc-coherence*:
  $[\![$ $C.ide\ f$; $C.ide\ g$; $C.ide\ h$; $src_C\ f = trg_C\ g$; $src_C\ g = trg_C\ h$ $]\!] \Longrightarrow$
    $F\ \mathrm{a}_C[f, g, h] \cdot_D \Phi\ (f \star_C g, h) \cdot_D (\Phi\ (f, g) \star_D F\ h) =$
    $\Phi\ (f, g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g, h)) \cdot_D \mathrm{a}_D[F\ f, F\ g, F\ h]$
begin

  no-notation $C.in\text{-}hom$ (‹«- : - $\rightarrow_C$ -»›)
  no-notation $D.in\text{-}hom$ (‹«- : - $\rightarrow_D$ -»›)
  notation $C.in\text{-}hhom$ (‹«- : - $\rightarrow_C$ -»›)

notation *C.in-hom* (‹«- : - $\Rightarrow_C$ -»›)
notation *D.in-hhom* (‹«- : - $\rightarrow_D$ -»›)
notation *D.in-hom* (‹«- : - $\Rightarrow_D$ -»›)

notation *C.lunit* (‹l_C[-]›)
notation *C.runit* (‹r_C[-]›)
notation *C.lunit′* (‹${\mathrm{l}_C}^{-1}$[-]›)
notation *C.runit′* (‹${\mathrm{r}_C}^{-1}$[-]›)
notation *C*.a′ (‹${\mathrm{a}_C}^{-1}$[-, -, -]›)
notation *D.lunit* (‹l_D[-]›)
notation *D.runit* (‹r_D[-]›)
notation *D.lunit′* (‹${\mathrm{l}_D}^{-1}$[-]›)
notation *D.runit′* (‹${\mathrm{r}_D}^{-1}$[-]›)
notation *D*.a′ (‹${\mathrm{a}_D}^{-1}$[-, -, -]›)

lemma *weakly-preserves-objects*:
assumes *C.obj a*
shows *D.isomorphic* (map_0 *a*) (*F a*)
  using *assms weakly-preserves-src* [*of a*] *D.isomorphic-symmetric* **by** *auto*

lemma *cmp-in-hom* [*intro*]:
assumes *C.ide a* **and** *C.ide b* **and** src_C *a* = trg_C *b*
shows «Φ (*a*, *b*) : map_0 (src_C *b*) $\rightarrow_D$ map_0 (trg_C *a*)»
and «Φ (*a*, *b*) : *F a* $\star_D$ *F b* $\Rightarrow_D$ *F* (*a* $\star_C$ *b*)»
proof −
  show «Φ (*a*, *b*) : *F a* $\star_D$ *F b* $\Rightarrow_D$ *F* (*a* $\star_C$ *b*)»
    using *assms* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def* **by** *auto*
  thus «Φ (*a*, *b*) : map_0 (src_C *b*) $\rightarrow_D$ map_0 (trg_C *a*)»
    using *assms D.vconn-implies-hpar* **by** *auto*
qed

lemma *cmp-simps* [*simp*]:
assumes *C.ide f* **and** *C.ide g* **and** src_C *f* = trg_C *g*
shows *D.arr* (Φ (*f*, *g*))
and src_D (Φ (*f*, *g*)) = src_D (*F g*) **and** trg_D (Φ (*f*, *g*)) = trg_D (*F f*)
and *D.dom* (Φ (*f*, *g*)) = *F f* $\star_D$ *F g* **and** *D.cod* (Φ (*f*, *g*)) = *F* (*f* $\star_C$ *g*)
  using *assms cmp-in-hom* **by** *simp-all blast+*

lemma *cmp-in-hom′*:
assumes *C.arr* μ **and** *C.arr* ν **and** src_C μ = trg_C ν
shows «Φ (μ, ν) : map_0 (src_C ν) $\rightarrow_D$ map_0 (trg_C μ)»
and «Φ (μ, ν) : *F* (*C.dom* μ) $\star_D$ *F* (*C.dom* ν) $\Rightarrow_D$ *F* (*C.cod* μ $\star_C$ *C.cod* ν)»
proof −
  show «Φ (μ, ν) : *F* (*C.dom* μ) $\star_D$ *F* (*C.dom* ν) $\Rightarrow_D$ *F* (*C.cod* μ $\star_C$ *C.cod* ν)»
    using *assms* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def* **by** *auto*
  thus «Φ (μ, ν) : map_0 (src_C ν) $\rightarrow_D$ map_0 (trg_C μ)»
    using *assms D.vconn-implies-hpar* **by** *auto*
qed

lemma *cmp-simps′*:
assumes $C.arr\ \mu$ **and** $C.arr\ \nu$ **and** $src_C\ \mu = trg_C\ \nu$
shows $D.arr\ (\Phi\ (\mu,\ \nu))$
and $src_D\ (\Phi\ (\mu,\ \nu)) = map_0\ (src_C\ \nu)$ **and** $trg_D\ (\Phi\ (\mu,\ \nu)) = map_0\ (trg_C\ \mu)$
and $D.dom\ (\Phi\ (\mu,\ \nu)) = F\ (C.dom\ \mu) \star_D F\ (C.dom\ \nu)$
and $D.cod\ (\Phi\ (\mu,\ \nu)) = F\ (C.cod\ \mu \star_C C.cod\ \nu)$
using *assms cmp-in-hom′* **by** *blast+*

lemma *cmp-components-are-iso* [*simp*]:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $src_C\ f = trg_C\ g$
shows $D.iso\ (\Phi\ (f,\ g))$
using *assms* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ **by** *simp*

lemma *weakly-preserves-hcomp*:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $src_C\ f = trg_C\ g$
shows $D.isomorphic\ (F\ f \star_D F\ g)\ (F\ (f \star_C g))$
using *assms D.isomorphic-def* **by** *auto*

end

context *pseudofunctor*
begin

The following defines the image of the unit isomorphism $\mathrm{i}_C[a]$ under F. We will use $(F\ a,\ \mathrm{i}[a])$ as an "alternate unit", to substitute for $(src_D\ (F\ a),\ \mathrm{i}_D[src_D\ (F\ a)])$.

abbreviation (*input*) i (‹i[-]›)
where $\mathrm{i}[a] \equiv F\ \mathrm{i}_C[a] \cdot_D \Phi\ (a,\ a)$

lemma i-*in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$F\ \mathrm{i}_C[a] \cdot_D \Phi\ (a,\ a) : map_0\ a \rightarrow_D map_0\ a$»
and «$\mathrm{i}[a] : F\ a \star_D F\ a \Rightarrow_D F\ a$»
proof (*unfold* map_0*-def*)
show «$F\ \mathrm{i}_C[a] \cdot_D \Phi\ (a,\ a) : F\ a \star_D F\ a \Rightarrow_D F\ a$»
using *assms preserves-hom cmp-in-hom*
by (*intro D.comp-in-homI*, *auto*)
show «$F\ \mathrm{i}_C[a] \cdot_D \Phi\ (a,\ a) : src_D\ (F\ a) \rightarrow_D src_D\ (F\ a)$»
using *assms* $C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
by (*intro D.vcomp-in-hhom D.seqI*, *auto*)
qed

lemma i-*simps* [*simp*]:
assumes $C.obj\ a$
shows $D.arr\ (\mathrm{i}\ a)$
and $src_D\ \mathrm{i}[a] = map_0\ a$ **and** $trg_D\ \mathrm{i}[a] = map_0\ a$
and $D.dom\ \mathrm{i}[a] = F\ a \star_D F\ a$ **and** $D.cod\ \mathrm{i}[a] = F\ a$
using *assms* i-*in-hom* **by** *auto*

lemma *iso*-i:

assumes $C.obj\ a$
shows $D.iso\ \mathrm{i}[a]$
  using *assms C.iso-unit C.obj-self-composable(1) C.seq-if-composable*
  by (*meson C.objE D.isos-compose* i-*simps(1) cmp-components-are-iso preserves-iso*)

If a is an object of C and we have an isomorphism «$\Phi\ (a,\ a) : F\ a \star_D F\ a \Rightarrow_D F\ (a \star_C a)$», then there is a canonical way to define a compatible isomorphism «$\Psi\ a : map_0\ a \Rightarrow_D F\ a$». Specifically, we take $\Psi\ a$ to be the unique isomorphism «$\psi : map_0\ a \Rightarrow_D F\ a$» such that $\psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$.

definition *unit*
where $unit\ a \equiv THE\ \psi.\ «\psi : map_0\ a \Rightarrow_D F\ a» \wedge D.iso\ \psi \wedge$
$\psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$

lemma *unit-char*:
assumes $C.obj\ a$
shows «$unit\ a : map_0\ a \Rightarrow_D F\ a$» **and** $D.iso\ (unit\ a)$
and $unit\ a \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (unit\ a \star_D unit\ a)$
and $\exists!\psi.\ «\psi : map_0\ a \Rightarrow_D F\ a» \wedge D.iso\ \psi \wedge \psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$
proof −
  let $?P = \lambda\psi.\ «\psi : map_0\ a \Rightarrow_D F\ a» \wedge D.iso\ \psi \wedge \psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$
  show $\exists!\psi.\ ?P\ \psi$
  proof −
    have $D.obj\ (map_0\ a)$
      using *assms* **by** *simp*
    moreover have $D.isomorphic\ (map_0\ a)\ (F\ a)$
      unfolding map_0-*def*
      using *assms isomorphic-src* **by** *simp*
    ultimately show *?thesis*
      using *assms D.unit-unique-upto-unique-iso* Φ.*preserves-hom* i-*in-hom iso*-i **by** *simp*
  qed
  hence *1*: $?P\ (unit\ a)$
    using *assms unit-def the1I2* [*of ?P ?P*] **by** *simp*
  show «$unit\ a : map_0\ a \Rightarrow_D F\ a$» **using** *1* **by** *simp*
  show $D.iso\ (unit\ a)$ **using** *1* **by** *simp*
  show $unit\ a \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (unit\ a \star_D unit\ a)$
    using *1* **by** *simp*
qed

lemma *unit-simps* [*simp*]:
assumes $C.obj\ a$
shows $D.arr\ (unit\ a)$
and $src_D\ (unit\ a) = map_0\ a$ **and** $trg_D\ (unit\ a) = map_0\ a$
and $D.dom\ (unit\ a) = map_0\ a$ **and** $D.cod\ (unit\ a) = F\ a$
  using *assms unit-char(1)*
      apply *auto*
   apply (*metis D.vconn-implies-hpar(1)* map_0-*simps(2)*)
  by (*metis D.vconn-implies-hpar(2)* map_0-*simps(3)*)

lemma *unit-in-hom* [*intro*]:

assumes $C.obj\ a$
shows «$unit\ a : map_0\ a \rightarrow_D map_0\ a$»
and «$unit\ a : map_0\ a \Rightarrow_D F\ a$»
  using *assms* **by** *auto*

lemma *unit-eqI*:
assumes $C.obj\ a$ **and** «μ: $map_0\ a \Rightarrow_D F\ a$» **and** $D.iso\ \mu$
and $\mu \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}\ a \cdot_D (\mu \star_D \mu)$
shows $\mu = unit\ a$
  using *assms unit-def unit-char*
    the1-equality [*of* $\lambda\mu$. «$\mu : map_0\ a \Rightarrow_D F\ a$» $\wedge$ $D.iso\ \mu$ $\wedge$
      $\mu \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\mu \star_D \mu)\ \mu$]
  by *simp*

The following defines the unique isomorphism satisfying the characteristic conditions for the left unitor $\mathrm{l}_D[trg_D\ (F\ f)]$, but using the "alternate unit" $\mathrm{i}[trg_C\ f]$ instead of $\mathrm{i}_D[trg_D\ (F\ f)]$, which is used to define $\mathrm{l}_D[trg_D\ (F\ f)]$.

definition *lF*
where $lF\ f \equiv THE\ \mu.$ «$\mu : F\ (trg_C\ f) \star_D F\ f \Rightarrow_D F\ f$» $\wedge$
    $F\ (trg_C\ f) \star_D \mu = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$

lemma *lF-char*:
assumes $C.ide\ f$
shows «$lF\ f : F\ (trg_C\ f) \star_D F\ f \Rightarrow_D F\ f$»
and $F\ (trg_C\ f) \star_D lF\ f = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$
and $\exists!\mu.$ «$\mu : F\ (trg_C\ f) \star_D F\ f \Rightarrow_D F\ f$» $\wedge$
    $F\ (trg_C\ f) \star_D \mu = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$
proof −
  let $?P = \lambda\mu.$ «$\mu : F\ (trg_C\ f) \star_D F\ f \Rightarrow_D F\ f$» $\wedge$
    $F\ (trg_C\ f) \star_D \mu = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$
  show $\exists!\mu.\ ?P\ \mu$
  proof −
    interpret *Df*: *prebicategory* ‹$(\cdot_D)$› ‹$(\star_D)$› a_D
      using *D.is-prebicategory* **by** *simp*
    interpret *S*: *subcategory* ‹$(\cdot_D)$› ‹$Df.left\ (F\ (trg_C\ f))$›
      using *assms Df.left-hom-is-subcategory* **by** *simp*
    interpret *Df*: *left-hom* ‹$(\cdot_D)$› ‹$(\star_D)$› ‹$F\ (trg_C\ f)$›
      using *assms D.weak-unit-char*
      by *unfold-locales simp*
    interpret *Df*: *left-hom-with-unit* ‹$(\cdot_D)$› ‹$(\star_D)$› a_D ‹$\mathrm{i}[trg_C\ f]$› ‹$F\ (trg_C\ f)$›
      using *assms* i-*in-hom iso*-i *D.weak-unit-char*(*1*) *assms weakly-preserves-trg*
      by *unfold-locales auto*
    have $\exists!\mu.$ «$\mu : Df.L\ (F\ f) \Rightarrow_S F\ f$» $\wedge$
      $Df.L\ \mu = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_S \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$
    proof −
      have $Df.left\ (F\ (trg_C\ f))\ (F\ f)$
        using *assms weakly-preserves-src D.isomorphic-def D.hseq-char D.hseq-char′*
          Df.left-def
        by *fastforce*

```
      thus ?thesis
        using assms Df.lunit-char(3) S.ide-char_SbC S.arr-char_SbC by simp
    qed
    moreover have Df.L (F f) = F (trg_C f) ⋆_D F f
      using assms by (simp add: Df.H_L-def)
    moreover have ⋀μ. Df.L μ = F (trg_C f) ⋆_D μ
      using Df.H_L-def by simp
    moreover have (i[trg_C f] ⋆_D F f) ·_S a_D^-1[F (trg_C f), F (trg_C f), F f] =
                  (i[trg_C f] ⋆_D F f) ·_D a_D^-1[F (trg_C f), F (trg_C f), F f]
      by (metis (no-types, lifting) D.arrI D.ext D.hseqI' D.hseq-char' D.seqE
          D.seq-if-composable D.vconn-implies-hpar(1) D.vconn-implies-hpar(2−3)
          D.vconn-implies-hpar(4) Df.ι-in-hom Df.arr-ω S.comp-char S.in-hom-char_SbC
          calculation(1,3))
    moreover have ⋀μ. «μ : F (trg_C f) ⋆_D F f ⇒_D F f» ⟷
                      «μ : F (trg_C f) ⋆_D F f ⇒_S F f»
      using assms S.in-hom-char_SbC S.arr-char_SbC
      by (metis D.in-homE Df.hom-connected(2) Df.left-def calculation(1−2))
    ultimately show ?thesis by simp
  qed
  hence 1: ?P (lF f)
    using lF-def the1I2 [of ?P ?P] by simp
  show «lF f : F (trg_C f) ⋆_D F f ⇒_D F f»
    using 1 by simp
  show F (trg_C f) ⋆_D lF f = (i[trg_C f] ⋆_D F f) ·_D a_D^-1[F (trg_C f), F (trg_C f), F f]
    using 1 by simp
qed

lemma lF-simps [simp]:
assumes C.ide f
shows D.arr (lF f)
and src_D (lF f) = map_0 (src_C f) and trg_D (lF f) = map_0 (trg_C f)
and D.dom (lF f) = F (trg_C f) ⋆_D F f and D.cod (lF f) = F f
  using assms lF-char(1)
      apply auto[5]
  unfolding map_0-def
  using assms
   apply (metis C.ideD(1) D.vconn-implies-hpar(1,3) map_0-def preserves-src)
  by (metis C.ideD(1) C.src-trg C.trg.preserves-arr D.in-homE D.trg-cod
      preserves-src preserves-trg)
```

The next two lemmas generalize the eponymous results from *MonoidalCategory.MonoidalFunctor*. See the proofs of those results for diagrams.

```
lemma lunit-coherence1:
assumes C.ide f
shows l_D[F f] ·_D D.inv (unit (trg_C f) ⋆_D F f) = lF f
proof −
  let ?b = trg_C f
  have 1: trg_D (F f) = map_0 ?b
    using assms by simp
```

```
have lF f ·D (unit ?b ⋆D F f) = 1D[F f]
proof −
  have D.par (lF f ·D (unit ?b ⋆D F f)) 1D[F f]
    using assms 1 D.lunit-in-hom unit-char(1−2) lF-char(1) D.ideD(1) by auto
  moreover have map0 ?b ⋆D (lF f ·D (unit ?b ⋆D F f)) = map0 ?b ⋆D 1D[F f]
  proof −
    have map0 ?b ⋆D (lF f ·D (unit ?b ⋆D F f)) =
          (map0 ?b ⋆D lF f) ·D (map0 ?b ⋆D unit ?b ⋆D F f)
      using assms D.objE [of map0 (trgC f)]
            D.whisker-left [of map0 ?b lF f unit ?b ⋆D F f]
      by auto
    also have ... = (map0 ?b ⋆D lF f) ·D
                    (D.inv (unit ?b) ⋆D F ?b ⋆D F f) ·D (unit ?b ⋆D unit ?b ⋆D F f)
    proof −
      have (D.inv (unit ?b) ⋆D F ?b ⋆D F f) ·D (unit ?b ⋆D unit ?b ⋆D F f) =
            D.inv (unit ?b) ·D unit ?b ⋆D F ?b ·D unit ?b ⋆D F f ·D F f
        using assms unit-char(1−2)
              D.interchange [of F ?b unit ?b F f F f]
              D.interchange [of D.inv (unit ?b) unit ?b F ?b ⋆D F f unit ?b ⋆D F f]
        by simp
      also have ... = map0 ?b ⋆D unit ?b ⋆D F f
       using assms unit-char(1−2) [of ?b] D.comp-arr-dom D.comp-cod-arr D.comp-inv-arr
        by (simp add: D.inv-is-inverse)
      finally show ?thesis by simp
    qed
    also have ... =
              (D.inv (unit ?b) ⋆D F f) ·D (F ?b ⋆D lF f) ·D (unit ?b ⋆D unit ?b ⋆D F f)
    proof −
      have (map0 ?b ⋆D lF f) ·D (D.inv (unit ?b) ⋆D F ?b ⋆D F f) =
            (D.inv (unit ?b) ⋆D F f) ·D (F ?b ⋆D lF f)
      proof −
        have (map0 ?b ⋆D lF f) ·D (D.inv (unit ?b) ⋆D F ?b ⋆D F f) =
              D.inv (unit ?b) ⋆D lF f
          using assms unit-char(1−2) lF-char(1) D.comp-arr-dom D.comp-cod-arr
                D.interchange [of map0 ?b D.inv (unit ?b) lF f F ?b ⋆D F f]
          by simp
        also have ... = D.inv (unit ?b) ·D F ?b ⋆D F f ·D lF f
          using assms unit-char(1−2) lF-char(1) D.comp-arr-dom D.comp-cod-arr
                D.inv-in-hom
          by auto
        also have ... = (D.inv (unit ?b) ⋆D F f) ·D (F ?b ⋆D lF f)
          using assms unit-char(1−2) lF-char(1) D.inv-in-hom
                D.interchange [of D.inv (unit ?b) F ?b F f lF f]
          by simp
        finally show ?thesis by simp
      qed
      thus ?thesis
        using assms unit-char(1−2) D.inv-in-hom D.comp-assoc by metis
    qed
```

```
      also have ... = (D.inv (unit ?b) ⋆D F f) ·D (i ?b ⋆D F f) ·D aD⁻¹[F ?b, F ?b, F f] ·D
                      (unit ?b ⋆D unit ?b ⋆D F f)
        using assms unit-char(1−2) lF-char(2) D.comp-assoc by auto
      also have ... = ((D.inv (unit ?b) ⋆D F f) ·D (i ?b ⋆D F f) ·D
                      ((unit ?b ⋆D unit ?b) ⋆D F f)) ·D aD⁻¹[map0 ?b, map0 ?b, F f]
       using assms unit-char(1−2) D.assoc′-naturality [of unit ?b unit ?b F f] D.comp-assoc
        by (simp add: ‹trgD (F f) = map0 (trgC f)›)
      also have ... = (iD[map0 ?b] ⋆D F f) ·D aD⁻¹[map0 ?b, map0 ?b, F f]
      proof −
        have ((D.inv (unit ?b) ⋆D F f) ·D (i ?b ⋆D F f) ·D ((unit ?b ⋆D unit ?b) ⋆D F f)) =
              iD[map0 ?b] ⋆D F f
        proof −
         have ((D.inv (unit ?b) ⋆D F f) ·D (i ?b ⋆D F f) ·D ((unit ?b ⋆D unit ?b) ⋆D F f)) =
                D.inv (unit ?b) ·D unit ?b ·D iD[map0 ?b] ⋆D F f
           using assms 1 D.unit-in-hom D.whisker-right [of F f] unit-char(2−3)
                 D.invert-side-of-triangle(1)
          by (metis C.ideD(1) C.obj-trg D.seqI′ map0-simps(1) unit-in-hom(2) preserves-ide)
          also have ... = iD[map0 ?b] ⋆D F f
          proof −
            have (D.inv (unit (trgC f)) ·D unit (trgC f)) ·D iD[map0 ?b] = iD[map0 ?b]
              by (simp add: D.comp-cod-arr D.comp-inv-arr D.inv-is-inverse unit-char(2)
                  assms)
            thus ?thesis
              by (simp add: D.comp-assoc)
          qed
          finally show ?thesis by blast
        qed
        thus ?thesis by simp
      qed
      also have ... = map0 ?b ⋆D lD[F f]
        using assms D.lunit-char [of F f] ‹trgD (F f) = map0 ?b› by simp
      finally show ?thesis by blast
    qed
    ultimately show ?thesis
      using assms D.L.is-faithful
      by (metis D.trg-cod D.trg-vcomp D.vseq-implies-hpar(2) lF-simps(3))
  qed
  thus ?thesis
    using assms 1 unit-char(1−2) C.ideD(1) C.obj-trg D.inverse-arrows-hcomp(1)
          D.invert-side-of-triangle(2) D.lunit-simps(1) unit-simps(2) preserves-ide
          D.iso-hcomp as-nat-iso.components-are-iso
    by metis
qed

lemma lunit-coherence2:
assumes C.ide f
shows lF f = F lC[f] ·D Φ (trgC f, f)
proof −
  let ?b = trgC f
```

```
have D.par (F l_C[f] ·_D Φ (?b, f)) (lF f)
  using assms cmp-simps'(1) cmp-simps(4−5) by force
moreover have F ?b ⋆_D F l_C[f] ·_D Φ (?b, f) = F ?b ⋆_D lF f
proof −
  have F ?b ⋆_D F l_C[f] ·_D Φ (?b, f) = (F ?b ⋆_D F l_C[f]) ·_D (F ?b ⋆_D Φ (?b, f))
    using assms cmp-in-hom D.whisker-left [of F ?b F l_C[f] Φ (?b, f)]
    by (simp add: calculation)
  also have ... = F ?b ⋆_D lF f
  proof −
    have (F ?b ⋆_D F l_C[f]) ·_D (F ?b ⋆_D Φ (?b, f))
          = (F ?b ⋆_D F l_C[f]) ·_D D.inv (Φ (?b, ?b ⋆_C f)) ·_D F a_C[?b, ?b, f] ·_D
            Φ (?b ⋆_C ?b, f) ·_D (Φ (?b, ?b) ⋆_D F f) ·_D a_D^−1[F ?b, F ?b, F f]
    proof −
      have 1: D.seq (F a_C[trg_C f, trg_C f, f])
                    (Φ (trg_C f ⋆_C trg_C f, f) ·_D (Φ (trg_C f, trg_C f) ⋆_D F f))
        using assms by fastforce
      hence 2: D.inv (Φ (?b, ?b ⋆_C f)) ·_D F a_C[?b, ?b, f] ·_D Φ (?b ⋆_C ?b, f) ·_D
                 (Φ (?b, ?b) ⋆_D F f)
                 = (F ?b ⋆_D Φ (?b, f)) ·_D a_D[F ?b, F ?b, F f]
        using assms cmp-in-hom assoc-coherence cmp-components-are-iso
              D.invert-side-of-triangle(1)
                [of F a_C[?b, ?b, f] ·_D Φ (?b ⋆_C ?b, f) ·_D (Φ (?b, ?b) ⋆_D F f)
                    Φ (?b, ?b ⋆_C f)
                    (F ?b ⋆_D Φ (?b, f)) ·_D a_D[F ?b, F ?b, F f]]
              C.ideD(1) C.ide-hcomp C.trg-hcomp C.trg-trg C.src-trg C.trg.preserves-ide
        by metis
      hence F ?b ⋆_D Φ (?b, f)
              = (D.inv (Φ (?b, ?b ⋆_C f)) ·_D F a_C[?b, ?b, f] ·_D Φ (?b ⋆_C ?b, f) ·_D
                (Φ (?b, ?b) ⋆_D F f)) ·_D a_D^−1[F ?b, F ?b, F f]
      proof −
        have D.seq (D.inv (Φ (trg_C f, trg_C f ⋆_C f)))
                   (F a_C[trg_C f, trg_C f, f] ·_D Φ (trg_C f ⋆_C trg_C f, f) ·_D
                     (Φ (trg_C f, trg_C f) ⋆_D F f))
          using assms 1 D.hseq-char by auto
        moreover have (F (trg_C f) ⋆_D Φ (trg_C f, f)) ·_D a_D[F (trg_C f), F (trg_C f), F f] =
                        D.inv (Φ (trg_C f, trg_C f ⋆_C f)) ·_D
                          F a_C[trg_C f, trg_C f, f] ·_D Φ (trg_C f ⋆_C trg_C f, f) ·_D
                          (Φ (trg_C f, trg_C f) ⋆_D F f)
          using assms 2 by simp
        ultimately show ?thesis
          using assms
                D.invert-side-of-triangle(2)
                  [of D.inv (Φ (?b, ?b ⋆_C f)) ·_D F a_C[?b, ?b, f] ·_D Φ (?b ⋆_C ?b, f) ·_D
                      (Φ (?b, ?b) ⋆_D F f)
                      F ?b ⋆_D Φ (?b, f) a_D[F ?b, F ?b, F f]]
          by fastforce
      qed
      thus ?thesis
        using D.comp-assoc by simp
```

qed
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D$
$(D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D F\ (\mathrm{i}_C[?b] \star_C f)) \cdot_D$
$\Phi\ (?b \star_C\ ?b,\ f) \cdot_D (\Phi\ (?b,\ ?b) \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ ?b,\ F\ ?b,\ F\ f]$
proof −
have *1*: $F\ (?b \star_C \mathrm{l}_C[f]) = F\ (\mathrm{i}_C[?b] \star_C f) \cdot_D D.inv\ (F\ \mathrm{a}_C[?b,\ ?b,\ f])$
using *assms C.lunit-char*(*1*−*2*) *C.unit-in-hom preserves-inv* **by** *auto*
have $F\ \mathrm{a}_C[?b,\ ?b,\ f] = D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D F\ (\mathrm{i}_C[?b] \star_C f)$
proof −
have $F\ \mathrm{a}_C[?b,\ ?b,\ f] \cdot_D D.inv\ (F\ (\mathrm{i}_C[?b] \star_C f)) =$
$D.inv\ (F\ (\mathrm{i}_C[?b] \star_C f) \cdot_D D.inv\ (F\ \mathrm{a}_C[?b,\ ?b,\ f]))$
using *assms* **by** (*simp add*: *C.iso-unit D.inv-comp*)
thus *?thesis*
using *assms 1 D.invert-side-of-triangle D.iso-inv-iso*
by (*metis C.iso-hcomp C.ideD*(*1*) *C.ide-is-iso C.iso-lunit C.iso-unit*
C.lunit-simps(*3*) *C.obj-trg C.src-trg C.trg.as-nat-iso.components-are-iso*
C.unit-simps(*2*) *D.arr-inv D.inv-inv preserves-iso*)
qed
thus *?thesis* **by** *argo*
qed
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D$
$D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D (F\ (\mathrm{i}_C[?b] \star_C f) \cdot_D \Phi\ (?b \star_C\ ?b,\ f)) \cdot_D$
$(\Phi\ (?b,\ ?b) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ ?b,\ F\ ?b,\ F\ f]$
using *D.comp-assoc* **by** *auto*
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D$
$D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D (\Phi\ (?b,\ f) \cdot_D (F\ \mathrm{i}_C[?b] \star_D F\ f)) \cdot_D$
$(\Phi\ (?b,\ ?b) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ ?b,\ F\ ?b,\ F\ f]$
using *assms* Φ*.naturality* [*of* $(\mathrm{i}_C[?b],\ f)$] *FF-def* $C.VV.arr\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
$C.VV.dom\text{-}char_{SbC}$
by *simp*
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D$
$D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D \Phi\ (?b,\ f) \cdot_D$
$((F\ \mathrm{i}_C[?b] \star_D F\ f)) \cdot_D (\Phi\ (?b,\ ?b) \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ ?b,\ F\ ?b,\ F\ f]$
using *D.comp-assoc* **by** *auto*
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D$
$D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D \Phi\ (?b,\ f) \cdot_D (\mathrm{i}\ ?b \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ ?b,\ F\ ?b,\ F\ f]$
using *assms* **by** (*simp add*: *D.comp-assoc D.whisker-right*)
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D$
$(D.inv\ (\Phi\ (?b,\ ?b \star_C f)) \cdot_D D.inv\ (F\ (?b \star_C \mathrm{l}_C[f])) \cdot_D \Phi\ (?b,\ f)) \cdot_D$
$(F\ ?b \star_D lF\ f)$
using *D.comp-assoc assms lF-char*(*2*) **by** *presburger*
also have ... $= (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D (F\ ?b \star_D lF\ f)$
proof −
have $D.inv\ (F\ ?b \star_D F\ \mathrm{l}_C[f]) =$
$D.inv\ (((F\ ?b \star_D F\ \mathrm{l}_C[f]) \cdot_D D.inv\ (\Phi\ (?b,\ ?b \star_C f))) \cdot_D \Phi\ (?b,\ ?b \star_C f))$
using *assms D.comp-inv-arr D.comp-inv-arr′ cmp-simps*(*4*)

D.comp-arr-dom D.comp-assoc
by *simp*
also have ... = *D.inv* (*D.inv* (Φ (*?b*, *f*)) $\cdot_D$ *F* (*?b* $\star_C$ $1_C[f]$) $\cdot_D$ Φ (*?b*, *?b* $\star_C$ *f*))
proof −
have *1*: Φ (*?b*, *f*) $\cdot_D$ (*F ?b* $\star_D$ *F* $1_C[f]$) = *F* (*?b* $\star_C$ $1_C[f]$) $\cdot_D$ Φ (*?b*, *?b* $\star_C$ *f*)
using *assms* Φ*.naturality* [*of* (*?b*, $1_C[f]$)] *FF-def* $C.VV.arr\text{-}char_{SbC}$
$C.VV.cod\text{-}char_{SbC}$ *D.VV.null-char C.VV.dom-simp*
by *simp*
have (*F ?b* $\star_D$ *F* $1_C[f]$) $\cdot_D$ *D.inv* (Φ (*?b*, *?b* $\star_C$ *f*)) =
D.inv (Φ (*?b*, *f*)) $\cdot_D$ *F* (*?b* $\star_C$ $1_C[f]$)
proof −
have *D.seq* (Φ (*?b*, *f*)) (*F ?b* $\star_D$ *F* $1_C[f]$)
using *assms cmp-in-hom*(*2*) [*of ?b f*] **by** *auto*
moreover have *D.iso* (Φ (*?b*, *f*)) $\wedge$ *D.iso* (Φ (*?b*, *?b* $\star_C$ *f*))
using *assms* **by** *simp*
ultimately show *?thesis*
using *1 D.invert-opposite-sides-of-square* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *auto*
qed
also have ... = *D.inv* (*F* (*?b* $\star_C$ $1_C[f]$) $\cdot_D$ Φ (*?b*, *?b* $\star_C$ *f*)) $\cdot_D$ Φ (*?b*, *f*)
proof −
have *D.iso* (*F* (*?b* $\star_C$ $1_C[f]$) $\cdot_D$ Φ (*?b*, *?b* $\star_C$ *f*))
using *assms D.isos-compose* $C.VV.arr\text{-}char_{SbC}$ *C.iso-lunit C.VV.dom-simp*
$C.VV.cod\text{-}simp$
by *simp*
moreover have *D.iso* (*D.inv* (Φ (*?b*, *f*)))
using *assms* **by** *simp*
moreover have *D.seq* (*D.inv* (Φ (*?b*, *f*))) (*F* (*?b* $\star_C$ $1_C[f]$) $\cdot_D$ Φ (*?b*, *?b* $\star_C$ *f*))
using *assms* $C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp* **by** *simp*
ultimately show *?thesis*
using *assms D.inv-comp* **by** *simp*
qed
also have ... = *D.inv* (Φ (*?b*, *?b* $\star_C$ *f*)) $\cdot_D$ *D.inv* (*F* (*?b* $\star_C$ $1_C[f]$)) $\cdot_D$ Φ (*?b*, *f*)
using *D.comp-assoc D.inv-comp assms cmp-simps′*(*1*) *cmp-simps*(*5*) **by** *force*
finally have *D.inv* (*F ?b* $\star_D$ *F* $1_C[f]$)
= *D.inv* (Φ (*?b*, *?b* $\star_C$ *f*)) $\cdot_D$ *D.inv* (*F* (*?b* $\star_C$ $1_C[f]$)) $\cdot_D$ Φ (*?b*, *f*)
by *blast*
thus *?thesis* **by** *argo*
qed
also have ... = ((*F ?b* $\star_D$ *F* $1_C[f]$) $\cdot_D$ *D.inv* (*F ?b* $\star_D$ *F* $1_C[f]$)) $\cdot_D$ (*F ?b* $\star_D$ *lF f*)
using *D.comp-assoc* **by** *simp*
also have ... = *F ?b* $\star_D$ *lF f*
using *assms D.comp-arr-inv′* [*of F ?b* $\star_D$ *F* $1_C[f]$] *D.comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed

ultimately show *?thesis*
using *assms D.L.is-faithful*
by (*metis D.in-homI lF-char*(*2*−*3*) *lF-simps*(*4*−*5*))
qed

lemma *lunit-coherence*:
assumes $C.ide\ f$
shows $\mathrm{l}_D[F\ f] = F\ \mathrm{l}_C[f] \cdot_D \Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f)$
proof −
have $\mathrm{l}_D[F\ f] = (F\ \mathrm{l}_C[f] \cdot_D \Phi\ (trg_C\ f, f)) \cdot_D (unit\ (trg_C\ f) \star_D F\ f)$
by (*metis C.ideD*(*1*) *C.obj-trg D.inv-inv D.invert-side-of-triangle*(*2*)
D.iso-hcomp D.iso-inv-iso as-nat-iso.components-are-iso assms lF-simps(*1*)
lunit-coherence1 lunit-coherence2 preserves-trg unit-char(*2*) *unit-simps*(*2*))
thus *?thesis*
using *assms D.comp-assoc* **by** *simp*
qed

We postpone proving the dual version of this result until after we have developed the notion of the "op bicategory" in the next section.

end

1.10.3 Pseudofunctors and Opposite Bicategories

There are three duals to a bicategory:

1. "op": sources and targets are exchanged;
2. "co": domains and codomains are exchanged;
3. "co-op": both sources and targets and domains and codomains are exchanged.

Here we consider the "op" case.

locale *op-bicategory* =
B: *bicategory* $V\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
for V :: $'a\ comp$ (**infixr** ‹·› *55*)
and H_B :: $'a\ comp$ (**infixr** ‹$\star_B$› *53*)
and a_B :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a_B[-, -, -]›)
and i_B :: $'a \Rightarrow 'a$ (‹i_B[-]›)
and src_B :: $'a \Rightarrow 'a$
and trg_B :: $'a \Rightarrow 'a$
begin

abbreviation H (**infixr** ‹$\star$› *53*)
where $H\ f\ g \equiv H_B\ g\ f$

abbreviation i (‹i[-]›)
where $\mathrm{i} \equiv \mathrm{i}_B$

abbreviation *src*

where $src \equiv trg_B$

abbreviation *trg*
where $trg \equiv src_B$

interpretation *horizontal-homs V src trg*
  by (*unfold-locales*, *auto*)

interpretation *H*: *functor VV.comp V* ‹$\lambda\mu\nu$. *fst* $\mu\nu \star$ *snd* $\mu\nu$›
  using $VV.arr\text{-}char_{SbC}$ *VV.dom-simp VV.cod-simp*
  apply *unfold-locales*
    apply (*metis* (*no-types*, *lifting*) *B.hseqE B.hseq-char′*)
   apply *auto*[*3*]
  using *VV.comp-char* $VV.seq\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *B.VxV.comp-char B.interchange*
  by (*metis* (*no-types*, *lifting*) $B.VxV.seqE_{PC}$ *fst-conv snd-conv*)

interpretation *horizontal-composition V H src trg*
  by (*unfold-locales*, *auto*)

abbreviation *UP*
where *UP* $\mu\nu\tau \equiv$ *if B.VVV.arr* $\mu\nu\tau$ *then*
        (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$)
      else VVV.null

abbreviation *DN*
where *DN* $\mu\nu\tau \equiv$ *if VVV.arr* $\mu\nu\tau$ *then*
        (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$)
      else B.VVV.null

lemma *VVV-arr-char*:
shows *VVV.arr* $\mu\nu\tau \longleftrightarrow$ *B.VVV.arr* (*DN* $\mu\nu\tau$)
using $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *B.VVV.not-arr-null*
  by *auto*

lemma *VVV-ide-char*:
shows *VVV.ide* $\mu\nu\tau \longleftrightarrow$ *B.VVV.ide* (*DN* $\mu\nu\tau$)
proof −
  have *VVV.ide* $\mu\nu\tau \longleftrightarrow$ *VVV.arr* $\mu\nu\tau \wedge$ *B.VxVxV.ide* $\mu\nu\tau$
    using $VVV.ide\text{-}char_{SbC}$ **by** *simp*
  also have ... $\longleftrightarrow$ *B.VVV.arr* (*DN* $\mu\nu\tau$) $\wedge$ *B.VxVxV.ide* (*DN* $\mu\nu\tau$)
    using *VVV-arr-char* $B.VxVxV.ide\text{-}char_{PC}$ **by** *auto*
  also have ... $\longleftrightarrow$ *B.VVV.ide* (*DN* $\mu\nu\tau$)
    using $B.VVV.ide\text{-}char_{SbC}$ [*of DN* $\mu\nu\tau$] **by** *blast*
  finally show *?thesis* **by** *fast*
qed

lemma *VVV-dom-char*:
shows *VVV.dom* $\mu\nu\tau$ = *UP* (*B.VVV.dom* (*DN* $\mu\nu\tau$))
proof (*cases VVV.arr* $\mu\nu\tau$)

show $\neg$ *VVV.arr* $\mu\nu\tau$ $\Longrightarrow$ *VVV.dom* $\mu\nu\tau$ = *UP* (*B.VVV.dom* (*DN* $\mu\nu\tau$))
using *VVV.dom-def VVV.has-domain-iff-arr VVV-arr-char B.VVV.dom-null*
by *auto*
show *VVV.arr* $\mu\nu\tau$ $\Longrightarrow$ *VVV.dom* $\mu\nu\tau$ = *UP* (*B.VVV.dom* (*DN* $\mu\nu\tau$))
proof −
assume $\mu\nu\tau$: *VVV.arr* $\mu\nu\tau$
have *VVV.dom* $\mu\nu\tau$ =
(*B.dom* (*fst* $\mu\nu\tau$), *B.dom* (*fst* (*snd* $\mu\nu\tau$)), *B.dom* (*snd* (*snd* $\mu\nu\tau$)))
using $\mu\nu\tau$ *VVV.dom-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$
by *simp*
also have ... = *UP* (*B.dom* (*snd* (*snd* $\mu\nu\tau$)), *B.dom* (*fst* (*snd* $\mu\nu\tau$)), *B.dom* (*fst* $\mu\nu\tau$))
by (*metis* (*no-types*, *lifting*) *B.VV.arrI*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.arr-dom VV.arrE VVV.arrE*
$\mu\nu\tau$ *fst-conv snd-conv src-dom trg-dom*)
also have ... = *UP* (*B.VVV.dom* (*DN* $\mu\nu\tau$))
using $\mu\nu\tau$ *B.VVV.dom-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by *simp*
finally show *?thesis* **by** *blast*
qed
qed

lemma *VVV-cod-char*:
shows *VVV.cod* $\mu\nu\tau$ = *UP* (*B.VVV.cod* (*DN* $\mu\nu\tau$))
proof (*cases VVV.arr* $\mu\nu\tau$)
show $\neg$ *VVV.arr* $\mu\nu\tau$ $\Longrightarrow$ *VVV.cod* $\mu\nu\tau$ = *UP* (*B.VVV.cod* (*DN* $\mu\nu\tau$))
using *VVV.cod-def VVV.has-codomain-iff-arr VVV-arr-char B.VVV.cod-null*
by *auto*
show *VVV.arr* $\mu\nu\tau$ $\Longrightarrow$ *VVV.cod* $\mu\nu\tau$ = *UP* (*B.VVV.cod* (*DN* $\mu\nu\tau$))
proof −
assume $\mu\nu\tau$: *VVV.arr* $\mu\nu\tau$
have *VVV.cod* $\mu\nu\tau$ = (*B.cod* (*fst* $\mu\nu\tau$), *B.cod* (*fst* (*snd* $\mu\nu\tau$)), *B.cod* (*snd* (*snd* $\mu\nu\tau$)))
using $\mu\nu\tau$ *VVV.cod-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$
by *simp*
also have ... = *UP* (*B.cod* (*snd* (*snd* $\mu\nu\tau$)), *B.cod* (*fst* (*snd* $\mu\nu\tau$)), *B.cod* (*fst* $\mu\nu\tau$))
by (*metis* (*no-types*, *lifting*) *B.VV.arrI*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.arr-cod VV.arrE VVV.arrE*
$\mu\nu\tau$ *fst-conv snd-conv src-cod trg-cod*)
also have ... = *UP* (*B.VVV.cod* (*DN* $\mu\nu\tau$))
using $\mu\nu\tau$ *B.VVV.cod-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by *simp*
finally show *?thesis* **by** *blast*
qed
qed

lemma *HoHV-char*:

shows *HoHV* $\mu\nu\tau$ = *B.HoVH* (*DN* $\mu\nu\tau$)
using *HoHV-def B.HoVH-def VVV-arr-char* **by** *simp*

lemma *HoVH-char*:
shows *HoVH* $\mu\nu\tau$ = *B.HoHV* (*DN* $\mu\nu\tau$)
using *HoVH-def B.HoHV-def VVV-arr-char* **by** *simp*

definition a (‹a[-, -, -]›)
where a[μ, ν, τ] ≡ *B.*α' (*DN* (μ, ν, τ))

interpretation *natural-isomorphism VVV.comp* ‹(·)› *HoHV HoVH*
‹$\lambda\mu\nu\tau$. a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]›
proof
fix $\mu\nu\tau$
show ¬ *VVV.arr* $\mu\nu\tau$ ⟹ a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)] = *B.null*
using *VVV.arr-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ a-*def B.*α'*.extensionality* **by** *auto*
assume $\mu\nu\tau$: *VVV.arr* $\mu\nu\tau$
show *B.arr* a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]
by (*metis* (*no-types*, *lifting*) *B.*α'*.preserves-arr VVV-arr-char* a-*def* $\mu\nu\tau$ *prod.collapse*)
show *HoVH* $\mu\nu\tau$ ·
a[*fst* (*VVV.dom* $\mu\nu\tau$), *fst* (*snd* (*VVV.dom* $\mu\nu\tau$)), *snd* (*snd* (*VVV.dom* $\mu\nu\tau$))] =
a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]
proof −
have *HoVH* $\mu\nu\tau$ ·
a[*fst* (*VVV.dom* $\mu\nu\tau$), *fst* (*snd* (*VVV.dom* $\mu\nu\tau$)), *snd* (*snd* (*VVV.dom* $\mu\nu\tau$))] =
HoVH $\mu\nu\tau$ · *B.*α' (*B.VVV.dom* (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$))
unfolding a-*def*
using $\mu\nu\tau$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VVV.dom-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VVV.dom-char*$_{SbC}$
by *auto*
also have ... = *B.*α' (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$)
using *B.*α'*.naturality1 VVV-arr-char* $\mu\nu\tau$ *HoVH-char* **by** *presburger*
also have ... = a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]
using $\mu\nu\tau$ a-*def* **by** *simp*
finally show *?thesis* **by** *blast*
qed
show a[*fst* (*VVV.cod* $\mu\nu\tau$), *fst* (*snd* (*VVV.cod* $\mu\nu\tau$)), *snd* (*snd* (*VVV.cod* $\mu\nu\tau$))] ·
HoHV $\mu\nu\tau$ =
a (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$))
proof −
have a[*fst* (*VVV.cod* $\mu\nu\tau$), *fst* (*snd* (*VVV.cod* $\mu\nu\tau$)), *snd* (*snd* (*VVV.cod* $\mu\nu\tau$))] ·
HoHV $\mu\nu\tau$ =
B.α' (*B.VVV.cod* (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$)) · *HoHV* $\mu\nu\tau$
unfolding a-*def*
using $\mu\nu\tau$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VVV.cod-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VVV.cod-char*$_{SbC}$
by *auto*
also have ... = *B.*α' (*snd* (*snd* $\mu\nu\tau$), *fst* (*snd* $\mu\nu\tau$), *fst* $\mu\nu\tau$)
using *B.*α'*.naturality2 VVV-arr-char* $\mu\nu\tau$ *HoHV-char* **by** *presburger*

also have ... = a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]
using $\mu\nu\tau$ a-*def* **by** *simp*
finally show *?thesis* **by** *blast*
qed
next
fix $\mu\nu\tau$
assume $\mu\nu\tau$: *VVV.ide* $\mu\nu\tau$
show *B.iso* a[*fst* $\mu\nu\tau$, *fst* (*snd* $\mu\nu\tau$), *snd* (*snd* $\mu\nu\tau$)]
proof −
have *B.VVV.ide* (*DN* $\mu\nu\tau$)
using $\mu\nu\tau$ *VVV-ide-char* **by** *blast*
thus *?thesis*
using $\mu\nu\tau$ a-*def* *B.*α'*.components-are-iso* **by** *force*
qed
qed

sublocale *bicategory V H* a i *src trg*
proof
show $\bigwedge a.$ *obj a* $\Longrightarrow$ «i *a* : *H a a* $\rightarrow_B$ *a*»
using *obj-def objE B.obj-def B.objE B.unit-in-hom* **by** *meson*
show $\bigwedge a.$ *obj a* $\Longrightarrow$ *B.iso* (i *a*)
using *obj-def objE B.obj-def B.objE B.iso-unit* **by** *meson*
show $\bigwedge f\ g\ h\ k.$ ⟦ *B.ide f*; *B.ide g*; *B.ide h*; *B.ide k*;
src f = *trg g*; *src g* = *trg h*; *src h* = *trg k* ⟧ $\Longrightarrow$
$(f \star \mathrm{a}[g,\ h,\ k]) \cdot \mathrm{a}[f,\ g \star h,\ k] \cdot (\mathrm{a}[f,\ g,\ h] \star k) = \mathrm{a}[f,\ g,\ h \star k] \cdot \mathrm{a}[f \star g,\ h,\ k]$
unfolding a-*def*
using *B.*a′*-def B.comp-assoc B.pentagon′ VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *simp*
qed

proposition *is-bicategory*:
shows *bicategory V H* a i *src trg*
..

lemma *assoc-ide-simp*:
assumes *B.ide f* **and** *B.ide g* **and** *B.ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows a[*f*, *g*, *h*] = *B.*a′ *h g f*
using *assms* a-*def B.*a′*-def* **by** *fastforce*

lemma *lunit-ide-simp*:
assumes *B.ide f*
shows *lunit f* = *B.runit f*
proof (*intro B.runit-eqI*)
show *B.ide f* **by** *fact*
show «*lunit f* : *H* (*trg f*) *f* $\rightarrow_B$ *f*»
using *assms* **by** *simp*
show *trg f* $\star$ *lunit f* = (i[*trg f*] $\star$ *f*) $\cdot$ a$_B$[*f*, *trg f*, *trg f*]
proof −
have *trg f* $\star$ *lunit f* = (i[*trg f*] $\star$ *f*) $\cdot$ a′ (*trg f*) (*trg f*) *f*

```
        using assms lunit-char(1−2) [of f] by simp
      moreover have a′ (trg f) (trg f) f = a_B[f, trg f, trg f]
      proof (unfold a′-def)
        have 1: VVV.ide (trg f, trg f, f)
          using assms VVV.ide-char_SbC VVV.arr-char_SbC VV.arr-char_SbC by simp
        have α′ (trg f, trg f, f) = B.inv a[trg f, trg f, f]
          using 1 B.α′.inverts-components by simp
        also have ... = B.inv (B.α′ (f, trg f, trg f))
          unfolding a-def using 1 by simp
        also have ... = a_B[f, trg f, trg f]
       using 1 B.VVV.ide-char_SbC B.VVV.arr-char_SbC B.VV.arr-char_SbC VVV.ide-char_SbC
              VVV.arr-char_SbC VV.arr-char_SbC B.α′.inverts-components B.α-def
          by force
        finally show α′ (trg f, trg f, f) = a_B[f, trg f, trg f]
          by blast
      qed
      ultimately show ?thesis by simp
    qed
  qed

  lemma runit-ide-simp:
  assumes B.ide f
  shows runit f = B.lunit f
    using assms runit-char(1−2) [of f] B.a′-def assoc-ide-simp
    by (intro B.lunit-eqI) auto

end

context pseudofunctor
begin

  interpretation C′: op-bicategory V_C H_C a_C i_C src_C trg_C ..
  interpretation D′: op-bicategory V_D H_D a_D i_D src_D trg_D ..

  notation C′.H (infixr ‹⋆_C^op› 53)
  notation D′.H (infixr ‹⋆_D^op› 53)

  interpretation F′: weak-arrow-of-homs V_C C′.src C′.trg V_D D′.src D′.trg F
    apply unfold-locales
    using weakly-preserves-src weakly-preserves-trg by simp-all
  interpretation H_D′oFF: composite-functor C′.VV.comp D′.VV.comp V_D F′.FF
                    ‹λμν. fst μν ⋆_D^op snd μν› ..
  interpretation FoH_C′: composite-functor C′.VV.comp V_C V_D
                    ‹λμν. fst μν ⋆_C^op snd μν› F
    ..
  interpretation Φ′: natural-isomorphism C′.VV.comp V_D H_D′oFF.map FoH_C′.map
                              ‹λf. Φ (snd f, fst f)›
      using C.VV.arr-char_SbC C′.VV.arr-char_SbC C′.VV.ide-char_SbC C.VV.ide-char_SbC
FF-def F′.FF-def
```

Φ.*extensionality* Φ.*naturality1* Φ.*naturality2*
C.VV.dom-simp C.VV.cod-simp C'.VV.dom-simp C'.VV.cod-simp
by *unfold-locales auto*
interpretation *F'*: *pseudofunctor* V_C *C'.H C'*.a i_C *C'.src C'.trg*
V_D *D'.H D'*.a i_D *D'.src D'.trg*
F ‹$\lambda f.\ \Phi$ (*snd f*, *fst f*)›
proof
fix *f g h*
assume *f*: *C.ide f* **and** *g*: *C.ide g* **and** *h*: *C.ide h*
and *fg*: *C'.src f* = *C'.trg g* **and** *gh*: *C'.src g* = *C'.trg h*
show *F* (*C'*.a *f g h*) $\cdot_D$ Φ (*snd* (*f* $\star_C{}^{op}$ *g*, *h*), *fst* (*f* $\star_C{}^{op}$ *g*, *h*)) $\cdot_D$
(Φ (*snd* (*f*, *g*), *fst* (*f*, *g*)) $\star_D{}^{op}$ *F h*) =
Φ (*snd* (*f*, *g* $\star_C{}^{op}$ *h*), *fst* (*f*, *g* $\star_C{}^{op}$ *h*)) $\cdot_D$ (*F f* $\star_D{}^{op}$ Φ (*snd* (*g*, *h*), *fst* (*g*, *h*))) $\cdot_D$
D'.a (*F f*) (*F g*) (*F h*)
using *f g h fg gh C.VV.in-hom-char*$_{SbC}$ *FF-def C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
C'.assoc-ide-simp D'.assoc-ide-simp preserves-inv assoc-coherence
D.invert-opposite-sides-of-square
[*of F* (a_C *h g f*) Φ (*g* $\star_C{}^{op}$ *h*, *f*) $\cdot_D$ (*F f* $\star_D{}^{op}$ Φ (*h*, *g*))
Φ (*h*, *f* $\star_C{}^{op}$ *g*) $\cdot_D$ (Φ (*g*, *f*) $\star_D{}^{op}$ *F h*) a_D (*F h*) (*F g*) (*F f*)]
D.comp-assoc
by *auto*
qed

lemma *induces-pseudofunctor-between-opposites*:
shows *pseudofunctor* ($\cdot_C$) ($\star_C{}^{op}$) *C'*.a i_C *C'.src C'.trg*
($\cdot_D$) ($\star_D{}^{op}$) *D'*.a i_D *D'.src D'.trg*
F ($\lambda f.\ \Phi$ (*snd f*, *fst f*))
..

It is now easy to dualize the coherence condition for F with respect to left unitors to obtain the corresponding condition for right unitors.

lemma *runit-coherence*:
assumes *C.ide f*
shows $\mathrm{r}_D[F\,f] = F\ \mathrm{r}_C[f] \cdot_D \Phi$ (*f*, src_C *f*) $\cdot_D$ (*F f* $\star_D$ *unit* (src_C *f*))
proof −
have *C'.lunit f* = $\mathrm{r}_C[f]$
using *assms C'.lunit-ide-simp* **by** *simp*
moreover have *D'.lunit* (*F f*) = $\mathrm{r}_D[F\,f]$
using *assms D'.lunit-ide-simp* **by** *simp*
moreover have *F'.unit* (src_C *f*) = *unit* (src_C *f*)
using *assms F'.unit-char F'.preserves-trg*
by (*intro unit-eqI*) *auto*
moreover have *D'.lunit* (*F f*) =
F (*C'.lunit f*) $\cdot_D$ Φ (*f*, src_C *f*) $\cdot_D$ (*F f* $\star_D$ *F'.unit* (src_C *f*))
using *assms F'.lunit-coherence* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

end

1.10.4 Preservation Properties

The objective of this section is to establish explicit formulas for the result of applying a pseudofunctor to expressions of various forms.

context *pseudofunctor*
begin

lemma *preserves-lunit*:
assumes $C.ide\ f$
shows $F\ \mathrm{l}_C[f] = \mathrm{l}_D[F\ f] \cdot_D (D.inv\ (unit\ (trg_C\ f)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (trg_C\ f, f))$
and $F\ \mathrm{l}_C{}^{-1}[f] = \Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f]$
proof −
  show *1*: $F\ \mathrm{l}_C[f] = \mathrm{l}_D[F\ f] \cdot_D (D.inv\ (unit\ (trg_C\ f)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (trg_C\ f, f))$
  proof −
    have $\mathrm{l}_D[F\ f] \cdot_D D.inv\ (\Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f)) = F\ \mathrm{l}_C[f]$
    proof −
      have $D.arr\ \mathrm{l}_D[F\ f]$
        using *assms* **by** *simp*
      moreover have $\mathrm{l}_D[F\ f] = F\ \mathrm{l}_C[f] \cdot_D \Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f)$
        using *assms lunit-coherence* **by** *simp*
      moreover have $D.iso\ (\Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f))$
        using *assms unit-char cmp-components-are-iso*
        by (*intro D.isos-compose D.seqI*) *auto*
      ultimately show *?thesis*
        using *assms D.invert-side-of-triangle*(*2*) **by** *metis*
    qed
    moreover have $D.inv\ (\Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f)) =$
      $(D.inv\ (unit\ (trg_C\ f)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (trg_C\ f, f))$
     using *assms* $C.VV.arr\text{-}char_{SbC}$ *unit-char FF-def D.inv-comp* $C.VV.dom\text{-}simp$ **by** *simp*
    ultimately show *?thesis* **by** *simp*
  qed
  show $F\ \mathrm{l}_C{}^{-1}[f] = \Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f]$
  proof −
    have $F\ \mathrm{l}_C{}^{-1}[f] =$
      $D.inv\ (\mathrm{l}_D[F\ f] \cdot_D (D.inv\ (unit\ (trg_C\ f)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (trg_C\ f, f)))$
      using *assms 1 preserves-inv* **by** *simp*
    also have ... $= \Phi\ (trg_C\ f, f) \cdot_D (unit\ (trg_C\ f) \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f]$
      using *assms unit-char D.comp-assoc D.isos-compose*
      by (*auto simp add*: *D.inv-comp*)
    finally show *?thesis* **by** *simp*
  qed
qed

lemma *preserves-runit*:
assumes $C.ide\ f$
shows $F\ \mathrm{r}_C[f] = \mathrm{r}_D[F\ f] \cdot_D (F\ f \star_D D.inv\ (unit\ (src_C\ f))) \cdot_D D.inv\ (\Phi\ (f, src_C\ f))$
and $F\ \mathrm{r}_C{}^{-1}[f] = \Phi\ (f, src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f)) \cdot_D \mathrm{r}_D{}^{-1}[F\ f]$

proof −
  show *1*: $F\ \mathrm{r}_C[f] = \mathrm{r}_D[F\ f] \cdot_D (F\ f \star_D D.inv\ (unit\ (src_C\ f))) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
  proof −
    have $F\ \mathrm{r}_C[f] = \mathrm{r}_D[F\ f] \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f)))$
    proof −
      have $\mathrm{r}_D[F\ f] = F\ \mathrm{r}_C[f] \cdot_D \Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f))$
        using *assms runit-coherence* **by** *simp*
      moreover have $D.iso\ (\Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f)))$
        using *assms unit-char D.iso-hcomp FF-def*
        apply (*intro D.isos-compose D.seqI*) **by** *auto*
      moreover have $D.arr\ \mathrm{r}_D[F\ f]$
        using *assms* **by** *simp*
      ultimately show *?thesis*
        using *assms D.invert-side-of-triangle*(*2*) **by** *metis*
    qed
    moreover have $D.inv\ (\Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f))) =$
      $(F\ f \star_D D.inv\ (unit\ (src_C\ f))) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
    using *assms* $C.VV.arr\text{-}char_{SbC}$ *unit-char D.iso-hcomp FF-def D.inv-comp* $C.VV.dom\text{-}simp$
      by *simp*
    ultimately show *?thesis* **by** *simp*
  qed
  show $F\ \mathrm{r}_C{}^{-1}[f] = \Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f)) \cdot_D \mathrm{r}_D{}^{-1}[F\ f]$
  proof −
    have $F\ \mathrm{r}_C{}^{-1}[f] =$
      $D.inv\ (\mathrm{r}_D[F\ f] \cdot_D (F\ f \star_D D.inv\ (unit\ (src_C\ f))) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f)))$
      using *assms 1 preserves-inv C.iso-runit* **by** *simp*
    also have ... $= \Phi\ (f,\ src_C\ f) \cdot_D (F\ f \star_D unit\ (src_C\ f)) \cdot_D \mathrm{r}_D{}^{-1}[F\ f]$
      using *assms unit-char D.comp-assoc D.isos-compose*
      by (*auto simp add*: *D.inv-comp*)
    finally show *?thesis* **by** *simp*
  qed
qed

lemma *preserves-assoc*:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $C.ide\ h$
and $src_C\ f = trg_C\ g$ **and** $src_C\ g = trg_C\ h$
shows $F\ \mathrm{a}_C[f,\ g,\ h] = \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h] \cdot_D$
  $(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
and $F\ \mathrm{a}_C{}^{-1}[f,\ g,\ h] = \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ h] \cdot_D$
  $(F\ f \star_D D.inv\ (\Phi\ (g,\ h))) \cdot_D D.inv\ (\Phi\ (f,\ g \star_C h))$
proof −
  show *1*: $F\ \mathrm{a}_C[f,\ g,\ h] =$
    $\Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h] \cdot_D$
    $(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
  proof −
    have $F\ \mathrm{a}_C[f,\ g,\ h] \cdot_D \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) =$
      $\Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$
      using *assms assoc-coherence* [*of f g h*] **by** *simp*
    moreover have $D.seq\ (\Phi\ (f,\ g \star_C h))\ ((F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h])$

using *assms* $C.VV.arr\text{-}char_{SbC}$ *FF-def* $C.VV.dom\text{-}simp$ $C.VV.cod\text{-}simp$ **by** *auto*
moreover have *2*: $D.iso\ (\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h))$
using *assms* $C.VV.arr\text{-}char_{SbC}$ *FF-def* $C.VV.dom\text{-}simp$ $C.VV.cod\text{-}simp$ **by** *auto*
moreover have $D.inv\ (\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h)) =$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
using *assms 2* $C.VV.arr\text{-}char_{SbC}$ *FF-def* $D.inv\text{-}comp$ $C.VV.dom\text{-}simp$ $C.VV.cod\text{-}simp$
by *simp*
ultimately show *?thesis*
using $D.invert\text{-}side\text{-}of\text{-}triangle(2)$
$[of\ \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$
$F\ \mathrm{a}_C[f,\ g,\ h]\ \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h)]$
$D.comp\text{-}assoc$
by *simp*
qed
show $F\ \mathrm{a}_C{}^{-1}[f,\ g,\ h] =$
$\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ h] \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ h))) \cdot_D D.inv\ (\Phi\ (f,\ g \star_C h))$
proof −
have $F\ \mathrm{a}_C{}^{-1}[f,\ g,\ h] =$
$D.inv\ (\Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h] \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h)))$
using *assms 1* $preserves\text{-}inv$ **by** *simp*
also have ... $= \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ h] \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ h))) \cdot_D D.inv\ (\Phi\ (f,\ g \star_C h))$
proof −
have «$\Phi\ (f,\ g \star_C h) : F\ f \star_D F\ (g \star_C h) \Rightarrow_D F\ (f \star_C g \star_C h)$» $\wedge$ $D.iso\ (\Phi\ (f,\ g \star_C h))$
using *assms* **by** *auto*
moreover have «$F\ f \star_D \Phi\ (g,\ h) : F\ f \star_D F\ g \star_D F\ h \Rightarrow_D F\ f \star_D F\ (g \star_C h)$» $\wedge$
$D.iso\ (F\ f \star_D \Phi\ (g,\ h))$
using *assms*
by (*intro conjI* $D.hcomp\text{-}in\text{-}vhom$, *auto*)
ultimately show *?thesis*
using *assms* $D.isos\text{-}compose$ $D.comp\text{-}assoc$ $D.inv\text{-}comp$
by (*elim conjE* $D.in\text{-}homE$) *auto*
qed
finally show *?thesis* **by** *simp*
qed
qed

lemma *preserves-hcomp*:
assumes $C.hseq\ \mu\ \nu$
shows $F\ (\mu \star_C \nu) =$
$\Phi\ (C.cod\ \mu,\ C.cod\ \nu) \cdot_D (F\ \mu \star_D F\ \nu) \cdot_D D.inv\ (\Phi\ (C.dom\ \mu,\ C.dom\ \nu))$
proof −
have $F\ (\mu \star_C \nu) \cdot_D \Phi\ (C.dom\ \mu,\ C.dom\ \nu) = \Phi\ (C.cod\ \mu,\ C.cod\ \nu) \cdot_D (F\ \mu \star_D F\ \nu)$
using *assms* $\Phi.naturality$ $C.VV.arr\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def* $C.VV.dom\text{-}simp$
by *auto*
thus *?thesis*
by (*metis* (*no-types*) *assms* $C.hcomp\text{-}simps(3)$ $C.hseqE$ $C.ide\text{-}dom$ $C.src\text{-}dom$ $C.trg\text{-}dom$

D.comp-arr-inv′ D.comp-assoc cmp-components-are-iso cmp-simps(*5*)
as-nat-trans.naturality1)
qed

lemma *preserves-adjunction-data*:
assumes *adjunction-data-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
shows *adjunction-data-in-bicategory* V_D H_D a_D i_D src_D trg_D
(F f) (F g) (*D.inv* (Φ (g, f)) $\cdot_D$ F η $\cdot_D$ *unit* (src_C f))
(*D.inv* (*unit* (trg_C f)) $\cdot_D$ F ε $\cdot_D$ Φ (f, g))
proof
interpret *adjunction-data-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
using *assms* **by** *auto*
show *D.ide* (F f)
using *preserves-ide* **by** *simp*
show *D.ide* (F g)
using *preserves-ide* **by** *simp*
show «*D.inv* (Φ (g, f)) $\cdot_D$ F η $\cdot_D$ *unit* (src_C f) : src_D (F f) $\Rightarrow_D$ F g $\star_D$ F f»
using *antipar* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *cmp-in-hom*(*2*) *unit-in-hom FF-def*
by *auto*
show «*D.inv* (*unit* (trg_C f)) $\cdot_D$ F ε $\cdot_D$ Φ (f, g) : F f $\star_D$ F g $\Rightarrow_D$ src_D (F g)»
using *antipar* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def cmp-in-hom*(*2*) *unit-in-hom*(*2*)
unit-char(*2*)
by *auto*
qed

lemma *preserves-equivalence*:
assumes *equivalence-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
shows *equivalence-in-bicategory* V_D H_D a_D i_D src_D trg_D
(F f) (F g) (*D.inv* (Φ (g, f)) $\cdot_D$ F η $\cdot_D$ *unit* (src_C f))
(*D.inv* (*unit* (trg_C f)) $\cdot_D$ F ε $\cdot_D$ Φ (f, g))
proof −
interpret *equivalence-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
using *assms* **by** *auto*
show *equivalence-in-bicategory* V_D H_D a_D i_D src_D trg_D
(F f) (F g) (*D.inv* (Φ (g, f)) $\cdot_D$ F η $\cdot_D$ *unit* (src_C f))
(*D.inv* (*unit* (trg_C f)) $\cdot_D$ F ε $\cdot_D$ Φ (f, g))
using *antipar unit-is-iso preserves-iso unit-char*(*2*) $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
FF-def D.isos-compose
by (*unfold-locales*) *auto*
qed

lemma *preserves-equivalence-maps*:
assumes *C.equivalence-map f*
shows *D.equivalence-map* (F f)
proof −
obtain g η ε **where** E: *equivalence-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
using *assms C.equivalence-map-def* **by** *auto*
interpret E: *equivalence-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
using E **by** *auto*

show *?thesis*
using *E preserves-equivalence C.equivalence-map-def D.equivalence-map-def* map_0*-simps*
by *blast*
qed

lemma *preserves-equivalent-objects*:
assumes *C.equivalent-objects a b*
shows *D.equivalent-objects* (map_0 *a*) (map_0 *b*)
using *assms D.equivalent-objects-def preserves-equivalence-maps*
unfolding *C.equivalent-objects-def* **by** *fast*

lemma *preserves-isomorphic*:
assumes *C.isomorphic f g*
shows *D.isomorphic* (*F f*) (*F g*)
using *assms C.isomorphic-def D.isomorphic-def preserves-iso* **by** *auto*

lemma *preserves-quasi-inverses*:
assumes *C.quasi-inverses f g*
shows *D.quasi-inverses* (*F f*) (*F g*)
using *assms C.quasi-inverses-def D.quasi-inverses-def preserves-equivalence* **by** *blast*

lemma *preserves-quasi-inverse*:
assumes *C.equivalence-map f*
shows *D.isomorphic* (*F* (*C.some-quasi-inverse f*)) (*D.some-quasi-inverse* (*F f*))
using *assms preserves-quasi-inverses C.quasi-inverses-some-quasi-inverse*
D.quasi-inverse-unique D.quasi-inverses-some-quasi-inverse
preserves-equivalence-maps
by *blast*

end

1.10.5 Identity Pseudofunctors

locale *identity-pseudofunctor* =
B: *bicategory* V_B H_B a_B i_B src_B trg_B
for V_B :: *'b comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: *'b comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: *'b* ⇒ *'b* ⇒ *'b* ⇒ *'b* (‹a_B[-, -, -]›)
and i_B :: *'b* ⇒ *'b* (‹i_B[-]›)
and src_B :: *'b* ⇒ *'b*
and trg_B :: *'b* ⇒ *'b*
begin

The underlying vertical functor is just the identity functor on the vertical category, which is already available as *B.map*.

abbreviation *map*
where *map* ≡ *B.map*

interpretation *I*: *weak-arrow-of-homs* V_B src_B trg_B V_B src_B trg_B *map*

using *B.isomorphic-reflexive* **by** *unfold-locales auto*

interpretation *II*: *functor B.VV.comp B.VV.comp I.FF*
using *I.functor-FF* **by** *simp*

interpretation $H_B oII$: *composite-functor B.VV.comp B.VV.comp* V_B *I.FF*
‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$›
..
interpretation IoH_B: *composite-functor B.VV.comp* V_B V_B ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$› *map*
..

The horizontal composition provides the compositor.

abbreviation *cmp*
where *cmp* $\equiv \lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$

interpretation *cmp*: *natural-transformation B.VV.comp* V_B $H_B oII$*.map* IoH_B*.map cmp*
using *B.VV.arr-char*$_{SbC}$ *B.VV.dom-simp B.VV.cod-simp B.H.as-nat-trans.naturality1*
B.H.as-nat-trans.naturality2 I.FF-def
apply *unfold-locales*
apply *auto*
by (*meson B.hseqE B.hseq-char′*)+

interpretation *cmp*: *natural-isomorphism B.VV.comp* V_B $H_B oII$*.map* IoH_B*.map cmp*
by *unfold-locales simp*

sublocale *pseudofunctor* V_B H_B a_B i_B src_B trg_B V_B H_B a_B i_B src_B trg_B *map cmp*
apply *unfold-locales*
by (*metis B.assoc-naturality2 B.assoc-naturality B.assoc-simps*(*1*) *B.comp-ide-self*
B.hcomp-simps(*1*) *B.ide-char B.ide-hcomp B.map-simp fst-conv snd-conv*)

lemma *is-pseudofunctor*:
shows *pseudofunctor* V_B H_B a_B i_B src_B trg_B V_B H_B a_B i_B src_B trg_B *map cmp*
..

lemma *unit-char′*:
assumes *B.obj a*
shows *unit a* = *a*
proof −
have src_B *a* = *a* ∧ *B.ide a*
using *assms* **by** *auto*
hence *a* = *unit a*
using *assms B.comp-arr-dom B.comp-cod-arr* $I.map_0$*-def map-def*
B.ide-in-hom(*2*) *B.objE* [*of a*] *B.ide-is-iso* [*of a*]
by (*intro unit-eqI*) *auto*
thus *?thesis* **by** *simp*
qed

end

lemma (**in** *identity-pseudofunctor*) map_0*-simp* [*simp*]:
assumes *B.obj a*
shows map_0 *a* = *a*
using *assms* map_0*-def* **by** *auto*

1.10.6 Embedding Pseudofunctors

In this section, we construct the embedding pseudofunctor of a sub-bicategory into the ambient bicategory.

locale *embedding-pseudofunctor* =
B: *bicategory V H* a_B i src_B trg_B +
S: *subbicategory*
begin

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)
notation *B.in-hhom* (‹«- : - $\rightarrow_B$ -»›)

definition *map*
where *map* μ = (*if S.arr* μ *then* μ *else B.null*)

lemma *map-in-hom* [*intro*]:
assumes *S.arr* μ
shows «*map* μ : src_B (*map* (*S.src* μ)) $\rightarrow_B$ src_B (*map* (*S.trg* μ))»
and «*map* μ : *map* (*S.dom* μ) $\Rightarrow_B$ *map* (*S.cod* μ)»
proof −
show *1*: «*map* μ : *map* (*S.dom* μ) $\Rightarrow_B$ *map* (*S.cod* μ)»
using *assms map-def* $S.in\text{-}hom\text{-}char_{SbC}$ **by** *fastforce*
show «*map* μ : src_B (*map* (*S.src* μ)) $\rightarrow_B$ src_B (*map* (*S.trg* μ))»
using *assms 1 map-def* $S.arr\text{-}char_{SbC}$ *S.src-def S.trg-def S.obj-char S.obj-src S.obj-trg*
by *auto*
qed

lemma *map-simps* [*simp*]:
assumes *S.arr* μ
shows *B.arr* (*map* μ)
and src_B (*map* μ) = src_B (*map* (*S.src* μ)) **and** trg_B (*map* μ) = src_B (*map* (*S.trg* μ))
and *B.dom* (*map* μ) = *map* (*S.dom* μ) **and** *B.cod* (*map* μ) = *map* (*S.cod* μ)
using *assms map-in-hom* **by** *blast+*

interpretation *functor S.comp V map*
apply *unfold-locales*
apply *auto*
using *map-def S.comp-char* $S.seq\text{-}char_{SbC}$ $S.arr\text{-}char_{SbC}$
apply *auto*[*1*]
using *map-def S.comp-simp* **by** *auto*

interpretation *weak-arrow-of-homs S.comp S.src S.trg V* src_B trg_B *map*
using $S.arr\text{-}char_{SbC}$ *map-def S.src-def S.trg-def S.src-closed S.trg-closed B.isomorphic-reflexive preserves-ide S.ide-src S.ide-trg*

apply *unfold-locales*
by *presburger+*

interpretation *HoFF*: *composite-functor S.VV.comp B.VV.comp V FF*
‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$›
..
interpretation *FoH*: *composite-functor S.VV.comp S.comp V* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$› *map*
..

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

definition *cmp*
where *cmp* $\mu\nu$ = (*if S.VV.arr* $\mu\nu$ *then fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$ *else B.null*)

lemma *cmp-in-hom* [*intro*]:
assumes *S.VV.arr* $\mu\nu$
shows «*cmp* $\mu\nu$: src_B (*snd* $\mu\nu$) $\rightarrow_B$ trg_B (*fst* $\mu\nu$)»
and «*cmp* $\mu\nu$: *map* (*S.dom* (*fst* $\mu\nu$)) $\star_B$ *map* (*S.dom* (*snd* $\mu\nu$))
$\Rightarrow_B$ *map* (*S.cod* (*fst* $\mu\nu$) $\star$ *S.cod* (*snd* $\mu\nu$))»
proof −
show «*cmp* $\mu\nu$: *map* (*S.dom* (*fst* $\mu\nu$)) $\star_B$ *map* (*S.dom* (*snd* $\mu\nu$))
$\Rightarrow_B$ *map* (*S.cod* (*fst* $\mu\nu$) $\star$ *S.cod* (*snd* $\mu\nu$))»
proof
show *1*: *B.arr* (*cmp* $\mu\nu$)
unfolding *cmp-def*
using *assms* $S.arr\text{-}char_{SbC}$ $S.VV.arr\text{-}char_{SbC}$ *S.inclusion S.src-def S.trg-def* **by** *auto*
show *B.dom* (*cmp* $\mu\nu$) = *map* (*S.dom* (*fst* $\mu\nu$)) $\star_B$ *map* (*S.dom* (*snd* $\mu\nu$))
unfolding *cmp-def map-def*
using *assms 1 cmp-def S.dom-simp S.cod-simp* $S.VV.arr\text{-}char_{SbC}$ $S.arr\text{-}char_{SbC}$ *S.hcomp-def*
S.inclusion S.dom-closed
by *auto*
show *B.cod* (*cmp* $\mu\nu$) = *map* (*S.cod* (*fst* $\mu\nu$) $\star$ *S.cod* (*snd* $\mu\nu$))
unfolding *cmp-def map-def*
using *assms 1 S.H.preserves-arr S.cod-simp S.hcomp-eqI S.hcomp-simps*(*4*) *S.hseq-char′*
by *auto*
qed
thus «*cmp* $\mu\nu$: src_B (*snd* $\mu\nu$) $\rightarrow_B$ trg_B (*fst* $\mu\nu$)»
using *cmp-def* **by** *auto*
qed

lemma *cmp-simps* [*simp*]:
assumes *S.VV.arr* $\mu\nu$
shows *B.arr* (*cmp* $\mu\nu$)
and src_B (*cmp* $\mu\nu$) = *S.src* (*snd* $\mu\nu$) **and** trg_B (*cmp* $\mu\nu$) = *S.trg* (*fst* $\mu\nu$)
and *B.dom* (*cmp* $\mu\nu$) = *map* (*S.dom* (*fst* $\mu\nu$)) $\star_B$ *map* (*S.dom* (*snd* $\mu\nu$))
and *B.cod* (*cmp* $\mu\nu$) = *map* (*S.cod* (*fst* $\mu\nu$) $\star$ *S.cod* (*snd* $\mu\nu$))
using *assms cmp-in-hom S.src-def S.trg-def* $S.VV.arr\text{-}char_{SbC}$
by *simp-all blast+*

lemma *iso-cmp*:
assumes $S.VV.ide\ \mu\nu$
shows $B.iso\ (cmp\ \mu\nu)$
using *assms* $S.VV.ide\text{-}char_{SbC}$ $S.VV.arr\text{-}char_{SbC}$ $S.arr\text{-}char_{SbC}$ *cmp-def* $S.ide\text{-}char_{SbC}$ *S.src-def S.trg-def*
by *auto*

interpretation Φ_E: *natural-isomorphism S.VV.comp V HoFF.map FoH.map cmp*
proof
show $\bigwedge\mu\nu.\ \neg\ S.VV.arr\ \mu\nu \Longrightarrow cmp\ \mu\nu = B.null$
using *cmp-def* **by** *simp*
fix $\mu\nu$
assume $\mu\nu$: $S.VV.arr\ \mu\nu$
have *1*: $S.arr\ (fst\ \mu\nu) \wedge S.arr\ (snd\ \mu\nu) \wedge S.src\ (fst\ \mu\nu) = S.trg\ (snd\ \mu\nu)$
using $\mu\nu$ $S.VV.arr\text{-}char_{SbC}$ **by** *simp*
show $B.arr\ (cmp\ \mu\nu)$
using $\mu\nu$ **by** *simp*
show $cmp\ (S.VV.cod\ \mu\nu) \cdot_B HoFF.map\ \mu\nu = cmp\ \mu\nu$
using $\mu\nu$ *1 cmp-def* $S.VV.arr\text{-}char_{SbC}$ $S.VV.cod\text{-}char_{SbC}$ *FF-def S.arr-cod S.cod-simp S.src-def S.trg-def map-def*
apply *simp*
by (*metis* (*no-types*, *lifting*) *B.comp-cod-arr B.hcomp-simps*(*4*) *cmp-simps*(*1*) $\mu\nu$)
show $FoH.map\ \mu\nu \cdot_B cmp\ (S.VV.dom\ \mu\nu) = cmp\ \mu\nu$
unfolding *cmp-def map-def*
using $\mu\nu$ $S.VV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $S.VV.dom\text{-}char_{SbC}$ $S.VV.cod\text{-}char_{SbC}$ *B.comp-arr-dom S.hcomp-def*
apply *simp*
by (*metis* (*no-types*, *lifting*) *B.hcomp-simps*(*3*) *cmp-def cmp-simps*(*1*) $S.arr\text{-}char_{SbC}$ $S.dom\text{-}char_{SbC}$ *S.hcomp-closed S.src-def S.trg-def*)
next
show $\bigwedge fg.\ S.VV.ide\ fg \Longrightarrow B.iso\ (cmp\ fg)$
using *iso-cmp* **by** *simp*
qed

sublocale *pseudofunctor S.comp S.hcomp* $S.\mathrm{a}\ \mathrm{i}$ *S.src S.trg V H* $\mathrm{a}_B\ \mathrm{i}$ src_B trg_B *map cmp*
proof
fix *f g h*
assume *f*: *S.ide f* **and** *g*: *S.ide g* **and** *h*: *S.ide h*
and *fg*: $S.src\ f = S.trg\ g$ **and** *gh*: $S.src\ g = S.trg\ h$
have *1*: $B.ide\ f \wedge B.ide\ g \wedge B.ide\ h \wedge src_B\ f = trg_B\ g \wedge src_B\ g = trg_B\ h$
using *f g h fg gh* $S.ide\text{-}char_{SbC}$ $S.arr\text{-}char_{SbC}$ *S.src-def S.trg-def* **by** *simp*
show $map\ (S.\mathrm{a}\ f\ g\ h) \cdot_B cmp\ (f \star g,\ h) \cdot_B cmp\ (f,\ g) \star_B map\ h =$
$cmp\ (f,\ g \star h) \cdot_B (map\ f \star_B cmp\ (g,\ h)) \cdot_B \mathrm{a}_B[map\ f,\ map\ g,\ map\ h]$
proof −
have $map\ (S.\mathrm{a}\ f\ g\ h) \cdot_B cmp\ (f \star g,\ h) \cdot_B cmp\ (f,\ g) \star_B map\ h =$
$\mathrm{a}_B[f,\ g,\ h] \cdot_B ((f \star_B g) \star_B h) \cdot_B ((f \star_B g) \star_B h)$
unfolding *map-def cmp-def*

using *1 f g h fg gh S.VVV.arr-char$_{SbC}$ S.VV.arr-char$_{SbC}$ B.VVV.arr-char$_{SbC}$ B.VV.arr-char$_{SbC}$*
B.comp-arr-dom S.arr-char$_{SbC}$ S.comp-char S.hcomp-closed S.hcomp-def
S.ideD(1) S.src-def
by (*simp add*: *S.assoc-closed*)
also have ... = *cmp* (*f*, *g* ⋆ *h*) ·$_B$ (*map f* ⋆$_B$ *cmp* (*g*, *h*)) ·$_B$ a$_B$[*map f*, *map g*, *map h*]
unfolding *cmp-def map-def*
using *1 f g h fg gh S.VV.arr-char$_{SbC}$ B.VVV.arr-char$_{SbC}$ B.VV.arr-char$_{SbC}$*
B.comp-arr-dom B.comp-cod-arr S.hcomp-def S.comp-char
S.arr-char$_{SbC}$ S.assoc-closed S.hcomp-closed S.ideD(1) S.trg-def
by *auto*
finally show *?thesis* **by** *blast*
qed
qed

lemma *is-pseudofunctor*:
shows *pseudofunctor S.comp S.hcomp S*.a i *S.src S.trg V H* a$_B$ i *src$_B$ trg$_B$ map cmp*
..

lemma *map$_0$-simp* [*simp*]:
assumes *S.obj a*
shows *map$_0$ a* = *a*
using *assms map$_0$-def map-def S.obj-char* **by** *auto*

lemma *unit-char′*:
assumes *S.obj a*
shows *unit a* = *a*
proof −
have *a*: *S.arr a*
using *assms* **by** *auto*
have *1*: *B.ide a*
using *assms S.obj-char* **by** *auto*
have *2*: *src$_B$ a* = *a*
using *assms S.obj-char* **by** *auto*
have *a* = *unit a*
proof (*intro unit-eqI*)
show *S.obj a* **by** *fact*
show «*a* : *map$_0$ a* ⇒$_B$ *map a*»
using *assms a 2 map$_0$-def map-def S.src-def S.dom-char$_{SbC}$ S.cod-char$_{SbC}$ S.obj-char*
by *auto*
show *B.iso a*
using *assms 1* **by** *simp*
show *a* ·$_B$ i[*map$_0$ a*] = (*map* i[*a*] ·$_B$ *cmp* (*a*, *a*)) ·$_B$ (*a* ⋆$_B$ *a*)
proof −
have *a* ·$_B$ i[*a*] = i[*a*] ·$_B$ *cmp* (*a*, *a*) ·$_B$ (*a* ⋆$_B$ *a*)
proof −
have *a* ·$_B$ i[*a*] = i[*a*]
using *assms 1 2 S.comp-cod-arr S.unitor-coincidence S.lunit-in-hom*
S.obj-char S.comp-simp

by *auto*
moreover have $(a \star_B a) \cdot_B (a \star_B a) = a \star_B a$
using *assms S.obj-char S.comp-ide-self* **by** *auto*
moreover have $B.dom\ \mathrm{i}[a] = a \star_B a$
using *assms S.obj-char* **by** *simp*
moreover have $\mathrm{i}[a] \cdot_B (a \star_B a) = \mathrm{i}[a]$
using *assms S.obj-char B.comp-arr-dom* **by** *simp*
ultimately show *?thesis*
using *assms cmp-def* $S.VV.arr\text{-}char_{SbC}$ **by** *auto*
qed
thus *?thesis*
using *assms a 2* map_0*-def map-def S.src-def B.comp-assoc* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed

end

1.10.7 Composition of Pseudofunctors

In this section, we show how pseudofunctors may be composed. The main work is to establish the coherence condition for associativity.

locale *composite-pseudofunctor* =
B: *bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ +
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$ +
G: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
for V_B :: *'b comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: *'b comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and i_B :: $'b \Rightarrow 'b$ (‹i_B[-]›)
and src_B :: $'b \Rightarrow 'b$
and trg_B :: $'b \Rightarrow 'b$
and V_C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *'c comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: *'d comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: *'d comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'b \Rightarrow 'c$
and Φ_F :: $'b * 'b \Rightarrow 'c$

```
and G :: 'c ⇒ 'd
and Φ_G :: 'c * 'c ⇒ 'd
begin

  sublocale composite-functor V_B V_C V_D F G ..

  sublocale weak-arrow-of-homs V_B src_B trg_B V_D src_D trg_D ‹G o F›
  proof
    show ⋀μ. B.arr μ ⟹ D.isomorphic ((G o F) (src_B μ)) (src_D ((G o F) μ))
    proof −
      fix μ
      assume μ: B.arr μ
      show D.isomorphic ((G o F) (src_B μ)) (src_D ((G o F) μ))
      proof −
        have (G o F) (src_B μ) = G (F (src_B μ))
          using μ by simp
        also have D.isomorphic ... (G (src_C (F μ)))
          using μ F.weakly-preserves-src G.preserves-iso
          by (meson C.isomorphicE D.isomorphic-def G.preserves-hom)
        also have D.isomorphic ... (src_D (G (F μ)))
          using μ G.weakly-preserves-src by blast
        also have ... = src_D ((G o F) μ)
          by simp
        finally show ?thesis by blast
      qed
    qed
    show ⋀μ. B.arr μ ⟹ D.isomorphic ((G o F) (trg_B μ)) (trg_D ((G o F) μ))
    proof −
      fix μ
      assume μ: B.arr μ
      show D.isomorphic ((G o F) (trg_B μ)) (trg_D ((G o F) μ))
      proof −
        have (G o F) (trg_B μ) = G (F (trg_B μ))
          using μ by simp
        also have D.isomorphic ... (G (trg_C (F μ)))
          using μ F.weakly-preserves-trg G.preserves-iso
          by (meson C.isomorphicE D.isomorphic-def G.preserves-hom)
        also have D.isomorphic ... (trg_D (G (F μ)))
          using μ G.weakly-preserves-trg by blast
        also have ... = trg_D ((G o F) μ)
          by simp
        finally show ?thesis by blast
      qed
    qed
  qed

  interpretation H_DoGF-GF: composite-functor B.VV.comp D.VV.comp V_D FF
                    ‹λμν. fst μν ⋆_D snd μν›
    ..
```

interpretation $GFoH_B$: *composite-functor* $B.VV.comp$ V_B V_D ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$›
‹*G o F*›
..

definition *cmp*
where *cmp* $\mu\nu$ = (*if* $B.VV.arr$ $\mu\nu$ *then*
G (F (H_B (*fst* $\mu\nu$) (*snd* $\mu\nu$))) $\cdot_D$ G (Φ_F ($B.VV.dom$ $\mu\nu$)) $\cdot_D$
Φ_G (F ($B.dom$ (*fst* $\mu\nu$)), F ($B.dom$ (*snd* $\mu\nu$)))
else $D.null$)

lemma *cmp-in-hom* [*intro*]:
assumes $B.VV.arr$ $\mu\nu$
shows «*cmp* $\mu\nu$: $H_D oGF$-$GF.map$ ($B.VV.dom$ $\mu\nu$) $\Rightarrow_D$ $GFoH_B.map$ ($B.VV.cod$ $\mu\nu$)»
proof −
have *cmp* $\mu\nu$ = G (F (H_B (*fst* $\mu\nu$) (*snd* $\mu\nu$))) $\cdot_D$ G (Φ_F ($B.VV.dom$ $\mu\nu$)) $\cdot_D$
Φ_G (F ($B.dom$ (*fst* $\mu\nu$)), F ($B.dom$ (*snd* $\mu\nu$)))
using *assms cmp-def* **by** *simp*
moreover have « ... : $H_D oGF$-$GF.map$ ($B.VV.dom$ $\mu\nu$) $\Rightarrow_D$ $GFoH_B.map$ ($B.VV.cod$ $\mu\nu$)»
proof (*intro* $D.comp$-in-$homI$)
show «Φ_G (F ($B.dom$ (*fst* $\mu\nu$)), F ($B.dom$ (*snd* $\mu\nu$))) :
$H_D oGF$-$GF.map$ ($B.VV.dom$ $\mu\nu$)
$\Rightarrow_D$ G (F ($B.dom$ (*fst* $\mu\nu$)) $\star_C$ F ($B.dom$ (*snd* $\mu\nu$)))»
using *assms* $F.FF$-def FF-def $B.VV.arr$-$char_{SbC}$ $B.VV.dom$-$simp$ **by** *auto*
show «G (Φ_F ($B.VV.dom$ $\mu\nu$)) :
G (F ($B.dom$ (*fst* $\mu\nu$)) $\star_C$ F ($B.dom$ (*snd* $\mu\nu$)))
$\Rightarrow_D$ $GFoH_B.map$ ($B.VV.dom$ $\mu\nu$)»
using *assms* $B.VV.arr$-$char_{SbC}$ $F.FF$-def $B.VV.dom$-$simp$ $B.VV.cod$-$simp$ **by** *auto*
show «G (F (*fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$)) :
$GFoH_B.map$ ($B.VV.dom$ $\mu\nu$) $\Rightarrow_D$ $GFoH_B.map$ ($B.VV.cod$ $\mu\nu$)»
proof −
have $B.VV.in$-hom $\mu\nu$ ($B.VV.dom$ $\mu\nu$) ($B.VV.cod$ $\mu\nu$)
using *assms* **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
ultimately show *?thesis* **by** *argo*
qed

lemma *cmp-simps* [*simp*]:
assumes $B.VV.arr$ $\mu\nu$
shows $D.arr$ (*cmp* $\mu\nu$)
and $D.dom$ (*cmp* $\mu\nu$) = $H_D oGF$-$GF.map$ ($B.VV.dom$ $\mu\nu$)
and $D.cod$ (*cmp* $\mu\nu$) = $GFoH_B.map$ ($B.VV.cod$ $\mu\nu$)
using *assms cmp-in-hom* **by** *blast+*

interpretation Φ: *natural-transformation*
$B.VV.comp$ V_D $H_D oGF$-$GF.map$ $GFoH_B.map$ *cmp*
proof

show $\bigwedge\mu\nu.\ \neg\ B.VV.arr\ \mu\nu \Longrightarrow cmp\ \mu\nu = D.null$
unfolding *cmp-def* **by** *simp*
fix $\mu\nu$
assume $\mu\nu$: $B.VV.arr\ \mu\nu$
show $D.arr\ (cmp\ \mu\nu)$
using $\mu\nu$ *cmp-in-hom* **by** *blast*
show $GFoH_B.map\ \mu\nu \cdot_D cmp\ (B.VV.dom\ \mu\nu) = cmp\ \mu\nu$
unfolding *cmp-def*
using $\mu\nu$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *D.comp-ide-arr* $B.VV.dom\text{-}char_{SbC}$ *D.comp-assoc*
as-nat-trans.naturality1
apply *simp*
by (*metis* (*no-types*, *lifting*) *B.H.preserves-arr B.hcomp-simps*(*3*))
show $cmp\ (B.VV.cod\ \mu\nu) \cdot_D H_D oGF\text{-}GF.map\ \mu\nu = cmp\ \mu\nu$
proof −
have $cmp\ (B.VV.cod\ \mu\nu) \cdot_D H_D oGF\text{-}GF.map\ \mu\nu =$
$(G\ (F\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))) \cdot_D$
$G\ (\Phi_F\ (B.cod\ (fst\ \mu\nu), B.cod\ (snd\ \mu\nu))) \cdot_D$
$\Phi_G\ (F\ (B.cod\ (fst\ \mu\nu)), F\ (B.cod\ (snd\ \mu\nu)))) \cdot_D$
$(fst\ (FF\ \mu\nu) \star_D snd\ (FF\ \mu\nu))$
unfolding *cmp-def*
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp* **by** *simp*
also have ... $= (G\ (\Phi_F\ (B.cod\ (fst\ \mu\nu), B.cod\ (snd\ \mu\nu))) \cdot_D$
$\Phi_G\ (F\ (B.cod\ (fst\ \mu\nu)), F\ (B.cod\ (snd\ \mu\nu)))) \cdot_D$
$(fst\ (FF\ \mu\nu) \star_D snd\ (FF\ \mu\nu))$
proof −
have $D.ide\ (G\ (F\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))))$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ **by** *simp*
moreover have $D.seq\ (G\ (F\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))))$
$(G\ (\Phi_F\ (B.cod\ (fst\ \mu\nu), B.cod\ (snd\ \mu\nu))) \cdot_D$
$\Phi_G\ (F\ (B.cod\ (fst\ \mu\nu)), F\ (B.cod\ (snd\ \mu\nu))))$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *F.FF-def*
apply (*intro D.seqI*)
apply *auto*
proof −
show $G\ (F\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))) =$
$D.cod\ (G\ (\Phi_F\ (B.cod\ (fst\ \mu\nu), B.cod\ (snd\ \mu\nu))) \cdot_D$
$\Phi_G\ (F\ (B.cod\ (fst\ \mu\nu)), F\ (B.cod\ (snd\ \mu\nu))))$
proof −
have $D.seq\ (G\ (\Phi_F\ (B.cod\ (fst\ \mu\nu), B.cod\ (snd\ \mu\nu))))$
$(\Phi_G\ (F\ (B.cod\ (fst\ \mu\nu)), F\ (B.cod\ (snd\ \mu\nu))))$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *F.FF-def* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$
by (*intro D.seqI*) *auto*
thus *?thesis*
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *B.VV.cod-simp* **by** *simp*
qed
qed
ultimately show *?thesis*

using $\mu\nu$ *D.comp-ide-arr* [*of* G (F ($B.cod$ (fst $\mu\nu$) $\star_B$ $B.cod$ (snd $\mu\nu$)))
G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$
Φ_G (F ($B.cod$ (fst $\mu\nu$)), F ($B.cod$ (snd $\mu\nu$)))]
by *simp*
qed
also have ... = G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$
(Φ_G (F ($B.cod$ (fst $\mu\nu$)), F ($B.cod$ (snd $\mu\nu$))) $\cdot_D$
(fst (FF $\mu\nu$) $\star_D$ snd (FF $\mu\nu$)))
using *D.comp-assoc* **by** *simp*
also have ... = G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$
Φ_G ($C.VV.cod$ ($F.FF$ $\mu\nu$)) $\cdot_D$ $G.H_D oFF.map$ ($F.FF$ $\mu\nu$)
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *F.FF-def G.FF-def FF-def C.VV.cod-simp* **by** *auto*
also have ... = G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$
$G.FoH_C.map$ ($F.FF$ $\mu\nu$) $\cdot_D$ Φ_G ($C.VV.dom$ ($F.FF$ $\mu\nu$))
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ $G.\Phi.naturality$ *C.VV.dom-simp C.VV.cod-simp* **by** *simp*
also have ... = (G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$
$G.FoH_C.map$ ($F.FF$ $\mu\nu$)) $\cdot_D$ Φ_G ($C.VV.dom$ ($F.FF$ $\mu\nu$))
using *D.comp-assoc* **by** *simp*
also have ... = (G (Φ_F ($B.VV.cod$ $\mu\nu$) $\cdot_C$ $F.H_D oFF.map$ $\mu\nu$)) $\cdot_D$
Φ_G ($C.VV.dom$ ($F.FF$ $\mu\nu$))
proof −
have ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$)) = $B.VV.cod$ $\mu\nu$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *B.VV.cod-simp* **by** *simp*
moreover have $G.FoH_C.map$ ($F.FF$ $\mu\nu$) = G ($F.H_D oFF.map$ $\mu\nu$)
using $\mu\nu$ *F.FF-def* **by** *simp*
moreover have G (Φ_F ($B.cod$ (fst $\mu\nu$), $B.cod$ (snd $\mu\nu$))) $\cdot_D$ G ($F.H_D oFF.map$ $\mu\nu$) =
G (Φ_F ($B.VV.cod$ $\mu\nu$) $\cdot_C$ $F.H_D oFF.map$ $\mu\nu$)
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *F.Φ.naturality2 F.Φ.preserves-reflects-arr*
G.preserves-comp calculation(*1*))
ultimately show *?thesis* **by** *argo*
qed
also have ... = G ($F.FoH_C.map$ $\mu\nu$ $\cdot_C$ Φ_F ($B.VV.dom$ $\mu\nu$)) $\cdot_D$
Φ_G ($C.VV.dom$ ($F.FF$ $\mu\nu$))
using $\mu\nu$ *F.Φ.naturality* [*of* $\mu\nu$] *F.FF-def* **by** *simp*
also have ... = G ($F.FoH_C.map$ $\mu\nu$) $\cdot_D$ G (Φ_F ($B.VV.dom$ $\mu\nu$)) $\cdot_D$
Φ_G ($C.VV.dom$ ($F.FF$ $\mu\nu$))
proof −
have G ($F.FoH_C.map$ $\mu\nu$ $\cdot_C$ Φ_F ($B.VV.dom$ $\mu\nu$)) =
G ($F.FoH_C.map$ $\mu\nu$) $\cdot_D$ G (Φ_F ($B.VV.dom$ $\mu\nu$))
using $\mu\nu$
by (*metis* (*mono-tags*, *lifting*) *F.Φ.naturality1 F.Φ.preserves-reflects-arr*
G.preserves-comp)
thus *?thesis*
using $\mu\nu$ *D.comp-assoc* **by** *simp*
qed
also have ... = cmp $\mu\nu$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *cmp-def F.FF-def F.FF.preserves-dom B.VV.dom-simp*
by *auto*

finally show *?thesis* **by** *simp*
qed
qed

interpretation Φ: *natural-isomorphism* $B.VV.comp$ V_D $H_D o GF\text{-}GF.map$ $GFoH_B.map$ *cmp*
proof
show $\bigwedge hk.\ B.VV.ide\ hk \Longrightarrow D.iso\ (cmp\ hk)$
proof −
fix *hk*
assume *hk*: $B.VV.ide\ hk$
have $D.iso\ (G\ (F\ (fst\ hk \star_B snd\ hk)) \cdot_D G\ (\Phi_F\ (B.VV.dom\ hk)) \cdot_D$
$\Phi_G\ (F\ (B.dom\ (fst\ hk)),\ F\ (B.dom\ (snd\ hk))))$
proof (*intro D.isos-compose*)
show $D.iso\ (\Phi_G\ (F\ (B.dom\ (fst\ hk)),\ F\ (B.dom\ (snd\ hk))))$
using *hk* $G.\Phi.components\text{-}are\text{-}iso$ [*of* $(F\ (B.dom\ (fst\ hk)),\ F\ (B.dom\ (snd\ hk)))$]
$C.VV.ide\text{-}char$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) $B.VV.ideD(1)$ $B.VV.ideD(2)$ $B.VxV.dom\text{-}char$
$F.FF\text{-}def$ $F.FF.as\text{-}nat\text{-}iso.components\text{-}are\text{-}iso$ $G.\Phi.preserves\text{-}iso$ *fst-conv snd-conv*)
show $D.iso\ (G\ (\Phi_F\ (B.VV.dom\ hk)))$
using *hk* $F.\Phi.components\text{-}are\text{-}iso$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.ideD(2)$
by *auto*
show $D.seq\ (G\ (\Phi_F\ (B.VV.dom\ hk)))\ (\Phi_G\ (F\ (B.dom\ (fst\ hk)),\ F\ (B.dom\ (snd\ hk))))$
using *hk* $B.VV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
$F.FF\text{-}def$
$C.VV.dom\text{-}simp$ $C.VV.cod\text{-}simp$
by *auto*
show $D.iso\ (G\ (F\ (fst\ hk \star_B snd\ hk)))$
using *hk* $F.\Phi.components\text{-}are\text{-}iso$ $B.VV.arr\text{-}char_{SbC}$ **by** *simp*
show $D.seq\ (G\ (F\ (fst\ hk \star_B snd\ hk)))$
$(G\ (\Phi_F\ (B.VV.dom\ hk)) \cdot_D \Phi_G\ (F\ (B.dom\ (fst\ hk)),\ F\ (B.dom\ (snd\ hk))))$
using *hk* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) $B.VV.ideD(1)$ *cmp-def cmp-simps*(*1*))
qed
thus $D.iso\ (cmp\ hk)$
unfolding *cmp-def* **using** *hk* **by** *simp*
qed
qed

sublocale *pseudofunctor* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D ‹*G o F*› *cmp*
proof
fix *f g h*
assume *f*: $B.ide\ f$ **and** *g*: $B.ide\ g$ **and** *h*: $B.ide\ h$
assume *fg*: $src_B\ f = trg_B\ g$ **and** *gh*: $src_B\ g = trg_B\ h$
show $map\ \mathrm{a}_B[f,\ g,\ h] \cdot_D cmp\ (f \star_B g,\ h) \cdot_D (cmp\ (f,\ g) \star_D map\ h) =$
$cmp\ (f,\ g \star_B h) \cdot_D (map\ f \star_D cmp\ (g,\ h)) \cdot_D \mathrm{a}_D[map\ f,\ map\ g,\ map\ h]$
proof −
have $map\ \mathrm{a}_B[f,\ g,\ h] \cdot_D cmp\ (f \star_B g,\ h) \cdot_D (cmp\ (f,\ g) \star_D map\ h) =$
$G\ (F\ \mathrm{a}_B[f,\ g,\ h]) \cdot_D$
$(G\ (F\ ((f \star_B g) \star_B h)) \cdot_D G\ (\Phi_F\ (f \star_B g,\ h)) \cdot_D \Phi_G\ (F\ (f \star_B g),\ F\ h)) \cdot_D$

$(G\ (F\ (f \star_B g)) \cdot_D G\ (\Phi_F\ (f,\ g)) \cdot_D \Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
unfolding *cmp-def*
using *f g h fg gh B.VV.arr-char$_{SbC}$ B.VV.dom-simp* **by** *simp*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h]) \cdot_D$
$(G\ (\Phi_F\ (f \star_B g,\ h)) \cdot_D \Phi_G\ (F\ (f \star_B g),\ F\ h)) \cdot_D$
$(G\ (F\ (f \star_B g)) \cdot_D G\ (\Phi_F\ (f,\ g)) \cdot_D \Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh D.comp-ide-arr D.comp-assoc*
by (*metis B.ideD(1) B.ide-hcomp B.src-hcomp F.cmp-simps(1) F.cmp-simps(5) G.as-nat-trans.naturality2*)
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h]) \cdot_D$
$(G\ (\Phi_F\ (f \star_B g,\ h)) \cdot_D \Phi_G\ (F\ (f \star_B g),\ F\ h)) \cdot_D$
$(G\ (\Phi_F\ (f,\ g)) \cdot_D \Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g fg*
by (*metis* (*no-types*) *D.comp-assoc F.cmp-simps(1) F.cmp-simps(5) G.as-nat-trans.naturality2*)
also have ... $= (G\ (F\ \mathrm{a}_B[f,\ g,\ h]) \cdot_D G\ (\Phi_F\ (f \star_B g,\ h))) \cdot_D$
$\Phi_G\ (F\ (f \star_B g),\ F\ h) \cdot_D (G\ (\Phi_F\ (f,\ g)) \cdot_D \Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h] \cdot_C \Phi_F\ (f \star_B g,\ h)) \cdot_D$
$\Phi_G\ (F\ (f \star_B g),\ F\ h) \cdot_D (G\ (\Phi_F\ (f,\ g)) \cdot_D \Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ B.VV.cod-simp* **by** *simp*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h] \cdot_C \Phi_F\ (f \star_B g,\ h)) \cdot_D$
$\Phi_G\ (F\ (f \star_B g),\ F\ h) \cdot_D (G\ (\Phi_F\ (f,\ g)) \star_D G\ (F\ h)) \cdot_D$
$(\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ C.VV.arr-char$_{SbC}$ F.FF-def D.whisker-right B.VV.dom-simp C.VV.cod-simp*
by *auto*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h] \cdot_C \Phi_F\ (f \star_B g,\ h)) \cdot_D$
$(\Phi_G\ (F\ (f \star_B g),\ F\ h) \cdot_D (G\ (\Phi_F\ (f,\ g)) \star_D G\ (F\ h))) \cdot_D$
$(\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h] \cdot_C \Phi_F\ (f \star_B g,\ h)) \cdot_D$
$(G\ (\Phi_F\ (f,\ g) \star_C F\ h) \cdot_D \Phi_G\ (F\ f \star_C F\ g,\ F\ h)) \cdot_D$
$(\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
proof −
have $\Phi_G\ (F\ (f \star_B g),\ F\ h) = \Phi_G\ (C.VV.cod\ (\Phi_F\ (f,\ g),\ F\ h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ C.VV.arr-char$_{SbC}$ B.VV.cod-simp C.VV.cod-simp*
by *simp*
moreover have $G\ (\Phi_F\ (f,\ g)) \star_D G\ (F\ h) = G.H_D oFF.map\ (\Phi_F\ (f,\ g),\ F\ h)$
using *f g h fg gh B.VV.arr-char$_{SbC}$ C.VV.arr-char$_{SbC}$ G.FF-def* **by** *simp*
moreover have $G.FoH_C.map\ (\Phi_F\ (f,\ g),\ F\ h) = G\ (\Phi_F\ (f,\ g) \star_C F\ h)$
using *f g h fg gh B.VV.arr-char$_{SbC}$* **by** *simp*
moreover have $\Phi_G\ (C.VV.dom\ (\Phi_F\ (f,\ g),\ F\ h)) = \Phi_G\ (F\ f \star_C F\ g,\ F\ h)$
using *f g h fg gh C.VV.arr-char$_{SbC}$ F.cmp-in-hom* [*of f g*] *C.VV.dom-simp* **by** *auto*
ultimately show *?thesis*
using *f g h fg gh B.VV.arr-char$_{SbC}$ G.Φ.naturality*
by (*metis* (*mono-tags*, *lifting*) *C.VV.arr-cod-iff-arr C.VV.arr-dom-iff-arr G.FoH$_C$.extensionality G.H$_D$oFF.extensionality G.Φ.extensionality*)
qed

also have ... $= (G\ (F\ \mathrm{a}_B[f,\ g,\ h]\ \cdot_C\ \Phi_F\ (f \star_B g,\ h))\ \cdot_D\ (G\ (\Phi_F\ (f,\ g) \star_C F\ h)))\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ ((F\ \mathrm{a}_B[f,\ g,\ h]\ \cdot_C\ \Phi_F\ (f \star_B g,\ h))\ \cdot_C\ (\Phi_F\ (f,\ g) \star_C F\ h))\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ F.FF-def B.VV.dom-simp B.VV.cod-simp* **by** *auto*
also have ... $= G\ (F\ \mathrm{a}_B[f,\ g,\ h]\ \cdot_C\ \Phi_F\ (f \star_B g,\ h)\ \cdot_C\ (\Phi_F\ (f,\ g) \star_C F\ h))\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *C.comp-assoc* **by** *simp*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h))\ \cdot_C\ \mathrm{a}_C[F\ f,\ F\ g,\ F\ h])\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh F.assoc-coherence* [*of f g h*] **by** *simp*
also have ... $= G\ ((\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_C\ \mathrm{a}_C[F\ f,\ F\ g,\ F\ h])\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *C.comp-assoc* **by** *simp*
also have ... $= (G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_D\ G\ \mathrm{a}_C[F\ f,\ F\ g,\ F\ h])\ \cdot_D$
$\Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D\ (\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ F.FF-def B.VV.dom-simp B.VV.cod-simp* **by** *auto*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_D$
$G\ \mathrm{a}_C[F\ f,\ F\ g,\ F\ h]\ \cdot_D\ \Phi_G\ (F\ f \star_C F\ g,\ F\ h)\ \cdot_D$
$(\Phi_G\ (F\ f,\ F\ g) \star_D G\ (F\ h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_D$
$\Phi_G\ (F\ f,\ F\ g \star_C F\ h)\ \cdot_D\ (G\ (F\ f) \star_D \Phi_G\ (F\ g,\ F\ h))\ \cdot_D$
$\mathrm{a}_D[G\ (F\ f),\ G\ (F\ g),\ G\ (F\ h)]$
using *f g h fg gh G.assoc-coherence* [*of F f F g F h*] **by** *simp*
also have ... $= (G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_D$
$\Phi_G\ (F\ f,\ F\ g \star_C F\ h)\ \cdot_D\ (G\ (F\ f) \star_D \Phi_G\ (F\ g,\ F\ h)))\ \cdot_D$
$\mathrm{a}_D[G\ (F\ f),\ G\ (F\ g),\ G\ (F\ h)]$
using *D.comp-assoc* **by** *simp*
also have ... $= (cmp\ (f,\ g \star_B h)\ \cdot_D\ (G\ (F\ f) \star_D cmp\ (g,\ h)))\ \cdot_D$
$\mathrm{a}_D[G\ (F\ f),\ G\ (F\ g),\ G\ (F\ h)]$
proof −
have $G\ (\Phi_F\ (f,\ g \star_B h)\ \cdot_C\ (F\ f \star_C \Phi_F\ (g,\ h)))\ \cdot_D$
$\Phi_G\ (F\ f,\ F\ g \star_C F\ h)\ \cdot_D\ (G\ (F\ f) \star_D \Phi_G\ (F\ g,\ F\ h)) =$
$cmp\ (f,\ g \star_B h)\ \cdot_D\ (G\ (F\ f) \star_D cmp\ (g,\ h))$
proof −
have $cmp\ (f,\ g \star_B h)\ \cdot_D\ (G\ (F\ f) \star_D cmp\ (g,\ h)) =$
$(G\ (F\ (f \star_B g \star_B h))\ \cdot_D\ G\ (\Phi_F\ (f,\ g \star_B h))\ \cdot_D\ \Phi_G\ (F\ f,\ F\ (g \star_B h)))\ \cdot_D$
$(G\ (F\ f) \star_D G\ (F\ (g \star_B h))\ \cdot_D\ G\ (\Phi_F\ (g,\ h))\ \cdot_D\ \Phi_G\ (F\ g,\ F\ h))$
unfolding *cmp-def*
using *f g h fg gh B.VV.arr-char$_{SbC}$ B.VV.dom-simp B.VV.cod-simp* **by** *simp*
also have ... $= (G\ (\Phi_F\ (f,\ g \star_B h))\ \cdot_D\ \Phi_G\ (F\ f,\ F\ (g \star_B h)))\ \cdot_D$
$(G\ (F\ f) \star_D G\ (F\ (g \star_B h))\ \cdot_D\ G\ (\Phi_F\ (g,\ h))\ \cdot_D\ \Phi_G\ (F\ g,\ F\ h))$
proof −
have $G\ (F\ (f \star_B g \star_B h))\ \cdot_D\ G\ (\Phi_F\ (f,\ g \star_B h)) = G\ (\Phi_F\ (f,\ g \star_B h))$
using *f g h fg gh B.VV.arr-char$_{SbC}$ D.comp-ide-arr B.VV.dom-simp B.VV.cod-simp*
by *simp*
thus *?thesis*

using *D.comp-assoc* **by** *metis*
qed
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D \Phi_G\ (F\,f,\ F\ (g \star_B h)) \cdot_D$
$(G\ (F\,f) \star_D G\ (F\ (g \star_B h)) \cdot_D G\ (\Phi_F\ (g,\ h)) \cdot_D \Phi_G\ (F\,g,\ F\,h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D \Phi_G\ (F\,f,\ F\ (g \star_B h)) \cdot_D$
$(G\ (F\,f) \star_D G\ (\Phi_F\ (g,\ h)) \cdot_D \Phi_G\ (F\,g,\ F\,h))$
proof −
have $G\ (F\ (g \star_B h)) \cdot_D G\ (\Phi_F\ (g,\ h)) = G\ (\Phi_F\ (g,\ h))$
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ *D.comp-ide-arr* $B.VV.dom\text{-}simp$ $B.VV.cod\text{-}simp$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *metis*
qed
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D \Phi_G\ (F\,f,\ F\ (g \star_B h)) \cdot_D$
$(G\ (F\,f) \star_D G\ (\Phi_F\ (g,\ h))) \cdot_D (G\ (F\,f) \star_D \Phi_G\ (F\,g,\ F\,h))$
using *f g h fg gh*
D.whisker-left [*of* $G\ (F\,f)\ G\ (\Phi_F\ (g,\ h))\ \Phi_G\ (F\,g,\ F\,h)$]
by *fastforce*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D$
$(\Phi_G\ (F\,f,\ F\ (g \star_B h)) \cdot_D (G\ (F\,f) \star_D G\ (\Phi_F\ (g,\ h)))) \cdot_D$
$(G\ (F\,f) \star_D \Phi_G\ (F\,g,\ F\,h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D$
$(G\ (F\,f \star_C \Phi_F\ (g,\ h)) \cdot_D \Phi_G\ (F\,f,\ F\,g \star_C F\,h)) \cdot_D$
$(G\ (F\,f) \star_D \Phi_G\ (F\,g,\ F\,h))$
proof −
have $\Phi_G\ (C.VV.cod\ (F\,f,\ \Phi_F\ (g,\ h))) = \Phi_G\ (F\,f,\ F\ (g \star_B h))$
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $B.VV.dom\text{-}simp$
$B.VV.cod\text{-}simp$
by *auto*
moreover have $G.H_D oFF.map\ (F\,f,\ \Phi_F\ (g,\ h)) = G\ (F\,f) \star_D G\ (\Phi_F\ (g,\ h))$
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ *G.FF-def* **by** *auto*
moreover have $G.FoH_C.map\ (F\,f,\ \Phi_F\ (g,\ h)) = G\ (F\,f \star_C \Phi_F\ (g,\ h))$
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ **by** *simp*
moreover have $C.VV.dom\ (F\,f,\ \Phi_F\ (g,\ h)) = (F\,f,\ F\,g \star_C F\,h)$
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$
F.cmp-in-hom [*of g h*]
by *auto*
ultimately show *?thesis*
using *f g h fg gh* $B.VV.arr\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
$G.\Phi.naturality$ [*of* $(F\,f,\ \Phi_F\ (g,\ h))$]
by *simp*
qed
also have ... $= (G\ (\Phi_F\ (f,\ g \star_B h)) \cdot_D (G\ (F\,f \star_C \Phi_F\ (g,\ h)))) \cdot_D$
$\Phi_G\ (F\,f,\ F\,g \star_C F\,h) \cdot_D (G\ (F\,f) \star_D \Phi_G\ (F\,g,\ F\,h))$
using *D.comp-assoc* **by** *simp*
also have ... $= G\ (\Phi_F\ (f,\ g \star_B h) \cdot_C (F\,f \star_C \Phi_F\ (g,\ h))) \cdot_D$
$\Phi_G\ (F\,f,\ F\,g \star_C F\,h) \cdot_D (G\ (F\,f) \star_D \Phi_G\ (F\,g,\ F\,h))$

using *f g h fg gh B.VV.arr-char*$_{SbC}$
by (*metis* (*no-types*, *lifting*) *B.assoc-simps*(*1*) *C.comp-assoc C.seqE*
F.preserves-assoc(*1*) *F.preserves-reflects-arr G.preserves-comp*)
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = *cmp* (*f*, *g* $\star_B$ *h*) $\cdot_D$ (*G* (*F f*) $\star_D$ *cmp* (*g*, *h*)) $\cdot_D$
a_D[*G* (*F f*), *G* (*F g*), *G* (*F h*)]
using *D.comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *is-pseudofunctor*:
shows *pseudofunctor* V_B H_B a_B i_B *src*$_B$ *trg*$_B$ V_D H_D a_D i_D *src*$_D$ *trg*$_D$ (*G o F*) *cmp*
..

lemma *map*$_0$*-simp* [*simp*]:
assumes *B.obj a*
shows *map*$_0$ *a* = *G.map*$_0$ (*F.map*$_0$ *a*)
using *assms map*$_0$*-def* **by** *auto*

lemma *unit-char*$'$:
assumes *B.obj a*
shows *unit a* = *G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*)
proof −
have *G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*) = *unit a*
proof (*intro unit-eqI* [*of a G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*)])
show *B.obj a* **by** *fact*
show «*G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*) : *map*$_0$ *a* $\Rightarrow_D$ *map a*»
using *assms* **by** *auto*
show *D.iso* (*G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*))
by (*simp add*: *D.isos-compose F.unit-char*(*2*) *G.unit-char*(*2*) *assms*)
show (*G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*)) $\cdot_D$ i_D[*map*$_0$ *a*] =
(*map* i_B[*a*] $\cdot_D$ *cmp* (*a*, *a*)) $\cdot_D$
(*G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*) $\star_D$ *G* (*F.unit a*) $\cdot_D$ *G.unit* (*F.map*$_0$ *a*))
proof −
have *1*: *cmp* (*a*, *a*) = *G* (Φ_F (*a*, *a*)) $\cdot_D$ Φ_G (*F a*, *F a*)
proof −
have *cmp* (*a*, *a*) = (*G* (*F* (*a* $\star_B$ *a*)) $\cdot_D$ *G* (Φ_F (*a*, *a*))) $\cdot_D$ Φ_G (*F a*, *F a*)
using *assms cmp-def B.VV.ide-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *B.VV.dom-char*$_{SbC}$
B.VV.cod-char$_{SbC}$
B.VxV.dom-char B.objE D.comp-assoc B.obj-simps
by *simp*
also have ... = *G* (Φ_F (*a*, *a*)) $\cdot_D$ Φ_G (*F a*, *F a*)
using *assms D.comp-cod-arr B.VV.arr-char*$_{SbC}$ *B.VV.ide-char*$_{SbC}$ **by** *auto*
finally show *?thesis* **by** *blast*
qed

```
      have (map i_B[a] ·_D cmp (a, a)) ·_D
             (G (F.unit a) ·_D G.unit (F.map_0 a) ⋆_D G (F.unit a) ·_D G.unit (F.map_0 a)) =
          map i_B[a] ·_D G (Φ_F (a, a)) ·_D
             (Φ_G (F a, F a) ·_D (G (F.unit a) ⋆_D G (F.unit a))) ·_D
             (G.unit (F.map_0 a) ⋆_D G.unit (F.map_0 a))
        using assms 1 D.comp-assoc D.interchange by simp
      also have ... = (map i_B[a] ·_D G (Φ_F (a, a)) ·_D G (F.unit a ⋆_C F.unit a)) ·_D
                      Φ_G (F.map_0 a, F.map_0 a) ·_D
                      (G.unit (F.map_0 a) ⋆_D G.unit (F.map_0 a))
        using assms G.Φ.naturality [of (F.unit a, F.unit a)]
          C.VV.arr-char_SbC C.VV.dom-char_SbC C.VV.cod-char_SbC G.FF-def D.comp-assoc
        by simp
      also have ... = (G (F i_B[a] ·_C Φ_F (a, a) ·_C (F.unit a ⋆_C F.unit a))) ·_D
                      Φ_G (F.map_0 a, F.map_0 a) ·_D
                      (G.unit (F.map_0 a) ⋆_D G.unit (F.map_0 a))
      proof −
        have C.arr (F i_B[a] ·_C Φ_F (a, a) ·_C (F.unit a ⋆_C F.unit a))
          using assms B.VV.ide-char_SbC B.VV.arr-char_SbC F.cmp-in-hom(2)
          by (intro C.seqI' C.comp-in-homI) auto
        hence map i_B[a] ·_D G (Φ_F (a, a)) ·_D G (F.unit a ⋆_C F.unit a) =
              G (F i_B[a] ·_C Φ_F (a, a) ·_C (F.unit a ⋆_C F.unit a))
          by auto
        thus ?thesis by argo
      qed
      also have ... = G (F.unit a ·_C i_C[F.map_0 a]) ·_D
                      Φ_G (F.map_0 a, F.map_0 a) ·_D
                      (G.unit (F.map_0 a) ⋆_D G.unit (F.map_0 a))
        using assms F.unit-char C.comp-assoc by simp
      also have ... = G (F.unit a) ·_D (G i_C[F.map_0 a] ·_D
                      Φ_G (F.map_0 a, F.map_0 a)) ·_D
                      (G.unit (F.map_0 a) ⋆_D G.unit (F.map_0 a))
        using assms D.comp-assoc by simp
      also have ... = (G (F.unit a) ·_D G.unit (F.map_0 a)) ·_D i_D[G.map_0 (F.map_0 a)]
        using assms G.unit-char D.comp-assoc by simp
      also have ... = (G (F.unit a) ·_D G.unit (F.map_0 a)) ·_D i_D[map_0 a]
        using assms map_0-def by auto
      finally show ?thesis by simp
    qed
  qed
  thus ?thesis by simp
qed

end
```

1.10.8 Restriction of Pseudofunctors

In this section, we construct the restriction and corestriction of a pseudofunctor to a subbicategory of its domain and codomain, respectively.

```
locale restricted-pseudofunctor =
```

C: *bicategory* V_C H_C a_C i_C src_C trg_C +
D: *bicategory* V_D H_D a_D i_D src_D trg_D +
F: *pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ +
C': *subbicategory* V_C H_C a_C i_C src_C trg_C *Arr*
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ :: $'c * 'c \Rightarrow 'd$
and *Arr* :: $'c \Rightarrow$ *bool*
begin

abbreviation *map*
where *map* $\equiv \lambda\mu$. *if* C'.*arr* μ *then* F μ *else* D.*null*

abbreviation *cmp*
where *cmp* $\equiv \lambda\mu\nu$. *if* C'.VV.*arr* $\mu\nu$ *then* Φ $\mu\nu$ *else* D.*null*

interpretation *functor* C'.*comp* V_D *map*
using C'.*inclusion* C'.*arr-char*$_{SbC}$ C'.*dom-char*$_{SbC}$ C'.*cod-char*$_{SbC}$ C'.*seq-char*$_{SbC}$ C'.*comp-char* C'.*arr-dom* C'.*arr-cod*
apply (*unfold-locales*)
apply *auto*
by *presburger*

interpretation *weak-arrow-of-homs* C'.*comp* C'.*src* C'.*trg* V_D src_D trg_D *map*
using C'.*arrE* C'.*ide-src* C'.*ide-trg* C'.*inclusion* C'.*src-def* C'.*trg-def* F.*weakly-preserves-src* F.*weakly-preserves-trg*
by *unfold-locales auto*

interpretation $H_D{}'oFF$: *composite-functor* C'.VV.*comp* D.VV.*comp* V_D *FF* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_D$ *snd* $\mu\nu$›
..
interpretation $FoH_{C'}$: *composite-functor* C'.VV.*comp* C'.*comp* V_D ‹$\lambda\mu\nu$. C'.*hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› *map*
..

interpretation Φ: *natural-transformation* C'.VV.*comp* V_D $H_D{}'oFF$.*map* $FoH_{C'}$.*map* *cmp*
using C'.*arr-char*$_{SbC}$ C'.*dom-char*$_{SbC}$ C'.*cod-char*$_{SbC}$ C'.VV.*arr-char*$_{SbC}$ C'.VV.*dom-char*$_{SbC}$ C'.VV.*cod-char*$_{SbC}$

FF-def C′.inclusion C′.dom-closed C′.cod-closed C′.src-def C′.trg-def
C′.hcomp-def C′.hcomp-closed F.Φ.naturality1 F.Φ.naturality2
C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ F.FF-def
by *unfold-locales auto*

interpretation Φ: *natural-isomorphism C′.VV.comp V$_D$ H$_D$′oFF.map FoH$_{C'}$.map cmp*
using *C.VV.ide-char$_{SbC}$ C.VV.arr-char$_{SbC}$ C′.VV.ide-char$_{SbC}$ C′.VV.arr-char$_{SbC}$ C′.arr-char$_{SbC}$*
C′.src-def C′.trg-def C′.ide-char$_{SbC}$ F.Φ.components-are-iso
by *unfold-locales auto*

sublocale *pseudofunctor C′.comp C′.hcomp C′.*a i$_C$ *C′.src C′.trg V$_D$ H$_D$* a$_D$ i$_D$ *src$_D$ trg$_D$ map cmp*
using *F.assoc-coherence C′.VVV.arr-char$_{SbC}$ C′.VV.arr-char$_{SbC}$ C′.arr-char$_{SbC}$ C′.hcomp-def*
C′.src-def C′.trg-def C′.assoc-closed C′.hcomp-closed C′.ide-char$_{SbC}$
by *unfold-locales auto*

lemma *is-pseudofunctor*:
shows *pseudofunctor C′.comp C′.hcomp C′.*a i$_C$ *C′.src C′.trg V$_D$ H$_D$* a$_D$ i$_D$ *src$_D$ trg$_D$ map cmp*
..

lemma *map$_0$-simp* [*simp*]:
assumes *C′.obj a*
shows *map$_0$ a = F.map$_0$ a*
using *assms map$_0$-def C′.obj-char* **by** *auto*

lemma *unit-char′*:
assumes *C′.obj a*
shows *F.unit a = unit a*
using *assms map$_0$-simp C′.obj-char F.unit-in-hom(2)* [*of a*] *F.unit-char(2−3)* i-*simps(1)*
apply (*intro unit-eqI*)
apply *auto*
by *blast*

end

We define the corestriction construction only for the case of sub-bicategories determined by a set of objects of the ambient bicategory. There are undoubtedly more general constructions, but this one is adequate for our present needs.

locale *corestricted-pseudofunctor* =
C: *bicategory V$_C$ H$_C$* a$_C$ i$_C$ *src$_C$ trg$_C$* +
D: *bicategory V$_D$ H$_D$* a$_D$ i$_D$ *src$_D$ trg$_D$* +
F: *pseudofunctor V$_C$ H$_C$* a$_C$ i$_C$ *src$_C$ trg$_C$ V$_D$ H$_D$* a$_D$ i$_D$ *src$_D$ trg$_D$ F* Φ +
D′: *subbicategory V$_D$ H$_D$* a$_D$ i$_D$ *src$_D$ trg$_D$* ‹λμ. *D.arr* μ ∧ *Obj* (*src$_D$* μ) ∧ *Obj* (*trg$_D$* μ)›
for *V$_C$* :: *′c comp* (**infixr** ‹·$_C$› *55*)
and *H$_C$* :: *′c comp* (**infixr** ‹⋆$_C$› *53*)
and a$_C$ :: *′c* ⇒ *′c* ⇒ *′c* ⇒ *′c* (‹a$_C$[-, -, -]›)
and i$_C$ :: *′c* ⇒ *′c* (‹i$_C$[-]›)

and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ :: $'c * 'c \Rightarrow 'd$
and *Obj* :: $'d \Rightarrow$ *bool* +
assumes *preserves-arr*: $\bigwedge\mu.\ C.arr\ \mu \Longrightarrow D'.arr\ (F\ \mu)$
begin

abbreviation *map*
where $map \equiv F$

abbreviation *cmp*
where $cmp \equiv \Phi$

interpretation *functor* V_C $D'.comp$ F
using *preserves-arr* *F.extensionality* $D'.arr\text{-}char_{SbC}$ $D'.dom\text{-}char_{SbC}$ $D'.cod\text{-}char_{SbC}$ $D'.comp\text{-}char$
by (*unfold-locales*) *auto*

interpretation *weak-arrow-of-homs* V_C src_C trg_C $D'.comp$ $D'.src$ $D'.trg$ F
proof
fix μ
assume μ: $C.arr\ \mu$
obtain φ **where** φ: «$\varphi : F\ (src_C\ \mu) \Rightarrow_D src_D\ (F\ \mu)$» $\wedge$ $D.iso\ \varphi$
using μ *F.weakly-preserves-src* **by** *auto*
have *2*: $D'.in\text{-}hom\ \varphi\ (F\ (src_C\ \mu))\ (D'.src\ (F\ \mu))$
using μ φ $D'.arr\text{-}char_{SbC}$ $D'.dom\text{-}char_{SbC}$ $D'.cod\text{-}char_{SbC}$ $D'.src\text{-}def$ *D.vconn-implies-hpar*(*1*−*2*) *preserves-arr*
by (*metis* (*no-types*, *lifting*) *C.src.preserves-arr* $D'.in\text{-}hom\text{-}char_{SbC}$ *D'.src.preserves-arr* *D.arrI*)
moreover have $D'.iso\ \varphi$
using *2* φ $D'.iso\text{-}char_{SbC}$ $D'.arr\text{-}char_{SbC}$ **by** *fastforce*
ultimately show $D'.isomorphic\ (F\ (src_C\ \mu))\ (D'.src\ (F\ \mu))$
using *D'.isomorphic-def* **by** *auto*
obtain ψ **where** ψ: «$\psi : F\ (trg_C\ \mu) \Rightarrow_D trg_D\ (F\ \mu)$» $\wedge$ $D.iso\ \psi$
using μ *F.weakly-preserves-trg* **by** *auto*
have *2*: $D'.in\text{-}hom\ \psi\ (F\ (trg_C\ \mu))\ (D'.trg\ (F\ \mu))$
using μ ψ $D'.arr\text{-}char_{SbC}$ $D'.dom\text{-}char_{SbC}$ $D'.cod\text{-}char_{SbC}$ $D'.trg\text{-}def$ *D.vconn-implies-hpar*(*1*−*2*) *preserves-arr*
by (*metis* (*no-types*, *lifting*) *C.trg.preserves-arr* $D'.in\text{-}hom\text{-}char_{SbC}$ *D'.trg.preserves-arr* *D.arrI*)
moreover have $D'.iso\ \psi$
using *2* ψ $D'.iso\text{-}char_{SbC}$ $D'.arr\text{-}char_{SbC}$ **by** *fastforce*

ultimately show $D'.isomorphic\ (F\ (trg_C\ \mu))\ (D'.trg\ (F\ \mu))$
using $D'.isomorphic\text{-}def$ **by** *auto*
qed

interpretation $H_{D'}oFF$: *composite-functor* $C.VV.comp\ D'.VV.comp\ D'.comp\ FF$
$\langle\lambda\mu\nu.\ D'.hcomp\ (fst\ \mu\nu)\ (snd\ \mu\nu)\rangle$
..
interpretation FoH_C: *composite-functor* $C.VV.comp\ V_C\ D'.comp\ \langle\lambda\mu\nu.\ fst\ \mu\nu \star_C snd\ \mu\nu\rangle$
F
..

interpretation *natural-transformation* $C.VV.comp\ D'.comp\ H_{D'}oFF.map\ FoH_C.map\ \Phi$
proof
show $\bigwedge\mu\nu.\ \neg\ C.VV.arr\ \mu\nu \Longrightarrow \Phi\ \mu\nu = D'.null$
by (*simp add*: $F.\Phi.extensionality$)
fix $\mu\nu$
assume $\mu\nu$: $C.VV.arr\ \mu\nu$
have *1*: $D'.arr\ (\Phi\ \mu\nu)$
using $\mu\nu\ D'.arr\text{-}char_{SbC}\ F.\Phi.naturality1\ F.\Phi.components\text{-}are\text{-}iso$
by (*metis* (*no-types*, *lifting*) $D.src\text{-}vcomp\ D.trg\text{-}vcomp\ FoH_C.preserves\text{-}arr$
$F.\Phi.preserves\text{-}reflects\text{-}arr$)
show $D'.arr\ (\Phi\ \mu\nu)$
using *1* **by** *simp*
show $D'.comp\ (FoH_C.map\ \mu\nu)\ (\Phi\ (C.VV.dom\ \mu\nu)) = \Phi\ \mu\nu$
using *1* $\mu\nu\ D'.arr\text{-}char_{SbC}\ D'.comp\text{-}char\ C.VV.dom\text{-}char_{SbC}\ F.\Phi.naturality1$
$C.VV.arr\text{-}dom\ D.src\text{-}vcomp\ D.trg\text{-}vcomp\ FoH_C.preserves\text{-}arr\ F.\Phi.preserves\text{-}reflects\text{-}arr$
by (*metis* (*mono-tags*, *lifting*))
show $D'.comp\ (\Phi\ (C.VV.cod\ \mu\nu))\ (H_{D'}oFF.map\ \mu\nu) = \Phi\ \mu\nu$
proof −
have $Obj\ (F.map_0\ (trg_C\ (fst\ \mu\nu))) \wedge Obj\ (F.map_0\ (trg_C\ (snd\ \mu\nu)))$
using $\mu\nu\ C.VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) $C.src\text{-}trg\ C.trg.preserves\text{-}reflects\text{-}arr\ D'.arr\text{-}char_{SbC}$
$F.map_0\text{-}def\ preserves\text{-}hseq$)
moreover have $Obj\ (F.map_0\ (src_C\ (snd\ \mu\nu)))$
using $\mu\nu\ C.VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) $C.src.preserves\text{-}reflects\text{-}arr\ C.trg\text{-}src\ D'.arr\text{-}char_{SbC}$
$F.map_0\text{-}def\ preserves\text{-}hseq$)
ultimately show *?thesis*
using $\mu\nu$ *1* $D'.arr\text{-}char_{SbC}\ D'.comp\text{-}char\ D'.hseq\text{-}char\ C.VV.arr\text{-}char_{SbC}\ C.VV.cod\text{-}char_{SbC}$
$C.VxV.cod\text{-}char\ FF\text{-}def\ F.FF\text{-}def\ D'.hcomp\text{-}char\ preserves\text{-}hseq$
apply *simp*
using $F.\Phi.naturality2$ **by** *force*
qed
qed

interpretation *natural-isomorphism* $C.VV.comp\ D'.comp\ H_{D'}oFF.map\ FoH_C.map\ \Phi$
apply *unfold-locales*
using $D'.iso\text{-}char_{SbC}\ D'.arr\text{-}char_{SbC}$ **by** *fastforce*

sublocale *pseudofunctor* V_C H_C a_C i_C src_C trg_C $D'.comp$ $D'.hcomp$ $D'.\mathrm{a}$ i_D $D'.src$ $D'.trg$
F Φ
proof
fix f g h
assume f: $C.ide\ f$ **and** g: $C.ide\ g$ **and** h: $C.ide\ h$
and fg: $src_C\ f = trg_C\ g$ **and** gh: $src_C\ g = trg_C\ h$
have $D'.comp\ (F\ \mathrm{a}_C[f,\ g,\ h])\ (D'.comp\ (\Phi\ (f \star_C g,\ h))\ (D'.hcomp\ (\Phi\ (f,\ g))\ (F\ h))) =$
$F\ \mathrm{a}_C[f,\ g,\ h] \cdot_D \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h)$
proof −
have *1*: $D'.arr\ (cmp\ (f,\ g) \star_D map\ h)$
by (*metis* (*mono-tags*, *lifting*) *C.ideD*(*1*) $D'.arr\text{-}char_{SbC}$ *D'.hcomp-closed*
F.Φ.preserves-reflects-arr F.cmp-simps(*1*−*2*) *F.preserves-hseq*
f fg g gh h preserves-reflects-arr)
moreover have *2*: $D.seq\ (cmp\ (f \star_C g,\ h))\ (cmp\ (f,\ g) \star_D map\ h)$
using *1 f g h fg gh* $D'.arr\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
F.FF-def
by (*intro D.seqI*) *auto*
moreover have $D'.arr\ (cmp\ (f \star_C g,\ h) \cdot_D (cmp\ (f,\ g) \star_D map\ h))$
using *1 2* $D'.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *D'.comp-char D'.seq-char*$_{SbC}$ *D.seqE F.Φ.preserves-reflects-arr*
preserves-reflects-arr)
ultimately show *?thesis*
using *f g h fg gh* $D'.dom\text{-}char_{SbC}$ $D'.cod\text{-}char_{SbC}$ *D'.comp-char D'.hcomp-def* $C.VV.arr\text{-}char_{SbC}$
apply *simp*
by *force*
qed
also have ... $= \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$
using *f g h fg gh F.assoc-coherence* [*of f g h*] **by** *blast*
also have ... $= D'.comp\ (\Phi\ (f,\ g \star_C h))$
$(D'.comp\ (D'.hcomp\ (F\ f)\ (\Phi\ (g,\ h)))\ (D'.\mathrm{a}\ (F\ f)\ (F\ g)\ (F\ h)))$
proof −
have $D.seq\ (map\ f \star_D cmp\ (g,\ h))\ \mathrm{a}_D[map\ f,\ map\ g,\ map\ h]$
using *f g h fg gh* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *F.FF-def*
by (*intro D.seqI*) *auto*
moreover have $D'.arr\ \mathrm{a}_D[map\ f,\ map\ g,\ map\ h]$
using *f g h fg gh* $D'.arr\text{-}char_{SbC}$ *as-nat-trans.preserves-arr* **by** *auto*
moreover have $D'.arr\ (map\ f \star_D cmp\ (g,\ h))$
using *f g h fg gh*
by (*metis* (*no-types*, *lifting*) $D'.arr\text{-}char_{SbC}$ *D.seqE D.vseq-implies-hpar*(*1*)
D.vseq-implies-hpar(*2*) *calculation*(*1*−*2*))
moreover have $D'.arr\ ((map\ f \star_D cmp\ (g,\ h)) \cdot_D \mathrm{a}_D[map\ f,\ map\ g,\ map\ h])$
using *f g h fg gh*
by (*metis* (*no-types*, *lifting*) $D'.arr\text{-}char_{SbC}$ *D'.comp-closed D.seqE*
calculation(*1*−*3*))
moreover have $D.seq\ (cmp\ (f,\ g \star_C h))$
$((map\ f \star_D cmp\ (g,\ h)) \cdot_D \mathrm{a}_D[map\ f,\ map\ g,\ map\ h])$
using *f g h fg gh F.cmp-simps′*(*1*) *F.cmp-simps*(*4*) *F.cmp-simps*(*5*) **by** *auto*
ultimately show *?thesis*
using *f g h fg gh* $C.VV.arr\text{-}char_{SbC}$ $D'.VVV.arr\text{-}char_{SbC}$ $D'.VV.arr\text{-}char_{SbC}$ *D'.comp-char*

D'.*hcomp-def*
by *simp*
qed
finally show D'.*comp* (F $\mathrm{a}_C[f, g, h]$)
(D'.*comp* (Φ ($f \star_C g$, h)) (D'.*hcomp* (Φ (f, g)) (F h))) =
D'.*comp* (Φ (f, $g \star_C h$))
(D'.*comp* (D'.*hcomp* (F f) (Φ (g, h))) (D'.a (F f) (F g) (F h)))
by *blast*
qed

lemma *is-pseudofunctor*:
shows *pseudofunctor* V_C H_C a_C i_C src_C trg_C D'.*comp* D'.*hcomp* D'.a i_D D'.*src* D'.*trg* F Φ
..

lemma map_0*-simp* [*simp*]:
assumes C.*obj* a
shows map_0 a = F.map_0 a
using *assms* map_0*-def* D'.*src-def* **by** *auto*

lemma *unit-char'*:
assumes C.*obj* a
shows F.*unit* a = *unit* a
proof (*intro unit-eqI*)
show C.*obj* a **by** *fact*
show *1*: D'.*in-hom* (F.*unit* a) (map_0 a) (*map* a)
using D'.*arr-char*$_{SbC}$ D'.*in-hom-char*$_{SbC}$ *assms unit-in-hom*(*2*) **by** *force*
show D'.*iso* (F.*unit* a)
using *assms* D'.*iso-char*$_{SbC}$ D'.*arr-char*$_{SbC}$ F.*unit-char*(*2*)
‹D'.*in-hom* (F.*unit* a) (map_0 a) (*map* a)›
by *auto*
show D'.*comp* (F.*unit* a) $\mathrm{i}_D[map_0\ a]$ =
D'.*comp* (D'.*comp* (*map* $\mathrm{i}_C[a]$) (*cmp* (a, a)))
(D'.*hcomp* (F.*unit* a) (F.*unit* a))
proof −
have D'.*comp* (F.*unit* a) $\mathrm{i}_D[map_0\ a]$ = F.*unit* a $\cdot_D$ $\mathrm{i}_D[src_D$ (*map* a)]
using *assms* D'.*comp-char* D'.*arr-char*$_{SbC}$ C.*obj-def* D'.*iso-char*$_{SbC}$
‹D'.*iso* (F.*unit* a)›
by *fastforce*
also have ... = (*map* $\mathrm{i}_C[a]$ $\cdot_D$ *cmp* (a, a)) $\cdot_D$ (F.*unit* a $\star_D$ F.*unit* a)
using *assms* F.*unit-char*(*3*) [*of* a] **by** *auto*
also have ... = D'.*comp* (D'.*comp* (*map* $\mathrm{i}_C[a]$) (*cmp* (a, a)))
(D'.*hcomp* (F.*unit* a) (F.*unit* a))
proof −
have D'.*arr* (*map* $\mathrm{i}_C[a]$ $\cdot_D$ *cmp* (a, a))
using *assms* D'.*comp-simp* **by** *auto*
moreover have D.*seq* (*map* $\mathrm{i}_C[a]$ $\cdot_D$ *cmp* (a, a)) (F.*unit* a $\star_D$ F.*unit* a)
using *assms* $C.VV$.*arr-char*$_{SbC}$ F.*cmp-simps*(*4*−*5*)
by (*intro* D.*seqI*) *auto*
ultimately show *?thesis*

```
        by (metis (no-types, lifting) 1 D'.comp-eqI' D'.hcomp-eqI' D'.hseqI'
            D'.iso-is-arr D'.seq-char_SbC D'.vconn-implies-hpar(1−2)
            i-simps(1) ‹D'.iso (F.unit a)› assms map_0-simps(2−3))
      qed
      finally show ?thesis by blast
    qed
  qed

end
```

1.10.9 Equivalence Pseudofunctors

In this section, we define "equivalence pseudofunctors", which are pseudofunctors that are locally fully faithful, locally essentially surjective, and biessentially surjective on objects. In a later section, we will show that a pseudofunctor is an equivalence pseudofunctor if and only if it can be extended to an equivalence of bicategories.

The definition below requires that an equivalence pseudofunctor be (globally) faithful with respect to vertical composition. Traditional formulations do not consider a pseudofunctor as a single global functor, so we have to consider whether this condition is too strong. In fact, a pseudofunctor (as defined here) is locally faithful if and only if it is globally faithful.

```
context pseudofunctor
begin

  definition locally-faithful
  where locally-faithful ≡
        ∀f g μ μ'. «μ : f ⇒_C g» ∧ «μ' : f ⇒_C g» ∧ F μ = F μ' ⟶ μ = μ'

  lemma locally-faithful-iff-faithful:
  shows locally-faithful ⟷ faithful-functor V_C V_D F
  proof
    show faithful-functor V_C V_D F ⟹ locally-faithful
      by (metis category.in-homE faithful-functor.is-faithful functor-axioms
          functor-def locally-faithful-def)
    show locally-faithful ⟹ faithful-functor V_C V_D F
    proof −
      assume 1: locally-faithful
      show faithful-functor V_C V_D F
      proof
        fix μ μ'
        assume par: C.par μ μ' and eq: F μ = F μ'
        obtain f g where fg: «μ : f ⇒_C g» ∧ «μ' : f ⇒_C g»
          using par by auto
        show μ = μ'
          using 1 fg locally-faithful-def eq by simp
      qed
    qed
  qed
```

end

In contrast, it is not true that a pseudofunctor that is locally full is also globally full, because we can have «$\nu : F\ h \Rightarrow_D F\ k$» even if h and k are not in the same hom-category. So it would be a mistake to require that an equivalence functor be globally full.

locale *equivalence-pseudofunctor* =
  pseudofunctor +
  faithful-functor V_C V_D F +
assumes *biessentially-surjective-on-objects*:
    $D.obj\ a' \Longrightarrow \exists\, a.\ C.obj\ a \wedge D.equivalent\text{-}objects\ (map_0\ a)\ a'$
and *locally-essentially-surjective*:
    $[\![\ C.obj\ a;\ C.obj\ b;\ «g : map_0\ a \rightarrow_D map_0\ b»;\ D.ide\ g\ ]\!] \Longrightarrow$
      $\exists f.\ «f : a \rightarrow_C b» \wedge C.ide\ f \wedge D.isomorphic\ (F\ f)\ g$
and *locally-full*:
    $[\![\ C.ide\ f;\ C.ide\ f';\ src_C\ f = src_C\ f';\ trg_C\ f = trg_C\ f';\ «\nu : F\ f \Rightarrow_D F\ f'»\ ]\!] \Longrightarrow$
      $\exists \mu.\ «\mu : f \Rightarrow_C f'» \wedge F\ \mu = \nu$
begin

  lemma *reflects-ide*:
  assumes $C.endo\ \mu$ **and** $D.ide\ (F\ \mu)$
  shows $C.ide\ \mu$
    using *assms is-faithful* [*of* $C.dom\ \mu\ \mu$] *C.ide-char′*
    by (*metis C.arr-dom C.cod-dom C.dom-dom C.seqE D.ide-char preserves-dom*)

  lemma *reflects-iso*:
  assumes $C.arr\ \mu$ **and** $D.iso\ (F\ \mu)$
  shows $C.iso\ \mu$
  proof −
    obtain μ' **where** μ': «$\mu' : C.cod\ \mu \Rightarrow_C C.dom\ \mu$» $\wedge\ F\ \mu' = D.inv\ (F\ \mu)$
      using *assms locally-full* [*of* $C.cod\ \mu\ C.dom\ \mu\ D.inv\ (F\ \mu)$]
        D.inv-in-hom C.in-homE preserves-hom C.in-homI
      by *auto*
    have *C.inverse-arrows* $\mu\ \mu'$
      using *assms* μ' *reflects-ide*
      apply (*intro C.inverse-arrowsI*)
       apply (*metis C.cod-comp C.dom-comp C.ide-dom C.in-homE C.seqI D.comp-inv-arr′*
          faithful-functor-axioms faithful-functor-def functor.preserves-ide
          as-nat-trans.preserves-comp-2 preserves-dom)
      by (*metis C.cod-comp C.dom-comp C.ide-cod C.in-homE C.seqI D.comp-arr-inv′*
        faithful-functor-axioms faithful-functor-def functor.preserves-ide
        preserves-cod as-nat-trans.preserves-comp-2)
    thus *?thesis* **by** *auto*
  qed

  lemma *reflects-isomorphic*:
  assumes $C.ide\ f$ **and** $C.ide\ f'$ **and** $src_C\ f = src_C\ f'$ **and** $trg_C\ f = trg_C\ f'$
  and $D.isomorphic\ (F\ f)\ (F\ f')$
  shows $C.isomorphic\ f\ f'$

using *assms C.isomorphic-def D.isomorphic-def locally-full reflects-iso C.arrI*
by *metis*

lemma *reflects-equivalence*:
assumes $C.ide\ f$ **and** $C.ide\ g$
and «$\eta : src_C\ f \Rightarrow_C g \star_C f$» **and** «$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»
and *equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ f)\ (F\ g)$
$(D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g))$
shows *equivalence-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
proof –
interpret E': *equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$F\ f$› ‹$F\ g$›
‹$D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)$›
‹$D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)$›
using *assms* **by** *auto*
show *?thesis*
proof
show $C.ide\ f$
using *assms*(*1*) **by** *simp*
show $C.ide\ g$
using *assms*(*2*) **by** *simp*
show «$\eta : src_C\ f \Rightarrow_C g \star_C f$»
using *assms*(*3*) **by** *simp*
show «$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»
using *assms*(*4*) **by** *simp*
have *0*: $src_C\ f = trg_C\ g \wedge src_C\ g = trg_C\ f$
using ‹«$\eta : src_C\ f \Rightarrow_C g \star_C f$»›
by (*metis C.hseqE C.ideD*(*1*) *C.ide-cod C.ide-dom C.in-homE assms*(*4*))
show $C.iso\ \eta$
proof –
have $D.iso\ (F\ \eta)$
proof –
have *1*: «$D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f) : map_0\ (src_C\ f) \Rightarrow_D F\ g \star_D F\ f$»
using ‹$C.ide\ f$› ‹$C.ide\ g$› ‹«$\eta : src_C\ f \Rightarrow_C g \star_C f$»›
unit-char cmp-in-hom cmp-components-are-iso
by (*metis* (*mono-tags*, *lifting*) *C.ideD*(*1*) E'.*unit-in-vhom preserves-src*)
have *2*: $D.iso\ (\Phi\ (g, f)) \wedge$ «$\Phi\ (g, f) : F\ g \star_D F\ f \Rightarrow_D F\ (g \star_C f)$»
using *0* ‹$C.ide\ f$› ‹$C.ide\ g$› *cmp-in-hom* **by** *simp*
have *3*: $D.iso\ (D.inv\ (unit\ (src_C\ f))) \wedge$
«$D.inv\ (unit\ (src_C\ f)) : F\ (src_C\ f) \Rightarrow_D map_0\ (src_C\ f)$»
using ‹$C.ide\ f$› *unit-char* **by** *simp*
have $D.iso\ (\Phi\ (g, f) \cdot_D (D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)) \cdot_D$
$D.inv\ (unit\ (src_C\ f)))$
using *1 2 3* E'.*unit-is-iso D.isos-compose* **by** *blast*
moreover have $\Phi\ (g, f) \cdot_D (D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)) \cdot_D$
$D.inv\ (unit\ (src_C\ f)) =$
$F\ \eta$
proof –
have $\Phi\ (g, f) \cdot_D (D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)) \cdot_D$

$D.inv\ (unit\ (src_C\ f))$
$= (\Phi\ (g, f) \cdot_D (D.inv\ (\Phi\ (g, f))) \cdot_D F\ \eta \cdot_D (unit\ (src_C\ f)) \cdot_D$
$D.inv\ (unit\ (src_C\ f)))$
using *D.comp-assoc* **by** *simp*
also have ... $= F\ (g \star_C f) \cdot_D F\ \eta \cdot_D F\ (src_C\ f)$
using *2 3 D.comp-arr-inv D.comp-inv-arr D.inv-is-inverse*
by (*metis C.ideD(1) C.obj-src D.comp-assoc D.dom-inv D.in-homE unit-char(2) assms(1)*)
also have ... $= F\ \eta$
using ‹«$\eta : src_C\ f \Rightarrow_C g \star_C f$»› *D.comp-arr-dom D.comp-cod-arr* **by** *auto*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *assms reflects-iso* **by** *auto*
qed
show $C.iso\ \varepsilon$
proof −
have $D.iso\ (F\ \varepsilon)$
proof −
have *1*: «$D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g) : F\ f \star_D F\ g \Rightarrow_D map_0\ (src_C\ g)$»
using ‹$C.ide\ f$› ‹$C.ide\ g$› ‹«$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»›
unit-char cmp-in-hom cmp-components-are-iso
by (*metis* (*mono-tags*, *lifting*) *C.ideD(1) E′.counit-in-vhom preserves-src*)
have *2*: $D.iso\ (D.inv\ (\Phi\ (f, g))) \wedge$
«$D.inv\ (\Phi\ (f, g)) : F\ (f \star_C g) \Rightarrow_D F\ f \star_D F\ g$»
using *0* ‹$C.ide\ f$› ‹$C.ide\ g$› ‹«$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»› *cmp-in-hom(2) D.inv-in-hom*
by *simp*
have *3*: $D.iso\ (unit\ (trg_C\ f)) \wedge$ «$unit\ (trg_C\ f) : map_0\ (trg_C\ f) \Rightarrow_D F\ (trg_C\ f)$»
using ‹$C.ide\ f$› *unit-char* **by** *simp*
have $D.iso\ (unit\ (trg_C\ f) \cdot_D (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)) \cdot_D$
$D.inv\ (\Phi\ (f, g)))$
using *0 1 2 3 E′.counit-is-iso D.isos-compose*
by (*metis D.arrI D.cod-comp D.cod-inv D.seqI D.seqI′*)
moreover have $unit\ (trg_C\ f) \cdot_D (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)) \cdot_D$
$D.inv\ (\Phi\ (f, g)) =$
$F\ \varepsilon$
proof −
have $unit\ (trg_C\ f) \cdot_D (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)) \cdot_D$
$D.inv\ (\Phi\ (f, g)) =$
$(unit\ (trg_C\ f) \cdot_D D.inv\ (unit\ (trg_C\ f))) \cdot_D F\ \varepsilon \cdot_D (\Phi\ (f, g) \cdot_D D.inv\ (\Phi\ (f, g)))$
using *D.comp-assoc* **by** *simp*
also have ... $= F\ (trg_C\ f) \cdot_D F\ \varepsilon \cdot_D F\ (f \star_C g)$
using *0 3 D.comp-arr-inv′ D.comp-inv-arr′*
by (*simp add*: $C.VV.arr\text{-}char_{SbC}$ $C.VV.ide\text{-}char_{SbC}$ *assms(1−2)*)
also have ... $= F\ \varepsilon$
using *0* ‹«$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»› *D.comp-arr-dom D.comp-cod-arr* **by** *auto*
finally show *?thesis* **by** *simp*

```
      qed
      ultimately show ?thesis by simp
    qed
    thus ?thesis
      using assms reflects-iso by auto
  qed
 qed
qed

lemma reflects-equivalence-map:
assumes C.ide f and D.equivalence-map (F f)
shows C.equivalence-map f
proof −
  obtain g′ φ ψ where E′: equivalence-in-bicategory V_D H_D a_D i_D src_D trg_D (F f) g′ φ ψ
    using assms D.equivalence-map-def by auto
  interpret E′: equivalence-in-bicategory V_D H_D a_D i_D src_D trg_D ‹F f› g′ φ ψ
    using E′ by auto
  obtain g where g: «g : trg_C f →_C src_C f» ∧ C.ide g ∧ D.isomorphic (F g) g′
    using assms E′.antipar locally-essentially-surjective [of trg_C f src_C f g′]
    by auto
  obtain μ where μ: «μ : g′ ⇒_D F g» ∧ D.iso μ
    using g D.isomorphic-def D.isomorphic-symmetric by blast
  interpret E″: equivalence-in-bicategory V_D H_D a_D i_D src_D trg_D ‹F f› ‹F g›
                ‹(F g ⋆_D F f) ·_D (μ ⋆_D F f) ·_D φ›
                ‹ψ ·_D (D.inv (F f) ⋆_D g′) ·_D (F f ⋆_D D.inv μ)›
    using assms μ E′.equivalence-in-bicategory-axioms D.ide-is-iso
          D.equivalence-respects-iso [of F f g′ φ ψ F f F f μ F g]
    by auto
  let ?η′ = Φ (g, f) ·_D (F g ⋆_D F f) ·_D (μ ⋆_D F f) ·_D φ ·_D D.inv (unit (src_C f))
  have η′: «?η′ : F (src_C f) ⇒_D F (g ⋆_C f)»
    using assms μ g unit-char  E′.unit-in-hom(2) E′.antipar E″.antipar cmp-in-hom
    apply (intro D.comp-in-homI)
        apply auto
    using E′.antipar(2) by blast
  have iso-η′: D.iso ?η′
    using assms g μ η′ E′.antipar unit-char
    by (metis C.in-hhomE D.arrI D.inv-comp-left(2) D.inv-comp-right(2) D.iso-hcomp
              D.iso-inv-iso D.isos-compose D.seqE E″.antipar(2) E″.unit-is-iso
              E′.unit-is-iso as-nat-iso.components-are-iso cmp-components-are-iso)
  let ?ε′ = unit (src_C g) ·_D ψ ·_D (D.inv (F f) ⋆_D g′) ·_D (F f ⋆_D D.inv μ) ·_D
            D.inv (Φ (f, g))
  have ε′: «?ε′ : F (f ⋆_C g) ⇒_D F (trg_C f)»
  proof (intro D.comp-in-homI)
    show «D.inv (Φ (f, g)) : F (f ⋆_C g) ⇒_D F f ⋆_D F g»
      using assms g cmp-in-hom C.VV.ide-char_SbC C.VV.arr-char_SbC by auto
    show «F f ⋆_D D.inv μ : F f ⋆_D F g ⇒_D F f ⋆_D g′»
      using assms g μ E″.antipar D.ide-in-hom(2) by auto
    show «D.inv (F f) ⋆_D g′ : F f ⋆_D g′ ⇒_D F f ⋆_D g′»
      using assms E′.antipar D.ide-is-iso by auto
```

show «ψ : $F\ f \star_D g' \Rightarrow_D trg_D\ (F\ f)$»
using E'.*counit-in-hom* **by** *simp*
show «*unit* (src_C g) : trg_D ($F\ f$) $\Rightarrow_D$ F (trg_C f)»
using *assms g unit-char* **by** *auto*
qed
have *iso-*ε': *D.iso* ?ε'
proof −
have *D.iso* (Φ (f, g))
using *assms g* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ **by** *auto*
thus *?thesis*
by (*metis C.in-hhomE D.arrI D.hseq-char′ D.ide-is-iso D.inv-comp-left*(*2*)
D.inv-comp-right(*2*) *D.iso-hcomp D.iso-inv-iso D.isos-compose D.seqE*
D.seq-if-composable E″.counit-is-iso E′.counit-is-iso E′.ide-left
ε' μ g *unit-char*(*2*))
qed
obtain η **where** η: «η : src_C $f \Rightarrow_C g \star_C f$» $\wedge$ $F\ \eta$ = ?η'
using *assms g E′.antipar* η' *locally-full* [*of* src_C f g $\star_C$ f ?η']
by (*metis C.ide-hcomp C.ideD*(*1*) *C.in-hhomE C.src.preserves-ide C.hcomp-simps*(*1*−*2*)
C.src-trg C.trg-trg)
have *iso-*η: *C.iso* η
using η η' *iso-*η' *reflects-iso* **by** *auto*
have *1*: $\exists\varepsilon$. «ε : $f \star_C g \Rightarrow_C src_C\ g$» $\wedge$ $F\ \varepsilon$ = ?ε'
using *assms g* ε' *locally-full* [*of* f $\star_C$ g src_C g ?ε'] **by** *force*
obtain ε **where** ε: «ε : $f \star_C g \Rightarrow_C src_C\ g$» $\wedge$ $F\ \varepsilon$ = ?ε'
using *1* **by** *blast*
have *iso-*ε: *C.iso* ε
using ε ε' *iso-*ε' *reflects-iso* **by** *auto*
have *equivalence-in-bicategory* $(\cdot_C)$ $(\star_C)$ a_C i_C src_C trg_C f g η ε
using *assms g* η ε *iso-*η *iso-*ε **by** (*unfold-locales*, *auto*)
thus *?thesis*
using *C.equivalence-map-def* **by** *auto*
qed

lemma *reflects-equivalent-objects*:
assumes *C.obj a* **and** *C.obj b* **and** *D.equivalent-objects* (map_0 a) (map_0 b)
shows *C.equivalent-objects a b*
proof −
obtain f' **where** f': «f' : $map_0\ a \rightarrow_D map_0\ b$» $\wedge$ *D.equivalence-map* f'
using *assms D.equivalent-objects-def D.equivalence-map-def* **by** *auto*
obtain f **where** f: «f : $a \rightarrow_C b$» $\wedge$ *C.ide* f $\wedge$ *D.isomorphic* ($F\ f$) f'
using *assms* f' *locally-essentially-surjective* [*of a b* f'] *D.equivalence-map-is-ide*
by *auto*
have *D.equivalence-map* ($F\ f$)
using f f' *D.equivalence-map-preserved-by-iso* [*of* f' $F\ f$] *D.isomorphic-symmetric*
by *simp*
hence *C.equivalence-map f*
using f f' *reflects-equivalence-map* [*of f*] **by** *simp*
thus *?thesis*
using f *C.equivalent-objects-def* **by** *auto*

qed

end

For each pair of objects a, b of C, an equivalence pseudofunctor restricts to an equivalence of categories between $C.hhom\ a\ b$ and $D.hhom\ (map_0\ a)\ (map_0\ b)$.

locale *equivalence-pseudofunctor-at-hom* =
  equivalence-pseudofunctor +
fixes a :: $'a$ **and** a' :: $'a$
assumes *obj-a*: $C.obj\ a$
and *obj-a'*: $C.obj\ a'$
begin

  sublocale $hhom_C$: *subcategory* V_C ‹$\lambda\mu$. «$\mu : a \to_C a'$»›
    using *C.hhom-is-subcategory* **by** *simp*
  sublocale $hhom_D$: *subcategory* V_D ‹$\lambda\mu$. «$\mu : map_0\ a \to_D map_0\ a'$»›
    using *D.hhom-is-subcategory* **by** *simp*

  definition F_1
  where $F_1 = (\lambda\mu.\ if\ hhom_C.arr\ \mu\ then\ F\ \mu\ else\ D.null)$

  interpretation F_1: *functor* $hhom_C.comp\ hhom_D.comp\ F_1$
    unfolding F_1*-def*
    using $hhom_C.arr\text{-}char_{SbC}$ $hhom_D.arr\text{-}char_{SbC}$ $hhom_C.dom\text{-}char_{SbC}$ $hhom_D.dom\text{-}char_{SbC}$
      $hhom_C.cod\text{-}char_{SbC}$ $hhom_D.cod\text{-}char_{SbC}$ $hhom_C.seq\text{-}char_{SbC}$ $hhom_C.comp\text{-}char$
$hhom_D.comp\text{-}char$
    by *unfold-locales auto*

  interpretation F_1: *fully-faithful-and-essentially-surjective-functor*
      $hhom_C.comp\ hhom_D.comp\ F_1$
  proof
    show $\bigwedge\mu\ \mu'.\ [\![hhom_C.par\ \mu\ \mu';\ F_1\ \mu = F_1\ \mu']\!] \Longrightarrow \mu = \mu'$
      unfolding F_1*-def*
    using *is-faithful* $hhom_C.dom\text{-}char_{SbC}$ $hhom_D.dom\text{-}char_{SbC}$ $hhom_C.cod\text{-}char_{SbC}$ $hhom_D.cod\text{-}char_{SbC}$
      by (*metis C.in-hhom-def* $hhom_C.arrE$)
    show $\bigwedge f\ f'\ \nu.\ [\![hhom_C.ide\ f;\ hhom_C.ide\ f';\ hhom_D.in\text{-}hom\ \nu\ (F_1\ f')\ (F_1\ f)]\!]$
      $\Longrightarrow \exists\mu.\ hhom_C.in\text{-}hom\ \mu\ f'\ f \wedge F_1\ \mu = \nu$
    proof (*unfold* F_1*-def*)
      fix $f\ f'\ \nu$
      assume f: $hhom_C.ide\ f$ **and** f': $hhom_C.ide\ f'$
      assume $hhom_D.in\text{-}hom\ \nu$ (*if* $hhom_C.arr\ f'$ *then* $F\ f'$ *else* $D.null$)
        (*if* $hhom_C.arr\ f$ *then* $F\ f$ *else* $D.null$)
      hence ν: $hhom_D.in\text{-}hom\ \nu\ (F\ f')\ (F\ f)$
        using $f\ f'$ **by** *simp*
      have $\exists\mu.\ hhom_C.in\text{-}hom\ \mu\ f'\ f \wedge F\ \mu = \nu$
      proof −
        have *1*: $src_C\ f' = src_C\ f \wedge trg_C\ f' = trg_C\ f$
          using $f\ f'$ $hhom_C.ide\text{-}char$ **by** (*metis C.in-hhomE* $hhom_C.arrE$)
        hence *ex*: $\exists\mu.\ C.in\text{-}hom\ \mu\ f'\ f \wedge F\ \mu = \nu$

by (*meson* ν f f' $hhom_D$.*in-hom-char*$_{SbC}$ *horizontal-homs.hhom-is-subcategory*
locally-full subcategory.ide-char$_{SbC}$ *weak-arrow-of-homs-axioms*
weak-arrow-of-homs-def)
obtain μ **where** μ: C.*in-hom* μ f' f $\wedge$ F $\mu = \nu$
using *ex* **by** *blast*
have $hhom_C$.*in-hom* μ f' f
by (*metis C.arrI C.in-hhom-def C.vconn-implies-hpar*(*1*−*2*) μ f f'
$hhom_C$.*arr-char*$_{SbC}$ $hhom_C$.*ide-char* $hhom_C$.*in-hom-char*$_{SbC}$)
thus *?thesis*
using μ **by** *auto*
qed
thus $\exists\mu$. $hhom_C$.*in-hom* μ f' f $\wedge$ (*if* $hhom_C$.*arr* μ *then* F μ *else D.null*) $= \nu$
by *auto*
qed
show $\bigwedge g$. $hhom_D$.*ide* $g \Longrightarrow \exists f$. $hhom_C$.*ide* f $\wedge$ $hhom_D$.*isomorphic* (F_1 f) g
proof (*unfold* F_1-*def*)
fix g
assume g: $hhom_D$.*ide* g
show $\exists f$. $hhom_C$.*ide* f $\wedge$ $hhom_D$.*isomorphic* (*if* $hhom_C$.*arr* f *then* F f *else D.null*) g
proof −
have C.*obj* a $\wedge$ C.*obj* a'
using *obj-a obj-a'* **by** *simp*
moreover have *1*: D.*ide* g $\wedge$ «g : map_0 $a \rightarrow_D map_0$ a'»
using g *obj-a obj-a'* $hhom_D$.*ide-char*$_{SbC}$ **by** *auto*
ultimately have *2*: $\exists f$. C.*in-hhom* f a a' $\wedge$ C.*ide* f $\wedge$ D.*isomorphic* (F f) g
using *locally-essentially-surjective* [*of* a a' g] **by** *simp*
obtain f φ **where** f: C.*in-hhom* f a a' $\wedge$ C.*ide* f $\wedge$ D.*in-hom* φ (F f) g $\wedge$ D.*iso* φ
using *2* **by** *auto*
have $hhom_C$.*ide* f
using f $hhom_C$.*ide-char*$_{SbC}$ $hhom_C$.*arr-char*$_{SbC}$ **by** *simp*
moreover have $hhom_D$.*isomorphic* (F f) g
proof −
have $hhom_D$.*arr* φ $\wedge$ $hhom_D$.*arr* (D.*inv* φ)
by (*metis 1 D.arrI D.in-hhom-def D.vconn-implies-hpar*(*1*−*4*) *D.inv-in-homI*
f $hhom_D$.*arrI*$_{SbC}$)
hence $hhom_D$.*in-hom* φ (F f) g $\wedge$ $hhom_D$.*iso* φ
by (*metis D.in-homE* f $hhom_D$.*cod-simp* $hhom_D$.*dom-simp* $hhom_D$.*in-homI* $hhom_D$.*iso-char*$_{SbC}$)
thus *?thesis*
unfolding $hhom_D$.*isomorphic-def* **by** *blast*
qed
ultimately show $\exists f$. $hhom_C$.*ide* f $\wedge$
$hhom_D$.*isomorphic* (*if* $hhom_C$.*arr* f *then* F f *else D.null*) g
by *force*
qed
qed
qed

lemma *equivalence-functor-*F_1:
shows *fully-faithful-and-essentially-surjective-functor* $hhom_C$.*comp* $hhom_D$.*comp* F_1

and *equivalence-functor* $hhom_C.comp\ hhom_D.comp\ F_1$
..

definition G_1
where $G_1 = (SOME\ G.\ \exists \eta\varepsilon.$
adjoint-equivalence $hhom_C.comp\ hhom_D.comp\ G\ F_1\ (fst\ \eta\varepsilon)\ (snd\ \eta\varepsilon))$

lemma G_1*-props*:
assumes $C.obj\ a$ **and** $C.obj\ a'$
shows $\exists \eta\ \varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon$
proof −
have $\exists G.\ \exists \eta\varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G\ F_1\ (fst\ \eta\varepsilon)\ (snd\ \eta\varepsilon)$
using F_1*.extends-to-adjoint-equivalence* **by** *simp*
hence $\exists \eta\varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ (fst\ \eta\varepsilon)\ (snd\ \eta\varepsilon)$
unfolding G_1*-def*
using *someI-ex*
[*of* $\lambda G.\ \exists \eta\varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G\ F_1\ (fst\ \eta\varepsilon)\ (snd\ \eta\varepsilon)$]
by *blast*
thus *?thesis* **by** *simp*
qed

definition η
where $\eta = (SOME\ \eta.\ \exists \varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon)$

definition ε
where $\varepsilon = (SOME\ \varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon)$

lemma $\eta\varepsilon$*-props*:
shows *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon$
using *obj-a obj-a'* η*-def* ε*-def* G_1*-props*
someI-ex [*of* $\lambda\eta.\ \exists \varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon$]
someI-ex [*of* $\lambda\varepsilon.$ *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon$]
by *simp*

sublocale $\eta\varepsilon$: *adjoint-equivalence* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\ \varepsilon$
using $\eta\varepsilon$*-props* **by** *simp*

sublocale $\eta\varepsilon$: *meta-adjunction* $hhom_C.comp\ hhom_D.comp\ G_1\ F_1\ \eta\varepsilon.\varphi\ \eta\varepsilon.\psi$
using $\eta\varepsilon$*.induces-meta-adjunction* **by** *simp*

end

context *identity-pseudofunctor*
begin

sublocale *equivalence-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
map cmp
using *B.isomorphic-reflexive B.arrI*
apply *unfold-locales*

by (*auto simp add*: *B.equivalent-objects-reflexive* map_0*-def B.obj-simps*)

lemma *is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* V_B H_B a_B i_B src_B trg_B V_B H_B a_B i_B src_B trg_B
map cmp
..

end

locale *composite-equivalence-pseudofunctor* =
composite-pseudofunctor +
F: *equivalence-pseudofunctor* V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F +
G: *equivalence-pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G
begin

interpretation *faithful-functor* V_B V_D ‹*G o F*›
using *F.faithful-functor-axioms G.faithful-functor-axioms faithful-functors-compose*
by *blast*

interpretation *equivalence-pseudofunctor* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
‹*G o F*› *cmp*
proof
show $\bigwedge c.$ *D.obj c* $\Longrightarrow \exists\, a.$ *B.obj a* $\wedge$ *D.equivalent-objects* (map_0 *a*) *c*
proof −
fix *c*
assume *c*: *D.obj c*
obtain *b* **where** *b*: *C.obj b* $\wedge$ *D.equivalent-objects* (*G.*map_0 *b*) *c*
using *c G.biessentially-surjective-on-objects* **by** *auto*
obtain *a* **where** *a*: *B.obj a* $\wedge$ *C.equivalent-objects* (*F.*map_0 *a*) *b*
using *b F.biessentially-surjective-on-objects* **by** *auto*
have *D.equivalent-objects* (map_0 *a*) *c*
using *a b* map_0*-def G.preserves-equivalent-objects D.equivalent-objects-transitive*
by *fastforce*
thus $\exists\, a.$ *B.obj a* $\wedge$ *D.equivalent-objects* (map_0 *a*) *c*
using *a* **by** *auto*
qed
show $\bigwedge a\ a'\ h.$ $[\![$*B.obj a*; *B.obj a'*; «*h* : map_0 *a* $\rightarrow_D$ map_0 *a'*»; *D.ide h*$]\!]$
$\Longrightarrow \exists f.$ *B.in-hhom f a a'* $\wedge$ *B.ide f* $\wedge$ *D.isomorphic* ((*G* $\circ$ *F*) *f*) *h*
proof −
fix *a a' h*
assume *a*: *B.obj a* **and** *a'*: *B.obj a'*
and *h-in-hom*: «*h* : map_0 *a* $\rightarrow_D$ map_0 *a'*» **and** *ide-h*: *D.ide h*
obtain *g*
where *g*: *C.in-hhom g* (*F.*map_0 *a*) (*F.*map_0 *a'*) $\wedge$ *C.ide g* $\wedge$ *D.isomorphic* (*G g*) *h*
using *a a' h-in-hom ide-h* map_0*-def B.obj-simps*
G.locally-essentially-surjective [*of F.*map_0 *a F.*map_0 *a' h*]
by *auto*
obtain *f* **where** *f*: *B.in-hhom f a a'* $\wedge$ *B.ide f* $\wedge$ *C.isomorphic* (*F f*) *g*
using *a a' g F.locally-essentially-surjective* **by** *blast*

```
      have D.isomorphic ((G o F) f) h
        by (metis D.isomorphic-transitive G.preserves-isomorphic comp-apply f g)
      thus ∃f. B.in-hhom f a a' ∧ B.ide f ∧ D.isomorphic ((G ∘ F) f) h
        using f by auto
    qed
    show ⋀f f' ν. ⟦B.ide f; B.ide f'; src_B f = src_B f'; trg_B f = trg_B f';
                  «ν : (G ∘ F) f ⇒_D (G ∘ F) f'»⟧
              ⟹ ∃τ. «τ : f →_B f'» ∧ (G ∘ F) τ = ν
    proof −
      fix f f' ν
      assume f: B.ide f and f': B.ide f'
      and src: src_B f = src_B f' and trg: trg_B f = trg_B f'
      and ν: «ν : (G ∘ F) f ⇒_D (G ∘ F) f'»
      have ν: «ν : G (F f) ⇒_D G (F f')»
        using ν by simp
      have 1: src_C (F f) = src_C (F f') ∧ trg_C (F f) = trg_C (F f')
        using f f' src trg by simp
      have 2: ∃μ. «μ : F f ⇒_C F f'» ∧ G μ = ν
        using f f' 1 ν G.locally-full F.preserves-ide by simp
      obtain μ where μ: «μ : F f ⇒_C F f'» ∧ G μ = ν
        using 2 by auto
      obtain τ where τ: «τ : f →_B f'» ∧ F τ = μ
        using f f' src trg 2 μ F.locally-full by blast
      show ∃τ. «τ : f →_B f'» ∧ (G ∘ F) τ = ν
        using μ τ by auto
    qed
  qed

  sublocale equivalence-pseudofunctor V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
            ‹G o F› cmp ..

  lemma is-equivalence-pseudofunctor:
  shows equivalence-pseudofunctor V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
            (G o F) cmp
    ..

 end

end
```

1.11 Strictness

theory *Strictness*
imports *Category3.ConcreteCategory Pseudofunctor CanonicalIsos*
begin

In this section we consider bicategories in which some or all of the canonical isomorphisms are assumed to be identities. A *normal* bicategory is one in which the unit isomorphisms are identities, so that unit laws for horizontal composition are satisfied "on

the nose". A *strict* bicategory (also known as a *2-category*) is a bicategory in which both the unit and associativity isomoprhisms are identities, so that horizontal composition is strictly associative as well as strictly unital.

From any given bicategory B we may construct a related strict bicategory S, its *strictification*, together with a pseudofunctor that embeds B in S. The Strictness Theorem states that this pseudofunctor is an equivalence pseudofunctor, so that bicategory B is biequivalent to its strictification. The Strictness Theorem is often used informally to justify suppressing canonical isomorphisms; which amounts to proving a theorem about 2-categories and asserting that it holds for all bicategories. Here we are working formally, so we can't just wave our hands and mutter something about the Strictness Theorem when we want to avoid dealing with units and associativities. However, in cases where we can establish that the property we would like to prove is reflected by the embedding of a bicategory in its strictification, then we can formally apply the Strictness Theorem to generalize to all bicategories a result proved for 2-categories. We will apply this approach here to simplify the proof of some facts about internal equivalences in a bicategory.

1.11.1 Normal and Strict Bicategories

A *normal* bicategory is one in which the unit isomorphisms are identities, so that unit laws for horizontal composition are satisfied "on the nose".

locale *normal-bicategory* =
  bicategory +
assumes *strict-lunit*: $\bigwedge f.\ ide\ f \Longrightarrow \mathrm{l}[f] = f$
and *strict-runit*: $\bigwedge f.\ ide\ f \Longrightarrow \mathrm{r}[f] = f$
begin

  lemma *strict-unit*:
  assumes *obj a*
  shows *ide* $\mathrm{i}[a]$
    using *assms strict-runit unitor-coincidence*(*2*) [*of a*] **by** *auto*

  lemma *strict-lunit'*:
  assumes *ide f*
  shows $\mathrm{l}^{-1}[f] = f$
    using *assms strict-lunit* **by** *simp*

  lemma *strict-runit'*:
  assumes *ide f*
  shows $\mathrm{r}^{-1}[f] = f$
    using *assms strict-runit* **by** *simp*

  lemma *hcomp-obj-arr*:
  assumes *obj b* **and** *arr f* **and** $b = trg\ f$
  shows $b \star f = f$
    using *assms strict-lunit*
    by (*metis comp-arr-dom comp-ide-arr ide-cod ide-dom lunit-naturality*)

lemma *hcomp-arr-obj*:
assumes *arr f* **and** *obj a* **and** *src f* = *a*
shows $f \star a = f$
using *assms strict-runit*
by (*metis comp-arr-dom comp-ide-arr ide-cod ide-dom runit-naturality*)

end

A *strict* bicategory is a normal bicategory in which the associativities are also identities, so that associativity of horizontal composition holds "on the nose".

locale *strict-bicategory* =
normal-bicategory +
assumes *strict-assoc*: $\bigwedge f\ g\ h.\ [\![ide\ f;\ ide\ g;\ ide\ h;\ src\ f = trg\ g;\ src\ g = trg\ h]\!] \Longrightarrow$
$ide\ \mathrm{a}[f,\ g,\ h]$
begin

lemma *strict-assoc'*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows $ide\ \mathrm{a}^{-1}[f,\ g,\ h]$
using *assms strict-assoc* **by** *simp*

lemma *hcomp-assoc*:
shows $(\mu \star \nu) \star \tau = \mu \star \nu \star \tau$
proof (*cases hseq* $\mu\ \nu \wedge$ *hseq* $\nu\ \tau$)
show $\neg$ (*hseq* $\mu\ \nu \wedge$ *hseq* $\nu\ \tau$) $\Longrightarrow$ *?thesis*
by (*metis hseqE hseq-char' match-1 match-2*)
show *hseq* $\mu\ \nu \wedge$ *hseq* $\nu\ \tau \Longrightarrow$ *?thesis*
proof −
assume *1*: *hseq* $\mu\ \nu \wedge$ *hseq* $\nu\ \tau$
have *2*: *arr* $\mu \wedge$ *arr* $\nu \wedge$ *arr* $\tau \wedge$ *src* μ = *trg* $\nu \wedge$ *src* ν = *trg* τ
using *1* **by** *blast*
have $(\mu \star \nu) \star \tau = \mathrm{a}[cod\ \mu,\ cod\ \nu,\ cod\ \tau] \cdot ((\mu \star \nu) \star \tau)$
using *1 assoc-in-hom strict-assoc comp-cod-arr assoc-simps*(*4*) *hseq-char*
by *simp*
also have ... $= (\mu \star \nu \star \tau) \cdot \mathrm{a}[dom\ \mu,\ dom\ \nu,\ dom\ \tau]$
using *1 assoc-naturality* **by** *auto*
also have ... $= \mu \star \nu \star \tau$
using *2 assoc-in-hom* [*of dom* μ *dom* ν *dom* τ] *strict-assoc comp-arr-dom*
by *auto*
finally show *?thesis* **by** *simp*
qed
qed

In a strict bicategory, every canonical isomorphism is an identity.

interpretation *bicategorical-language* ..
interpretation *E*: *self-evaluation-map V H* a i *src trg* ..
notation *E.eval* (‹⦃-⦄›)

lemma *ide-eval-Can*:

```
  assumes Can t
  shows ide ⦃t⦄
  proof −
    have 1: ⋀u1 u2. ⟦ ide ⦃u1⦄; ide ⦃u2⦄; Arr u1; Arr u2; Dom u1 = Cod u2 ⟧
                    ⟹ ide (⦃u1⦄ · ⦃u2⦄)
      by (metis (no-types, lifting) E.eval-simps'(4−5) comp-ide-self ide-char)
    have ⋀u. Can u ⟹ ide ⦃u⦄
    proof −
      fix u
      show Can u ⟹ ide ⦃u⦄

        using 1 α-def a'-def strict-lunit strict-runit strict-assoc strict-assoc'
              ℓ-ide-simp r-ide-simp Can-implies-Arr comp-ide-arr E.eval-simps'(2−3)
        by (induct u) auto
    qed
    thus ?thesis
      using assms by simp
  qed

  lemma ide-can:
  assumes Ide f and Ide g and ⌊f⌋ = ⌊g⌋
  shows ide (can g f)
    using assms Can-red Can-Inv red-in-Hom Inv-in-Hom can-def ide-eval-Can
          Can.simps(4) Dom-Inv red-simps(4)
    by presburger

end

context bicategory
begin
```

The following result gives conditions for strictness of a bicategory that are typically somewhat easier to verify than those used for the definition.

```
  lemma is-strict-if:
  assumes ⋀f. ide f ⟹ f ⋆ src f = f
  and ⋀f. ide f ⟹ trg f ⋆ f = f
  and ⋀a. obj a ⟹ ide i[a]
  and ⋀f g h. ⟦ide f; ide g; ide h; src f = trg g; src g = trg h⟧ ⟹ ide a[f, g, h]
  shows strict-bicategory V H a i src trg
  proof
    show ⋀f g h. ⟦ide f; ide g; ide h; src f = trg g; src g = trg h⟧ ⟹ ide a[f, g, h]
      by fact
    fix f
    assume f: ide f
    show l[f] = f
    proof −
      have f = l[f]
        using assms f unit-simps(5)
        by (intro lunit-eqI) (auto simp add: comp-arr-ide)
```

```
      thus ?thesis by simp
    qed
    show r[f] = f
    proof −
      have f = r[f]
      proof (intro runit-eqI)
        show ide f by fact
        show «f : f ⋆ src f ⇒ f»
          using f assms(1) by auto
        show f ⋆ src f = (f ⋆ i[src f]) · a[f, src f, src f]
        proof −
          have (f ⋆ i[src f]) · a[f, src f, src f] = (f ⋆ src f) · a[f, src f, src f]
            using f assms(2−3) unit-simps(5) by simp
          also have ... = (f ⋆ src f ⋆ src f) · a[f, src f, src f]
            using f assms(1−2) ideD(1) trg-src src.preserves-ide by metis
          also have ... = f ⋆ src f
            using f comp-arr-ide assms(1,4) assoc-in-hom [of f src f src f] by auto
          finally show ?thesis by simp
        qed
      qed
      thus ?thesis by simp
    qed
  qed

end
```

1.11.2 Strictification

The Strictness Theorem asserts that every bicategory is biequivalent to a strict bicategory. More specifically, it shows how to construct, given an arbitrary bicategory, a strict bicategory (its *strictification*) that is biequivalent to it. Consequently, given a property P of bicategories that is "bicategorical" (*i.e.* respects biequivalence), if we want to show that P holds for a bicategory B then it suffices to show that P holds for the strictification of B, and if we want to show that P holds for all bicategories, it is sufficient to show that it holds for all strict bicategories. This is very useful, because it becomes quite tedious, even with the aid of a proof assistant, to do "diagram chases" with all the units and associativities fully spelled out.

Given a bicategory B, the strictification S of B may be constructed as the bicategory whose arrows are triples (A, B, μ), where X and Y are "normal identity terms" (essentially, nonempty horizontally composable lists of 1-cells of B) having the same syntactic source and target, and «$\mu : \{\!|X|\!\} \Rightarrow \{\!|Y|\!\}$» in B. Vertical composition in S is given by composition of the underlying arrows in B. Horizontal composition in S is given by $(A, B, \mu) \star (A', B', \mu') = (AA', BB', \nu)$, where AA' and BB' denote concatenations of lists and where ν is defined as the composition $can\ BB'\ (B \star B') \cdot (\mu \star \mu') \cdot can\ (A \star A')\ AA'$, where $can\ (A \star A')\ AA'$ and $can\ BB'\ (B \star B')$ are canonical isomorphisms in B. The canonical isomorphism $can\ (A \star A')\ AA'$ corresponds to taking a pair of lists $A \star A'$ and "shifting the parentheses to the right" to obtain a single list AA'. The canonical

isomorphism can BB' ($B \star B'$) corresponds to the inverse rearrangement.

The bicategory B embeds into its strictification S via the functor UP that takes each arrow μ of B to ($\langle dom\ \mu\rangle$, $\langle cod\ \mu\rangle$, μ), where $\langle dom\ \mu\rangle$ and $\langle cod\ \mu\rangle$ denote one-element lists. This mapping extends to a pseudofunctor. There is also a pseudofunctor DN, which maps (A, B, μ) in S to μ in B; this is such that $DN\ o\ UP$ is the identity on B and $UP\ o\ DN$ is equivalent to the identity on S, so we obtain a biequivalence between B and S.

It seems difficult to find references that explicitly describe a strictification construction in elementary terms like this (in retrospect, it ought to have been relatively easy to rediscover such a construction, but my thinking got off on the wrong track). One reference that I did find useful was [1], which discusses strictification for monoidal categories.

locale *strictified-bicategory* =
  B: *bicategory* V_B H_B a_B i_B src_B trg_B
for V_B :: $'a$ *comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹$\star_B$› *53*)
and a_B :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a_B[-, -, -]›)
and i_B :: $'a \Rightarrow 'a$ (‹i_B[-]›)
and src_B :: $'a \Rightarrow 'a$
and trg_B :: $'a \Rightarrow 'a$
begin

  sublocale E: *self-evaluation-map* V_B H_B a_B i_B src_B trg_B **..**

  notation *B.in-hhom* (‹«- : - $\rightarrow_B$ -»›)
  notation *B.in-hom* (‹«- : - $\Rightarrow_B$ -»›)

  notation *E.eval* (‹⦃-⦄›)
  notation *E.Nmlize* (‹⌊-⌋›)

The following gives the construction of a bicategory whose arrows are triples (A, B, μ), where $Nml\ A \wedge Ide\ A$, $Nml\ B \wedge Ide\ B$, $Src\ A = Src\ B$, $Trg\ A = Trg\ B$, and $\mu : ⦃A⦄ \Rightarrow ⦃B⦄$. We use *concrete-category* to construct the vertical composition, so formally the arrows of the bicategory will be of the form $MkArr\ A\ B\ \mu$.

The 1-cells of the bicategory correspond to normal, identity terms A in the bicategorical language associated with B.

abbreviation *IDE*
where $IDE \equiv \{A.\ E.Nml\ A \wedge E.Ide\ A\}$

If terms A and B determine 1-cells of the strictification and have a common source and target, then the 2-cells between these 1-cells correspond to arrows μ of the underlying bicategory such that «$\mu : ⦃A⦄ \Rightarrow_B ⦃B⦄$».

abbreviation *HOM*
where $HOM\ A\ B \equiv \{\mu.\ E.Src\ A = E.Src\ B \wedge E.Trg\ A = E.Trg\ B \wedge «\mu : ⦃A⦄ \Rightarrow_B ⦃B⦄»\}$

The map taking term $A \in OBJ$ to its evaluation $⦃A⦄ \in HOM\ A\ A$ defines the embedding of 1-cells as identity 2-cells.

abbreviation *EVAL*

where $EVAL \equiv E.eval$

sublocale *concrete-category IDE HOM EVAL* ‹λ- - - μ ν. $\mu \cdot_B \nu$›
  using *E.ide-eval-Ide B.comp-arr-dom B.comp-cod-arr B.comp-assoc*
  by (*unfold-locales*, *auto*)

lemma *is-concrete-category*:
shows *concrete-category IDE HOM EVAL* (λ- - - μ ν. $\mu \cdot_B \nu$)
  ..

abbreviation *vcomp* (**infixr** ‹·› *55*)
where $vcomp \equiv COMP$

lemma *arr-char*:
shows $arr\ F \longleftrightarrow$
  $E.Nml\ (Dom\ F) \wedge E.Ide\ (Dom\ F) \wedge E.Nml\ (Cod\ F) \wedge E.Ide\ (Cod\ F) \wedge$
  $E.Src\ (Dom\ F) = E.Src\ (Cod\ F) \wedge E.Trg\ (Dom\ F) = E.Trg\ (Cod\ F) \wedge$
  $«Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\}» \wedge F \neq Null$
  using *arr-char* **by** *auto*

lemma *arrI* :
assumes $E.Nml\ (Dom\ F)$ **and** $E.Ide\ (Dom\ F)$ **and** $E.Nml\ (Cod\ F)$ **and** $E.Ide\ (Cod\ F)$
and $E.Src\ (Dom\ F) = E.Src\ (Cod\ F)$ **and** $E.Trg\ (Dom\ F) = E.Trg\ (Cod\ F)$
and $«Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\}»$ **and** $F \neq Null$
shows $arr\ F$
  using *assms arr-char* **by** *blast*

lemma *arrE* [*elim*]:
assumes $arr\ F$
shows $(\![\ E.Nml\ (Dom\ F);\ E.Ide\ (Dom\ F);\ E.Nml\ (Cod\ F);\ E.Ide\ (Cod\ F);$
  $E.Src\ (Dom\ F) = E.Src\ (Cod\ F);\ E.Trg\ (Dom\ F) = E.Trg\ (Cod\ F);$
  $«Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\}»;\ F \neq Null\ ]\!] \Longrightarrow T) \Longrightarrow T$
  using *assms arr-char* **by** *simp*

lemma *ide-char*:
shows $ide\ F \longleftrightarrow endo\ F \wedge B.ide\ (Map\ F)$
proof
  show $ide\ F \Longrightarrow endo\ F \wedge B.ide\ (Map\ F)$
    using $ide\text{-}char_{CC}$ **by** (*simp add*: *E.ide-eval-Ide*)
  show $endo\ F \wedge B.ide\ (Map\ F) \Longrightarrow ide\ F$
    by (*metis* (*no-types*, *lifting*) *B.ide-char B.in-homE arr-char* $ide\text{-}char_{CC}$
      mem-Collect-eq seq-char)
qed

lemma *ideI* [*intro*]:
assumes $arr\ F$ **and** $Dom\ F = Cod\ F$ **and** $B.ide\ (Map\ F)$
shows $ide\ F$
  using *assms ide-char dom-char cod-char seq-char* **by** *presburger*

lemma *ideE* [*elim*]:
assumes *ide F*
shows (⟦ *arr F*; *Dom F* = *Cod F*; *B.ide* (*Map F*); *Map F* = ⦃*Dom F*⦄;
Map F = ⦃*Cod F*⦄ ⟧ ⟹ *T*) ⟹ *T*
using *assms*
by (*metis* (*no-types*, *lifting*) *Map-ide*(*2*) *ide-char seq-char*)

Source and target are defined by the corresponding syntactic operations on terms.

definition *src*
where *src F* ≡ *if arr F then MkIde* (*E.Src* (*Dom F*)) *else null*

definition *trg*
where *trg F* ≡ *if arr F then MkIde* (*E.Trg* (*Dom F*)) *else null*

lemma *src-simps* [*simp*]:
assumes *arr F*
shows *Dom* (*src F*) = *E.Src* (*Dom F*) **and** *Cod* (*src F*) = *E.Src* (*Dom F*)
and *Map* (*src F*) = ⦃*E.Src* (*Dom F*)⦄
using *assms src-def arr-char* **by** *auto*

lemma *trg-simps* [*simp*]:
assumes *arr F*
shows *Dom* (*trg F*) = *E.Trg* (*Dom F*) **and** *Cod* (*trg F*) = *E.Trg* (*Dom F*)
and *Map* (*trg F*) = ⦃*E.Trg* (*Dom F*)⦄
using *assms trg-def arr-char* **by** *auto*

interpretation *src*: *endofunctor vcomp src*
using *src-def comp-char E.Obj-implies-Ide*
apply (*unfold-locales*)
apply *auto*[*4*]
proof −
show ⋀*g f*. *seq g f* ⟹ *src* (*g* · *f*) = *src g* · *src f*
proof −
fix *g f*
assume *gf*: *seq g f*
have *src* (*g* · *f*) = *MkIde* (*E.Src* (*Dom* (*g* · *f*)))
using *gf src-def comp-char* **by** *simp*
also have ... = *MkIde* (*E.Src* (*Dom f*))
using *gf* **by** (*simp add*: *seq-char*)
also have ... = *MkIde* (*E.Src* (*Dom g*)) · *MkIde* (*E.Src* (*Dom f*))
using *gf seq-char E.Obj-implies-Ide* **by** *auto*
also have ... = *src g* · *src f*
using *gf src-def comp-char* **by** *auto*
finally show *src* (*g* · *f*) = *src g* · *src f* **by** *blast*
qed
qed

interpretation *trg*: *endofunctor vcomp trg*
using *trg-def comp-char E.Obj-implies-Ide*

```
  apply (unfold-locales)
      apply auto[4]
proof −
  show ⋀g f. seq g f ⟹ trg (g · f) = trg g · trg f
  proof −
    fix g f
    assume gf: seq g f
    have trg (g · f) = MkIde (E.Trg (Dom (g · f)))
      using gf trg-def comp-char by simp
    also have ... = MkIde (E.Trg (Dom f))
      using gf by (simp add: seq-char)
    also have ... = MkIde (E.Trg (Dom g)) · MkIde (E.Trg (Dom f))
      using gf seq-char E.Obj-implies-Ide by auto
    also have ... = trg g · trg f
      using gf trg-def comp-char by auto
    finally show trg (g · f) = trg g · trg f by blast
  qed
qed

interpretation horizontal-homs vcomp src trg
  using src-def trg-def Cod-in-Obj Map-in-Hom E.Obj-implies-Ide
  by unfold-locales auto

notation in-hhom (‹«- : - → -»›)

definition hcomp    (infixr ‹⋆› 53)
where μ ⋆ ν ≡ if arr μ ∧ arr ν ∧ src μ = trg ν
            then MkArr (Dom μ ⌊⋆⌋ Dom ν) (Cod μ ⌊⋆⌋ Cod ν)
                    (B.can (Cod μ ⌊⋆⌋ Cod ν) (Cod μ ⋆ Cod ν) ·B
                      (Map μ ⋆B Map ν) ·B
                      B.can (Dom μ ⋆ Dom ν) (Dom μ ⌊⋆⌋ Dom ν))
            else null

lemma arr-hcomp:
assumes arr μ and arr ν and src μ = trg ν
shows arr (μ ⋆ ν)
proof −
  have 1: E.Ide (Dom μ ⌊⋆⌋ Dom ν) ∧ E.Nml (Dom μ ⌊⋆⌋ Dom ν) ∧
          E.Ide (Cod μ ⌊⋆⌋ Cod ν) ∧ E.Nml (Cod μ ⌊⋆⌋ Cod ν)
    using assms arr-char src-def trg-def E.Ide-HcompNml E.Nml-HcompNml(1) by auto
  moreover
  have «B.can (Cod μ ⌊⋆⌋ Cod ν) (Cod μ ⋆ Cod ν) ·B (Map μ ⋆B Map ν) ·B
          B.can (Dom μ ⋆ Dom ν) (Dom μ ⌊⋆⌋ Dom ν) :
            ⦃Dom μ ⌊⋆⌋ Dom ν⦄ ⇒B ⦃Cod μ ⌊⋆⌋ Cod ν⦄»
  proof −
    have «Map μ ⋆B Map ν : ⦃Dom μ ⋆ Dom ν⦄ ⇒B ⦃Cod μ ⋆ Cod ν⦄»
      using assms arr-char dom-char cod-char src-def trg-def E.eval-simps'(2−3)
      by auto
    moreover
```

have «$B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu)$:
$\{\!|Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu|\!\} \Rightarrow_B \{\!|Dom\ \mu \star Dom\ \nu|\!\}$» $\wedge$
«$B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu)$:
$\{\!|Cod\ \mu \star Cod\ \nu|\!\} \Rightarrow_B \{\!|Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu|\!\}$»
using *assms 1 arr-char src-def trg-def*
apply (*intro conjI B.in-homI*) **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
moreover have $E.Src\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) = E.Src\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ \wedge$
$E.Trg\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) = E.Trg\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)$
using *assms arr-char src-def trg-def*
by (*simp add*: *E.Src-HcompNml E.Trg-HcompNml*)
ultimately show *?thesis*
unfolding *hcomp-def*
using *assms* **by** (*intro arrI*, *auto*)
qed

lemma *src-hcomp* [*simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \mu = trg\ \nu$
shows $src\ (\mu \star \nu) = src\ \nu$
using *assms arr-char hcomp-def src-def trg-def arr-hcomp E.Src-HcompNml* **by** *simp*

lemma *trg-hcomp* [*simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \mu = trg\ \nu$
shows $trg\ (hcomp\ \mu\ \nu) = trg\ \mu$
using *assms arr-char hcomp-def src-def trg-def arr-hcomp E.Trg-HcompNml* **by** *simp*

lemma *hseq-char*:
shows $arr\ (\mu \star \nu) \longleftrightarrow arr\ \mu \wedge arr\ \nu \wedge src\ \mu = trg\ \nu$
using *arr-hcomp hcomp-def* **by** *simp*

lemma *Dom-hcomp* [*simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \mu = trg\ \nu$
shows $Dom\ (\mu \star \nu) = Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu$
using *assms hcomp-def* [*of* $\mu\ \nu$] **by** *simp*

lemma *Cod-hcomp* [*simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \mu = trg\ \nu$
shows $Cod\ (\mu \star \nu) = Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu$
using *assms hcomp-def* [*of* $\mu\ \nu$] **by** *simp*

lemma *Map-hcomp* [*simp*]:
assumes $arr\ \mu$ **and** $arr\ \nu$ **and** $src\ \mu = trg\ \nu$
shows $Map\ (\mu \star \nu) = B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \cdot_B$
$(Map\ \mu \star_B Map\ \nu) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu)$
using *assms hcomp-def* [*of* $\mu\ \nu$] **by** *simp*

interpretation *functor VV.comp vcomp* ‹$\lambda\mu\nu.\ hcomp\ (fst\ \mu\nu)\ (snd\ \mu\nu)$›

```
proof
  show ⋀f. ¬ VV.arr f ⟹ fst f ⋆ snd f = null
    using hcomp-def by auto
  show A: ⋀f. VV.arr f ⟹ arr (fst f ⋆ snd f)
    using VV.arrE hseq-char by blast
  show ⋀f. VV.arr f ⟹ dom (fst f ⋆ snd f) = fst (VV.dom f) ⋆ snd (VV.dom f)
  proof −
    fix f
    assume f: VV.arr f
    have dom (fst f ⋆ snd f) = MkIde (Dom (fst f) ⌊⋆⌋ Dom (snd f))
      using f VV.arrE [of f] dom-char arr-hcomp hcomp-def by simp
    also have ... = fst (VV.dom f) ⋆ snd (VV.dom f)
    proof −
      have hcomp (fst (VV.dom f)) (snd (VV.dom f)) =
            MkArr (Dom (fst f) ⌊⋆⌋ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
                (B.can (Dom (fst f) ⌊⋆⌋ Dom (snd f)) (Dom (fst f) ⋆ Dom (snd f)) ·B
                  (⦃Dom (fst f)⦄ ⋆B ⦃Dom (snd f)⦄) ·B
                  B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f)))
        using f VV.arrE [of f] arr-hcomp hcomp-def VV.dom-simp by simp
      moreover have B.can (Dom (fst f) ⌊⋆⌋ Dom (snd f)) (Dom (fst f) ⋆ Dom (snd f)) ·B
                      (⦃Dom (fst f)⦄ ⋆B ⦃Dom (snd f)⦄) ·B
                        B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f)) =
                    ⦃Dom (fst f) ⌊⋆⌋ Dom (snd f)⦄
      proof −
        have 1: E.Ide (Dom (fst f) ⋆ Dom (snd f))
          using f VV.arr-char_SbC arr-char dom-char
          apply simp
          by (metis (no-types, lifting) src-simps(1) trg-simps(1))
        have 2: E.Ide (Dom (fst f) ⌊⋆⌋ Dom (snd f))
          using f VV.arr-char_SbC arr-char dom-char
          apply simp
          by (metis (no-types, lifting) E.Ide-HcompNml src-simps(1) trg-simps(1))
        have 3: ⌊Dom (fst f) ⋆ Dom (snd f)⌋ = ⌊Dom (fst f) ⌊⋆⌋ Dom (snd f)⌋
          using f VV.arr-char_SbC arr-char dom-char
          apply simp
          by (metis (no-types, lifting) E.Nml-HcompNml(1) E.Nmlize-Nml
              src-simps(1) trg-simps(1))
        have (⦃Dom (fst f)⦄ ⋆B ⦃Dom (snd f)⦄) ·B
                B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f)) =
              B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
          using 1 2 3 B.comp-cod-arr by force
        thus ?thesis
          using 1 2 3 f VV.arr-char_SbC B.can-Ide-self B.vcomp-can by simp
      qed
      ultimately show ?thesis by simp
    qed
    finally show dom (fst f ⋆ snd f) = fst (VV.dom f) ⋆ snd (VV.dom f)
      by simp
  qed
```

```
show ⋀f. VV.arr f ⟹ cod (fst f ⋆ snd f) = fst (VV.cod f) ⋆ snd (VV.cod f)
proof −
  fix f
  assume f: VV.arr f
  have cod (fst f ⋆ snd f) = MkIde (Cod (fst f) ⌊⋆⌋ Cod (snd f))
    using f VV.arrE [of f] cod-char arr-hcomp hcomp-def by simp
  also have ... = fst (VV.cod f) ⋆ snd (VV.cod f)
  proof −
    have hcomp (fst (VV.cod f)) (snd (VV.cod f)) =
          MkArr (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f))
                (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f)) ·B
                  (⦃Cod (fst f)⦄ ⋆B ⦃Cod (snd f)⦄) ·B
                  B.can (Cod (fst f) ⋆ Cod (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f)))
      using f VV.arrE [of f] arr-hcomp hcomp-def VV.cod-simp by simp
    moreover have B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f)) ·B
                    (⦃Cod (fst f)⦄ ⋆B ⦃Cod (snd f)⦄) ·B
                      B.can (Cod (fst f) ⋆ Cod (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f)) =
                  ⦃Cod (fst f) ⌊⋆⌋ Cod (snd f)⦄
    proof −
      have 1: E.Ide (Cod (fst f) ⋆ Cod (snd f))
        using f VV.arr-char_SbC arr-char dom-char
        apply simp
        by (metis (no-types, lifting) src-simps(1) trg-simps(1))
      have 2: E.Ide (Cod (fst f) ⌊⋆⌋ Cod (snd f))
        using f VV.arr-char_SbC arr-char dom-char
        apply simp
        by (metis (no-types, lifting) E.Ide-HcompNml src-simps(1) trg-simps(1))
      have 3: ⌊Cod (fst f) ⋆ Cod (snd f)⌋ = ⌊Cod (fst f) ⌊⋆⌋ Cod (snd f)⌋
        using f VV.arr-char_SbC arr-char dom-char
        apply simp
        by (metis (no-types, lifting) E.Nml-HcompNml(1) E.Nmlize-Nml
            src-simps(1) trg-simps(1))
      have (⦃Cod (fst f)⦄ ⋆B ⦃Cod (snd f)⦄) ·B
              B.can (Cod (fst f) ⋆ Cod (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f)) =
            B.can (Cod (fst f) ⋆ Cod (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f))
        using 1 2 3 B.comp-cod-arr by force
      thus ?thesis
        using 1 2 3 f VV.arr-char_SbC B.can-Ide-self B.vcomp-can by simp
    qed
    ultimately show ?thesis by simp
  qed
  finally show cod (fst f ⋆ snd f) = fst (VV.cod f) ⋆ snd (VV.cod f)
    by simp
qed
show ⋀g f. VV.seq g f ⟹
          fst (VV.comp g f) ⋆ snd (VV.comp g f) = (fst g ⋆ snd g) · (fst f ⋆ snd f)
proof −
  fix f g
  assume fg: VV.seq g f
```

```
have f: arr (fst f) ∧ arr (snd f) ∧ src (fst f) = trg (snd f)
  using fg VV.seq-char_SbC VV.arr-char_SbC by simp
have g: arr (fst g) ∧ arr (snd g) ∧ src (fst g) = trg (snd g)
  using fg VV.seq-char_SbC VV.arr-char_SbC by simp
have 1: arr (fst (VV.comp g f)) ∧ arr (snd (VV.comp g f)) ∧
        src (fst (VV.comp g f)) = trg (snd (VV.comp g f))
  using fg VV.arrE by blast
have 0: VV.comp g f = (fst g · fst f, snd g · snd f)
  using fg 1 VV.comp-char VxV.comp-char
  by (metis (no-types, lifting) VV.seq-char_SbC VxV.seqE_PC)
let ?X = MkArr (Dom (fst (VV.comp g f)) ⌊⋆⌋ Dom (snd (VV.comp g f)))
               (Cod (fst (VV.comp g f)) ⌊⋆⌋ Cod (snd (VV.comp g f)))
               (B.can (Cod (fst (VV.comp g f)) ⌊⋆⌋ Cod (snd (VV.comp g f)))
                      (Cod (fst (VV.comp g f)) ⋆ Cod (snd (VV.comp g f))) ·_B
                 (Map (fst (VV.comp g f)) ⋆_B Map (snd (VV.comp g f))) ·_B
                 B.can (Dom (fst (VV.comp g f)) ⋆ Dom (snd (VV.comp g f)))
                       (Dom (fst (VV.comp g f)) ⌊⋆⌋ Dom (snd (VV.comp g f))))
have 2: fst (VV.comp g f) ⋆ snd (VV.comp g f) = ?X
  unfolding hcomp-def using 1 by simp
also have ... = (fst g ⋆ snd g) · (fst f ⋆ snd f)
proof −
  let ?GG = MkArr (Dom (fst g) ⌊⋆⌋ Dom (snd g)) (Cod (fst g) ⌊⋆⌋ Cod (snd g))
                  (B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·_B
                    (Map (fst g) ⋆_B Map (snd g)) ·_B
                    B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g)))
  let ?FF = MkArr (Dom (fst f) ⌊⋆⌋ Dom (snd f)) (Cod (fst f) ⌊⋆⌋ Cod (snd f))
                  (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f)) ·_B
                    (Map (fst f) ⋆_B Map (snd f)) ·_B
                    B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f)))
  have 4: arr ?FF ∧ arr ?GG ∧ Dom ?GG = Cod ?FF
  proof −
    have arr ?FF ∧ arr ?GG
      using f g fg VV.arr-char_SbC VV.seqE hcomp-def A by presburger
    thus ?thesis
      using 0 1 by (simp add: fg seq-char)
  qed
  have (fst g ⋆ snd g) · (fst f ⋆ snd f) = ?GG · ?FF
    unfolding hcomp-def
    using 1 f g fg VV.arr-char_SbC VV.seqE by simp
  also have ... = ?X
  proof (intro arr-eqI)
    show seq ?GG ?FF
      using fg 4 seq-char by blast
    show arr ?X
      using fg 1 arr-hcomp hcomp-def by simp
    show Dom (?GG · ?FF) = Dom ?X
      using fg 0 1 4 seq-char by simp
    show Cod (?GG · ?FF) = Cod ?X
      using fg 0 1 4 seq-char by simp
```

```
show Map (?GG · ?FF) = Map ?X
proof −
  have Map (?GG · ?FF) = Map ?GG ·B Map ?FF
    using 4 by auto
  also have
    ... = (B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
             (Map (fst g) ⋆B Map (snd g)) ·B
             B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g))) ·B
           (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f)) ·B
             (Map (fst f) ⋆B Map (snd f)) ·B
             B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f)))
    using fg by simp
  also have
    ... = B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
             ((Map (fst g) ⋆B Map (snd g)) ·B (Map (fst f) ⋆B Map (snd f))) ·B
             B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
  proof −
    have (B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
            (Map (fst g) ⋆B Map (snd g)) ·B
            B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g))) ·B
            (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f)) ·B
            (Map (fst f) ⋆B Map (snd f)) ·B
            B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))) =
          B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
            ((Map (fst g) ⋆B Map (snd g)) ·B
            (B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g)) ·B
            B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f))) ·B
            (Map (fst f) ⋆B Map (snd f))) ·B
            B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
      using B.comp-assoc by simp
   also have ... = B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
                   ((Map (fst g) ⋆B Map (snd g)) ·B (Map (fst f) ⋆B Map (snd f))) ·B
                   B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
    proof −
      have (B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g))) ·B
              (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f))) =
            ⦃Cod (fst f) ⋆ Cod (snd f)⦄
      proof −
        have (B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g))) ·B
                (B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f))) =
                B.can (Dom (fst g) ⋆ Dom (snd g)) (Cod (fst f) ⋆ Cod (snd f))
        proof −
          have E.Ide (Dom (fst g) ⋆ Dom (snd g))
            using g arr-char
            by (metis (no-types, lifting) E.Ide.simps(3) src-simps(2) trg-simps(2))
          moreover have E.Ide (Dom (fst g) ⌊⋆⌋ Dom (snd g))
            using 4 by auto
          moreover have E.Ide (Cod (fst f) ⋆ Cod (snd f))
            using f arr-char
```

by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *src-simps*(*2*) *trg-simps*(*2*))
moreover have
$\lfloor Dom\ (fst\ g) \star Dom\ (snd\ g)\rfloor = \lfloor Dom\ (fst\ g)\ \lfloor\star\rfloor\ Dom\ (snd\ g)\rfloor$
using *g E.Nml-HcompNml*(*1*) *calculation*(*1*) **by** *fastforce*
moreover have
$\lfloor Dom\ (fst\ g)\ \lfloor\star\rfloor\ Dom\ (snd\ g)\rfloor = \lfloor Cod\ (fst\ f) \star Cod\ (snd\ f)\rfloor$
using *g fg seq-char*
by (*metis* (*no-types*, *lifting*) $VV.seq\text{-}char_{SbC}$ $VxV.seqE_{PC}$ *calculation*(*4*))
moreover have
$Dom\ (fst\ g)\ \lfloor\star\rfloor\ Dom\ (snd\ g) = Cod\ (fst\ f)\ \lfloor\star\rfloor\ Cod\ (snd\ f)$
using *0 1* **by** (*simp add*: *seq-char*)
ultimately show *?thesis*
using *B.vcomp-can* **by** *simp*
qed
also have ... $= \{\!|Cod\ (fst\ f) \star Cod\ (snd\ f)|\!\}$
proof −
have $Dom\ (fst\ g) \star Dom\ (snd\ g) = Cod\ (fst\ f) \star Cod\ (snd\ f)$
using *0 f g fg seq-char* $VV.seq\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
by *simp*
thus *?thesis*
using *f B.can-Ide-self* [*of* $Dom\ (fst\ f) \star Dom\ (snd\ f)$]
by (*metis* (*no-types*, *lifting*) *B.can-Ide-self E.Ide.simps*(*3*)
arrE src-simps(*2*) *trg-simps*(*2*))
qed
finally show *?thesis* **by** *simp*
qed
hence $(B.can\ (Dom\ (fst\ g) \star Dom\ (snd\ g))\ (Dom\ (fst\ g)\ \lfloor\star\rfloor\ Dom\ (snd\ g)) \cdot_B$
$B.can\ (Cod\ (fst\ f)\ \lfloor\star\rfloor\ Cod\ (snd\ f))\ (Cod\ (fst\ f) \star Cod\ (snd\ f))) \cdot_B$
$(Map\ (fst\ f) \star_B Map\ (snd\ f)) =$
$\{\!|Cod\ (fst\ f) \star Cod\ (snd\ f)|\!\} \cdot_B (Map\ (fst\ f) \star_B Map\ (snd\ f))$
by *simp*
also have ... $= Map\ (fst\ f) \star_B Map\ (snd\ f)$
proof −
have *1*: $\forall p.\ arr\ p \longrightarrow map\ (cod\ p) \cdot map\ p = map\ p$
by *blast*
have *3*: $\{\!|Cod\ (fst\ f)|\!\} \cdot_B Map\ (fst\ f) = Map\ (map\ (cod\ (fst\ f)) \cdot map\ (fst\ f))$
by (*simp add*: *f*)
have *4*: $map\ (cod\ (fst\ f)) \cdot map\ (fst\ f) = fst\ f$
using *1 f map-simp* **by** *simp*
show *?thesis*
proof −
have *2*: $\{\!|Cod\ (snd\ f)|\!\} \cdot_B Map\ (snd\ f) = Map\ (snd\ f)$
using *1 f map-simp*
by (*metis* (*no-types*, *lifting*) *Dom-cod Map-cod Map-comp arr-cod*)
have $B.seq\ \{\!|Cod\ (snd\ f)|\!\}\ (Map\ (snd\ f))$
using *f 2* **by** *auto*
moreover have $B.seq\ \{\!|Cod\ (fst\ f)|\!\}\ (Map\ (fst\ f))$
using *4 f 3* **by** *auto*
moreover have

```
                {|Cod (fst f)|} ·B Map (fst f) ⋆B {|Cod (snd f)|} ·B Map (snd f) =
                 Map (fst f) ⋆B Map (snd f)
                using 2 3 4 by presburger
              ultimately show ?thesis
                by (simp add: B.interchange)
            qed
          qed
          finally have
            (B.can (Dom (fst g) ⋆ Dom (snd g)) (Dom (fst g) ⌊⋆⌋ Dom (snd g)) ·B
               B.can (Cod (fst f) ⌊⋆⌋ Cod (snd f)) (Cod (fst f) ⋆ Cod (snd f))) ·B
               (Map (fst f) ⋆B Map (snd f)) =
             Map (fst f) ⋆B Map (snd f)
            by simp
          thus ?thesis
            using fg B.comp-cod-arr by simp
        qed
        finally show ?thesis by simp
      qed
      also have ... = B.can (Cod (fst g) ⌊⋆⌋ Cod (snd g)) (Cod (fst g) ⋆ Cod (snd g)) ·B
                      (Map (fst g · fst f) ⋆B Map (snd g · snd f)) ·B
                      B.can (Dom (fst f) ⋆ Dom (snd f)) (Dom (fst f) ⌊⋆⌋ Dom (snd f))
      proof −
        have 2: Dom (fst g) = Cod (fst f)
          using 0 f g fg VV.seq-char_SbC [of g f] VV.arr-char_SbC arr-char seq-char
          by (metis (no-types, lifting) fst-conv)
        hence Map (fst g · fst f) = Map (fst g) ·B Map (fst f)
          using f g Map-comp [of fst f fst g] by simp
        moreover have B.seq (Map (fst g)) (Map (fst f)) ∧
                      B.seq (Map (snd g)) (Map (snd f))
          using f g 0 1 2 arr-char
          by (metis (no-types, lifting) B.seqI' prod.sel(2) seq-char)
        ultimately show ?thesis
          using 0 1 seq-char Map-comp B.interchange by auto
      qed
      also have ... = Map ?X
        using fg 0 1 by (simp add: seq-char)
      finally show ?thesis by simp
    qed
  qed
  finally show ?thesis by simp
qed
finally show fst (VV.comp g f) ⋆ snd (VV.comp g f) = (fst g ⋆ snd g) · (fst f ⋆ snd f)
  by simp
qed
qed

interpretation horizontal-composition vcomp hcomp src trg
  using hseq-char by (unfold-locales, auto)
```

lemma *hcomp-assoc*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ
and *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $(\mu \star \nu) \star \tau = \mu \star \nu \star \tau$
proof (*intro arr-eqI*)
  have $\mu\nu$: «*Map* $\mu \star_B$ *Map* ν : $\{\!|$*Dom* $\mu \star$ *Dom* $\nu|\!\}$ $\Rightarrow_B$ $\{\!|$*Cod* $\mu \star$ *Cod* $\nu|\!\}$»
    using *assms src-def trg-def arr-char*
    by (*auto simp add*: *E.eval-simps'*(*2*−*3*) *Pair-inject*)
  have $\nu\tau$: «*Map* $\nu \star_B$ *Map* τ : $\{\!|$*Dom* $\nu \star$ *Dom* $\tau|\!\}$ $\Rightarrow_B$ $\{\!|$*Cod* $\nu \star$ *Cod* $\tau|\!\}$»
    using *assms src-def trg-def arr-char*
    by (*auto simp add*: *E.eval-simps'*(*2*−*3*) *Pair-inject*)
  show *hseq* $(\mu \star \nu)$ τ
    using *assms* $\mu\nu$ $\nu\tau$ **by** *auto*
  show *hseq* μ $(\nu \star \tau)$
    using *assms* $\mu\nu$ $\nu\tau$ **by** *auto*
  show *Dom* $((\mu \star \nu) \star \tau)$ = *Dom* $(\mu \star \nu \star \tau)$
  proof −
    have *E.Nml* (*Dom* μ) $\wedge$ *E.Nml* (*Dom* ν) $\wedge$ *E.Nml* (*Dom* τ)
      using *assms* **by** *blast*
    moreover have *E.Src* (*Dom* μ) = *E.Trg* (*Dom* ν) $\wedge$ *E.Src* (*Dom* ν) = *E.Trg* (*Dom* τ)
      using *assms* $\mu\nu$ $\nu\tau$
      by (*metis* (*no-types*, *lifting*) *src-simps*(*2*) *trg-simps*(*2*))
    ultimately show *?thesis*
      using *assms* $\mu\nu$ $\nu\tau$ *E.HcompNml-assoc* **by** *simp*
  qed
  show *Cod* $((\mu \star \nu) \star \tau)$ = *Cod* $(\mu \star \nu \star \tau)$
  proof −
    have *E.Nml* (*Cod* μ) $\wedge$ *E.Nml* (*Cod* ν) $\wedge$ *E.Nml* (*Cod* τ)
      using *assms* **by** *blast*
    moreover have *E.Src* (*Cod* μ) = *E.Trg* (*Cod* ν) $\wedge$ *E.Src* (*Cod* ν) = *E.Trg* (*Cod* τ)
      using *assms* $\mu\nu$ $\nu\tau$
      by (*metis* (*no-types*, *lifting*) *arrE src-simps*(*2*) *trg-simps*(*2*))
    ultimately show *?thesis*
      using *assms* $\mu\nu$ $\nu\tau$ *E.HcompNml-assoc* **by** *simp*
  qed
 show *Map* $((\mu \star \nu) \star \tau)$ = *Map* $(\mu \star \nu \star \tau)$
  proof −
    have *Map* $((\mu \star \nu) \star \tau)$ =
        B.can (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν $\lfloor\star\rfloor$ *Cod* τ) (*Cod* $\mu \star$ *Cod* $\nu \star$ *Cod* τ) $\cdot_B$
          (*Map* $\mu \star_B$ *Map* $\nu \star_B$ *Map* τ) $\cdot_B$
            B.can (*Dom* $\mu \star$ *Dom* $\nu \star$ *Dom* τ) (*Dom* μ $\lfloor\star\rfloor$ *Dom* ν $\lfloor\star\rfloor$ *Dom* τ)
    proof −
      have *1*: *Map* $((\mu \star \nu) \star \tau)$ =
            B.can (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν $\lfloor\star\rfloor$ *Cod* τ) ((*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) $\star$ *Cod* τ) $\cdot_B$
              (*B.can* (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) (*Cod* $\mu \star$ *Cod* ν) $\cdot_B$
                (*Map* $\mu \star_B$ *Map* ν) $\cdot_B$
                  B.can (*Dom* $\mu \star$ *Dom* ν) (*Dom* μ $\lfloor\star\rfloor$ *Dom* ν) $\star_B$ *Map* τ) $\cdot_B$
                B.can ((*Dom* μ $\lfloor\star\rfloor$ *Dom* ν) $\star$ *Dom* τ) (*Dom* μ $\lfloor\star\rfloor$ *Dom* ν $\lfloor\star\rfloor$ *Dom* τ)
      using *assms* $\mu\nu$ $\nu\tau$ *hcomp-def E.HcompNml-assoc src-def trg-def arr-char*

E.Nml-HcompNml E.Ide-HcompNml
by *auto*
also have
... = $B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau) \cdot_B$
$(B.can\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau)) \cdot_B$
$B.can\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
proof −
have
$B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \cdot_B$
$(Map\ \mu \star_B Map\ \nu) \cdot_B B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star_B Map\ \tau =$
$B.can\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau)$
proof −
have $B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \cdot_B$
$(Map\ \mu \star_B Map\ \nu) \cdot_B B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu)$
$\star_B Map\ \tau =$
$(B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \star_B B.can\ (Cod\ \tau)\ (Cod\ \tau)) \cdot_B$
$((Map\ \mu \star_B Map\ \nu) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star_B Map\ \tau)$
proof −
have $B.seq\ (B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu))$
$((Map\ \mu \star_B Map\ \nu) \cdot_B B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu))$
by (*metis* (*no-types*, *lifting*) *B.arrI Map-hcomp arrE arr-hcomp assms*(*1*) *assms*(*2*) *assms*(*4*))
moreover have $B.seq\ (B.can\ (Cod\ \tau)\ (Cod\ \tau))\ (Map\ \tau)$
using *B.can-in-hom assms*(*3*) **by** *blast*
moreover have $B.ide\ (B.can\ (Cod\ \tau)\ (Cod\ \tau))$
using *B.can-Ide-self E.ide-eval-Ide arr-char assms*(*3*) **by** *presburger*
ultimately show *?thesis*
by (*metis* (*no-types*) *B.comp-ide-arr B.interchange*)
qed
also have
... = $(B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \star_B B.can\ (Cod\ \tau)\ (Cod\ \tau)) \cdot_B$
$((Map\ \mu \star_B Map\ \nu) \star_B Map\ \tau) \cdot_B$
$(B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star_B$
$B.can\ (Dom\ \tau)\ (Dom\ \tau))$
proof −
have $B.seq\ (Map\ \mu \star_B Map\ \nu)\ (B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu))$
by (*metis* (*no-types*, *lifting*) *B.arrI B.null-is-zero*(*2*) *B.ext Map-hcomp arrE arr-hcomp assms*(*1*) *assms*(*2*) *assms*(*4*))
moreover have $B.seq\ (Map\ \tau)\ (B.can\ (Dom\ \tau)\ (Dom\ \tau))$
using *assms*(*3*) **by** *fastforce*
ultimately show *?thesis*
using *B.interchange*
by (*metis* (*no-types*, *lifting*) *B.can-Ide-self B.comp-arr-ide E.ide-eval-Ide arrE assms*(*3*))

qed
also have
  ... = (*B.can* (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) (*Cod* μ $\star$ *Cod* ν) $\star_B$ *B.can* (*Cod* τ) (*Cod* τ)) $\cdot_B$
    (*B.can* ((*Cod* μ $\star$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ) $\cdot_B$
      (*Map* μ $\star_B$ *Map* ν $\star_B$ *Map* τ) $\cdot_B$
        B.can (*Dom* μ $\star$ *Dom* ν $\star$ *Dom* τ) ((*Dom* μ $\star$ *Dom* ν) $\star$ *Dom* τ)) $\cdot_B$
      (*B.can* (*Dom* μ $\star$ *Dom* ν) (*Dom* μ $\lfloor\star\rfloor$ *Dom* ν) $\star_B$
        B.can (*Dom* τ) (*Dom* τ))
proof −
  have (*Map* μ $\star_B$ *Map* ν) $\star_B$ *Map* τ =
      B.a′ $\{\!|$*Cod* $\mu|\!\}$ $\{\!|$*Cod* $\nu|\!\}$ $\{\!|$*Cod* $\tau|\!\}$ $\cdot_B$
        (*Map* μ $\star_B$ *Map* ν $\star_B$ *Map* τ) $\cdot_B$
          a_B $\{\!|$*Dom* $\mu|\!\}$ $\{\!|$*Dom* $\nu|\!\}$ $\{\!|$*Dom* $\tau|\!\}$
    using *B.hcomp-reassoc*(*1*)
    by (*metis* (*no-types*, *lifting*) *B.hcomp-in-vhomE B.in-homE* $\mu\nu$ $\nu\tau$ *arrE*
      assms(*1*) *assms*(*2*) *assms*(*3*))
  also have ... = *B.can* ((*Cod* μ $\star$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ) $\cdot_B$
      (*Map* μ $\star_B$ *Map* ν $\star_B$ *Map* τ) $\cdot_B$
        B.can (*Dom* μ $\star$ *Dom* ν $\star$ *Dom* τ) ((*Dom* μ $\star$ *Dom* ν) $\star$ *Dom* τ)
    using *assms arr-char src-def trg-def arr-char B.canE-associator* **by** *simp*
 finally show *?thesis* **by** *simp*
qed
also have
  ... = ((*B.can* (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) (*Cod* μ $\star$ *Cod* ν) $\star_B$ *B.can* (*Cod* τ) (*Cod* τ)) $\cdot_B$
      (*B.can* ((*Cod* μ $\star$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ))) $\cdot_B$
    (*Map* μ $\star_B$ *Map* ν $\star_B$ *Map* τ) $\cdot_B$
    (*B.can* (*Dom* μ $\star$ *Dom* ν $\star$ *Dom* τ) ((*Dom* μ $\star$ *Dom* ν) $\star$ *Dom* τ) $\cdot_B$
      (*B.can* (*Dom* μ $\star$ *Dom* ν) (*Dom* μ $\lfloor\star\rfloor$ *Dom* ν) $\star_B$
        B.can (*Dom* τ) (*Dom* τ)))
  using *B.comp-assoc* **by** *simp*
also have
  ... = *B.can* ((*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ) $\cdot_B$
      (*Map* μ $\star_B$ *Map* ν $\star_B$ *Map* τ) $\cdot_B$
      B.can (*Dom* μ $\star$ *Dom* ν $\star$ *Dom* τ) ((*Dom* μ $\lfloor\star\rfloor$ *Dom* ν) $\star$ *Dom* τ)
proof −
  have (*B.can* (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) (*Cod* μ $\star$ *Cod* ν) $\star_B$ *B.can* (*Cod* τ) (*Cod* τ)) $\cdot_B$
      (*B.can* ((*Cod* μ $\star$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ)) =
    B.can ((*Cod* μ $\lfloor\star\rfloor$ *Cod* ν) $\star$ *Cod* τ) (*Cod* μ $\star$ *Cod* ν $\star$ *Cod* τ)
  proof −
    have *E.Ide* (*Cod* μ $\star$ *Cod* ν)
      by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *arrE assms*(*1*−*2*,*4*)
        src-simps(*1*) *trg-simps*(*1*))
    moreover have *E.Ide* (*Cod* μ $\lfloor\star\rfloor$ *Cod* ν)
      using *E.Ide-HcompNml assms*(*1*) *assms*(*2*) *calculation* **by** *auto*
    moreover have $\lfloor$*Cod* μ $\star$ *Cod* $\nu\rfloor$ = $\lfloor$*Cod* μ $\lfloor\star\rfloor$ *Cod* $\nu\rfloor$
      using *E.Nml-HcompNml*(*1*) *assms*(*1*) *assms*(*2*) *calculation*(*1*) **by** *fastforce*
    moreover have *E.Src* (*Cod* μ $\star$ *Cod* ν) = *E.Trg* (*Cod* τ)
      by (*metis* (*no-types*, *lifting*) *E.Src.simps*(*3*) *arrE assms*(*2*−*3*,*5*)
        src-simps(*2*) *trg-simps*(*2*))

```
      moreover have E.Src (Cod μ ⌊⋆⌋ Cod ν) = E.Trg (Cod τ)
        using E.Src-HcompNml assms(1) assms(2) calculation(1) calculation(4)
        by fastforce
      moreover have ⌊Cod μ ⋆ Cod ν ⋆ Cod τ⌋ = ⌊(Cod μ ⋆ Cod ν) ⋆ Cod τ⌋
        by (metis (no-types, lifting) E.Arr.simps(3) E.Nmlize-Hcomp-Hcomp
            E.Nmlize-Hcomp-Hcomp' E.Ide-implies-Arr E.Src.simps(3) arrE assms(3)
            calculation(1) calculation(4))
      ultimately show ?thesis
        using assms(3) B.hcomp-can B.vcomp-can by auto
    qed
    moreover have
      B.can (Dom μ ⋆ Dom ν ⋆ Dom τ) ((Dom μ ⋆ Dom ν) ⋆ Dom τ) ·B
       (B.can (Dom μ ⋆ Dom ν) (Dom μ ⌊⋆⌋ Dom ν) ⋆B B.can (Dom τ) (Dom τ)) =
       B.can (Dom μ ⋆ Dom ν ⋆ Dom τ) ((Dom μ ⌊⋆⌋ Dom ν) ⋆ Dom τ)
    proof −
      have E.Ide (Dom μ ⌊⋆⌋ Dom ν)
        by (metis (no-types, lifting) E.Ide-HcompNml arrE assms(1−2,4)
            src-simps(2) trg-simps(2))
      moreover have E.Ide (Dom μ ⋆ Dom ν)
        by (metis (no-types, lifting) E.Ide.simps(3) arrE assms(1−2,4)
            src-simps(1) trg-simps(1))
      moreover have ⌊Dom μ ⌊⋆⌋ Dom ν⌋ = ⌊Dom μ ⋆ Dom ν⌋
        using E.Nml-HcompNml(1) assms(1−2) calculation(2) by fastforce
      moreover have E.Src (Dom μ ⌊⋆⌋ Dom ν) = E.Trg (Dom τ)
        by (metis (no-types, lifting) E.Ide.simps(3) E.Src-HcompNml arrE
            assms(1−3,5) calculation(2) src-simps(2) trg-simps(2))
      moreover have E.Src (Dom μ ⋆ Dom ν) = E.Trg (Dom τ)
        using E.Src-HcompNml assms(1−2) calculation(2) calculation(4)
        by fastforce
      moreover have E.Ide ((Dom μ ⋆ Dom ν) ⋆ Dom τ)
        using E.Ide.simps(3) assms(3) calculation(2) calculation(5) by blast
      moreover have ⌊(Dom μ ⋆ Dom ν) ⋆ Dom τ⌋ = ⌊Dom μ ⋆ Dom ν ⋆ Dom τ⌋
        using E.Nmlize-Hcomp-Hcomp calculation(6) by auto
      ultimately show ?thesis
        using assms(3) B.hcomp-can B.vcomp-can by auto
    qed
    ultimately show ?thesis by simp
  qed
  finally show ?thesis by simp
 qed
 thus ?thesis by simp
qed
also have
 ... = (B.can (Cod μ ⌊⋆⌋ Cod ν ⌊⋆⌋ Cod τ) ((Cod μ ⌊⋆⌋ Cod ν) ⋆ Cod τ) ·B
        B.can ((Cod μ ⌊⋆⌋ Cod ν) ⋆ Cod τ) (Cod μ ⋆ Cod ν ⋆ Cod τ)) ·B
         (Map μ ⋆B Map ν ⋆B Map τ) ·B
          B.can (Dom μ ⋆ Dom ν ⋆ Dom τ) ((Dom μ ⌊⋆⌋ Dom ν) ⋆ Dom τ) ·B
           B.can ((Dom μ ⌊⋆⌋ Dom ν) ⋆ Dom τ) (Dom μ ⌊⋆⌋ Dom ν ⌊⋆⌋ Dom τ)
 using B.comp-assoc by simp
```

also have ... = $B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
proof −
have $B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau) \cdot_B$
$B.can\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau) =$
$B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau)$
proof −
have $E.Ide\ (Cod\ \mu \star Cod\ \nu \star Cod\ \tau)$
using *assms src-def trg-def* **by** *fastforce*
moreover have $E.Ide\ ((Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau)$
using *assms arr-char src-def trg-def E.Ide-HcompNml E.Src-HcompNml*
by *auto*
moreover have $E.Ide\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)$
using *assms arr-char src-def trg-def*
by (*simp add*: *E.Nml-HcompNml*(*1*) *E.Ide-HcompNml E.Trg-HcompNml*)
moreover have $\lfloor Cod\ \mu \star Cod\ \nu \star Cod\ \tau\rfloor = \lfloor(Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau\rfloor$
using *assms arr-char src-def trg-def E.Nml-HcompNml E.HcompNml-assoc* **by** *simp*
moreover have $\lfloor(Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu) \star Cod\ \tau\rfloor = \lfloor Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau\rfloor$
using *assms arr-char src-def trg-def E.Nml-HcompNml E.HcompNml-assoc*
by *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have
$B.can\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau) \cdot_B$
$B.can\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau) =$
$B.can\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
proof −
have $E.Ide\ (Dom\ \mu \star Dom\ \nu \star Dom\ \tau)$
using *assms src-def trg-def* **by** *fastforce*
moreover have $E.Ide\ ((Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau)$
using *assms arr-char src-def trg-def E.Ide-HcompNml E.Src-HcompNml*
by *auto*
moreover have $E.Ide\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
using *assms arr-char src-def trg-def*
by (*simp add*: *E.Nml-HcompNml*(*1*) *E.Ide-HcompNml E.Trg-HcompNml*)
moreover have $\lfloor Dom\ \mu \star Dom\ \nu \star Dom\ \tau\rfloor = \lfloor(Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau\rfloor$
using *assms arr-char src-def trg-def E.Nml-HcompNml E.HcompNml-assoc* **by** *simp*
moreover have
$\lfloor(Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu) \star Dom\ \tau\rfloor = \lfloor Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau\rfloor$
using *assms arr-char src-def trg-def E.Nml-HcompNml E.HcompNml-assoc*
by *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
also have ... = $Map\ (\mu \star \nu \star \tau)$

proof −
have *1*: $Map\ (\mu \star \nu \star \tau) =$
$B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau) \cdot_B$
$(Map\ \mu \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
using *assms Map-hcomp* [*of* $\mu\ \nu \star \tau$] *Map-hcomp* [*of* $\nu\ \tau$] **by** *simp*
also have
$... = B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau) \cdot_B$
$((B.can\ (Cod\ \mu)\ (Cod\ \mu) \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau)) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$(B.can\ (Dom\ \mu)\ (Dom\ \mu) \star_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau))) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)$
proof −
have $Map\ \mu \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau) =$
$(B.can\ (Cod\ \mu)\ (Cod\ \mu) \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau)) \cdot_B$
$(Map\ \mu \star_B (Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau))$
using *assms B.interchange B.comp-cod-arr*
by (*metis* (*no-types*, *lifting*) *B.can-Ide-self B.in-homE Map-hcomp arrE hseq-char*)
also have $... = (B.can\ (Cod\ \mu)\ (Cod\ \mu) \star_B$
$B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau)) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$(B.can\ (Dom\ \mu)\ (Dom\ \mu) \star_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau))$
using *assms B.interchange B.comp-arr-dom* [*of Map* μ *B.can* $(Dom\ \mu)\ (Dom\ \mu)$]
by (*metis* (*no-types*, *lifting*) *B.can-Ide-self B.null-is-zero*(*2*) *B.ext B.in-homE Map-hcomp arrE hseq-char*)
finally have
$Map\ \mu \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau) \cdot_B$
$(Map\ \nu \star_B Map\ \tau) \cdot_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau) =$
$(B.can\ (Cod\ \mu)\ (Cod\ \mu) \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau)) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$(B.can\ (Dom\ \mu)\ (Dom\ \mu) \star_B B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau))$
by *simp*
thus *?thesis* **by** *simp*
qed
also have
$... = (B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \mu \star Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau) \cdot_B$
$(B.can\ (Cod\ \mu)\ (Cod\ \mu) \star_B B.can\ (Cod\ \nu\ \lfloor\star\rfloor\ Cod\ \tau)\ (Cod\ \nu \star Cod\ \tau))) \cdot_B$
$(Map\ \mu \star_B Map\ \nu \star_B Map\ \tau) \cdot_B$
$((B.can\ (Dom\ \mu)\ (Dom\ \mu) \star_B$
$B.can\ (Dom\ \nu \star Dom\ \tau)\ (Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu\ \lfloor\star\rfloor\ Dom\ \tau))$

```
      using B.comp-assoc by simp
    also have ... = B.can (Cod μ ⌊⋆⌋ Cod ν ⌊⋆⌋ Cod τ) (Cod μ ⋆ Cod ν ⋆ Cod τ) ·B
                    (Map μ ⋆B Map ν ⋆B Map τ) ·B
                      B.can (Dom μ ⋆ Dom ν ⋆ Dom τ) (Dom μ ⌊⋆⌋ Dom ν ⌊⋆⌋ Dom τ)
      using assms μν ντ E.HcompNml-assoc src-def trg-def arr-char
          E.Src-HcompNml E.Trg-HcompNml E.Nml-HcompNml E.Ide-HcompNml
      by simp
    finally show ?thesis by simp
  qed
  ultimately show ?thesis by metis
 qed
qed

lemma obj-char:
shows obj a ⟷ endo a ∧ E.Obj (Dom a) ∧ Map a = ⦃Dom a⦄
proof
 assume a: obj a
 show endo a ∧ E.Obj (Dom a) ∧ Map a = ⦃Dom a⦄
 proof (intro conjI)
  show endo a
   using a ide-char by blast
  show E.Obj (Dom a)
   using a ide-char src-def
   by (metis (no-types, lifting) E.Ide-implies-Arr E.Obj-Trg arrE obj-def
       trg-simps(1) trg-src)
  show Map a = ⦃Dom a⦄
   using a ide-char src-def by blast
 qed
 next
 assume a: endo a ∧ E.Obj (Dom a) ∧ Map a = ⦃Dom a⦄
 show obj a
 proof −
  have arr a using a by auto
  moreover have src a = a
   using a E.Obj-in-Hom(1) seq-char by (intro arr-eqI, auto)
  ultimately show ?thesis
   using obj-def by simp
 qed
qed

lemma hcomp-obj-self:
assumes obj a
shows a ⋆ a = a
proof (intro arr-eqI)
 show hseq a a
  using assms by auto
 show arr a
  using assms by auto
 show 1: Dom (a ⋆ a) = Dom a
```

```
    unfolding hcomp-def
    using assms arr-char E.HcompNml-Trg-Nml
    apply simp
    by (metis (no-types, lifting) objE obj-def trg-simps(1))
  show 2: Cod (a ⋆ a) = Cod a
    unfolding hcomp-def
    using assms 1 arr-char E.HcompNml-Trg-Nml
    apply simp
    by (metis (no-types, lifting) Dom-hcomp ideE objE)
  show Map (a ⋆ a) = Map a
    using 1 Map-ide(1) assms by fastforce
qed

lemma hcomp-ide-src:
assumes ide f
shows f ⋆ src f = f
proof (intro arr-eqI)
  show hseq f (src f)
    using assms by simp
  show arr f
    using assms by simp
  show 1: Dom (f ⋆ src f) = Dom f
    unfolding hcomp-def
    using assms apply simp
    using assms ide-char arr-char E.HcompNml-Nml-Src
    by (metis (no-types, lifting) ideD(1))
  show Cod (f ⋆ src f) = Cod f
    unfolding hcomp-def
    using assms apply simp
    using assms ide-char arr-char E.HcompNml-Nml-Src
    by (metis (no-types, lifting) ideD(1))
  show Map (f ⋆ src f) = Map f
    by (simp add: 1 Map-ide(1) assms)
qed

lemma hcomp-trg-ide:
assumes ide f
shows trg f ⋆ f = f
proof (intro arr-eqI)
  show hseq (trg f) f
    using assms by auto
  show arr f
    using assms by auto
  show 1: Dom (trg f ⋆ f) = Dom f
    unfolding hcomp-def
    using assms apply simp
    using assms ide-char arr-char E.HcompNml-Trg-Nml
    by (metis (no-types, lifting) ideD(1))
  show Cod (trg f ⋆ f) = Cod f
```

```
    unfolding hcomp-def
    using assms apply simp
    using assms ide-char arr-char E.HcompNml-Trg-Nml
    by (metis (no-types, lifting)  ideD(1))
  show Map (trg f ⋆ f) = Map f
    by (simp add: 1 Map-ide(1) assms)
qed

interpretation L: full-functor vcomp vcomp L
proof
  fix a a′ g
  assume a: ide a and a′: ide a′
  assume g: in-hom g (L a′) (L a)
  have a-eq: a = MkIde (Dom a)
    using a dom-char [of a] by simp
  have a′-eq: a′ = MkIde (Dom a′)
    using a′ dom-char [of a′] by simp
  have 1: Cod g = Dom a
  proof −
    have Dom (L a) = Dom a
      by (simp add: a hcomp-trg-ide)
    thus ?thesis
      using g cod-char [of g]
      by (metis (no-types, lifting) Dom-cod in-homE)
  qed
  have 2: Dom g = Dom a′
    using a′ g hcomp-trg-ide in-hom-char by auto
  let ?f = MkArr (Dom a′) (Cod a) (Map g)
  have f: in-hom ?f a′ a
    by (metis (no-types, lifting) 1 2 MkArr-Map a a′ g ideE in-hom-char)
  moreover have L ?f = g
  proof −
    have L ?f =
          trg (MkArr (Dom a′) (Cod a) (Map g)) ⋆ MkArr (Dom a′) (Cod a) (Map g)
      using f by auto
    also have ... = MkIde (E.Trg (Cod a)) ⋆ MkArr (Dom a′) (Cod a) (Map g)
      using a a′ f trg-def [of a] vconn-implies-hpar by auto
    also have ... = MkArr (E.Trg (Cod a) ⌊⋆⌋ Dom a′) (E.Trg (Cod a) ⌊⋆⌋ Cod a)
                        (B.can (E.Trg (Cod a) ⌊⋆⌋ Cod a) (E.Trg (Cod a) ⋆ Cod a) ·B
                          (⦃E.Trg (Cod a)⦄ ⋆B Map g) ·B
                          B.can (E.Trg (Cod a) ⋆ Dom a′) (E.Trg (Cod a) ⌊⋆⌋ Dom a′))
      using hcomp-def
      apply simp
      by (metis (no-types, lifting) Cod.simps(1) arrE f in-homE src-trg trg.preserves-arr
          trg-def)
    also have ... = MkArr (Dom a′) (Cod a)
                        (B.can (Cod a) (E.Trg (Cod a) ⋆ Cod a) ·B
                          (trgB ⦃Cod a⦄ ⋆B Map g) ·B
                          B.can (E.Trg (Cod a) ⋆ Dom a′) (Dom a′))
```

```
proof −
  have E.Trg (Cod a) ⌊⋆⌋ Dom a′ = Dom a′
    using a a′ arr-char E.HcompNml-Trg-Nml
    by (metis (no-types, lifting) f ideE trg-simps(1) vconn-implies-hpar(4))
  moreover have E.Trg (Cod a) ⌊⋆⌋ Cod a = Cod a
    using a a′ arr-char E.HcompNml-Trg-Nml by blast
  moreover have ⦃E.Trg (Cod a)⦄ = trg_B ⦃Cod a⦄
    using a a′ arr-char E.eval-simps′(3) by fastforce
  ultimately show ?thesis by simp
qed
also have ... = MkArr (Dom a′) (Cod a)
                (B.lunit ⦃Cod a⦄ ·_B (trg_B ⦃Cod a⦄ ⋆_B Map g) ·_B B.lunit′ ⦃Dom a′⦄)
proof −
  have E.Trg (Cod a) = E.Trg (Dom a′)
    using a a′ a-eq g ide-char arr-char src-def trg-def trg-hcomp
          ‹Cod g = Dom a› ‹Dom g = Dom a′›
    by (metis (no-types, lifting) Cod.simps(1) in-homE)
  moreover have B.can (Cod a) (E.Trg (Cod a) ⋆ Cod a) = B.lunit ⦃Cod a⦄
    using a ide-char arr-char B.canE-unitor(2) by blast
  moreover have B.can (E.Trg (Dom a′) ⋆ Dom a′) (Dom a′) = B.lunit′ ⦃Dom a′⦄
    using a′ ide-char arr-char B.canE-unitor(4) by blast
  ultimately show ?thesis by simp
qed
also have ... = MkArr (Dom g) (Cod g) (Map g)
proof −
  have src_B ⦃Cod a⦄ = src_B (Map g)
    using a f g ide-char arr-char src-def B.comp-cod-arr
    by (metis (no-types, lifting) B.vconn-implies-hpar(1) B.vconn-implies-hpar(3)
        Cod.simps(1) Map.simps(1) in-homE)
  moreover have
    B.lunit ⦃Cod g⦄ ·_B (trg_B (Map g) ⋆_B Map g) ·_B B.lunit′ ⦃Dom g⦄ = Map g
  proof −
    have B.lunit ⦃Cod g⦄ ·_B (trg_B (Map g) ⋆_B Map g) ·_B B.lunit′ ⦃Dom g⦄ =
          B.lunit ⦃Cod g⦄ ·_B B.lunit′ ⦃Cod g⦄ ·_B Map g
      using g ide-char arr-char B.lunit′-naturality
      by (metis (no-types, lifting) partial-composition-axioms B.in-homE
          partial-composition.arrI)
    also have ... = (B.lunit ⦃Cod g⦄ ·_B B.lunit′ ⦃Cod g⦄) ·_B Map g
      using B.comp-assoc by simp
    also have ... = ⦃Cod g⦄ ·_B Map g
      using g E.ide-eval-Ide B.comp-arr-inv′ by fastforce
    also have ... = Map g
      using g E.ide-eval-Ide B.comp-cod-arr by fastforce
    finally show ?thesis by simp
  qed
  ultimately have
    B.lunit ⦃Cod a⦄ ·_B (trg_B ⦃Cod a⦄ ⋆_B Map g) ·_B B.lunit′ ⦃Dom a′⦄ = Map g
    using a a′ 1 2 f g hcomp-def dom-char cod-char
   by (metis (no-types, lifting) B.null-is-zero(2) B.ext B.lunit-simps(2) B.lunit-simps(3)
```

```
        B.src.preserves-reflects-arr B.trg-vcomp B.vseq-implies-hpar(1) ideE)
      thus ?thesis
        using a 1 2 by auto
    qed
    also have ... = g
      using g MkArr-Map by blast
    finally show ?thesis by simp
  qed
  ultimately show ∃f. in-hom f a′ a ∧ L f = g
    by blast
qed

interpretation R: full-functor vcomp vcomp R
proof
  fix a a′ g
  assume a: ide a and a′: ide a′
  assume g: in-hom g (R a′) (R a)
  have a-eq: a = MkIde (Dom a)
    using a dom-char [of a] by simp
  have a′-eq: a′ = MkIde (Dom a′)
    using a′ dom-char [of a′] by simp
  have 1: Cod g = Dom a
    using a g hcomp-ide-src in-hom-char by force
  have 2: Dom g = Dom a′
    using a′ g hcomp-ide-src by auto
  let ?f = MkArr (Dom a′) (Cod a) (Map g)
  have f: in-hom ?f a′ a
  proof (intro in-homI)
    show 3: arr (MkArr (Dom a′) (Cod a) (Map g))
      by (metis (no-types, lifting) 1 2 Cod.simps(1) MkArr-Map a-eq g in-homE)
    show dom (MkArr (Dom a′) (Cod a) (Map g)) = a′
      using a a′ 3 dom-char by auto
    show cod (MkArr (Dom a′) (Cod a) (Map g)) = a
      using a a′ 3 cod-char by auto
  qed
  moreover have R ?f = g
  proof −
    have R ?f =
          MkArr (Dom a′) (Cod a) (Map g) ⋆ src (MkArr (Dom a′) (Cod a) (Map g))
      using f by auto
    also have ... = MkArr (Dom a′) (Cod a) (Map g) ⋆ MkIde (E.Src (Cod a))
      using a a′ f src-def [of a] vconn-implies-hpar by auto
    also have ... = MkArr (Dom a′ ⌊⋆⌋ E.Src (Cod a)) (Cod a ⌊⋆⌋ E.Src (Cod a))
                      (B.can (Cod a ⌊⋆⌋ E.Src (Cod a)) (Cod a ⋆ E.Src (Cod a)) ·B
                        (Map g ⋆B ⦃E.Src (Cod a)⦄) ·B
                        B.can (Dom a′ ⋆ E.Src (Cod a)) (Dom a′ ⌊⋆⌋ E.Src (Cod a)))
      using hcomp-def
      apply simp
      by (metis (no-types, lifting) Cod-cod arrE f in-homE trg-src src.preserves-arr src-def)
```

```
also have ... = MkArr (Dom a') (Cod a)
                  (B.can (Cod a) (Cod a ⋆ E.Src (Cod a)) ·B
                    (Map g ⋆B srcB ⦃Cod a⦄) ·B
                    B.can (Dom a' ⋆ E.Src (Cod a)) (Dom a'))
proof −
  have Dom a' ⌊⋆⌋ E.Src (Cod a) = Dom a'
    using a a' arr-char E.HcompNml-Nml-Src
    by (metis (no-types, lifting) f ideE src-simps(1) vconn-implies-hpar(3))
  moreover have Cod a ⌊⋆⌋ E.Src (Cod a) = Cod a
    using a a' arr-char E.HcompNml-Nml-Src by blast
  moreover have ⦃E.Src (Cod a)⦄ = srcB ⦃Cod a⦄
    using a a' arr-char E.eval-simps'(2) by fastforce
  ultimately show ?thesis by simp
qed
also have ... = MkArr (Dom a') (Cod a)
                  (B.runit ⦃Cod a⦄ ·B (Map g ⋆B srcB ⦃Cod a⦄) ·B B.runit' ⦃Dom a'⦄)
  by (metis (no-types, lifting) B.canE-unitor(1) B.canE-unitor(3) a a' arrE f ideE
      src-simps(1) vconn-implies-hpar(3))
also have ... = MkArr (Dom g) (Cod g) (Map g)
proof −
  have srcB ⦃Cod a⦄ = srcB (Map g)
    using a f g ide-char arr-char src-def B.comp-cod-arr
    by (metis (no-types, lifting) B.vconn-implies-hpar(1) B.vconn-implies-hpar(3)
        Cod.simps(1) Map.simps(1) in-homE)
  moreover have
    B.runit ⦃Cod g⦄ ·B (Map g ⋆B srcB (Map g)) ·B B.runit' ⦃Dom g⦄ = Map g
  proof −
    have B.runit ⦃Cod g⦄ ·B (Map g ⋆B srcB (Map g)) ·B B.runit' ⦃Dom g⦄ =
          B.runit ⦃Cod g⦄ ·B B.runit'⦃Cod g⦄ ·B Map g
      using g ide-char arr-char B.runit'-naturality [of Map g]
      by (metis (no-types, lifting) partial-composition-axioms B.in-homE
          partial-composition.arrI)
    also have ... = (B.runit ⦃Cod g⦄ ·B B.runit' ⦃Cod g⦄) ·B Map g
      using B.comp-assoc by simp
    also have ... = ⦃Cod g⦄ ·B Map g
      using g E.ide-eval-Ide B.comp-arr-inv' by fastforce
    also have ... = Map g
      using g E.ide-eval-Ide B.comp-cod-arr by fastforce
    finally show ?thesis by simp
  qed
  ultimately have
    B.runit ⦃Cod a⦄ ·B (Map g ⋆B srcB ⦃Cod a⦄) ·B B.runit' ⦃Dom a'⦄ = Map g
    using a a' 1 2 f g hcomp-def dom-char cod-char
    by (metis (no-types, lifting) ideE)
  thus ?thesis
    using a 1 2 by auto
qed
also have ... = g
   using g MkArr-Map by blast
```

```
    finally show ?thesis by simp
  qed
  ultimately show ∃f. in-hom f a′ a ∧ R f = g
    by blast
qed

interpretation L: faithful-functor vcomp vcomp L
proof
  fix f f′
  assume par: par f f′ and eq: L f = L f′
  show f = f′
  proof (intro arr-eqI)
    have 1: Dom f = Dom f′ ∧ Cod f = Cod f′
      using par dom-char cod-char by auto
    show arr f
      using par by simp
    show arr f′
      using par by simp
    show 2: Dom f = Dom f′ and 3: Cod f = Cod f′
      using 1 by auto
    show Map f = Map f′
    proof −
      have B.L (Map f) = trg_B (Map f) ⋆_B Map f
        using par by auto
      also have ... = trg_B (Map f′) ⋆_B Map f′
      proof −
        have ⦃E.Trg (Dom f)⦄ ⋆_B Map f = ⦃E.Trg (Dom f′)⦄ ⋆_B Map f′
        proof −
          have A: «B.can (E.Trg (Dom f) ⋆ Dom f) (E.Trg (Dom f) ⌊⋆⌋ Dom f) :
                    ⦃E.Trg (Dom f) ⌊⋆⌋ Dom f⦄ ⇒_B ⦃E.Trg (Dom f)⦄ ⋆_B ⦃Dom f⦄»
            using par arr-char B.can-in-hom E.Ide-HcompNml
                  E.Ide-Nmlize-Ide E.Nml-Trg E.Nmlize-Nml E.HcompNml-Trg-Nml
                  src-def trg-def
            by (metis (no-types, lifting) E.eval-simps(3) E.ide-eval-Ide E.Ide-implies-Arr
                B.canE-unitor(4) B.lunit′-in-vhom)
          have B: «B.can (E.Trg (Dom f) ⌊⋆⌋ Cod f) (E.Trg (Dom f) ⋆ Cod f) :
                    ⦃E.Trg (Dom f)⦄ ⋆_B ⦃Cod f⦄ ⇒_B ⦃E.Trg (Dom f) ⌊⋆⌋ Cod f⦄»
            using par arr-char B.can-in-hom E.Ide-HcompNml
                  E.Ide-Nmlize-Ide E.Nml-Trg E.Nmlize-Nml E.HcompNml-Trg-Nml
                  src-def trg-def
            by (metis (no-types, lifting) E.Nmlize.simps(3) E.eval.simps(3) E.Ide.simps(3)
                E.Ide-implies-Arr E.Src-Trg trg.preserves-arr trg-simps(2))
          have C: «⦃E.Trg (Dom f)⦄ ⋆_B Map f :
                    ⦃E.Trg (Dom f)⦄ ⋆_B ⦃Dom f⦄ ⇒_B ⦃E.Trg (Dom f)⦄ ⋆_B ⦃Cod f⦄»
            using par arr-char
            by (metis (no-types, lifting) E.eval-simps′(1) E.eval-simps(3) E.ide-eval-Ide
                E.Ide-implies-Arr E.Obj-Trg E.Obj-implies-Ide B.hcomp-in-vhom
                B.ide-in-hom(2) B.src-trg)
          have 3: (⦃E.Trg (Dom f)⦄ ⋆_B Map f) ·_B
```

$B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f) =$
$(\{\!|E.Trg\ (Dom\ f')|\!\}\ \star_B\ Map\ f') \cdot_B$
$B.can\ (E.Trg\ (Dom\ f') \star Dom\ f')\ (E.Trg\ (Dom\ f')\ \lfloor\star\rfloor\ Dom\ f')$

proof −
have *2*: $B.can\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Cod\ f)\ (E.Trg\ (Dom\ f) \star Cod\ f) \cdot_B$
$(\{\!|E.Trg\ (Dom\ f)|\!\}\ \star_B\ Map\ f) \cdot_B$
$B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f) =$
$B.can\ (E.Trg\ (Dom\ f')\ \lfloor\star\rfloor\ Cod\ f')\ (E.Trg\ (Dom\ f') \star Cod\ f') \cdot_B$
$(\{\!|E.Trg\ (Dom\ f')|\!\}\ \star_B\ Map\ f') \cdot_B$
$B.can\ (E.Trg\ (Dom\ f') \star Dom\ f')\ (E.Trg\ (Dom\ f')\ \lfloor\star\rfloor\ Dom\ f')$
using *par eq hcomp-def trg-def src-trg trg.preserves-arr Map-hcomp trg-simps(1) trg-simps(2) trg-simps(3)*
by *auto*
have $B.mono\ (B.can\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Cod\ f)\ (E.Trg\ (Dom\ f) \star Cod\ f))$
using *par arr-char B.inverse-arrows-can B.iso-is-section B.section-is-mono src-def trg-def E.Nmlize-Nml E.HcompNml-Trg-Nml E.Ide-implies-Arr trg.preserves-arr trg-simps(1)*
by *auto*
moreover have
$B.seq\ (B.can\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Cod\ f)\ (E.Trg\ (Dom\ f) \star Cod\ f))$
$((\{\!|E.Trg\ (Dom\ f)|\!\}\ \star_B\ Map\ f) \cdot_B$
$B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f))$
using *A B C* **by** *auto*
moreover have
$B.seq\ (B.can\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Cod\ f)\ (E.Trg\ (Dom\ f) \star Cod\ f))$
$((\{\!|E.Trg\ (Dom\ f')|\!\}\ \star_B\ Map\ f') \cdot_B$
$B.can\ (E.Trg\ (Dom\ f') \star Dom\ f')\ (E.Trg\ (Dom\ f')\ \lfloor\star\rfloor\ Dom\ f'))$
using *par 1 2 arr-char calculation(2)* **by** *auto*
moreover have $B.can\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Cod\ f)\ (E.Trg\ (Dom\ f) \star Cod\ f) =$
$B.can\ (E.Trg\ (Dom\ f')\ \lfloor\star\rfloor\ Cod\ f')\ (E.Trg\ (Dom\ f') \star Cod\ f')$
using *par 1 arr-char* **by** *simp*
ultimately show *?thesis*
using *2 B.mono-cancel cod-char* **by** *auto*
qed
show *?thesis*
proof −
have $B.epi\ (B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f))$
using *par arr-char B.inverse-arrows-can B.iso-is-retraction B.retraction-is-epi E.Nmlize-Nml E.HcompNml-Trg-Nml src-def trg-def E.Ide-implies-Arr*
by (*metis* (*no-types*, *lifting*) *E.Nmlize.simps(3) E.Ide.simps(3) E.Src-Trg trg.preserves-arr trg-simps(1)*)
moreover have $B.seq\ (\{\!|E.Trg\ (Dom\ f)|\!\}\ \star_B\ Map\ f)$
$(B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f))$
using *A C* **by** *auto*
moreover have $B.seq\ (\{\!|E.Trg\ (Dom\ f')|\!\}\ \star_B\ Map\ f')$
$(B.can\ (E.Trg\ (Dom\ f) \star Dom\ f)\ (E.Trg\ (Dom\ f)\ \lfloor\star\rfloor\ Dom\ f))$
using *1 3 calculation(2)* **by** *auto*
ultimately show *?thesis*

```
            using par 1 3 arr-char B.epi-cancel by simp
          qed
        qed
        moreover have trg_B (Map f) = ⦃E.Trg (Dom f)⦄ ∧
                      trg_B (Map f′) = ⦃E.Trg (Dom f′)⦄
          using par arr-char trg-def E.Ide-implies-Arr B.comp-arr-dom
                B.vseq-implies-hpar(2) E.eval-simps(3)
          by (metis (no-types, lifting) B.vconn-implies-hpar(2))
        ultimately show ?thesis by simp
      qed
      also have ... = B.L (Map f′)
        using par B.hseqE B.hseq-char′ by auto
      finally have B.L (Map f) = B.L (Map f′)
        by simp
      thus ?thesis
        using 2 3 par arr-char B.L.is-faithful
        by (metis (no-types, lifting) B.in-homE)
    qed
  qed
qed

interpretation R: faithful-functor vcomp vcomp R
proof
  fix f f′
  assume par: par f f′ and eq: R f = R f′
  show f = f′
  proof (intro arr-eqI)
    have 1: Dom f = Dom f′ ∧ Cod f = Cod f′
      using par dom-char cod-char by auto
    show arr f
      using par by simp
    show arr f′
      using par by simp
    show 2: Dom f = Dom f′ and 3: Cod f = Cod f′
      using 1 by auto
    show Map f = Map f′
    proof −
      have B.R (Map f) = Map f ⋆_B src_B (Map f)
        using par apply simp by (metis B.hseqE B.hseq-char′)
      also have ... = Map f′ ⋆_B src_B (Map f′)
      proof −
        have Map f ⋆_B ⦃E.Src (Dom f)⦄ = Map f′ ⋆_B ⦃E.Src (Dom f′)⦄
        proof −
          have 2: E.Ide (Cod f ⋆ E.Src (Dom f))
            using par arr-char src.preserves-arr by auto
          hence 3: E.Ide (Cod f ⌊⋆⌋ E.Src (Dom f))
            using par arr-char E.Nml-Src E.Ide-HcompNml calculation by auto
          have 4: ⌊Cod f ⋆ E.Src (Dom f)⌋ = ⌊Cod f ⌊⋆⌋ E.Src (Dom f)⌋
            using par arr-char by (simp add: E.Nml-HcompNml(1))
```

have *A*: «*B.can* (*Dom f* ⋆ *E.Src* (*Dom f*)) (*Dom f* ⌊⋆⌋ *E.Src* (*Dom f*)) :
⦃*Dom f* ⌊⋆⌋ *E.Src* (*Dom f*)⦄ $\Rightarrow_B$ ⦃*Dom f*⦄ $\star_B$ ⦃*E.Src* (*Dom f*)⦄»
using *par arr-char B.can-in-hom E.Ide-HcompNml*
E.Ide-Nmlize-Ide E.Nml-Src E.Nmlize-Nml E.HcompNml-Nml-Src
src-def trg-def
by (*metis* (*no-types*, *lifting*) *E.eval-simps*(*2*) *E.ide-eval-Ide E.Ide-implies-Arr*
B.canE-unitor(*3*) *B.runit′-in-vhom*)
have *B*: «*B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)) :
⦃*Cod f*⦄ $\star_B$ ⦃*E.Src* (*Dom f*)⦄ $\Rightarrow_B$ ⦃*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)⦄»
using *2 3 4 B.can-in-hom* [*of Cod f* ⋆ *E.Src* (*Dom f*) *Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)]
by *simp*
have *C*: «*Map f* $\star_B$ ⦃*E.Src* (*Dom f*)⦄ :
⦃*Dom f*⦄ $\star_B$ ⦃*E.Src* (*Dom f*)⦄ $\Rightarrow_B$ ⦃*Cod f*⦄ $\star_B$ ⦃*E.Src* (*Dom f*)⦄»
using *par arr-char E.Ide-Nmlize-Ide E.Nml-Trg E.Nmlize-Nml E.HcompNml-Trg-Nml*
src-def trg-def E.ide-eval-Ide E.Ide-implies-Arr E.Obj-implies-Ide
apply (*intro B.hcomp-in-vhom*)
apply (*simp add*: *B.ide-in-hom*(*2*))
apply *simp*
by (*metis* (*no-types*, *lifting*) *A B.ideD*(*1*) *B.not-arr-null B.seq-if-composable*
B.src.preserves-reflects-arr B.vconn-implies-hpar(*3*) *E.HcompNml-Nml-Src*)
have *5*: (*Map f* $\star_B$ ⦃*E.Src* (*Dom f*)⦄) $\cdot_B$
B.can (*Dom f* ⋆ *E.Src* (*Dom f*)) (*Dom f* ⌊⋆⌋ *E.Src* (*Dom f*)) =
(*Map f′* $\star_B$ ⦃*E.Src* (*Dom f′*)⦄) $\cdot_B$
B.can (*Dom f′* ⋆ *E.Src* (*Dom f′*)) (*Dom f′* ⌊⋆⌋ *E.Src* (*Dom f′*))
proof −
have *6*: *B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)) $\cdot_B$
(*Map f* $\star_B$ ⦃*E.Src* (*Dom f*)⦄) $\cdot_B$
B.can (*Dom f* ⋆ *E.Src* (*Dom f*)) (*Dom f* ⌊⋆⌋ *E.Src* (*Dom f*)) =
B.can (*Cod f′* ⌊⋆⌋ *E.Src* (*Dom f′*)) (*Cod f′* ⋆ *E.Src* (*Dom f′*)) $\cdot_B$
(*Map f′* $\star_B$ ⦃*E.Src* (*Dom f′*)⦄) $\cdot_B$
B.can (*Dom f′* ⋆ *E.Src* (*Dom f′*)) (*Dom f′* ⌊⋆⌋ *E.Src* (*Dom f′*))
using *par eq hcomp-def src-def trg-src src.preserves-arr Map-hcomp*
src-simps(*1*) *src-simps*(*2*) *src-simps*(*3*)
by *auto*
have *B.mono* (*B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)))
using *2 3 4 B.inverse-arrows-can*(*1*) *B.iso-is-section B.section-is-mono*
by *simp*
moreover have
B.seq (*B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)))
((*Map f* $\star_B$ ⦃*E.Src* (*Dom f*)⦄) $\cdot_B$
B.can (*Dom f* ⋆ *E.Src* (*Dom f*)) (*Dom f* ⌊⋆⌋ *E.Src* (*Dom f*)))
using *A B C* **by** *auto*
moreover have
B.seq (*B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)))
((*Map f′* $\star_B$ ⦃*E.Src* (*Dom f′*)⦄) $\cdot_B$
B.can (*Dom f′* ⋆ *E.Src* (*Dom f′*)) (*Dom f′* ⌊⋆⌋ *E.Src* (*Dom f′*)))
using *par 1 6 arr-char calculation*(*2*) **by** *auto*
moreover have *B.can* (*Cod f* ⌊⋆⌋ *E.Src* (*Dom f*)) (*Cod f* ⋆ *E.Src* (*Dom f*)) =
B.can (*Cod f′* ⌊⋆⌋ *E.Src* (*Dom f′*)) (*Cod f′* ⋆ *E.Src* (*Dom f′*))

```
            using par 1 arr-char by simp
          ultimately show ?thesis
            using 6 B.mono-cancel cod-char by auto
        qed
        show ?thesis
        proof −
          have B.epi (B.can (Dom f ⋆ E.Src (Dom f)) (Dom f ⌊⋆⌋ E.Src (Dom f)))
            using 2 3 4 B.inverse-arrows-can(1) B.iso-is-retraction B.retraction-is-epi
            by (metis (no-types, lifting) E.Nml-Src E.Nmlize.simps(3) E.Nmlize-Nml
                E.HcompNml-Nml-Src E.Ide.simps(3) par arrE)
          moreover have B.seq (Map f ⋆B ⦃E.Src (Dom f)⦄)
                          (B.can (Dom f ⋆ E.Src (Dom f)) (Dom f ⌊⋆⌋ E.Src (Dom f)))
            using A C by auto
          moreover have B.seq (Map f′ ⋆B ⦃E.Src (Dom f′)⦄)
                          (B.can (Dom f ⋆ E.Src (Dom f)) (Dom f ⌊⋆⌋ E.Src (Dom f)))
            using 1 5 calculation(2) by auto
          ultimately show ?thesis
            using par 1 5 arr-char B.epi-cancel by simp
        qed
      qed
      moreover have srcB (Map f) = ⦃E.Src (Dom f)⦄ ∧
                    srcB (Map f′) = ⦃E.Src (Dom f′)⦄
        using par arr-char src-def
        by (metis (no-types, lifting) B.vconn-implies-hpar(1) E.Nml-implies-Arr
            E.eval-simps(2))
      ultimately show ?thesis by simp
    qed
    also have ... = B.R (Map f′)
      using par B.hseqE B.hseq-char′ by auto
    finally have B.R (Map f) = B.R (Map f′)
      by simp
    thus ?thesis
      using 2 3 par arr-char B.R.is-faithful
      by (metis (no-types, lifting) B.in-homE)
  qed
 qed
qed

definition a
where a τ μ ν ≡ if VVV.arr (τ, μ, ν) then hcomp τ (hcomp μ ν) else null

interpretation natural-isomorphism VVV.comp vcomp HoHV HoVH
              ‹λτμν. a (fst τμν) (fst (snd τμν)) (snd (snd τμν))›
proof
  show ⋀τμν. ¬ VVV.arr τμν ⟹ a (fst τμν) (fst (snd τμν)) (snd (snd τμν)) = null
    using a-def by simp
  show 1: ⋀τμν. VVV.arr τμν ⟹ arr (a (fst τμν) (fst (snd τμν)) (snd (snd τμν)))
    using VVV.arr-charSbC VV.arr-charSbC a-def hcomp-assoc HoHV-def VVV.dom-simp
VV.dom-simp
```

```
    by force
  show ⋀τμν. VVV.arr τμν ⟹
            HoVH τμν ·
              a (fst (VVV.dom τμν)) (fst (snd (VVV.dom τμν)))
                (snd (snd (VVV.dom τμν))) =
            a (fst τμν) (fst (snd τμν)) (snd (snd τμν))
    using a-def HoVH.as-nat-trans.naturality1 HoVH-def by auto
  show ⋀τμν. VVV.arr τμν ⟹
              a (fst (VVV.cod τμν)) (fst (snd (VVV.cod τμν)))
                (snd (snd (VVV.cod τμν))) · HoHV τμν =
              a (fst τμν) (fst (snd τμν)) (snd (snd τμν))
  proof −
    fix τμν
    assume τμν: VVV.arr τμν
    have HoHV τμν = a (fst τμν) (fst (snd τμν)) (snd (snd τμν))
      unfolding a-def HoHV-def
      using τμν HoHV.preserves-cod hcomp-assoc VVV.arr-char_SbC VV.arr-char_SbC
      by simp
    thus a (fst (VVV.cod τμν)) (fst (snd (VVV.cod τμν))) (snd (snd (VVV.cod τμν))) ·
           HoHV τμν =
         a (fst τμν) (fst (snd τμν)) (snd (snd τμν))
      using 1 τμν comp-cod-arr a-def HoVH.as-nat-trans.naturality2
      by (metis (no-types, lifting) HoVH-def prod.collapse)
  qed
  show ⋀fgh. VVV.ide fgh ⟹ iso (a (fst fgh) (fst (snd fgh)) (snd (snd fgh)))
    using a-def HoVH.preserves-ide HoVH-def by auto
qed

definition i
where i ≡ λa. a

sublocale bicategory vcomp hcomp a i src trg
  using hcomp-obj-self a-def hcomp-assoc VVV.arr-char_SbC VV.arr-char_SbC
  apply unfold-locales
  by (auto simp add: i-def ide-in-hom(2))

lemma is-bicategory:
shows bicategory vcomp hcomp a i src trg
  ..

sublocale strict-bicategory vcomp hcomp a i src trg
proof
  show ⋀fgh. ide fgh ⟹ lunit fgh = fgh
  proof −
    fix fgh
    assume fgh: ide fgh
    have fgh = lunit fgh
    proof (intro lunit-eqI)
      show ide fgh using fgh by simp
```

```
      show «fgh : trg fgh ⋆ fgh ⇒ fgh»
        using fgh hcomp-def hcomp-trg-ide by auto
      show trg fgh ⋆ fgh = (i (trg fgh) ⋆ fgh) · a′ (trg fgh) (trg fgh) fgh
      proof −
        have (i (trg fgh) ⋆ fgh) · a′ (trg fgh) (trg fgh) fgh =
              (trg fgh ⋆ fgh) · a′ (trg fgh) (trg fgh) fgh
          using fgh i-def by metis
        also have ... = (trg fgh ⋆ fgh) · (trg fgh ⋆ trg fgh ⋆ fgh)
          using fgh a-def by fastforce
        also have ... = trg fgh ⋆ fgh
          using fgh hcomp-obj-self hcomp-assoc
          by (simp add: hcomp-trg-ide)
        finally show ?thesis by simp
      qed
    qed
    thus lunit fgh = fgh by simp
  qed
  show ⋀fgh. ide fgh ⟹ runit fgh = fgh
  proof −
    fix fgh
    assume fgh: ide fgh
    have fgh = runit fgh
    proof (intro runit-eqI)
      show ide fgh using fgh by simp
      show «fgh : fgh ⋆ src fgh ⇒ fgh»
        using fgh hcomp-def hcomp-ide-src by auto
      show fgh ⋆ src fgh = (fgh ⋆ i (src fgh)) · a fgh (src fgh) (src fgh)
      proof −
        have (fgh ⋆ i (src fgh)) · a fgh (src fgh) (src fgh) =
              (fgh ⋆ src fgh) · a fgh (src fgh) (src fgh)
          using fgh i-def by metis
        also have ... = (fgh ⋆ src fgh) · (fgh ⋆ src fgh ⋆ src fgh)
          using fgh a-def by fastforce
        also have ... = fgh ⋆ src fgh
          using fgh comp-arr-dom hcomp-obj-self by simp
        finally show ?thesis by simp
      qed
    qed
    thus runit fgh = fgh by simp
  qed
  show ⋀f g h. ⟦ ide f; ide g; ide h; src f = trg g; src g = trg h ⟧ ⟹ ide (a f g h)
    using a-def VV.arr-char_SbC VVV.arr-char_SbC by auto
qed

theorem is-strict-bicategory:
shows strict-bicategory vcomp hcomp a i src trg
  ..

lemma iso-char:
```

shows *iso* $\mu \longleftrightarrow$ *arr* $\mu \wedge$ *B.iso* (*Map* μ)
and *iso* $\mu \Longrightarrow$ *inv* μ = *MkArr* (*Cod* μ) (*Dom* μ) (*B.inv* (*Map* μ))
proof −
  have *1*: *iso* $\mu \Longrightarrow$ *arr* $\mu \wedge$ *B.iso* (*Map* μ)
  proof −
    assume μ: *iso* μ
    obtain ν **where** ν: *inverse-arrows* μ ν
      using μ **by** *auto*
    have *B.inverse-arrows* (*Map* μ) (*Map* ν)
    proof
      show *B.ide* (*Map* $\mu \cdot_B$ *Map* ν)
      proof −
        have *Map* $\mu \cdot_B$ *Map* ν = *Map* ($\mu \cdot \nu$)
          using μ ν *inverse-arrows-def Map-comp arr-char seq-char*
          by (*metis* (*no-types*, *lifting*) *ide-compE*)
        moreover have *B.ide* ...
          using ν *ide-char* **by** *blast*
        ultimately show *?thesis* **by** *simp*
      qed
      show *B.ide* (*Map* $\nu \cdot_B$ *Map* μ)
      proof −
        have *Map* $\nu \cdot_B$ *Map* μ = *Map* ($\nu \cdot \mu$)
          using μ ν *inverse-arrows-def comp-char* [*of* ν μ] **by** *simp*
        moreover have *B.ide* ...
          using ν *ide-char* **by** *blast*
        ultimately show *?thesis* **by** *simp*
      qed
    qed
    thus *arr* $\mu \wedge$ *B.iso* (*Map* μ)
      using μ **by** *auto*
  qed
  let *?*ν = *MkArr* (*Cod* μ) (*Dom* μ) (*B.inv* (*Map* μ))
  have *2*: *arr* $\mu \wedge$ *B.iso* (*Map* μ) $\Longrightarrow$ *iso* $\mu \wedge$ *inv* μ = *?*ν
  proof
    assume μ: *arr* $\mu \wedge$ *B.iso* (*Map* μ)
    have ν: «*?*ν : *cod* $\mu \Rightarrow$ *dom* μ»
      using μ *arr-char dom-char cod-char* **by** *auto*
    have *4*: *inverse-arrows* μ *?*ν
    proof
      show *ide* (*?*$\nu \cdot \mu$)
      proof −
        have *?*$\nu \cdot \mu$ = *dom* μ
          using μ ν *MkArr-Map comp-char seq-char B.comp-inv-arr′ dom-char* **by** *auto*
        thus *?thesis*
          using μ **by** *simp*
      qed
      show *ide* ($\mu \cdot$ *?*ν)
      proof −
        have $\mu \cdot$ *?*ν = *cod* μ

using μ ν *MkArr-Map comp-char seq-char B.comp-arr-inv′ cod-char* **by** *auto*
thus *?thesis*
using μ **by** *simp*
qed
qed
thus *iso* μ **by** *auto*
show *inv* μ = *?*ν
using *4 inverse-unique* **by** *simp*
qed
have *3*: *arr* $\mu \wedge$ *B.iso* (*Map* μ) $\Longrightarrow$ *iso* μ
using *2* **by** *simp*
show *iso* $\mu \longleftrightarrow$ *arr* $\mu \wedge$ *B.iso* (*Map* μ)
using *1 3* **by** *blast*
show *iso* $\mu \Longrightarrow$ *inv* μ = *?*ν
using *1 2* **by** *blast*
qed

1.11.3 The Strictness Theorem

The Strictness Theorem asserts: "Every bicategory is biequivalent to a strict bicategory." This amounts to an equivalent (and perhaps more desirable) formulation of the Coherence Theorem. In this section we prove the Strictness Theorem by constructing an equivalence pseudofunctor from a bicategory to its strictification.

We define a map *UP* from the given bicategory *B* to its strictification, and show that it is an equivalence pseudofunctor. The following auxiliary definition is not logically necessary, but it provides some terms that can be the targets of simplification rules and thereby enables some proofs to be done by simplification that otherwise could not be. Trying to eliminate it breaks some short proofs below, so I have kept it.

definition UP_0
where UP_0 $a \equiv$ *if B.obj a then MkIde* $\langle a \rangle_0$ *else null*

lemma *obj-*UP_0 [*simp*]:
assumes *B.obj a*
shows *obj* (UP_0 *a*)
unfolding *obj-def*
using *assms* UP_0*-def ide-MkIde* [*of* $\langle a \rangle_0$] *src-def* **by** *simp*

lemma UP_0*-in-hom* [*intro*]:
assumes *B.obj a*
shows «UP_0 *a* : UP_0 *a* $\rightarrow$ UP_0 *a*»
and «UP_0 *a* : UP_0 *a* $\Rightarrow$ UP_0 *a*»
using *assms obj-*UP_0 **by** *blast+*

lemma UP_0*-simps* [*simp*]:
assumes *B.obj a*
shows *ide* (UP_0 *a*) *arr* (UP_0 *a*)
and *src* (UP_0 *a*) = UP_0 *a* **and** *trg* (UP_0 *a*) = UP_0 *a*
and *dom* (UP_0 *a*) = UP_0 *a* **and** *cod* (UP_0 *a*) = UP_0 *a*

using *assms obj-UP*$_0$
apply *blast*
using *assms obj-UP*$_0$ *obj-simps*
by *simp-all*

definition *UP*
where *UP* $\mu \equiv$ *if B.arr* μ *then MkArr* $\langle$*B.dom* $\mu\rangle$ $\langle$*B.cod* $\mu\rangle$ μ *else null*

lemma *Dom-UP* [*simp*]:
assumes *B.arr* μ
shows *Dom* (*UP* μ) = $\langle$*B.dom* $\mu\rangle$
using *assms UP-def* **by** *fastforce*

lemma *Cod-UP* [*simp*]:
assumes *B.arr* μ
shows *Cod* (*UP* μ) = $\langle$*B.cod* $\mu\rangle$
using *assms UP-def* **by** *fastforce*

lemma *Map-UP* [*simp*]:
assumes *B.arr* μ
shows *Map* (*UP* μ) = μ
using *assms UP-def* **by** *fastforce*

lemma *arr-UP*:
assumes *B.arr* μ
shows *arr* (*UP* μ)
using *assms UP-def*
by (*intro arrI*, *fastforce*+)

lemma *UP-in-hom* [*intro*]:
assumes *B.arr* μ
shows «*UP* μ : *UP*$_0$ (*src*$_B$ μ) $\rightarrow$ *UP*$_0$ (*trg*$_B$ μ)»
and «*UP* μ : *UP* (*B.dom* μ) $\Rightarrow$ *UP* (*B.cod* μ)»
using *assms arr-UP UP-def UP*$_0$*-def dom-char cod-char src-def trg-def* **by** *auto*

lemma *UP-simps* [*simp*]:
assumes *B.arr* μ
shows *arr* (*UP* μ)
and *src* (*UP* μ) = *UP*$_0$ (*src*$_B$ μ) **and** *trg* (*UP* μ) = *UP*$_0$ (*trg*$_B$ μ)
and *dom* (*UP* μ) = *UP* (*B.dom* μ) **and** *cod* (*UP* μ) = *UP* (*B.cod* μ)
using *assms arr-UP UP-in-hom* **by** *auto*

interpretation *UP*: *functor* V_B *vcomp UP*
using *UP-def arr-UP UP-simps*(*4*−*5*) *arr-UP UP-def comp-char seq-char*
by *unfold-locales auto*

interpretation *UP*: *weak-arrow-of-homs* V_B *src*$_B$ *trg*$_B$ *vcomp src trg UP*
proof
fix μ

assume μ: $B.arr\ \mu$
show $UP\ (src_B\ \mu) \cong src\ (UP\ \mu)$
proof −
let $?\varphi = MkArr\ \langle src_B\ \mu\rangle\ \langle src_B\ \mu\rangle_0\ (src_B\ \mu)$
have φ: «$?\varphi : UP\ (src_B\ \mu) \Rightarrow src\ (UP\ \mu)$»
using μ *UP-def src-def arr-UP*
by (*intro MkArr-in-hom*) *auto*
have *iso* $?\varphi$
using μ φ *iso-char src-def* **by** *auto*
thus *?thesis*
using φ *isomorphic-def* **by** *auto*
qed
show $UP\ (trg_B\ \mu) \cong trg\ (UP\ \mu)$
proof −
let $?\varphi = MkArr\ \langle trg_B\ \mu\rangle\ \langle trg_B\ \mu\rangle_0\ (trg_B\ \mu)$
have φ: «$?\varphi : UP\ (trg_B\ \mu) \Rightarrow trg\ (UP\ \mu)$»
using μ *UP-def trg-def arr-UP*
by (*intro MkArr-in-hom*) *auto*
have *iso* $?\varphi$
using μ φ *iso-char trg-def* **by** *auto*
thus *?thesis*
using φ *isomorphic-def* **by** *auto*
qed
qed

interpretation *HoUP-UP*: *composite-functor B.VV.comp VV.comp vcomp*
UP.FF ‹$\lambda\mu\nu$. *hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› **..**
interpretation *UPoH*: *composite-functor B.VV.comp* V_B *vcomp*
‹$\lambda\mu\nu$. *fst* $\mu\nu \star_B$ *snd* $\mu\nu$› *UP* **..**

abbreviation Φ_o
where $\Phi_o\ fg \equiv MkArr\ (\langle fst\ fg\rangle \star \langle snd\ fg\rangle)\ \langle fst\ fg \star_B snd\ fg\rangle\ (fst\ fg \star_B snd\ fg)$

interpretation Φ: *transformation-by-components*
B.VV.comp vcomp HoUP-UP.map UPoH.map Φ_o
proof
fix *fg*
assume *fg*: *B.VV.ide fg*
show «$\Phi_o\ fg$: *HoUP-UP.map fg* $\Rightarrow$ *UPoH.map fg*»
proof (*intro in-homI*)
show *1*: *arr* ($\Phi_o\ fg$)
using *fg arr-char* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ **by** *auto*
show *dom* ($\Phi_o\ fg$) = *HoUP-UP.map fg*
using *1 fg UP.FF-def* $B.VV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ *dom-char hcomp-def B.can-Ide-self*
by *simp*
show *cod* ($\Phi_o\ fg$) = *UPoH.map fg*
using *fg arr-char cod-char* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *UP-def* **by** *auto*
qed

next
fix $\mu\nu$
assume $\mu\nu$: $B.VV.arr\ \mu\nu$
show $\Phi_o\ (B.VV.cod\ \mu\nu) \cdot HoUP\text{-}UP.map\ \mu\nu = UPoH.map\ \mu\nu \cdot \Phi_o\ (B.VV.dom\ \mu\nu)$
proof −
  have $\Phi_o\ (B.VV.cod\ \mu\nu) \cdot HoUP\text{-}UP.map\ \mu\nu =$
    $MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
      $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
      $(fst\ \mu\nu \star_B snd\ \mu\nu)$
  proof −
    have $\Phi_o\ (B.VV.cod\ \mu\nu) \cdot HoUP\text{-}UP.map\ \mu\nu =$
      $MkArr\ (\langle B.cod\ (fst\ \mu\nu)\rangle \star \langle B.cod\ (snd\ \mu\nu)\rangle)\ (\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
        $(B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)) \cdot$
      $MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu)\rangle \star \langle B.cod\ (snd\ \mu\nu)\rangle)$
        $(fst\ \mu\nu \star_B snd\ \mu\nu)$
      using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *arr-char UP.FF-def hcomp-def UP-def* $B.VV.cod\text{-}simp$
        src-def trg-def B.can-in-hom B.can-Ide-self B.comp-arr-dom B.comp-cod-arr
      by *auto*
    also have ... $= MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
        $((B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)) \cdot_B (fst\ \mu\nu \star_B snd\ \mu\nu))$
      using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$
      by (*intro comp-MkArr arr-MkArr*, *auto*)
    also have ... $= MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
        $(fst\ \mu\nu \star_B snd\ \mu\nu)$
      using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *B.comp-cod-arr* **by** *auto*
    finally show *?thesis* **by** *simp*
  qed
  also have ... $= UPoH.map\ \mu\nu \cdot \Phi_o\ (B.VV.dom\ \mu\nu)$
  proof −
    have $UPoH.map\ \mu\nu \cdot \Phi_o\ (B.VV.dom\ \mu\nu) =$
      $MkArr\ (\langle B.dom\ (fst\ \mu\nu) \star_B B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
        $(fst\ \mu\nu \star_B snd\ \mu\nu) \cdot$
      $MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.dom\ (fst\ \mu\nu) \star_B B.dom\ (snd\ \mu\nu)\rangle)$
        $(B.dom\ (fst\ \mu\nu) \star_B B.dom\ (snd\ \mu\nu))$
      using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *arr-char UP.FF-def hcomp-def UP-def* $B.VV.dom\text{-}simp$
        src-def trg-def B.can-in-hom B.can-Ide-self B.comp-arr-dom B.comp-cod-arr
      by *auto*
    also have ... $= MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$
        $((fst\ \mu\nu \star_B snd\ \mu\nu) \cdot_B (B.dom\ (fst\ \mu\nu) \star_B B.dom\ (snd\ \mu\nu)))$
      using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ *arr-MkArr*
      apply (*intro comp-MkArr arr-MkArr*) **by** *auto*
    also have ... $= MkArr\ (\langle B.dom\ (fst\ \mu\nu)\rangle \star \langle B.dom\ (snd\ \mu\nu)\rangle)$
        $(\langle B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu)\rangle)$

$(fst\ \mu\nu \star_B snd\ \mu\nu)$
using $\mu\nu$ $B.VV.arr\text{-}char_{SbC}$ $B.comp\text{-}arr\text{-}dom$ **by** *auto*
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
qed

abbreviation cmp_{UP}
where $cmp_{UP} \equiv \Phi.map$

lemma $cmp_{UP}\text{-}in\text{-}hom$ [*intro*]:
assumes $B.arr\ (fst\ \mu\nu)$ **and** $B.arr\ (snd\ \mu\nu)$ **and** $src_B\ (fst\ \mu\nu) = trg_B\ (snd\ \mu\nu)$
shows «$cmp_{UP}\ \mu\nu : UP_0\ (src_B\ (snd\ \mu\nu)) \rightarrow UP_0\ (trg_B\ (fst\ \mu\nu))$»
and «$cmp_{UP}\ \mu\nu : UP\ (B.dom\ (fst\ \mu\nu)) \star UP\ (B.dom\ (snd\ \mu\nu))$
$\Rightarrow UP\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))$»
proof −
let $?\mu = fst\ \mu\nu$ **and** $?\nu = snd\ \mu\nu$
show *1*: «$cmp_{UP}\ \mu\nu$:
$UP\ (B.dom\ ?\mu) \star UP\ (B.dom\ ?\nu) \Rightarrow UP\ (B.cod\ ?\mu \star_B B.cod\ ?\nu)$»
proof
show $arr\ (cmp_{UP}\ \mu\nu)$
using *assms* **by** *auto*
show $dom\ (cmp_{UP}\ \mu\nu) = UP\ (B.dom\ ?\mu) \star UP\ (B.dom\ ?\nu)$
proof −
have $B.VV.in\text{-}hom\ (?\mu,\ ?\nu)\ (B.dom\ ?\mu,\ B.dom\ ?\nu)\ (B.cod\ ?\mu,\ B.cod\ ?\nu)$
using *assms* $B.VV.in\text{-}hom\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ **by** *auto*
hence $dom\ (cmp_{UP}\ \mu\nu) = HoUP\text{-}UP.map\ (B.dom\ ?\mu,\ B.dom\ ?\nu)$
by *auto*
also have ... $= UP\ (B.dom\ ?\mu) \star UP\ (B.dom\ ?\nu)$
using *assms* $UP.FF\text{-}def$ **by** *fastforce*
finally show *?thesis* **by** *simp*
qed
show $cod\ (cmp_{UP}\ \mu\nu) = UP\ (B.cod\ ?\mu \star_B B.cod\ ?\nu)$
using *assms* $B.VV.in\text{-}hom\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.cod\text{-}simp$ **by** *auto*
qed
show «$cmp_{UP}\ \mu\nu : UP_0\ (src_B\ ?\nu) \rightarrow UP_0\ (trg_B\ ?\mu)$»
using *assms 1 src-dom* [*of* $cmp_{UP}\ \mu\nu$] *trg-dom* [*of* $cmp_{UP}\ \mu\nu$] **by** *fastforce*
qed

lemma $cmp_{UP}\text{-}simps$ [*simp*]:
assumes $B.arr\ (fst\ \mu\nu)$ **and** $B.arr\ (snd\ \mu\nu)$ **and** $src_B\ (fst\ \mu\nu) = trg_B\ (snd\ \mu\nu)$
shows $arr\ (cmp_{UP}\ \mu\nu)$
and $src\ (cmp_{UP}\ \mu\nu) = UP_0\ (src_B\ (snd\ \mu\nu))$ **and** $trg\ (cmp_{UP}\ \mu\nu) = UP_0\ (trg_B\ (fst\ \mu\nu))$
and $dom\ (cmp_{UP}\ \mu\nu) = UP\ (B.dom\ (fst\ \mu\nu)) \star UP\ (B.dom\ (snd\ \mu\nu))$
and $cod\ (cmp_{UP}\ \mu\nu) = UP\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))$
using *assms* $cmp_{UP}\text{-}in\text{-}hom$ [*of* $\mu\nu$] **by** *auto*

lemma $cmp_{UP}\text{-}ide\text{-}simps$ [*simp*]:

assumes $B.ide\ (fst\ fg)$ **and** $B.ide\ (snd\ fg)$ **and** $src_B\ (fst\ fg) = trg_B\ (snd\ fg)$
shows $Dom\ (cmp_{UP}\ fg) = \langle fst\ fg\rangle \star \langle snd\ fg\rangle$
and $Cod\ (cmp_{UP}\ fg) = \langle fst\ fg \star_B snd\ fg\rangle$
and $Map\ (cmp_{UP}\ fg) = fst\ fg \star_B snd\ fg$
  using $assms\ B.VV.ide\text{-}char_{SbC}\ B.VV.arr\text{-}char_{SbC}$ **by** *auto*

interpretation Φ: *natural-isomorphism*
$B.VV.comp\ vcomp\ HoUP\text{-}UP.map\ UPoH.map\ cmp_{UP}$
proof
  fix *fg*
  assume *fg*: $B.VV.ide\ fg$
  have $arr\ (cmp_{UP}\ fg)$
    using $fg\ \Phi.preserves\text{-}reflects\text{-}arr\ [of\ fg]$ **by** *simp*
  thus $iso\ (cmp_{UP}\ fg)$
    using $fg\ iso\text{-}char$ **by** *simp*
qed

lemma $cmp_{UP}\text{-}ide\text{-}simp$:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $cmp_{UP}\ (f,\ g) = MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g)$
  using $assms\ B.VV.ide\text{-}char_{SbC}\ B.VV.arr\text{-}char_{SbC}$ **by** *simp*

lemma $cmp_{UP}'\text{-}ide\text{-}simp$:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $inv\ (cmp_{UP}\ (f,\ g)) = MkArr\ \langle f \star_B g\rangle\ (\langle f\rangle \star \langle g\rangle)\ (f \star_B g)$
  using $assms\ cmp_{UP}\text{-}ide\text{-}simp\ iso\text{-}char\ \Phi.components\text{-}are\text{-}iso\ [of\ (f,\ g)]$
    $B.VV.ide\text{-}char_{SbC}\ B.VV.arr\text{-}char_{SbC}$
  by *simp*

interpretation *UP*: *pseudofunctor*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ vcomp\ hcomp\ \mathrm{a}\ \mathrm{i}\ src\ trg\ UP\ cmp_{UP}$
proof
  fix $f\ g\ h$
  assume *f*: $B.ide\ f$ **and** *g*: $B.ide\ g$ **and** *h*: $B.ide\ h$
  and *fg*: $src_B\ f = trg_B\ g$ **and** *gh*: $src_B\ g = trg_B\ h$
  show $UP\ \mathrm{a}_B[f,\ g,\ h] \cdot cmp_{UP}\ (f \star_B g,\ h) \cdot (cmp_{UP}\ (f,\ g) \star UP\ h) =$
    $cmp_{UP}\ (f,\ g \star_B h) \cdot (UP\ f \star cmp_{UP}\ (g,\ h)) \cdot \mathrm{a}\ (UP\ f)\ (UP\ g)\ (UP\ h)$
  proof −
    have $UP\ \mathrm{a}_B[f,\ g,\ h] \cdot cmp_{UP}\ (f \star_B g,\ h) \cdot (cmp_{UP}\ (f,\ g) \star UP\ h) =$
      $MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h)$
    proof −
      have *1*: $UP.hseq_D\ (MkIde\ \langle f\rangle)\ (MkIde\ \langle g\rangle)$
        using $f\ g\ fg\ hseq\text{-}char\ src\text{-}def\ trg\text{-}def\ arr\text{-}char$ **by** *auto*
      have *2*: $UP.hseq_D\ (MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g) \cdot MkIde\ (\langle f\rangle \star \langle g\rangle))$
        $(MkIde\ \langle h\rangle)$
      proof −
        have $MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g) \cdot MkIde\ (\langle f\rangle \star \langle g\rangle) =$
          $MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g)$
        proof −

have $MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g) \cdot MkIde\ (\langle f\rangle \star \langle g\rangle) =$
$MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g) \cdot MkArr\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle)\ (f \star_B g)$
using *f g fg* **by** *simp*
also have ... $= MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ ((f \star_B g) \cdot_B (f \star_B g))$
using *f g fg* **by** (*intro comp-MkArr arr-MkArr*, *auto*)
also have ... $= MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g)$
using *f g fg* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *f g h fg gh arr-char src-def trg-def* **by** *auto*
qed
have $UP\ \mathrm{a}_B[f, g, h] = MkArr\ \langle (f \star_B g) \star_B h\rangle\ \langle f \star_B g \star_B h\rangle\ \mathrm{a}_B[f, g, h]$
using *f g h fg gh UP-def B.HoHV-def B.HoVH-def* $B.VVV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VVV.dom\text{-}char_{SbC}$ $B.VVV.cod\text{-}char_{SbC}$
by *simp*
moreover have
$cmp_{UP}\ (f \star_B g, h) = MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ \langle (f \star_B g) \star_B h\rangle\ ((f \star_B g) \star_B h) \cdot$
$MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ ((f \star_B g) \star_B h)$
using *f g h fg gh* Φ*.map-simp-ide* Φ*.map-def* $B.VV.arr\text{-}char_{SbC}$ *UP.FF-def* $B.VV.cod\text{-}simp$ *hcomp-def B.can-Ide-self*
by *simp*
moreover have $cmp_{UP}\ (f, g) \star UP\ h =$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (B.inv\ \mathrm{a}_B[f, g, h])$
proof −
have $MkArr\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle)$
$(B.can\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle) \cdot_B (f \star_B g) \cdot_B B.can\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle)) =$
$MkArr\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle)\ (f \star_B g)$
using *f g fg B.can-Ide-self B.comp-arr-dom B.comp-cod-arr* **by** *simp*
moreover have $MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g) \cdot$
$MkArr\ (\langle f\rangle \star \langle g\rangle)\ (\langle f\rangle \star \langle g\rangle)\ (f \star_B g) =$
$MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g)$
by (*metis* (*no-types*, *lifting*) *2 B.ideD*(*1*) *E.eval.simps*(*2*−*3*) *cod-MkArr comp-arr-ide f g ide-char′ seq-char*)
moreover have $B.can\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle)\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle) = B.inv\ \mathrm{a}_B[f, g, h]$
using *f g h fg gh B.canI-associator-0 B.inverse-arrows-can* **by** *simp*
ultimately show *?thesis*
using *1 2 f g h fg gh* Φ*.map-def UP-def hcomp-def UP.FF-def* $B.VV.arr\text{-}char_{SbC}$ *B.can-Ide-self B.comp-cod-arr* $B.VV.cod\text{-}simp$
by *simp*
qed
ultimately have $UP\ \mathrm{a}_B[f, g, h] \cdot cmp_{UP}\ (f \star_B g, h) \cdot (cmp_{UP}\ (f, g) \star UP\ h) =$
$MkArr\ \langle (f \star_B g) \star_B h\rangle\ \langle f \star_B g \star_B h\rangle\ \mathrm{a}_B[f, g, h] \cdot$
$MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ \langle (f \star_B g) \star_B h\rangle\ ((f \star_B g) \star_B h) \cdot$
$MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ ((f \star_B g) \star_B h) \cdot$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (B.inv\ \mathrm{a}_B[f, g, h])$
using *comp-assoc* **by** *presburger*
also have ... $= MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ \langle f \star_B g \star_B h\rangle$
$(\mathrm{a}_B[f, g, h] \cdot_B ((f \star_B g) \star_B h) \cdot_B ((f \star_B g) \star_B h) \cdot_B$

$B.inv\ \mathrm{a}_B[f,\ g,\ h])$

proof −

have $Arr\ (MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (B.inv\ \mathrm{a}_B[f,\ g,\ h])) \wedge$
$Arr\ (MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)\ ((f \star_B g) \star_B h)) \wedge$
$Arr\ (MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f \star_B g\rangle \star \langle h\rangle)$
$(((f \star_B g) \star_B h) \cdot_B B.inv\ \mathrm{a}_B[f,\ g,\ h])) \wedge$
$Arr\ (MkArr\ (\langle f \star_B g\rangle \star \langle h\rangle)\ \langle (f \star_B g) \star_B h\rangle\ ((f \star_B g) \star_B h)) \wedge$
$Arr\ (MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ \langle (f \star_B g) \star_B h\rangle$
$(((f \star_B g) \star_B h) \cdot_B ((f \star_B g) \star_B h) \cdot_B B.inv\ \mathrm{a}_B[f,\ g,\ h])) \wedge$
$Arr\ (MkArr\ \langle (f \star_B g) \star_B h\rangle\ \langle f \star_B g \star_B h\rangle\ \mathrm{a}_B[f,\ g,\ h])$

using *f g h fg gh B.α.preserves-hom B.HoHV-def B.HoVH-def* **by** *auto*

thus *?thesis*

using *f g h fg gh comp-def B.comp-arr-dom B.comp-cod-arr* **by** *simp*

qed

also have ... $= MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h)$

using *B.comp-cod-arr B.comp-arr-inv′*

by (*auto simp add*: *f fg g gh h*)

finally show *?thesis* **by** *simp*

qed

also have ... $= cmp_{UP}\ (f,\ g \star_B h) \cdot (UP\ f \star cmp_{UP}\ (g,\ h)) \cdot \mathrm{a}\ (UP\ f)\ (UP\ g)\ (UP\ h)$

proof −

have $cmp_{UP}\ (f,\ g \star_B h) \cdot (UP\ f \star cmp_{UP}\ (g,\ h)) \cdot \mathrm{a}\ (UP\ f)\ (UP\ g)\ (UP\ h) =$
$cmp_{UP}\ (f,\ g \star_B h) \cdot (MkIde\ \langle f\rangle \star cmp_{UP}\ (g,\ h)) \cdot$
$(MkIde\ \langle f\rangle \star MkIde\ \langle g\rangle \star MkIde\ \langle h\rangle)$

using *f g h fg gh VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char src-def trg-def UP-def* a-*def*

by *auto*

also have ... $= MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h) \cdot$
$MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h) \cdot$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h) \cdot$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (f \star_B g \star_B h)$

proof −

have $cmp_{UP}\ (f,\ g \star_B h) = MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h) \cdot$
$MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h)$

using *f g h fg gh Φ.map-simp-ide Φ.map-def UP.FF-def UP-def hcomp-def*
B.VV.arr-char$_{SbC}$ *B.can-Ide-self B.comp-arr-dom B.comp-cod-arr src-def trg-def*
arr-char B.VV.cod-simp

by *auto*

moreover

have $cmp_{UP}\ (g,\ h) = MkArr\ (\langle g\rangle \star \langle h\rangle)\ \langle g \star_B h\rangle\ (g \star_B h)$

using *g h gh cmp*$_{UP}$*-ide-simp* **by** *blast*

moreover have $MkIde\ \langle f\rangle \star MkArr\ (\langle g\rangle \star \langle h\rangle)\ \langle g \star_B h\rangle\ (g \star_B h) =$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h)$

using *f g h fg gh hcomp-def arr-char src-def trg-def B.can-Ide-self*
B.comp-arr-dom B.comp-cod-arr

by *auto*

moreover

have $MkIde\ \langle f\rangle \star MkIde\ \langle g\rangle \star MkIde\ \langle h\rangle =$
$MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (f \star_B g \star_B h)$

proof −
have «$f : f \Rightarrow_B f$» ∧ «$g : g \Rightarrow_B g$» ∧ «$h : h \Rightarrow_B h$»
using *f g h* **by** *auto*
thus *?thesis*
using *f g h fg gh hcomp-def arr-char src-def trg-def B.can-Ide-self B.comp-arr-dom B.comp-cod-arr*
by *auto*
qed
ultimately show *?thesis*
using *comp-assoc* **by** *auto*
qed
also have ... = $MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h)$
proof −
have $Arr\ (MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (f \star_B g \star_B h)) \wedge$
$Arr\ (MkArr\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h)) \wedge$
$Arr\ (MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (\langle f\rangle \star \langle g \star_B h\rangle)\ (f \star_B g \star_B h)) \wedge$
$Arr\ (MkArr\ (\langle f\rangle \star \langle g \star_B h\rangle)\ \langle f \star_B g \star_B h\rangle\ (f \star_B g \star_B h))$
using *f g h fg gh* **by** *auto*
thus *?thesis*
using *f g h fg gh comp-def* **by** *auto*
qed
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
qed

lemma *UP-is-pseudofunctor*:
shows *pseudofunctor* V_B H_B $\mathtt{a}_B$ $\mathtt{i}_B$ src_B trg_B *vcomp hcomp* a i *src trg UP* cmp_{UP} **..**

lemma *UP-*map_0*-obj* [*simp*]:
assumes *B.obj a*
shows $UP.map_0\ a = UP_0\ a$
using *assms UP.*map_0*-def* **by** *auto*

interpretation *UP*: *full-functor* V_B *vcomp UP*
proof
fix μ *f g*
assume *f*: *B.ide f* **and** *g*: *B.ide g*
assume μ: «$\mu : UP\ f \Rightarrow UP\ g$»
show $\exists \nu.$ «$\nu : f \Rightarrow_B g$» ∧ $UP\ \nu = \mu$
proof −
have *1*: «$Map\ \mu : f \Rightarrow_B g$»
using *f g* μ *UP-def arr-char in-hom-char* **by** *auto*
moreover have $UP\ (Map\ \mu) = \mu$
proof −
have $\mu = MkArr\ (Dom\ \mu)\ (Cod\ \mu)\ (Map\ \mu)$
using μ *MkArr-Map* **by** *auto*
also have ... = $UP\ (Map\ \mu)$

```
      using 1 B.arrI UP.as-nat-trans.preserves-hom UP-def μ in-hom-char by force
    finally show ?thesis by auto
  qed
  ultimately show ?thesis by blast
 qed
qed

interpretation UP: faithful-functor V_B vcomp UP
 using arr-char UP-def
 by (unfold-locales, simp-all)

interpretation UP: fully-faithful-functor V_B vcomp UP ..

lemma UP-is-fully-faithful-functor:
shows fully-faithful-functor V_B vcomp UP
 ..

no-notation B.in-hom (‹«- : - →_B -»›)

lemma Map-reflects-hhom:
assumes B.obj a and B.obj b and ide g
and «g : UP.map_0 a → UP.map_0 b»
shows «Map g : a →_B b»
proof
 have 1: B.ide (Map g)
  using assms ide-char by blast
 show B.arr (Map g)
  using 1 by simp
 show src_B (Map g) = a
 proof −
  have src_B (Map g) = Map (src g)
   using assms src-def apply simp
   by (metis (no-types, lifting) E.eval-simps(2) E.Ide-implies-Arr arr-char ideE)
  also have ... = Map (UP.map_0 a)
   using assms by (metis (no-types, lifting) in-hhomE)
  also have ... = a
   using assms UP.map_0-def UP-def [of a] src-def by auto
  finally show ?thesis by simp
 qed
 show trg_B (Map g) = b
 proof −
  have trg_B (Map g) = Map (trg g)
   using assms trg-def apply simp
   by (metis (no-types, lifting) E.eval-simps(3) E.Ide-implies-Arr arr-char ideE)
  also have ... = Map (UP.map_0 b)
   using assms by (metis (no-types, lifting) in-hhomE)
  also have ... = b
   using assms UP.map_0-def UP-def [of b] src-def by auto
  finally show ?thesis by simp
```

qed
qed

lemma *eval-Dom-ide* [*simp*]:
assumes *ide g*
shows $\{\!|Dom\ g|\!\} = Map\ g$
using *assms dom-char ideD* **by** *auto*

lemma *Cod-ide*:
assumes *ide f*
shows $Cod\ f = Dom\ f$
using *assms dom-char* **by** *auto*

lemma *Map-preserves-objects*:
assumes *obj a*
shows *B.obj* (*Map a*)
proof −
have src_B (*Map a*) = *Map* (*src a*)
using *assms src-def* **apply** *simp*
using *E.eval-simps*(*2*) *E.Ide-implies-Arr arr-char ideE*
by (*metis* (*no-types*, *lifting*) *objE*)
also have *1*: ... = $\{\!|E.Src\ (Dom\ a)|\!\}$
using *assms src-def* **by** *auto*
also have ... = $\{\!|\langle Map\ a\rangle_0|\!\}$
using *assms B.src.extensionality 1 obj-simps*(*2*) **by** *force*
also have ... = *Map a*
using *assms* **by** *auto*
finally have src_B (*Map a*) = *Map a* **by** *simp*
moreover have *B.arr* (*Map a*)
using *assms B.ideD arr-char* **by** *auto*
ultimately show *?thesis*
using *B.obj-def* **by** *simp*
qed

interpretation *UP*: *equivalence-pseudofunctor*
V_B H_B a_B i_B src_B trg_B *vcomp hcomp* a i *src trg UP* cmp_{UP}
proof

show $\bigwedge f\, f'\, \nu.\ [\![\ B.ide\ f;\ B.ide\ f';\ src_B\ f = src_B\ f';\ trg_B\ f = trg_B\ f';$
$«\nu : UP\ f \Rightarrow UP\ f'»\ ]\!] \Longrightarrow \exists\, \mu.\ «\mu : f \Rightarrow_B f'» \wedge UP\ \mu = \nu$
using *UP.is-full* **by** *simp*

show $\bigwedge b.\ obj\ b \Longrightarrow \exists\, a.\ B.obj\ a \wedge equivalent\text{-}objects\ (UP.map_0\ a)\ b$
proof −
fix *b*
assume *b*: *obj b*
have *1*: *B.obj* (*Map b*)
using *b Map-preserves-objects* **by** *simp*
have *3*: $UP.map_0$ (*Map b*) = *MkArr* $\langle Map\ b\rangle_0$ $\langle Map\ b\rangle_0$ (*Map b*)

```
    using b 1 UP.map₀-def [of Map b] UP-def src-def arr-char by auto
  have 4: b = MkArr (Dom b) (Dom b) (Map b)
    using b objE eval-Dom-ide
    by (metis (no-types, lifting) dom-char ideD(2) obj-def)
  let ?φ = MkArr ⟨Map b⟩₀ (Dom b) (Map b)
  have φ: arr ?φ
  proof −
    have 2: E.Obj (Dom b)
      using b obj-char by blast
    have E.Nml ⟨Map b⟩₀ ∧ E.Ide ⟨Map b⟩₀
      using 1 by auto
    moreover have E.Nml (Dom b) ∧ E.Ide (Dom b)
      using b arr-char [of b] by auto
    moreover have E.Src ⟨Map b⟩₀ = E.Src (Dom b)
      using b 1 2
      apply (cases Dom b)
            apply simp-all
      by fastforce
    moreover have E.Trg ⟨Map b⟩₀ = E.Trg (Dom b)
      using b 1 2
      apply (cases Dom b)
            apply simp-all
      by fastforce
    moreover have «Map b : ⦃⟨Map b⟩₀⦄ ⇒_B ⦃Dom b⦄»
      using b 1 by (elim objE, auto)
    ultimately show ?thesis
      using arr-char ‹E.Nml ⟨Map b⟩₀ ∧ E.Ide ⟨Map b⟩₀› by auto
  qed
  hence iso ?φ
    using 1 iso-char by auto
  moreover have dom ?φ = UP.map₀ (Map b)
    using φ dom-char b 1 3 B.objE UP.map₀-def UP-def src-def by auto
  moreover have cod ?φ = b
    using φ cod-char b 4 1 by auto
  ultimately have isomorphic (UP.map₀ (Map b)) b
    using φ 3 4 isomorphic-def by blast
  moreover have 5: obj (UP.map₀ (Map b))
    using 1 UP.map₀-simps(2) by simp
  ultimately have 6: UP.map₀ (Map b) = b
    using b isomorphic-objects-are-equal by simp
  have equivalent-objects (UP.map₀ (Map b)) b
    using b 6 equivalent-objects-reflexive [of b] by simp
  thus ∃a. B.obj a ∧ equivalent-objects (UP.map₀ a) b
    using 1 6 by auto
qed

show ⋀a b g. ⟦ B.obj a; B.obj b; «g : UP.map₀ a → UP.map₀ b»; ide g ⟧ ⟹
               ∃f. «f : a →_B b» ∧ B.ide f ∧ isomorphic (UP f) g
proof −
```

```
fix a b g
assume a: B.obj a and b: B.obj b
assume g-in-hhom: «g : UP.map₀ a → UP.map₀ b»
assume ide-g: ide g
have 1: B.ide (Map g)
  using ide-g ide-char by blast
have arr (UP a)
  using a by auto
have arr (UP b)
  using b by auto
have Map-g-eq: Map g = ⦃Dom g⦄
  using ide-g by simp
have Map-g-in-hhom: «Map g : a →_B b»
  using a b ide-g g-in-hhom Map-reflects-hhom by simp

let ?φ = MkArr ⟨Map g⟩ (Dom g) (Map g)
have φ: arr ?φ
proof −
  have «Map ?φ : ⦃Dom ?φ⦄ ⇒_B ⦃Cod ?φ⦄»
    using 1 Map-g-eq by auto
  moreover have E.Ide ⟨Map g⟩ ∧ E.Nml ⟨Map g⟩
    using 1 by simp
  moreover have E.Ide (Dom g) ∧ E.Nml (Dom g)
    using ide-g arr-char ide-char by blast
  moreover have E.Src ⟨Map g⟩ = E.Src (Dom g)
    using ide-g g-in-hhom src-def Map-g-in-hhom
    by (metis (no-types, lifting) B.ideD(2) B.in-hhom-def B.objE B.obj-def′
        Dom.simps(1) E.Src.simps(2) UP.map₀-def ‹arr (UP a)› a in-hhomE UP-def)
  moreover have E.Trg ⟨Map g⟩ = E.Trg (Dom g)
  proof −
    have E.Trg ⟨Map g⟩ = ⟨b⟩₀
      using Map-g-in-hhom by auto
    also have ... = E.Trg (Dom g)
    proof −
      have E.Trg (Dom g) = Dom (trg g)
        using ide-g trg-def by simp
      also have ... = Dom (UP.map₀ b)
        using g-in-hhom by auto
      also have ... = ⟨b⟩₀
        using b ‹arr (UP b)› UP.map₀-def src-def UP-def B.objE by auto
      finally show ?thesis by simp
    qed
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using arr-char by simp
qed
have «?φ : UP (Map g) ⇒ g»
  by (simp add: 1 φ ide-g in-hom-char)
```

moreover have *iso* $?\varphi$
using φ *1 iso-char* **by** *simp*
ultimately have *isomorphic* (*UP* (*Map g*)) *g*
using *isomorphic-def* **by** *auto*
thus $\exists f.$ «$f : a \rightarrow_B b$» $\wedge$ *B.ide f* $\wedge$ *isomorphic* (*UP f*) *g*
using *1 Map-g-in-hhom* **by** *auto*
qed
qed

theorem *UP-is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* V_B H_B a_B i_B src_B trg_B *vcomp hcomp* a i *src trg*
UP cmp_{UP}
..

Next, we work out the details of the equivalence pseudofunctor *DN* in the converse direction.

definition *DN*
where *DN* $\mu \equiv$ *if arr* μ *then Map* μ *else B.null*

lemma *DN-in-hom* [*intro*]:
assumes *arr* μ
shows «*DN* μ : *DN* (*src* μ) $\rightarrow_B$ *DN* (*trg* μ)»
and «*DN* μ : *DN* (*dom* μ) $\Rightarrow_B$ *DN* (*cod* μ)»
using *assms DN-def arr-char* [*of* μ] *B.vconn-implies-hpar*(*1*−*2*) *E.eval-in-hom*(*1*)
B.in-hhom-def
by *auto*

lemma *DN-simps* [*simp*]:
assumes *arr* μ
shows *B.arr* (*DN* μ)
and src_B (*DN* μ) = *DN* (*src* μ) **and** trg_B (*DN* μ) = *DN* (*trg* μ)
and *B.dom* (*DN* μ) = *DN* (*dom* μ) **and** *B.cod* (*DN* μ) = *DN* (*cod* μ)
using *assms DN-in-hom* **by** *auto*

interpretation *functor vcomp* V_B *DN*
using *DN-def seqE Map-comp seq-char*
by *unfold-locales auto*

interpretation *DN*: *weak-arrow-of-homs vcomp src trg* V_B src_B trg_B *DN*
proof
fix μ
assume μ: *arr* μ
show *B.isomorphic* (*DN* (*src* μ)) (src_B (*DN* μ))
proof −
have *DN* (*src* μ) = src_B (*DN* μ)
using *B.src.extensionality DN-def DN-simps*(*2*) **by** *auto*
moreover have *B.ide* (*DN* (*src* μ))
using μ **by** *simp*
ultimately show *?thesis*

```
      using μ B.isomorphic-reflexive by auto
  qed
  show B.isomorphic (DN (trg μ)) (trg_B (DN μ))
  proof −
    have DN (trg μ) = trg_B (DN μ)
      using ‹B.isomorphic (DN (src μ)) (src_B (DN μ))› by fastforce
    moreover have B.ide (DN (trg μ))
      using μ by simp
    ultimately show ?thesis
      using B.isomorphic-reflexive by auto
  qed
qed

interpretation functor VV.comp B.VV.comp DN.FF
  using DN.functor-FF by auto
interpretation HoDN-DN: composite-functor VV.comp B.VV.comp V_B
                DN.FF ‹λμν. H_B (fst μν) (snd μν)› ..
interpretation DNoH: composite-functor VV.comp vcomp V_B
                ‹λμν. fst μν ⋆ snd μν› DN ..

abbreviation Ψ_o
where Ψ_o fg ≡ B.can (Dom (fst fg) ⌊⋆⌋ Dom (snd fg)) (Dom (fst fg) ⋆ Dom (snd fg))

abbreviation Ψ_o'
where Ψ_o' fg ≡ B.can (Dom (fst fg) ⋆ Dom (snd fg)) (Dom (fst fg) ⌊⋆⌋ Dom (snd fg))

lemma Ψ_o-in-hom:
assumes VV.ide fg
shows «Ψ_o fg : Map (fst fg) ⋆_B Map (snd fg) ⇒_B ⦃Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⦄»
and «Ψ_o' fg : ⦃Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⦄ ⇒_B Map (fst fg) ⋆_B Map (snd fg)»
and B.inverse-arrows (Ψ_o fg) (Ψ_o' fg)
proof −
  have 1: E.Ide (Dom (fst fg) ⋆ Dom (snd fg))
    unfolding E.Ide.simps(3)
    apply (intro conjI)
    using assms VV.ide-char VV.arr-char_SbC arr-char
      apply simp
    using VV.arr-char_SbC VV.ideD(1) assms
     apply blast
    by (metis (no-types, lifting) VV.arrE VV.ideD(1) assms src-simps(1) trg-simps(1))
  have 2: E.Ide (Dom (fst fg) ⌊⋆⌋ Dom (snd fg))
    using 1
  by (meson E.Ide.simps(3) E.Ide-HcompNml VV.arr-char_SbC VV.ideD(1) arr-char assms)
  have 3: ⌊Dom (fst fg) ⋆ Dom (snd fg)⌋ = ⌊Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⌋
    by (metis (no-types, lifting) E.Ide.simps(3) E.Nml-HcompNml(1) E.Nmlize.simps(3)
        E.Nmlize-Nml VV.arr-char_SbC VV.ideD(1) arr-char assms 1)
  have 4: ⦃Dom (fst fg) ⋆ Dom (snd fg)⦄ = Map (fst fg) ⋆_B Map (snd fg)
    using assms VV.ide-char_SbC VV.arr-char_SbC arr-char by simp
  show «Ψ_o fg : Map (fst fg) ⋆_B Map (snd fg) ⇒_B ⦃Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⦄»
```

```
    using 1 2 3 4 by auto
  show «Ψo' fg : ⦃Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⦄ ⇒B Map (fst fg) ⋆B Map (snd fg)»
    using 1 2 3 4 by auto
  show B.inverse-arrows (Ψo fg) (Ψo' fg)
    using 1 2 3 B.inverse-arrows-can by blast
qed

interpretation Ψ: transformation-by-components
                  VV.comp VB HoDN-DN.map DNoH.map Ψo
proof
  fix fg
  assume fg: VV.ide fg
  have 1: ⦃Dom (fst fg) ⋆ Dom (snd fg)⦄ = Map (fst fg) ⋆B Map (snd fg)
    using fg VV.ide-charSbC VV.arr-charSbC arr-char by simp
  show «Ψo fg : HoDN-DN.map fg ⇒B DNoH.map fg»
  proof
    show B.arr (Ψo fg)
      using fg Ψo-in-hom by blast
    show B.dom (Ψo fg) = HoDN-DN.map fg
    proof −
      have B.dom (Ψo fg) = Map (fst fg) ⋆B Map (snd fg)
        using fg Ψo-in-hom by blast
      also have ... = HoDN-DN.map fg
        using fg DN.FF-def DN-def VV.arr-charSbC src-def trg-def VV.ide-charSbC by auto
      finally show ?thesis by simp
    qed
    show B.cod (Ψo fg) = DNoH.map fg
    proof −
      have B.cod (Ψo fg) = ⦃Dom (fst fg) ⌊⋆⌋ Dom (snd fg)⦄
        using fg Ψo-in-hom by blast
      also have ... = DNoH.map fg
      proof −
        have DNoH.map fg =
              B.can (Cod (fst fg) ⌊⋆⌋ Cod (snd fg)) (Cod (fst fg) ⋆ Cod (snd fg)) ·B
                (Map (fst fg) ⋆B Map (snd fg)) ·B
                  B.can (Dom (fst fg) ⋆ Dom (snd fg)) (Dom (fst fg) ⌊⋆⌋ Dom (snd fg))
          using fg DN-def Map-hcomp VV.arr-charSbC
          apply simp
          using VV.ideD(1) by blast
        also have ... =
                  B.can (Cod (fst fg) ⌊⋆⌋ Cod (snd fg)) (Cod (fst fg) ⋆ Cod (snd fg)) ·B
                    B.can (Dom (fst fg) ⋆ Dom (snd fg)) (Dom (fst fg) ⌊⋆⌋ Dom (snd fg))
        proof −
          have (Map (fst fg) ⋆B Map (snd fg)) ·B
                  B.can (Dom (fst fg) ⋆ Dom (snd fg)) (Dom (fst fg) ⌊⋆⌋ Dom (snd fg)) =
                B.can (Dom (fst fg) ⋆ Dom (snd fg)) (Dom (fst fg) ⌊⋆⌋ Dom (snd fg))
            using fg 1 Ψo-in-hom B.comp-cod-arr by blast
          thus ?thesis by simp
        qed
```

also have ... = ⦃*Dom* (*fst fg*) ⌊⋆⌋ *Dom* (*snd fg*)⦄
proof −
have *B.can* (*Cod* (*fst fg*) ⌊⋆⌋ *Cod* (*snd fg*)) (*Cod* (*fst fg*) ⋆ *Cod* (*snd fg*)) = Ψ_o *fg*
using *fg* $VV.ide\text{-}char_{SbC}$ *Cod-ide* **by** *simp*
thus *?thesis*
using *fg 1* Ψ_o*-in-hom* [*of fg*] *B.comp-arr-inv′* **by** *fastforce*
qed
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
qed
next
show ⋀*f*. *VV.arr f* ⟹
Ψ_o (*VV.cod f*) $\cdot_B$ *HoDN-DN.map f* = *DNoH.map f* $\cdot_B$ Ψ_o (*VV.dom f*)
proof −
fix $\mu\nu$
assume $\mu\nu$: *VV.arr* $\mu\nu$
show Ψ_o (*VV.cod* $\mu\nu$) $\cdot_B$ *HoDN-DN.map* $\mu\nu$ = *DNoH.map* $\mu\nu$ $\cdot_B$ Ψ_o (*VV.dom* $\mu\nu$)
proof −
have *1*: *E.Ide* (*Dom* (*fst* $\mu\nu$) ⋆ *Dom* (*snd* $\mu\nu$))
unfolding *E.Ide.simps*(*3*)
by (*metis* (*no-types*, *lifting*) *VV.arrE* $\mu\nu$ *arrE src-simps*(*2*) *trg-simps*(*2*))
have *2*: *E.Ide* (*Dom* (*fst* $\mu\nu$) ⌊⋆⌋ *Dom* (*snd* $\mu\nu$))
using *1*
by (*meson E.Ide.simps*(*3*) *E.Ide-HcompNml* $VV.arr\text{-}char_{SbC}$ *VV.ideD*(*1*) *arr-char* $\mu\nu$)
have *3*: ⌊*Dom* (*fst* $\mu\nu$) ⋆ *Dom* (*snd* $\mu\nu$)⌋ = ⌊*Dom* (*fst* $\mu\nu$) ⌊⋆⌋ *Dom* (*snd* $\mu\nu$)⌋
by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *E.Nml-HcompNml*(*1*) *E.Nmlize.simps*(*3*) *E.Nmlize-Nml* $VV.arr\text{-}char_{SbC}$ *arr-char* $\mu\nu$ *1*)
have *4*: *E.Ide* (*Cod* (*fst* $\mu\nu$) ⋆ *Cod* (*snd* $\mu\nu$))
unfolding *E.Ide.simps*(*3*)
by (*metis* (*no-types*, *lifting*) *1 E.Ide.simps*(*3*) *VV.arrE* $\mu\nu$ *arr-char*)
have *5*: *E.Ide* (*Cod* (*fst* $\mu\nu$) ⌊⋆⌋ *Cod* (*snd* $\mu\nu$))
using *4*
by (*meson E.Ide.simps*(*3*) *E.Ide-HcompNml* $VV.arr\text{-}char_{SbC}$ *VV.ideD*(*1*) *arr-char* $\mu\nu$)
have *6*: ⌊*Cod* (*fst* $\mu\nu$) ⋆ *Cod* (*snd* $\mu\nu$)⌋ = ⌊*Cod* (*fst* $\mu\nu$) ⌊⋆⌋ *Cod* (*snd* $\mu\nu$)⌋
by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *E.Nml-HcompNml*(*1*) *E.Nmlize.simps*(*3*) *E.Nmlize-Nml* $VV.arr\text{-}char_{SbC}$ *arr-char* $\mu\nu$ *1*)
have *A*: «$\Psi_o{}'$ $\mu\nu$: ⦃*Dom* (*fst* $\mu\nu$) ⌊⋆⌋ *Dom* (*snd* $\mu\nu$)⦄
$\Rightarrow_B$ ⦃*Dom* (*fst* $\mu\nu$) ⋆ *Dom* (*snd* $\mu\nu$)⦄»
using *1 2 3* **by** *auto*
have *B*: «*Map* (*fst* $\mu\nu$) $\star_B$ *Map* (*snd* $\mu\nu$) :
⦃*Dom* (*fst* $\mu\nu$) ⋆ *Dom* (*snd* $\mu\nu$)⦄ $\Rightarrow_B$ ⦃*Cod* (*fst* $\mu\nu$) ⋆ *Cod* (*snd* $\mu\nu$)⦄»
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ *arr-char src-def trg-def E.Nml-implies-Arr E.eval-simps′*(*2*−*3*)
by *auto*
have *C*: «*B.can* (*Cod* (*fst* $\mu\nu$) ⌊⋆⌋ *Cod* (*snd* $\mu\nu$)) (*Cod* (*fst* $\mu\nu$) ⋆ *Cod* (*snd* $\mu\nu$)) :
⦃*Cod* (*fst* $\mu\nu$) ⋆ *Cod* (*snd* $\mu\nu$)⦄ $\Rightarrow_B$ ⦃*Cod* (*fst* $\mu\nu$) ⌊⋆⌋ *Cod* (*snd* $\mu\nu$)⦄»

using *4 5 6* **by** *auto*
have Ψ_o (*VV.cod* $\mu\nu$) $\cdot_B$ *HoDN-DN.map* $\mu\nu$ =
B.can (*Cod* (*fst* $\mu\nu$) $\lfloor\star\rfloor$ *Cod* (*snd* $\mu\nu$)) (*Cod* (*fst* $\mu\nu$) $\star$ *Cod* (*snd* $\mu\nu$)) $\cdot_B$
(*Map* (*fst* $\mu\nu$) $\star_B$ *Map* (*snd* $\mu\nu$))
using $\mu\nu$ *VV.arr-char*$_{SbC}$ *VV.cod-char*$_{SbC}$ *arr-char src-def trg-def cod-char DN.FF-def DN-def*
by *auto*
also have ... = *B.can* (*Cod* (*fst* $\mu\nu$) $\lfloor\star\rfloor$ *Cod* (*snd* $\mu\nu$))
(*Cod* (*fst* $\mu\nu$) $\star$ *Cod* (*snd* $\mu\nu$)) $\cdot_B$
(*Map* (*fst* $\mu\nu$) $\star_B$ *Map* (*snd* $\mu\nu$)) $\cdot_B$ $\Psi_o{}'$ $\mu\nu$ $\cdot_B$ Ψ_o $\mu\nu$
using *B* $\mu\nu$ *VV.arr-char*$_{SbC}$ *arr-char src-def trg-def*
E.Ide-HcompNml E.Nml-HcompNml(1) B.can-Ide-self B.comp-arr-dom
by *auto*
also have ... = *DNoH.map* $\mu\nu$ $\cdot_B$ Ψ_o (*VV.dom* $\mu\nu$)
proof −
have *DNoH.map* $\mu\nu$ $\cdot_B$ Ψ_o (*VV.dom* $\mu\nu$) =
B.can (*Cod* (*fst* $\mu\nu$) $\lfloor\star\rfloor$ *Cod* (*snd* $\mu\nu$)) (*Cod* (*fst* $\mu\nu$) $\star$ *Cod* (*snd* $\mu\nu$)) $\cdot_B$
(*Map* (*fst* $\mu\nu$) $\star_B$ *Map* (*snd* $\mu\nu$)) $\cdot_B$ $\Psi_o{}'$ $\mu\nu$ $\cdot_B$ Ψ_o (*VV.dom* $\mu\nu$)
using $\mu\nu$ *DN-def VV.arr-char*$_{SbC}$ *B.comp-assoc* **by** *simp*
moreover have Ψ_o (*VV.dom* $\mu\nu$) = Ψ_o $\mu\nu$
using $\mu\nu$ *VV.dom-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *auto*
ultimately show *?thesis*
using *B.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
qed
qed

abbreviation *cmp*$_{DN}$
where *cmp*$_{DN}$ $\equiv$ Ψ*.map*

interpretation Ψ: *natural-isomorphism VV.comp* V_B *HoDN-DN.map DNoH.map cmp*$_{DN}$
using Ψ_o*-in-hom B.iso-def* Ψ*.map-simp-ide*
apply *unfold-locales*
apply *auto*
by *blast*

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

lemma *cmp*$_{DN}$*-in-hom* [*intro*]:
assumes *arr* (*fst* $\mu\nu$) **and** *arr* (*snd* $\mu\nu$) **and** *src* (*fst* $\mu\nu$) = *trg* (*snd* $\mu\nu$)
shows «*cmp*$_{DN}$ $\mu\nu$: *DN* (*src* (*snd* $\mu\nu$)) $\rightarrow_B$ *DN* (*trg* (*fst* $\mu\nu$))»
and «*cmp*$_{DN}$ $\mu\nu$: *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))
$\Rightarrow_B$ *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))»
proof −
have *1*: *VV.arr* $\mu\nu$
using *assms VV.arr-char*$_{SbC}$ **by** *simp*
show *2*: «*cmp*$_{DN}$ $\mu\nu$: *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))

$\Rightarrow_B$ *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))»
proof −
have *HoDN-DN.map* (*VV.dom* $\mu\nu$) = *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))
using *assms 1 DN.FF-def VV.dom-simp* **by** *auto*
moreover have *DNoH.map* (*VV.cod* $\mu\nu$) = *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))
using *assms 1 VV.cod-simp* **by** *simp*
ultimately show *?thesis*
using *assms 1* Ψ*.preserves-hom* **by** *auto*
qed
show «cmp_{DN} $\mu\nu$: *DN* (*src* (*snd* $\mu\nu$)) $\rightarrow_B$ *DN* (*trg* (*fst* $\mu\nu$))»
using *assms 2 B.src-dom* [*of* cmp_{DN} $\mu\nu$] *B.trg-dom* [*of* cmp_{DN} $\mu\nu$]
by (*elim B.in-homE*, *intro B.in-hhomI*) *auto*
qed

lemma cmp_{DN}*-simps* [*simp*]:
assumes *arr* (*fst* $\mu\nu$) **and** *arr* (*snd* $\mu\nu$) **and** *src* (*fst* $\mu\nu$) = *trg* (*snd* $\mu\nu$)
shows *B.arr* (cmp_{DN} $\mu\nu$)
and src_B (cmp_{DN} $\mu\nu$) = *DN* (*src* (*snd* $\mu\nu$)) **and** trg_B (cmp_{DN} $\mu\nu$) = *DN* (*trg* (*fst* $\mu\nu$))
and *B.dom* (cmp_{DN} $\mu\nu$) = *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))
and *B.cod* (cmp_{DN} $\mu\nu$) = *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))
proof
show *VV.arr* $\mu\nu$
using *assms* **by** *blast*
have *1*: «cmp_{DN} $\mu\nu$: *DN* (*src* (*snd* $\mu\nu$)) $\rightarrow_B$ *DN* (*trg* (*fst* $\mu\nu$))»
using *assms* **by** *blast*
show src_B (cmp_{DN} $\mu\nu$) = *DN* (*src* (*snd* $\mu\nu$))
using *1* **by** *fast*
show trg_B (cmp_{DN} $\mu\nu$) = *DN* (*trg* (*fst* $\mu\nu$))
using *1* **by** *fast*
have *2*: «cmp_{DN} $\mu\nu$: *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))
$\Rightarrow_B$ *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))»
using *assms* **by** *blast*
show *B.dom* (cmp_{DN} $\mu\nu$) = *DN* (*dom* (*fst* $\mu\nu$)) $\star_B$ *DN* (*dom* (*snd* $\mu\nu$))
using *2* **by** *fast*
show *B.cod* (cmp_{DN} $\mu\nu$) = *DN* (*cod* (*fst* $\mu\nu$) $\star$ *cod* (*snd* $\mu\nu$))
using *2* **by** *fast*
qed

interpretation *DN*: *pseudofunctor vcomp hcomp* a i *src trg* V_B H_B a_B i_B src_B trg_B
DN cmp_{DN}
proof
show $\bigwedge f\ g\ h.$ ⟦ *ide f*; *ide g*; *ide h*; *src f* = *trg g*; *src g* = *trg h* ⟧ $\Longrightarrow$
DN (a *f g h*) $\cdot_B$ cmp_{DN} (*f* $\star$ *g*, *h*) $\cdot_B$ (cmp_{DN} (*f*, *g*) $\star_B$ *DN h*) =
cmp_{DN} (*f*, *g* $\star$ *h*) $\cdot_B$ (*DN f* $\star_B$ cmp_{DN} (*g*, *h*)) $\cdot_B$ a_B[*DN f*, *DN g*, *DN h*]
proof −
fix *f g h*
assume *f*: *ide f* **and** *g*: *ide g* **and** *h*: *ide h*
and *fg*: *src f* = *trg g* **and** *gh*: *src g* = *trg h*
show *DN* (a *f g h*) $\cdot_B$ cmp_{DN} (*f* $\star$ *g*, *h*) $\cdot_B$ (cmp_{DN} (*f*, *g*) $\star_B$ *DN h*) =

$cmp_{DN}\ (f,\ g \star h) \cdot_B (DN\ f \star_B cmp_{DN}\ (g,\ h)) \cdot_B \mathrm{a}_B[DN\ f,\ DN\ g,\ DN\ h]$
proof −
have *1*: $E.Trg\ (Dom\ g) = E.Trg\ (Dom\ g \lfloor\star\rfloor\ Dom\ h) \wedge$
$\{\!|E.Trg\ (Dom\ g)|\!\} = \{\!|E.Trg\ (Dom\ g \lfloor\star\rfloor\ Dom\ h)|\!\}$
using *f g h fg gh arr-char src-def trg-def E.Trg-HcompNml*
by (*metis* (*no-types*, *lifting*) *ideD*(*1*) *src-simps*(*2*) *trg-simps*(*2*))
have *2*: $arr\ (MkArr\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h)\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)$
$(B.can\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h) \cdot_B$
$(Map\ f \star_B B.can\ (Cod\ g \lfloor\star\rfloor\ Cod\ h)\ (Cod\ g \star Cod\ h) \cdot_B$
$(Map\ g \star_B Map\ h) \cdot_B B.can\ (Dom\ g \star Dom\ h)\ (Dom\ g \lfloor\star\rfloor\ Dom\ h)) \cdot_B$
$B.can\ (Dom\ f \star Dom\ g \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h)))$
proof −
have «$B.can\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h) \cdot_B$
$(Map\ f \star_B$
$B.can\ (Cod\ g \lfloor\star\rfloor\ Cod\ h)\ (Cod\ g \star Cod\ h) \cdot_B$
$(Map\ g \star_B Map\ h) \cdot_B B.can\ (Dom\ g \star Dom\ h)\ (Dom\ g \lfloor\star\rfloor\ Dom\ h)) \cdot_B$
$B.can\ (Dom\ f \star Dom\ g \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h) :$
$EVAL\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h)$
$\Rightarrow_B EVAL\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)$»
proof (*intro B.comp-in-homI*)
show *2*: «$B.can\ (Dom\ f \star Dom\ g \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h) :$
$EVAL\ (Dom\ f \lfloor\star\rfloor\ Dom\ g \lfloor\star\rfloor\ Dom\ h) \Rightarrow_B$
$EVAL\ (Dom\ f \star Dom\ g \lfloor\star\rfloor\ Dom\ h)$»
using *f g h fg gh 1*
apply (*intro B.can-in-hom*)
apply (*metis* (*no-types*, *lifting*) *E.Ide-HcompNml E.Nml-HcompNml*(*1*)
arr-char ideD(*1*) *src-simps*(*1*) *trg-simps*(*1*))
apply (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *E.Ide-HcompNml ideD*(*1*)
arr-char src-simps(*1*) *trg-simps*(*1*))
by (*metis* (*no-types*, *lifting*) *E.Nml-HcompNml*(*1*) *E.Nmlize.simps*(*3*)
E.Nmlize-Nml ideD(*1*) *arr-char src-simps*(*1*) *trg-simps*(*1*))
show «$B.can\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h) :$
$EVAL\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h) \Rightarrow_B$
$EVAL\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)$»
proof −
have $E.Ide\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h)$
using *f g h fg gh 1 Cod-ide E.Ide-HcompNml arr-char*
apply *simp*
by (*metis* (*no-types*, *lifting*) *ideD*(*1*) *src-simps*(*1*) *trg-simps*(*1*))
moreover have $E.Ide\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)$
using *f g h fg gh*
by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *E.Ide-HcompNml E.Nml-HcompNml*(*1*)
arr-char calculation ideD(*1*) *src-simps*(*1*) *trg-simps*(*1*))
moreover have $E.Nmlize\ (Cod\ f \star Cod\ g \lfloor\star\rfloor\ Cod\ h) =$
$E.Nmlize\ (Cod\ f \lfloor\star\rfloor\ Cod\ g \lfloor\star\rfloor\ Cod\ h)$
using *f g h fg gh*
by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *E.Nml-HcompNml*(*1*) *E.Nmlize.simps*(*3*)
E.Nmlize-Nml arr-char calculation(*1*) *ideD*(*1*) *src-simps*(*1*) *trg-simps*(*1*))
ultimately show *?thesis*

```
      using B.can-in-hom [of Cod f ⋆ Cod g ⌊⋆⌋ Cod h Cod f ⌊⋆⌋ Cod g ⌊⋆⌋ Cod h]
      by blast
    qed
    show «Map f ⋆B B.can (Cod g ⌊⋆⌋ Cod h) (Cod g ⋆ Cod h) ·B
          (Map g ⋆B Map h) ·B B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) :
           EVAL (Dom f ⋆ Dom g ⌊⋆⌋ Dom h) ⇒B EVAL (Cod f ⋆ Cod g ⌊⋆⌋ Cod h)»
      using f g h fg gh B.can-in-hom
      apply simp
    proof (intro B.hcomp-in-vhom B.comp-in-homI)
      show 1: «B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) :
                EVAL (Dom g ⌊⋆⌋ Dom h) ⇒B EVAL (Dom g ⋆ Dom h)»
        using g h gh B.can-in-hom
      by (metis (no-types, lifting) E.Ide.simps(3) E.Ide-HcompNml E.Nml-HcompNml(1)
          E.Nmlize.simps(3) E.Nmlize-Nml arr-char ideD(1) src-simps(1) trg-simps(1))
      show «B.can (Cod g ⌊⋆⌋ Cod h) (Cod g ⋆ Cod h) :
              EVAL (Cod g ⋆ Cod h) ⇒B EVAL (Cod g ⌊⋆⌋ Cod h)»
        using g h gh B.can-in-hom
        by (metis (no-types, lifting) Cod-ide E.Ide.simps(3) E.Ide-HcompNml
            E.Nml-HcompNml(1) E.Nmlize.simps(3) E.Nmlize-Nml arr-char ideD(1)
            src-simps(2) trg-simps(2))
      show «Map g ⋆B Map h : EVAL (Dom g ⋆ Dom h) ⇒B EVAL (Cod g ⋆ Cod h)»
      proof
        show 2: B.hseq (Map g) (Map h)
          using g h gh
          by (metis (no-types, lifting) B.null-is-zero(1−2) B.hseq-char′
              B.ideD(1) Map-hcomp ideE ide-hcomp)
        show B.dom (Map g ⋆B Map h) = EVAL (Dom g ⋆ Dom h)
          using g h gh 2 by fastforce
        show B.cod (Map g ⋆B Map h) = EVAL (Cod g ⋆ Cod h)
          using g h gh 2 by fastforce
      qed
      show «Map f : Map f ⇒B EVAL (Cod f)»
        using f arr-char Cod-ide by auto
      show srcB (Map f) = trgB ⦃Dom g ⌊⋆⌋ Dom h⦄
        using f g h fg gh 1 2 src-def trg-def B.arrI B.hseqE B.not-arr-null
              B.trg.extensionality B.trg.preserves-hom B.vconn-implies-hpar(2)
              B.vconn-implies-hpar(4) E.eval.simps(3)
        by (metis (no-types, lifting) Map-ide(1))
    qed
  qed
  thus ?thesis
    using f g h fg gh arr-char src-def trg-def E.Nml-HcompNml E.Ide-HcompNml
          ideD(1)
    apply (intro arr-MkArr) by auto
qed
have 3: E.Ide (Dom g ⌊⋆⌋ Dom h)
  using g h gh ide-char arr-char src-def trg-def E.Ide-HcompNml Cod-ide
  by (metis (no-types, lifting) ideD(1) src-simps(2) trg-simps(2))
have 4: E.Ide (Dom g ⋆ Dom h)
```

by (*metis* (*no-types*, *lifting*) *E.Ide.simps*(*3*) *arrE g gh h ideE src-simps*(*1*) *trg-simps*(*1*))
have *5*: *E.Nmlize* (*Dom g* $\lfloor\star\rfloor$ *Dom h*) = *E.Nmlize* (*Dom g* $\star$ *Dom h*)
  using *g h gh ide-char arr-char src-def trg-def E.Nml-HcompNml*
  by (*metis* (*no-types*, *lifting*) *4 E.Ide.simps*(*3*) *E.Nmlize.simps*(*3*) *E.Nmlize-Nml ideD*(*1*))
have *6*: *E.Ide* (*Dom f* $\lfloor\star\rfloor$ *Dom g* $\lfloor\star\rfloor$ *Dom h*)
  using *f g h fg gh arr-char src-def trg-def*
  by (*metis* (*no-types*, *lifting*) *1 E.Nml-HcompNml*(*1*) *E.Ide-HcompNml ideD*(*1*) *src-simps*(*2*) *trg-simps*(*2*))
have *7*: *E.Ide* (*Dom f* $\star$ *Dom g* $\lfloor\star\rfloor$ *Dom h*)
  using *f g h fg gh arr-char src-def trg-def*
  by (*metis* (*no-types*, *lifting*) *1 3 E.Ide.simps*(*3*) *ideD*(*1*) *src-simps*(*2*) *trg-simps*(*2*))
have *8*: *E.Nmlize* (*Dom f* $\lfloor\star\rfloor$ *Dom g* $\lfloor\star\rfloor$ *Dom h*) =
      E.Nmlize (*Dom f* $\star$ *Dom g* $\lfloor\star\rfloor$ *Dom h*)
  using *f g h fg gh arr-char src-def trg-def*
      7 E.Nml-HcompNml(*1*) *ideD*(*1*)
  by *auto*
have *DN* (a *f g h*) $\cdot_B$ cmp_{DN} (*f* $\star$ *g*, *h*) $\cdot_B$ (cmp_{DN} (*f*, *g*) $\star_B$ *DN h*) =
    B.can (*Dom f* $\lfloor\star\rfloor$ *Dom g* $\lfloor\star\rfloor$ *Dom h*) ((*Dom f* $\star$ *Dom g*) $\star$ *Dom h*)
proof −
  have *9*: *VVV.arr* (*f*, *g*, *h*)
    using *f g h fg gh VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char ideD* **by** *simp*
  have *10*: *VV.ide* (*f*, *g*)
    using *f g fg VV.ide-char*$_{SbC}$ **by** *auto*
  have *11*: *VV.ide* (*hcomp f g*, *h*)
    using *f g h fg gh VV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *simp*
  have *12*: *arr* (*MkArr* (*Dom g* $\lfloor\star\rfloor$ *Dom h*) (*Cod g* $\lfloor\star\rfloor$ *Cod h*)
                (*B.can* (*Cod g* $\lfloor\star\rfloor$ *Cod h*) (*Cod g* $\star$ *Cod h*) $\cdot_B$
                  (*Map g* $\star_B$ *Map h*) $\cdot_B$
                  B.can (*Dom g* $\star$ *Dom h*) (*Dom g* $\lfloor\star\rfloor$ *Dom h*)))
  proof (*intro arr-MkArr*)
    show *Dom g* $\lfloor\star\rfloor$ *Dom h* $\in$ *IDE*
      using *g h gh*
      by (*metis* (*no-types*, *lifting*) *3 E.Nml-HcompNml*(*1*) *arr-char ideD*(*1*) *mem-Collect-eq src-simps*(*2*) *trg-simps*(*2*))
    show *Cod g* $\lfloor\star\rfloor$ *Cod h* $\in$ *IDE*
      using *g h gh Cod-ide* ‹*Dom g* $\lfloor\star\rfloor$ *Dom h* $\in$ *IDE*› **by** *auto*
    show *B.can* (*Cod g* $\lfloor\star\rfloor$ *Cod h*) (*Cod g* $\star$ *Cod h*) $\cdot_B$
          (*Map g* $\star_B$ *Map h*) $\cdot_B$
          B.can (*Dom g* $\star$ *Dom h*) (*Dom g* $\lfloor\star\rfloor$ *Dom h*)
        $\in$ *HOM* (*Dom g* $\lfloor\star\rfloor$ *Dom h*) (*Cod g* $\lfloor\star\rfloor$ *Cod h*)
    proof
      show *E.Src* (*Dom g* $\lfloor\star\rfloor$ *Dom h*) = *E.Src* (*Cod g* $\lfloor\star\rfloor$ *Cod h*) $\wedge$
          E.Trg (*Dom g* $\lfloor\star\rfloor$ *Dom h*) = *E.Trg* (*Cod g* $\lfloor\star\rfloor$ *Cod h*) $\wedge$
          «*B.can* (*Cod g* $\lfloor\star\rfloor$ *Cod h*) (*Cod g* $\star$ *Cod h*) $\cdot_B$
            (*Map g* $\star_B$ *Map h*) $\cdot_B$ *B.can* (*Dom g* $\star$ *Dom h*) (*Dom g* $\lfloor\star\rfloor$ *Dom h*) :
            $\{\!|$*Dom g* $\lfloor\star\rfloor$ *Dom h*$|\!\}$ $\Rightarrow_B$ $\{\!|$*Cod g* $\lfloor\star\rfloor$ *Cod h*$|\!\}$»
      proof (*intro conjI*)
        show *E.Src* (*Dom g* $\lfloor\star\rfloor$ *Dom h*) = *E.Src* (*Cod g* $\lfloor\star\rfloor$ *Cod h*)

```
        using g h gh Cod-ide by simp
      show E.Trg (Dom g ⌊⋆⌋ Dom h) = E.Trg (Cod g ⌊⋆⌋ Cod h)
        using g h gh Cod-ide by simp
      show «B.can (Cod g ⌊⋆⌋ Cod h) (Cod g ⋆ Cod h) ·B
              (Map g ⋆B Map h) ·B B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) :
              ⦃Dom g ⌊⋆⌋ Dom h⦄ ⇒B ⦃Cod g ⌊⋆⌋ Cod h⦄»
      proof (intro B.comp-in-homI)
        show «B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) :
                ⦃Dom g ⌊⋆⌋ Dom h⦄ ⇒B ⦃Dom g ⋆ Dom h⦄»
          using 3 4 5 by blast
        show «Map g ⋆B Map h : ⦃Dom g ⋆ Dom h⦄ ⇒B ⦃Cod g ⋆ Cod h⦄»
          using g h gh
          by (metis (no-types, lifting) 4 B.ide-in-hom(2) Cod-ide E.eval.simps(3)
              E.ide-eval-Ide Map-ide(2))
        show «B.can (Cod g ⌊⋆⌋ Cod h) (Cod g ⋆ Cod h) :
                ⦃Cod g ⋆ Cod h⦄ ⇒B ⦃Cod g ⌊⋆⌋ Cod h⦄»
          using 3 4 5 Cod-ide g h by auto
      qed
    qed
  qed
qed
have DN (a f g h) = ⦃Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h⦄
proof −
  have DN (a f g h) =
        (B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) (Dom f ⋆ Dom g ⌊⋆⌋ Dom h) ·B
          ((Map f ⋆B B.can (Dom g ⌊⋆⌋ Dom h) (Dom g ⋆ Dom h) ·B
            (Map g ⋆B Map h) ·B B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h))) ·B
          B.can (Dom f ⋆ Dom g ⌊⋆⌋ Dom h) (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h))
    using f g h fg gh 1 2 9 12 DN-def a-def Cod-ide by simp
  also have
    ... = B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) (Dom f ⋆ Dom g ⌊⋆⌋ Dom h) ·B
            (Map f ⋆B ⦃Dom g ⌊⋆⌋ Dom h⦄) ·B
              B.can (Dom f ⋆ Dom g ⌊⋆⌋ Dom h) (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h)
  proof −
    have «B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) :
                    ⦃Dom g ⌊⋆⌋ Dom h⦄ ⇒B Map g ⋆B Map h»
      using g h 3 4 5 B.can-in-hom [of Dom g ⌊⋆⌋ Dom h Dom g ⋆ Dom h]
      by simp
    hence Map f ⋆B B.can (Dom g ⌊⋆⌋ Dom h) (Dom g ⋆ Dom h) ·B
                (Map g ⋆B Map h) ·B B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) =
          Map f ⋆B B.can (Dom g ⌊⋆⌋ Dom h) (Dom g ⋆ Dom h) ·B
                  B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h)
      using B.comp-cod-arr by auto
    also have ... = Map f ⋆B ⦃Dom g ⌊⋆⌋ Dom h⦄
      using f g h fg gh 3 4 5 B.can-Ide-self by auto
    finally have Map f ⋆B B.can (Dom g ⌊⋆⌋ Dom h) (Dom g ⋆ Dom h) ·B
                (Map g ⋆B Map h) ·B B.can (Dom g ⋆ Dom h) (Dom g ⌊⋆⌋ Dom h) =
                Map f ⋆B ⦃Dom g ⌊⋆⌋ Dom h⦄
      by simp
```

thus *?thesis* **by** *simp*
qed
also have
... = *B.can* (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*) $\cdot_B$
B.can (*Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*)
proof −
have «*B.can* (*Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) :
⦃*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*⦄ $\Rightarrow_B$ *Map f* $\star_B$ ⦃*Dom g* ⌊⋆⌋ *Dom h*⦄»
using *f g h 6 7 8*
B.can-in-hom [*of Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*]
by *simp*
hence (*Map f* $\star_B$ ⦃*Dom g* ⌊⋆⌋ *Dom h*⦄) $\cdot_B$
B.can (*Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) =
B.can (*Dom f* ⋆ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*)
using *B.comp-cod-arr* **by** *auto*
thus *?thesis* **by** *simp*
qed
also have
... = *B.can* (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*)
using *f g h fg gh 6 7 8* **by** *auto*
also have ... = ⦃*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*⦄
using *f g h fg gh 6 B.can-Ide-self* **by** *blast*
finally show *?thesis* **by** *simp*
qed
have *DN* (a *f g h*) $\cdot_B$ cmp_{DN} (*f* ⋆ *g*, *h*) $\cdot_B$ (cmp_{DN} (*f*, *g*) $\star_B$ *DN h*) =
B.can (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⌊⋆⌋ *Dom h*) $\cdot_B$
B.can ((*Dom f* ⌊⋆⌋ *Dom g*) ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⋆ *Dom h*) $\cdot_B$
(*B.can* (*Dom f* ⌊⋆⌋ *Dom g*) (*Dom f* ⋆ *Dom g*) $\star_B$ *Map h*)
proof −
have *DN* (a *f g h*) =
B.can (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⌊⋆⌋ *Dom h*)
using *f g h fg gh DN-def 1 4 6 7 B.can-Ide-self E.HcompNml-assoc*
E.Ide.simps(*3*) ‹*DN* (a *f g h*) = ⦃*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*⦄› *ide-char*
by (*metis* (*no-types*, *lifting*) *arr-char ideD*(*1*))
thus *?thesis*
using *f g h fg gh 10 11 DN-def* Ψ*.map-simp-ide* **by** *simp*
qed
also have
... = (*B.can* (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⌊⋆⌋ *Dom h*) $\cdot_B$
B.can ((*Dom f* ⌊⋆⌋ *Dom g*) ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⋆ *Dom h*)) $\cdot_B$
(*B.can* (*Dom f* ⌊⋆⌋ *Dom g*) (*Dom f* ⋆ *Dom g*) $\star_B$ *Map h*)
using *B.comp-assoc* **by** *simp*
also have
... = *B.can* (*Dom f* ⌊⋆⌋ *Dom g* ⌊⋆⌋ *Dom h*) ((*Dom f* ⌊⋆⌋ *Dom g*) ⋆ *Dom h*) $\cdot_B$
B.can ((*Dom f* ⌊⋆⌋ *Dom g*) ⋆ *Dom h*) ((*Dom f* ⋆ *Dom g*) ⋆ *Dom h*)
proof −
have *B.can* (*Dom f* ⌊⋆⌋ *Dom g*) (*Dom f* ⋆ *Dom g*) $\star_B$ *Map h* =
B.can ((*Dom f* ⌊⋆⌋ *Dom g*) ⋆ *Dom h*) ((*Dom f* ⋆ *Dom g*) ⋆ *Dom h*)
proof −

```
    have B.can (Dom f ⌊⋆⌋ Dom g) (Dom f ⋆ Dom g) ⋆B Map h =
          B.can (Dom f ⌊⋆⌋ Dom g) (Dom f ⋆ Dom g) ⋆B B.can (Dom h) (Dom h)
      using h B.can-Ide-self by fastforce
   also have ... = B.can ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h) ((Dom f ⋆ Dom g) ⋆ Dom h)
      using f g h 1 4 7 arr-char E.Nml-HcompNml(1) E.Src-HcompNml
            B.hcomp-can [of Dom f ⋆ Dom g Dom f ⌊⋆⌋ Dom g Dom h Dom h]
      by (metis (no-types, lifting) E.Nmlize.simps(3) E.Nmlize-Nml
          E.Ide.simps(3) E.Ide-HcompNml E.Src.simps(3) arrE ideD(1))
    finally show ?thesis by simp
  qed
  moreover have
    B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) ((Dom f ⌊⋆⌋ Dom g) ⌊⋆⌋ Dom h) ·B
      B.can ((Dom f ⌊⋆⌋ Dom g) ⌊⋆⌋ Dom h) ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h) =
        B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h)
  proof −
    have E.Ide ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h)
      using f g h 1 4 7
      by (metis (no-types, lifting) E.Ide.simps(3) E.Ide-HcompNml E.Src-HcompNml
          arrE ideD(1))
    moreover have E.Ide ((Dom f ⌊⋆⌋ Dom g) ⌊⋆⌋ Dom h)
      using f g h 1 7 E.Ide-HcompNml E.Nml-HcompNml(1) arr-char calculation
            ideD(1)
      by auto
    moreover have E.Ide (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h)
      using f g h 1 4 6 by blast
    moreover have E.Nmlize ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h) =
                  E.Nmlize ((Dom f ⌊⋆⌋ Dom g) ⌊⋆⌋ Dom h)
      using f g h 1 4 7
      by (metis (no-types, lifting) E.Nml-HcompNml(1) E.Nmlize.simps(3)
          E.Nmlize-Nml E.Ide.simps(3) arrE calculation(1) ideD(1))
    moreover have E.Nmlize ((Dom f ⌊⋆⌋ Dom g) ⌊⋆⌋ Dom h) =
                  E.Nmlize (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h)
      using f g h 1 4 7 E.HcompNml-assoc by fastforce
    ultimately show ?thesis
      using B.vcomp-can by simp
  qed
  ultimately show ?thesis by simp
qed
also have ... = B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) ((Dom f ⋆ Dom g) ⋆ Dom h)
proof −
  have E.Ide ((Dom f ⋆ Dom g) ⋆ Dom h)
    using 1 4 7 by simp
  moreover have E.Ide ((Dom f ⌊⋆⌋ Dom g) ⋆ Dom h)
    using f g 1 4 7
    by (metis (no-types, lifting) E.Ide.simps(3) E.Ide-HcompNml E.Src-HcompNml
        arrE ideD(1))
  moreover have E.Ide (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h)
    using f g h 6 by blast
  moreover have E.Nmlize ((Dom f ⋆ Dom g) ⋆ Dom h) =
```

$E.Nmlize\ ((Dom\ f\ \lfloor\star\rfloor\ Dom\ g)\ \star\ Dom\ h)$
using *f g h 1 7 E.Nml-HcompNml(1)* **by** *fastforce*
moreover have $E.Nmlize\ ((Dom\ f\ \lfloor\star\rfloor\ Dom\ g)\ \star\ Dom\ h) =$
$E.Nmlize\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)$
using *f g h 1 4 7*
by (*metis* (*no-types*, *lifting*) *E.Nml-HcompNml(1) E.Nmlize.simps(3) E.Nmlize-Nml E.HcompNml-assoc E.Ide.simps(3) arrE ideD(1)*)
ultimately show *?thesis*
using *B.vcomp-can* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
also have $... = cmp_{DN}\ (f,\ g \star h)\ \cdot_B\ (DN\ f\ \star_B\ cmp_{DN}\ (g,\ h))\ \cdot_B$
$\mathrm{a}_B[DN\ f,\ DN\ g,\ DN\ h]$
proof −
have $cmp_{DN}\ (f,\ g \star h)\ \cdot_B\ (DN\ f\ \star_B\ cmp_{DN}\ (g,\ h))\ \cdot_B\ \mathrm{a}_B[DN\ f,\ DN\ g,\ DN\ h] =$
$(cmp_{DN}\ (f,\ g \star h)\ \cdot_B\ (DN\ f\ \star_B\ cmp_{DN}\ (g,\ h)))\ \cdot_B\ \mathrm{a}_B[DN\ f,\ DN\ g,\ DN\ h]$
using *B.comp-assoc* **by** *simp*
also have
$... = B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g \star Dom\ h)\ \cdot_B$
$B.can\ (Dom\ f \star Dom\ g \star Dom\ h)\ ((Dom\ f \star Dom\ g)\ \star\ Dom\ h)$
proof −
have $cmp_{DN}\ (f,\ g \star h)\ \cdot_B\ (DN\ f\ \star_B\ cmp_{DN}\ (g,\ h)) =$
$B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g \star Dom\ h)$
proof −
have $cmp_{DN}\ (f,\ g \star h)\ \cdot_B\ (DN\ f\ \star_B\ cmp_{DN}\ (g,\ h)) =$
$B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ \cdot_B$
$(Map\ f\ \star_B\ B.can\ (Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ g \star Dom\ h))$
using *f g h fg gh* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *DN-def* **by** *simp*
also have
$... = B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ \cdot_B$
$(B.can\ (Dom\ f)\ (Dom\ f)\ \star_B\ B.can\ (Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ g \star Dom\ h))$
proof −
have $Map\ f = B.can\ (Dom\ f)\ (Dom\ f)$
using *f arr-char B.can-Ide-self* [*of Dom f*] *Map-ide*
by (*metis* (*no-types*, *lifting*) *ide-char'*)
thus *?thesis* **by** *simp*
qed
also have
$... = B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ \cdot_B$
$B.can\ (Dom\ f \star Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g \star Dom\ h)$
using *1 4 5 7 B.hcomp-can* **by** *auto*
also have $... = B.can\ (Dom\ f\ \lfloor\star\rfloor\ Dom\ g\ \lfloor\star\rfloor\ Dom\ h)\ (Dom\ f \star Dom\ g \star Dom\ h)$
using *1 4 5 6 7 8 B.vcomp-can* **by** *auto*
finally show *?thesis* **by** *simp*
qed
moreover have $\mathrm{a}_B[DN\ f,\ DN\ g,\ DN\ h] =$
$B.can\ (Dom\ f \star Dom\ g \star Dom\ h)\ ((Dom\ f \star Dom\ g)\ \star\ Dom\ h)$
using *f g h 1 4 7 DN-def B.canE-associator(1)* **by** *auto*

```
        ultimately show ?thesis by simp
      qed
      also have ... = B.can (Dom f ⌊⋆⌋ Dom g ⌊⋆⌋ Dom h) ((Dom f ⋆ Dom g) ⋆ Dom h)
        using 1 4 5 6 7 8 E.Nmlize-Hcomp-Hcomp B.vcomp-can by simp
      finally show ?thesis by simp
    qed
    finally show ?thesis by blast
  qed
 qed
qed

lemma DN-is-pseudofunctor:
shows pseudofunctor vcomp hcomp a i src trg V_B H_B a_B i_B src_B trg_B DN cmp_DN
  ..

interpretation faithful-functor vcomp V_B DN
  using arr-char dom-char cod-char DN-def
  apply unfold-locales
  by (metis (no-types, lifting) Cod-dom Dom-cod MkArr-Map)

no-notation B.in-hom (‹«- : - →_B -»›)

lemma DN-UP:
assumes B.arr μ
shows DN (UP μ) = μ
  using assms UP-def DN-def arr-UP by auto

interpretation DN: equivalence-pseudofunctor
                 vcomp hcomp a i src trg V_B H_B a_B i_B src_B trg_B DN cmp_DN
proof

  show ⋀f f' ν. ⟦ ide f; ide f'; src f = src f'; trg f = trg f'; «ν : DN f ⇒_B DN f'» ⟧
                 ⟹ ∃μ. «μ : f ⇒ f'» ∧ DN μ = ν
  proof −
    fix f f' ν
    assume f: ide f and f': ide f'
    and eq-src: src f = src f' and eq-trg: trg f = trg f'
    and ν: «ν : DN f ⇒_B DN f'»
    show ∃μ. «μ : f ⇒ f'» ∧ DN μ = ν
    proof −
      let ?μ = MkArr (Dom f) (Dom f') ν
      have μ: «?μ : f ⇒ f'»
      proof −
        have E.Src (Dom f) = E.Src (Dom f')
          using f f'
          by (metis (no-types, lifting) eq-src ideD(1) src-simps(2))
        moreover have E.Trg (Dom f) = E.Trg (Dom f')
          using f f'
          by (metis (no-types, lifting) eq-trg ideD(1) trg-simps(2))
```

```
      ultimately show ?thesis
        using f f' ν DN-def MkArr-Map [of f] MkArr-Map [of f']
        by (intro MkArr-in-hom) auto
    qed
    moreover have DN ?μ = ν
      using μ DN-def by auto
    ultimately show ?thesis by blast
  qed
qed

show ⋀a'. B.obj a' ⟹ ∃ a. obj a ∧ B.equivalent-objects (DN.map₀ a) a'
proof −
  fix a'
  assume a': B.obj a'
  have obj (UP.map₀ a')
    using a' UP.map₀-simps(1) by simp
  moreover have B.equivalent-objects (DN.map₀ (UP.map₀ a')) a'
  proof −
    have arr (MkArr ⟨a'⟩ ⟨a'⟩ a')
      using a' UP-def [of a'] UP.preserves-reflects-arr [of a'] by auto
    moreover have arr (MkArr ⟨a'⟩₀ ⟨a'⟩₀ a')
      using a' arr-char obj-simps by auto
    ultimately have DN.map₀ (UP.map₀ a') = a'
      using a' UP.map₀-def DN.map₀-def DN-def src-def by auto
    thus ?thesis
      using a' B.equivalent-objects-reflexive by simp
  qed
  ultimately show ∃ a. obj a ∧ B.equivalent-objects (DN.map₀ a) a'
    by blast
qed

show ⋀a b g. ⟦ obj a; obj b; «g : DN.map₀ a →B DN.map₀ b»; B.ide g ⟧ ⟹
              ∃f. «f : a → b» ∧ ide f ∧ B.isomorphic (DN f) g
proof −
  fix a b g
  assume a: obj a and b: obj b
  and g: «g : DN.map₀ a →B DN.map₀ b» and ide-g: B.ide g
  have ide (UP g)
    using ide-g UP.preserves-ide by simp
  moreover have B.isomorphic (DN (UP g)) g
    using ide-g DN-UP B.isomorphic-reflexive by simp
  moreover have «UP g : a → b»
  proof
    show arr (UP g)
      using g UP.preserves-reflects-arr by auto
    show src (UP g) = a
    proof −
      have src (UP g) = MkArr ⟨srcB g⟩₀ ⟨srcB g⟩₀ (srcB g)
        using ide-g src-def UP-def UP.preserves-reflects-arr [of g] B.ideD(1) by simp
```

```
        also have ... = a
        proof −
          have src_B g = src_B (DN.map_0 a)
            using a g B.in-hhom-def by simp
          also have ... = Map a
            using a Map-preserves-objects DN.map_0-def DN-def B.src-src B.obj-simps
            by auto
          finally have src_B g = Map a by simp
          hence MkArr ⟨src_B g⟩_0 ⟨src_B g⟩_0 (src_B g) = MkArr ⟨Map a⟩_0 ⟨Map a⟩_0 (Map a)
            by simp
          also have ... = a
            using a obj-char [of a] MkIde-Dom [of a]
            apply (cases Dom a)
                    apply force
            by simp-all
          finally show ?thesis by simp
        qed
        finally show ?thesis by simp
      qed
      show trg (UP g) = b
      proof −
        have trg (UP g) = MkArr ⟨trg_B g⟩_0 ⟨trg_B g⟩_0 (trg_B g)
          using ide-g trg-def UP-def UP.preserves-reflects-arr [of g] B.ideD(1) by simp
        also have ... = b
        proof −
          have trg_B g = trg_B (DN.map_0 b)
            using b g B.in-hhom-def by simp
          also have ... = Map b
            using b Map-preserves-objects DN.map_0-def DN-def B.src-src B.obj-simps
            by auto
          finally have trg_B g = Map b by simp
          hence MkArr ⟨trg_B g⟩_0 ⟨trg_B g⟩_0 (trg_B g) = MkArr ⟨Map b⟩_0 ⟨Map b⟩_0 (Map b)
            by simp
          also have ... = b
            using b obj-char [of b] MkIde-Dom [of b]
            apply (cases Dom b)
                    apply force
            by simp-all
          finally show ?thesis by simp
        qed
        finally show ?thesis by simp
      qed
    qed
    ultimately show ∃f. «f : a → b» ∧ ide f ∧ B.isomorphic (DN f) g
      by blast
  qed
qed

theorem DN-is-equivalence-pseudofunctor:
```

shows *equivalence-pseudofunctor vcomp hcomp* a i *src trg* V_B H_B a_B i_B src_B trg_B
DN cmp_{DN}
..

The following gives an explicit formula for a component of the unit isomorphism of the pseudofunctor *UP* from a bicategory to its strictification. It is not currently being used – I originally proved it in order to establish something that I later proved in a more abstract setting – but it might be useful at some point.

interpretation *UP*: *equivalence-pseudofunctor*
V_B H_B a_B i_B src_B trg_B *vcomp hcomp* a i *src trg UP* cmp_{UP}
using *UP-is-equivalence-pseudofunctor* **by** *auto*

lemma *UP-unit-char*:
assumes *B.obj a*
shows *UP.unit a* = *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*
proof –
have *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* = *UP.unit a*
proof (*intro UP.unit-eqI*)
show *B.obj a*
using *assms* **by** *simp*
have *0*: «$a : a \Rightarrow_B a$»
using *assms* **by** *auto*
have *1*: *arr* (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*)
apply (*unfold arr-char*, *intro conjI*)
using *assms* **by** *auto*
have *2*: *arr* (*MkArr* $\langle a\rangle$ $\langle a\rangle$ *a*)
apply (*unfold arr-char*, *intro conjI*)
using *assms* **by** *auto*
have *3*: *arr* (*MkArr* $\langle a\rangle_0$ $\langle a\rangle_0$ *a*)
apply (*unfold arr-char*, *intro conjI*)
using *assms* **by** *auto*
show «*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* : $UP.map_0$ *a* ⇒ *UP a*»
using *assms 1 2 UP-def* $UP.map_0$*-def src-def*
by (*intro MkArr-in-hom*) *auto*
show *iso* (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*)
using *assms 1 iso-char* **by** *auto*
show *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* · i ($UP.map_0$ *a*) =
(*UP* $\mathrm{i}_B[a]$ · cmp_{UP} (*a*, *a*)) · (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* ⋆ *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*)
proof –
have *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* · i ($UP.map_0$ *a*) = *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*
unfolding i*-def* $UP.map_0$*-def UP-def*
using *assms 0 1 2 src-def* **by** *auto*
also have ... = (*UP* $\mathrm{i}_B[a]$ · cmp_{UP} (*a*, *a*)) · (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* ⋆ *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*)
proof –
have (*UP* $\mathrm{i}_B[a]$ · cmp_{UP} (*a*, *a*)) · (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* ⋆ *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*) =
(*MkArr* $\langle a \star_B a\rangle$ $\langle a\rangle$ $\mathrm{i}_B[a]$ · *MkArr* ($\langle a\rangle$ ⋆ $\langle a\rangle$) $\langle a \star_B a\rangle$ ($a \star_B a$))
· (*MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a* ⋆ *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ *a*)
using *assms UP-def* cmp_{UP}*-ide-simp* **by** *auto*
also have ... = (*MkArr* $\langle a \star_B a\rangle$ $\langle a\rangle$ $\mathrm{i}_B[a]$ · *MkArr* ($\langle a\rangle$ ⋆ $\langle a\rangle$) $\langle a \star_B a\rangle$ ($a \star_B a$))

$\cdot$ *MkArr* $\langle a\rangle_0$ $(\langle a\rangle \star \langle a\rangle)$ $(B.runit'\ a)$
using *assms 0 1 2 3 hcomp-def B.comp-cod-arr src-def trg-def*
B.can-Ide-self B.canE-unitor [*of* $\langle a\rangle_0$] *B.comp-cod-arr*
by *auto*
also have ... = *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ $((\mathrm{i}_B[a] \cdot_B (a \star_B a)) \cdot_B B.runit'\ a)$
proof −
have *MkArr* $\langle a \star_B a\rangle$ $\langle a\rangle$ $\mathrm{i}_B[a]$ $\cdot$ *MkArr* $(\langle a\rangle \star \langle a\rangle)$ $\langle a \star_B a\rangle$ $(a \star_B a)$ =
MkArr $(\langle a\rangle \star \langle a\rangle)$ $\langle a\rangle$ $(\mathrm{i}_B[a] \cdot_B (a \star_B a))$
using *assms*
by (*intro comp-MkArr arr-MkArr*) *auto*
moreover have *MkArr* $(\langle a\rangle \star \langle a\rangle)$ $\langle a\rangle$ $(\mathrm{i}_B[a] \cdot_B (a \star_B a))$
$\cdot$ *MkArr* $\langle a\rangle_0$ $(\langle a\rangle \star \langle a\rangle)$ $(B.runit'\ a)$ =
MkArr $\langle a\rangle_0$ $\langle a\rangle$ $((\mathrm{i}_B[a] \cdot_B (a \star_B a)) \cdot_B B.runit'\ a)$
using *assms 0 B.comp-arr-dom*
by (*intro comp-MkArr arr-MkArr*, *auto*)
ultimately show *?thesis* **by** *argo*
qed
also have ... = *MkArr* $\langle a\rangle_0$ $\langle a\rangle$ a
using *assms B.comp-arr-dom B.comp-arr-inv' B.iso-unit B.unitor-coincidence*(*2*)
by *simp*
finally show *?thesis* **by** *argo*
qed
finally show *?thesis* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed

end

1.11.4 Pseudofunctors into a Strict Bicategory

In the special case of a pseudofunctor into a strict bicategory, we can obtain explicit formulas for the images of the units and associativities under the pseudofunctor, which only involve the structure maps of the pseudofunctor, since the units and associativities in the target bicategory are all identities. This is useful in applying strictification.

locale *pseudofunctor-into-strict-bicategory* =
pseudofunctor +
D: *strict-bicategory* V_D H_D a_D i_D src_D trg_D
begin

lemma *image-of-unitor*:
assumes *C.ide g*
shows $F\ \mathrm{l}_C[g] = (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g))$
and $F\ \mathrm{r}_C[g] = (F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g))$
and $F\ (C.lunit'\ g) = \Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (trg_C\ g) \star_D F\ g)$
and $F\ (C.runit'\ g) = \Phi\ (g,\ src_C\ g) \cdot_D (F\ g \star_D unit\ (src_C\ g))$
proof −
show *A*: $F\ \mathrm{l}_C[g] = (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g))$

proof −
  have *1*: «$(D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g)) :$
      $F\ (trg_C\ g \star_C g) \Rightarrow_D F\ g$»
  proof
    show «$D.inv\ (unit\ (trg_C\ g)) \star_D F\ g : F\ (trg_C\ g) \star_D F\ g \Rightarrow_D F\ g$»
      using *assms unit-char* **by** (*auto simp add*: *D.hcomp-obj-arr*)
    show «$D.inv\ (\Phi\ (trg_C\ g,\ g)) : F\ (trg_C\ g \star_C g) \Rightarrow_D F\ (trg_C\ g) \star_D F\ g$»
      using *assms cmp-in-hom*(*2*) *D.inv-is-inverse* **by** *simp*
  qed
  have $(D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g)) =$
      $F\ g \cdot_D (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g))$
    using *1 D.comp-cod-arr* **by** *auto*
  also have ... $= (F\ \mathrm{l}_C[g] \cdot_D \Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (trg_C\ g) \star_D F\ g)) \cdot_D$
      $(D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g))$
    using *assms lunit-coherence* [*of g*] *D.strict-lunit* **by** *simp*
  also have ... $= F\ \mathrm{l}_C[g] \cdot_D \Phi\ (trg_C\ g,\ g) \cdot_D$
      $((unit\ (trg_C\ g) \star_D F\ g) \cdot_D (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g)) \cdot_D$
      $D.inv\ (\Phi\ (trg_C\ g,\ g))$
    using *D.comp-assoc* **by** *simp*
  also have ... $= F\ \mathrm{l}_C[g]$
  proof −
    have $(unit\ (trg_C\ g) \star_D F\ g) \cdot_D (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) = F\ (trg_C\ g) \star_D F\ g$
      using *assms unit-char D.whisker-right*
      by (*metis C.ideD*(*1*) *C.obj-trg C.trg.preserves-reflects-arr D.comp-arr-inv′*
        unit-simps(*5*) *preserves-arr preserves-ide*)
    moreover have $\Phi\ (trg_C\ g,\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g)) = F\ (trg_C\ g \star_C g)$
      using *assms D.comp-arr-inv D.inv-is-inverse* **by** *simp*
    ultimately show *?thesis*
      using *assms D.comp-arr-dom D.comp-cod-arr unit-char* **by** *auto*
  qed
  finally show *?thesis* **by** *simp*
qed
show *B*: $F\ \mathrm{r}_C[g] = (F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g))$
proof −
  have *1*: «$(F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) :$
      $F\ (g \star_C src_C\ g) \Rightarrow_D F\ g$»
  proof
    show «$F\ g \star_D D.inv\ (unit\ (src_C\ g)) : F\ g \star_D F\ (src_C\ g) \Rightarrow_D F\ g$»
      using *assms unit-char* **by** (*auto simp add*: *D.hcomp-arr-obj*)
    show «$D.inv\ (\Phi\ (g,\ src_C\ g)) : F\ (g \star_C src_C\ g) \Rightarrow_D F\ g \star_D F\ (src_C\ g)$»
      using *assms cmp-in-hom*(*2*) **by** *simp*
  qed
  have $(F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) =$
      $F\ g \cdot_D (F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g))$
    using *1 D.comp-cod-arr* **by** *auto*
  also have ... $= (F\ \mathrm{r}_C[g] \cdot_D \Phi\ (g,\ src_C\ g) \cdot_D (F\ g \star_D unit\ (src_C\ g))) \cdot_D$
      $(F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g))$
    using *assms runit-coherence* [*of g*] *D.strict-runit* **by** *simp*
  also have ... $= F\ \mathrm{r}_C[g] \cdot_D (\Phi\ (g,\ src_C\ g) \cdot_D ((F\ g \star_D unit\ (src_C\ g)) \cdot_D$

$(F\ g \star_D D.inv\ (unit\ (src_C\ g))))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g))$
using *D.comp-assoc* **by** *simp*
also have ... $= F\ \mathrm{r}_C[g]$
proof −
have $(F\ g \star_D unit\ (src_C\ g)) \cdot_D (F\ g \star_D D.inv\ (unit\ (src_C\ g))) = F\ g \star_D F\ (src_C\ g)$
using *assms D.whisker-left unit-char*
by (*metis C.ideD*(*1*) *C.obj-src C.src.preserves-ide D.comp-arr-inv′ D.ideD*(*1*) *unit-simps*(*5*) *preserves-ide*)
moreover have $\Phi\ (g,\ src_C\ g) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) = F\ (g \star_C src_C\ g)$
using *assms D.comp-arr-inv D.inv-is-inverse* **by** *simp*
ultimately show *?thesis*
using *assms D.comp-arr-dom D.comp-cod-arr unit-char cmp-in-hom*(*2*) [*of g* src_C *g*]
by *auto*
qed
finally show *?thesis* **by** *simp*
qed
show $F\ (C.lunit'\ g) = \Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (trg_C\ g) \star_D F\ g)$
proof −
have $F\ (C.lunit'\ g) = D.inv\ (F\ \mathrm{l}_C[g])$
using *assms C.iso-lunit preserves-inv* **by** *simp*
also have ... $= D.inv\ ((D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \cdot_D D.inv\ (\Phi\ (trg_C\ g,\ g)))$
using *A* **by** *simp*
also have ... $= \Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (trg_C\ g) \star_D F\ g)$
proof −
have $D.iso\ (D.inv\ (\Phi\ (trg_C\ g,\ g))) \wedge D.inv\ (D.inv\ (\Phi\ (trg_C\ g,\ g))) = \Phi\ (trg_C\ g,\ g)$
using *assms* **by** *simp*
moreover have $D.iso\ (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) \wedge$
$D.inv\ (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g) = unit\ (trg_C\ g) \star_D F\ g$
using *assms unit-char* **by** *simp*
moreover have $D.seq\ (D.inv\ (unit\ (trg_C\ g)) \star_D F\ g)\ (D.inv\ (\Phi\ (trg_C\ g,\ g)))$
using *assms unit-char* **by** (*metis A C.lunit-simps*(*1*) *preserves-arr*)
ultimately show *?thesis*
using *D.inv-comp* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
show $F\ (C.runit'\ g) = \Phi\ (g,\ src_C\ g) \cdot_D (F\ g \star_D unit\ (src_C\ g))$
proof −
have $F\ (C.runit'\ g) = D.inv\ (F\ \mathrm{r}_C[g])$
using *assms C.iso-runit preserves-inv* **by** *simp*
also have ... $= D.inv\ ((F\ g \star_D D.inv\ (unit\ (src_C\ g))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)))$
using *B* **by** *simp*
also have ... $= \Phi\ (g,\ src_C\ g) \cdot_D (F\ g \star_D unit\ (src_C\ g))$
proof −
have $D.iso\ (D.inv\ (\Phi\ (g,\ src_C\ g))) \wedge D.inv\ (D.inv\ (\Phi\ (g,\ src_C\ g))) = \Phi\ (g,\ src_C\ g)$
using *assms* **by** *simp*
moreover have $D.iso\ (F\ g \star_D D.inv\ (unit\ (src_C\ g))) \wedge$
$D.inv\ (F\ g \star_D D.inv\ (unit\ (src_C\ g))) = F\ g \star_D unit\ (src_C\ g)$
using *assms unit-char* **by** *simp*

moreover have $D.seq\ (F\ g \star_D D.inv\ (unit\ (src_C\ g)))\ (D.inv\ (\Phi\ (g,\ src_C\ g)))$
using *assms unit-char* **by** (*metis B C.runit-simps*(*1*) *preserves-arr*)
ultimately show *?thesis*
using *D.inv-comp* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
qed

lemma *image-of-associator*:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $C.ide\ h$ **and** $src_C\ f = trg_C\ g$ **and** $src_C\ g = trg_C\ h$
shows $F\ \mathrm{a}_C[f,\ g,\ h] = \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
and $F\ (C.\mathrm{a}'\ f\ g\ h) = \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ h))) \cdot_D D.inv\ (\Phi\ (f,\ g \star_C h))$
proof −
show *1*: $F\ \mathrm{a}_C[f,\ g,\ h] = \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
proof −
have *2*: $D.seq\ (\Phi\ (f,\ g \star_C h))\ ((F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h])$
using *assms D.assoc-in-hom*(*2*) [*of F f F g F h*] *cmp-simps*(*1*,*4*) *C.VV.cod-simp*
by (*intro D.seqI*) *auto*
moreover have *3*: $F\ \mathrm{a}_C[f,\ g,\ h] \cdot_D \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h) =$
$\Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$
using *assms assoc-coherence* [*of f g h*] **by** *blast*
moreover have $D.iso\ (\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h))$
using *assms 2 3* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def D.isos-compose*
by *auto*
ultimately have $F\ \mathrm{a}_C[f,\ g,\ h] =$
$(\Phi\ (f,\ g \star_C h) \cdot_D ((F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h])) \cdot_D$
$D.inv\ (\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h))$
using *D.invert-side-of-triangle*(*2*)
[*of* $\Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h]$
$F\ \mathrm{a}_C[f,\ g,\ h]\ \Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h)$]
by *presburger*
also have ... $= \Phi\ (f,\ g \star_C h) \cdot_D (F\ f \star_D \Phi\ (g,\ h)) \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
proof −
have $D.inv\ (\Phi\ (f \star_C g,\ h) \cdot_D (\Phi\ (f,\ g) \star_D F\ h)) =$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ h) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ h))$
proof −
have $D.seq\ (\Phi\ (f \star_C g,\ h))\ (\Phi\ (f,\ g) \star_D F\ h)$
using *assms* **by** *fastforce*
thus *?thesis*
using *assms D.inv-comp* **by** *simp*
qed
moreover have $(F\ f \star_D \Phi\ (g,\ h)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ h] = (F\ f \star_D \Phi\ (g,\ h))$
using *assms D.comp-arr-dom D.assoc-in-hom* [*of F f F g F h*] *cmp-in-hom*

```
        by (metis 2 3 D.comp-arr-ide D.hseq-char D.seqE D.strict-assoc
            cmp-simps(2) cmp-simps(3) preserves-ide)
      ultimately show ?thesis
        using D.comp-assoc by simp
    qed
    finally show ?thesis
      by simp
  qed
  show F (C.a' f g h) = Φ (f ⋆C g, h) ·D (Φ (f, g) ⋆D F h) ·D
                        (F f ⋆D D.inv (Φ (g, h))) ·D D.inv (Φ (f, g ⋆C h))
  proof −
    have F (C.a' f g h) = D.inv (F aC[f, g, h])
      using assms preserves-inv by simp
    also have ... = D.inv (Φ (f, g ⋆C h) ·D (F f ⋆D Φ (g, h)) ·D
                          (D.inv (Φ (f, g)) ⋆D F h) ·D D.inv (Φ (f ⋆C g, h)))
      using 1 by simp
    also have ... = Φ (f ⋆C g, h) ·D (Φ (f, g) ⋆D F h) ·D
                          (F f ⋆D D.inv (Φ (g, h))) ·D D.inv (Φ (f, g ⋆C h))
      using assms C.VV.arr-charSbC FF-def D.hcomp-assoc D.comp-assoc
            C.VV.dom-simp C.VV.cod-simp

      by (simp add: D.inv-comp D.isos-compose)
    finally show ?thesis by simp
  qed
qed

end
```

1.11.5 Internal Equivalences in a Strict Bicategory

In this section we prove a useful fact about internal equivalences in a strict bicategory: namely, that if the "right" triangle identity holds for such an equivalence then the "left" does, as well. Later we will dualize this property, and use strictification to extend it to arbitrary bicategories.

```
locale equivalence-in-strict-bicategory =
  strict-bicategory +
  equivalence-in-bicategory
begin

  lemma triangle-right-implies-left:
  shows (g ⋆ ε) · (η ⋆ g) = g ⟹ (ε ⋆ f) · (f ⋆ η) = f
  proof −
```

The formal proof here was constructed following the string diagram sketch below, which appears in [12] (see it also in context in [9]). The diagram is reproduced here by permission of its author, Mike Shulman, who says (private communication): "Just don't give the impression that the proof itself is due to me, because it's not. I don't know who first gave that proof; it's probably folklore."

assume *1*: $(g \star \varepsilon) \cdot (\eta \star g) = g$
have *2*: $(inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) = g$
proof −
  have $(inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) = inv\ ((g \star \varepsilon) \cdot (\eta \star g))$
    using *antipar inv-comp hcomp-assoc* **by** *simp*
  also have ... $= g$
    using *1* **by** *simp*
  finally show *?thesis* **by** *blast*
qed
have $(\varepsilon \star f) \cdot (f \star \eta) = (\varepsilon \star f) \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \star f) \cdot (f \star \eta)$
proof −
  have $(f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \star f) \cdot (f \star \eta) = f \star \eta$
    using *2 ide-left ide-right antipar whisker-left*
    by (*metis comp-cod-arr unit-simps*(*1*) *unit-simps*(*3*))
  thus *?thesis* **by** *simp*
qed
also have ... $= (\varepsilon \star f) \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \star f) \cdot (f \star \eta) \star (inv\ \eta \cdot \eta)$
proof −
  have $inv\ \eta \cdot \eta = src\ f$
    by (*simp add*: *comp-inv-arr′*)
  thus *?thesis*
    by (*metis antipar*(*1*) *antipar*(*2*) *arrI calculation*
      comp-ide-arr hcomp-arr-obj ideD(*1*) *ide-left ide-right obj-src seqE*
      strict-assoc′ triangle-in-hom(*1*) *vconn-implies-hpar*(*1*))
qed
also have ... $= (\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)) \star ((f \star inv\ \eta) \cdot (f \star \eta))) \cdot (f \star \eta)$
  using *ide-left ide-right antipar unit-is-iso*
  by (*metis 2 arr-inv calculation comp-arr-dom comp-inv-arr′ counit-simps*(*1*)
    counit-simps(*2*) *dom-inv hcomp-arr-obj ideD*(*1*) *unit-simps*(*1*) *unit-simps*(*2*)
    unit-simps(*5*) *obj-trg seqI whisker-left*)
also have ... $= (\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)) \star$

$((f \star inv\ \eta) \cdot ((inv\ \varepsilon \star f) \cdot (\varepsilon \star f)) \cdot (f \star \eta))) \cdot (f \star \eta)$
proof −
  have $(inv\ \varepsilon \star f) \cdot (\varepsilon \star f) = (f \star g) \star f$
    using *whisker-right* [*of f inv* ε ε] *counit-in-hom*
    by (*simp add*: *antipar*(*1*) *comp-inv-arr′*)
  thus *?thesis*
    using *hcomp-assoc comp-arr-dom*
    by (*metis comp-cod-arr ide-left local.unit-simps*(*1*) *local.unit-simps*(*3*) *whisker-left*)
qed
also have ... $= (((\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon))) \cdot (f \star g)) \star$
$(((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot (\varepsilon \star f) \cdot (f \star \eta))) \cdot$
$(f \star \eta)$
  using *ide-left ide-right antipar comp-assoc whisker-right comp-cod-arr*
  by (*metis 2 comp-arr-dom counit-simps*(*1*−*2*))
also have ... $= (((\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon))) \star ((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f))) \cdot$
$((f \star g) \star (\varepsilon \star f) \cdot (f \star \eta))) \cdot$
$(f \star \eta)$
proof −
  have *3*: *seq* $(\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)))\ (f \star g)$
    using *2 antipar* **by** *auto*
  moreover have *4*: *seq* $((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f))\ ((\varepsilon \star f) \cdot (f \star \eta))$
    using *antipar hcomp-assoc* **by** *auto*
  ultimately show *?thesis*
    using *interchange* **by** *simp*
qed
also have ... $= ((\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon))) \star ((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f))) \cdot$
$((f \star g) \star (\varepsilon \star f) \cdot (f \star \eta)) \cdot (f \star \eta)$
  using *comp-assoc* **by** *presburger*
also have ... $= ((\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot$
$((f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)) \star f)) \cdot$
$(f \star (g \star \varepsilon) \cdot (\eta \star g) \star f) \cdot (f \star \eta)$
proof −
  have $(\varepsilon \cdot (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon))) \star ((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) =$
$(\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot ((f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)) \star f)$
  proof −
    have *seq* $\varepsilon\ (f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon))$
      using *2 antipar* **by** *simp*
    moreover have *seq* $((f \star inv\ \eta) \cdot (inv\ \varepsilon \star f))\ f$
      using *antipar hcomp-assoc hcomp-obj-arr* **by** *auto*
    ultimately show *?thesis*
      using *comp-assoc comp-arr-dom hcomp-obj-arr*
interchange [*of* $\varepsilon\ f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon)\ (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)\ f$]
      by *simp*
  qed
  moreover have $((f \star g) \star (\varepsilon \star f) \cdot (f \star \eta)) \cdot (f \star \eta) =$
$(f \star (g \star \varepsilon) \cdot (\eta \star g) \star f) \cdot (f \star \eta)$
  proof −
    have $((f \star g) \star (\varepsilon \star f) \cdot (f \star \eta)) \cdot (f \star \eta) =$

$(f \star g \star \varepsilon \star f) \cdot (f \star (g \star f) \star \eta) \cdot (f \star \eta \star src\ f)$
using *antipar comp-assoc hcomp-assoc whisker-left hcomp-arr-obj* **by** *simp*
also have ... $= (f \star g \star \varepsilon \star f) \cdot (f \star ((g \star f) \star \eta) \cdot (\eta \cdot src\ f))$
using *antipar comp-arr-dom whisker-left hcomp-arr-obj* **by** *simp*
also have ... $= (f \star g \star \varepsilon \star f) \cdot (f \star \eta \star \eta)$
using *antipar comp-arr-dom comp-cod-arr hcomp-arr-obj*
interchange [*of* $g \star f\ \eta\ \eta\ src\ f$]
by *simp*
also have ... $= (f \star g \star \varepsilon \star f) \cdot (f \star \eta \star g \star f) \cdot (f \star src\ f \star \eta)$
using *antipar comp-arr-dom comp-cod-arr whisker-left*
interchange [*of* $\eta\ src\ f\ g \star f\ \eta$]
by *simp*
also have ... $= ((f \star g \star \varepsilon \star f) \cdot (f \star \eta \star g \star f)) \cdot (f \star \eta)$
using *antipar comp-assoc* **by** (*simp add*: *hcomp-obj-arr*)
also have ... $= (f \star (g \star \varepsilon) \cdot (\eta \star g) \star f) \cdot (f \star \eta)$
using *antipar comp-assoc whisker-left whisker-right hcomp-assoc* **by** *simp*
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
qed
also have ... $= (\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot$
$((f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \star f) \cdot$
$(f \star (g \star \varepsilon) \cdot (\eta \star g) \star f)) \cdot (f \star \eta)$
using *comp-assoc hcomp-assoc antipar*(*1*) *antipar*(*2*) **by** *auto*
also have ... $= (\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot$
$((f \star (inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \cdot (g \star \varepsilon) \cdot (\eta \star g) \star f)) \cdot$
$(f \star \eta)$
using *ide-left ide-right antipar comp-cod-arr comp-assoc whisker-left*
by (*metis 1 2 comp-ide-self unit-simps*(*1*) *unit-simps*(*3*))
also have ... $= (\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot (f \star \eta)$
proof −
have $(inv\ \eta \star g) \cdot (g \star inv\ \varepsilon) \cdot (g \star \varepsilon) \cdot (\eta \star g) = g$
using *ide-left ide-right antipar comp-arr-dom comp-assoc*
by (*metis 1 2 comp-ide-self*)
thus *?thesis*
using *antipar comp-cod-arr* **by** *simp*
qed
also have ... $= (f \star inv\ \eta) \cdot ((inv\ \varepsilon \star f) \cdot (\varepsilon \star f)) \cdot (f \star \eta)$
proof −
have $(\varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot (f \star \eta) =$
$(trg\ f \cdot \varepsilon \star (f \star inv\ \eta) \cdot (inv\ \varepsilon \star f)) \cdot (f \star \eta)$
using *hcomp-obj-arr comp-cod-arr* **by** *simp*
also have ... $= ((trg\ f \star f \star inv\ \eta) \cdot (\varepsilon \star inv\ \varepsilon \star f)) \cdot (f \star \eta)$
using *antipar hcomp-arr-obj hcomp-assoc interchange* **by** *auto*
also have ... $= (f \star inv\ \eta) \cdot ((inv\ \varepsilon \star f) \cdot (\varepsilon \star f)) \cdot (f \star \eta)$
proof −
have $(inv\ \varepsilon \star f) \cdot (\varepsilon \star f) = (trg\ f \star inv\ \varepsilon \star f) \cdot (\varepsilon \star trg\ f \star f)$
using *hseqI′* **by** (*simp add*: *hcomp-obj-arr*)
also have ... $= \varepsilon \star inv\ \varepsilon \star f$

```
        using antipar comp-arr-dom comp-cod-arr
              interchange [of trg f ε inv ε ⋆ f trg f ⋆ f]
        by force
      finally have (inv ε ⋆ f) · (ε ⋆ f) = ε ⋆ inv ε ⋆ f by simp
      moreover have trg f ⋆ f ⋆ inv η = f ⋆ inv η
        using hcomp-obj-arr [of trg f f ⋆ inv η] by fastforce
      ultimately have ((trg f ⋆ f ⋆ inv η) · (ε ⋆ inv ε ⋆ f)) · (f ⋆ η) =
                      ((f ⋆ inv η) · ((inv ε ⋆ f) · (ε ⋆ f))) · (f ⋆ η)
        by simp
      thus ?thesis
        using comp-assoc by simp
    qed
    finally show ?thesis by simp
  qed
  also have ... = f ⋆ inv η · η
  proof −
    have (inv ε ⋆ f) · (ε ⋆ f) = f ⋆ g ⋆ f
      using ide-left ide-right antipar counit-is-iso whisker-right hcomp-assoc
      by (metis comp-arr-dom comp-inv-arr′ counit-simps(1) counit-simps(2) seqE)
    thus ?thesis
      using ide-left ide-right antipar unit-is-iso comp-cod-arr
      by (metis arr-inv dom-inv unit-simps(1) unit-simps(3) seqI whisker-left)
  qed
  also have ... = f ⋆ src f
    using antipar by (simp add: comp-inv-arr′)
  also have ... = f
    using hcomp-arr-obj by simp
  finally show ?thesis by simp
qed

end
```

Now we use strictification to generalize the preceding result to arbitrary bicategories. I originally attempted to generalize this proof directly from the strict case, by filling in the necessary canonical isomorphisms, but it turned out to be too daunting. The proof using strictification is still fairly tedious, but it is manageable.

```
context equivalence-in-bicategory
begin

  interpretation S: strictified-bicategory V H a i src trg ..

  notation S.vcomp (infixr ‹·S› 55)
  notation S.hcomp (infixr ‹⋆S› 53)
  notation S.in-hom (‹«- : - ⇒S -»›)
  notation S.in-hhom (‹«- : - →S -»›)

  interpretation UP: equivalence-pseudofunctor V H a i src trg
                     S.vcomp S.hcomp S.a S.i S.src S.trg S.UP S.cmpUP
    using S.UP-is-equivalence-pseudofunctor by auto
```

interpretation *UP*: *pseudofunctor-into-strict-bicategory V H* a i *src trg*
S.vcomp S.hcomp S.a *S*.i *S.src S.trg S.UP* $S.cmp_{UP}$
..
interpretation *E*: *equivalence-in-bicategory S.vcomp S.hcomp S*.a *S*.i *S.src S.trg*
‹*S.UP f*› ‹*S.UP g*›
‹*S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η $\cdot_S$ *UP.unit* (*src f*)›
‹*S.inv* (*UP.unit* (*trg f*)) $\cdot_S$ *S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*)›
using *equivalence-in-bicategory-axioms UP.preserves-equivalence* [*of f g* η ε] **by** *auto*
interpretation *E*: *equivalence-in-strict-bicategory S.vcomp S.hcomp S*.a *S*.i *S.src S.trg*
‹*S.UP f*› ‹*S.UP g*›
‹*S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η $\cdot_S$ *UP.unit* (*src f*)›
‹*S.inv* (*UP.unit* (*trg f*)) $\cdot_S$ *S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*)›
..

lemma *UP-triangle*:
shows *S.UP* (($g \star \varepsilon$) $\cdot$ a[*g*, *f*, *g*] $\cdot$ ($\eta \star g$)) =
S.cmp_{UP} (*g*, *src g*) $\cdot_S$ (*S.UP g* $\star_S$ *S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*)) $\cdot_S$
(*S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η $\star_S$ *S.UP g*) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*trg g*, *g*))
and *S.UP* ($\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$) =
(*S.*cmp_{UP} (*g*, *src g*) $\cdot_S$ (*S.UP g* $\star_S$ *UP.unit* (*src g*))) $\cdot_S$
(*S.inv* (*UP.unit* (*trg g*)) $\star_S$ *S.UP g*) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*trg g*, *g*))
and *S.UP* (($\varepsilon \star f$) $\cdot$ a^{-1}[*f*, *g*, *f*] $\cdot$ ($f \star \eta$)) =
S.cmp_{UP} (*trg f*, *f*) $\cdot_S$ (*S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*) $\star_S$ *S.UP f*) $\cdot_S$
(*S.UP f* $\star_S$ *S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*f*, *src f*))
and *S.UP* ($\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$) =
(*S.*cmp_{UP} (*trg f*, *f*) $\cdot_S$ (*UP.unit* (*trg f*) $\star_S$ *S.UP f*)) $\cdot_S$
(*S.UP f* $\star_S$ *S.inv* (*UP.unit* (*src f*))) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*f*, *src f*))
and ($g \star \varepsilon$) $\cdot$ a[*g*, *f*, *g*] $\cdot$ ($\eta \star g$) = $\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$ $\Longrightarrow$
S.cmp_{UP} (*trg f*, *f*) $\cdot_S$ (*S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*) $\star_S$ *S.UP f*) $\cdot_S$
(*S.UP f* $\star_S$ *S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*f*, *src f*)) =
(*S.*cmp_{UP} (*trg f*, *f*) $\cdot_S$ (*UP.unit* (*trg f*) $\star_S$ *S.UP f*)) $\cdot_S$
(*S.UP f* $\star_S$ *S.inv* (*UP.unit* (*src f*))) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*f*, *src f*))
proof −
show *T1*: *S.UP* (($g \star \varepsilon$) $\cdot$ a[*g*, *f*, *g*] $\cdot$ ($\eta \star g$)) =
S.cmp_{UP} (*g*, *src g*) $\cdot_S$ (*S.UP g* $\star_S$ *S.UP* ε $\cdot_S$ *S.*cmp_{UP} (*f*, *g*)) $\cdot_S$
(*S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\cdot_S$ *S.UP* η $\star_S$ *S.UP g*) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*trg g*, *g*))
proof −
have *S.UP* (($g \star \varepsilon$) $\cdot$ a[*g*, *f*, *g*] $\cdot$ ($\eta \star g$)) =
S.UP ($g \star \varepsilon$) $\cdot_S$ *S.UP* a[*g*, *f*, *g*] $\cdot_S$ *S.UP* ($\eta \star g$)
using *antipar* **by** *simp*
also have ... =
(*S.*cmp_{UP} (*g*, *src g*) $\cdot_S$ (*S.UP g* $\star_S$ *S.UP* ε) $\cdot_S$
((*S.inv* (*S.*cmp_{UP} (*g*, $f \star g$)) $\cdot_S$ *S.*cmp_{UP} (*g*, $f \star g$)) $\cdot_S$
(*S.UP g* $\star_S$ *S.*cmp_{UP} (*f*, *g*))) $\cdot_S$
(((*S.inv* (*S.*cmp_{UP} (*g*, *f*)) $\star_S$ *S.UP g*) $\cdot_S$ (*S.inv* (*S.*cmp_{UP} ($g \star f$, *g*)))) $\cdot_S$
S.cmp_{UP} ($g \star f$, *g*)) $\cdot_S$ (*S.UP* η $\star_S$ *S.UP g*)) $\cdot_S$ *S.inv* (*S.*cmp_{UP} (*trg g*, *g*))
proof −
have *S.UP* a[*g*, *f*, *g*] =
S.cmp_{UP} (*g*, $f \star g$) $\cdot_S$ (*S.UP g* $\star_S$ *S.*cmp_{UP} (*f*, *g*)) $\cdot_S$

$(S.inv\ (S.cmp_{UP}\ (g,\ f))\ \star_S\ S.UP\ g)\ \cdot_S\ S.inv\ (S.cmp_{UP}\ (g \star f,\ g))$
using *ide-left ide-right antipar UP.image-of-associator* [*of g f g*] **by** *simp*
moreover have
$S.UP\ (g \star \varepsilon) =$
$S.cmp_{UP}\ (g,\ src\ g)\ \cdot_S\ (S.UP\ g \star_S S.UP\ \varepsilon)\ \cdot_S\ S.inv\ (S.cmp_{UP}\ (g,\ f \star g))$
proof −
have $S.seq\ (S.cmp_{UP}\ (g,\ src\ g))\ (S.UP\ g \star_S S.UP\ \varepsilon)$
using *antipar UP.FF-def UP.cmp-in-hom* [*of g src g*]
apply (*intro S.seqI*) **by** *auto*
moreover have
$S.UP\ (g \star \varepsilon)\ \cdot_S\ S.cmp_{UP}\ (g,\ f \star g) = S.cmp_{UP}\ (g,\ src\ g)\ \cdot_S\ (S.UP\ g \star_S S.UP\ \varepsilon)$
using *antipar* $UP.\Phi.naturality$ [*of* $(g,\ \varepsilon)$] *UP.FF-def* $VV.arr\text{-}char_{SbC}$
$VV.dom\text{-}simp\ VV.cod\text{-}simp$
by *simp*
moreover have $S.iso\ (S.cmp_{UP}\ (g,\ f \star g))$
using *antipar* **by** *simp*
ultimately show *?thesis*
using *S.invert-side-of-triangle*(*2*)
[*of* $S.cmp_{UP}\ (g,\ src\ g)\ \cdot_S\ (S.UP\ g \star_S S.UP\ \varepsilon)\ S.UP\ (g \star \varepsilon)$
$S.cmp_{UP}\ (g,\ f \star g)$] *S.comp-assoc*
by *presburger*
qed
moreover have $S.UP\ (\eta \star g) =$
$(S.cmp_{UP}\ (g \star f,\ g)\ \cdot_S\ (S.UP\ \eta \star_S S.UP\ g))\ \cdot_S\ S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
proof −
have $S.UP\ (\eta \star g)\ \cdot_S\ S.cmp_{UP}\ (trg\ g,\ g) =$
$S.cmp_{UP}\ (g \star f,\ g)\ \cdot_S\ (S.UP\ \eta \star_S S.UP\ g)$
using *antipar* $UP.\Phi.naturality$ [*of* $(\eta,\ g)$] *UP.FF-def* $VV.arr\text{-}char_{SbC}$
$VV.dom\text{-}simp\ VV.cod\text{-}simp$
by *simp*
moreover have $S.seq\ (S.cmp_{UP}\ (g \star f,\ g))\ (S.UP\ \eta \star_S S.UP\ g)$
using *antipar UP.cmp-in-hom*(*2*) **by** (*intro S.seqI*, *auto*)
ultimately show *?thesis*
using *antipar S.invert-side-of-triangle*(*2*) **by** *simp*
qed
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have ... $= S.cmp_{UP}\ (g,\ src\ g)\ \cdot_S$
$((S.UP\ g \star_S S.UP\ \varepsilon)\ \cdot_S\ (S.UP\ g \star_S S.cmp_{UP}\ (f,\ g)))\ \cdot_S$
$((S.inv\ (S.cmp_{UP}\ (g,\ f))\ \star_S S.UP\ g)\ \cdot_S\ (S.UP\ \eta \star_S S.UP\ g))\ \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
proof −
have $(S.inv\ (S.cmp_{UP}\ (g \star f,\ g))\ \cdot_S\ S.cmp_{UP}\ (g \star f,\ g))\ \cdot_S\ (S.UP\ \eta \star_S S.UP\ g) =$
$(S.UP\ \eta \star_S S.UP\ g)$
using *antipar S.comp-inv-arr′ S.comp-cod-arr* **by** *auto*
moreover have $(S.inv\ (S.cmp_{UP}\ (g,\ f \star g))\ \cdot_S\ S.cmp_{UP}\ (g,\ f \star g))\ \cdot_S$
$(S.UP\ g \star_S S.cmp_{UP}\ (f,\ g))$
$= (S.UP\ g \star_S S.cmp_{UP}\ (f,\ g))$

proof −
have $S.inv\ (S.cmp_{UP}\ (g,\ f \star g)) \cdot_S S.cmp_{UP}\ (g,\ f \star g) = S.UP\ g \star_S S.UP\ (f \star g)$
using *antipar S.comp-inv-arr′ UP.cmp-in-hom* **by** *auto*
thus *?thesis*
using *antipar VV.arr-char*$_{SbC}$ *S.comp-cod-arr* **by** *simp*
qed
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have ... $= S.cmp_{UP}\ (g,\ src\ g) \cdot_S (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
using *antipar VV.arr-char*$_{SbC}$ *S.whisker-left S.whisker-right* **by** *auto*
finally show *?thesis* **by** *simp*
qed
show *T2*: $S.UP\ (\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]) =$
$(S.cmp_{UP}\ (g,\ src\ g) \cdot_S (S.UP\ g \star_S UP.unit\ (src\ g))) \cdot_S$
$(S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g) \cdot_S S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
using *UP.image-of-unitor* **by** *simp*
show $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) =$
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) =$
$S.UP\ (\varepsilon \star f) \cdot_S S.UP\ \mathrm{a}^{-1}[f,\ g,\ f] \cdot_S S.UP\ (f \star \eta)$
using *antipar* **by** *simp*
also have ... $= S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \star_S S.UP\ f) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (f \star g,\ f)) \cdot_S S.cmp_{UP}\ (f \star g,\ f) \cdot_S$
$(S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.inv\ (S.cmp_{UP}\ (f,\ g \star f)) \cdot_S$
$S.cmp_{UP}\ (f,\ g \star f) \cdot_S (S.UP\ f \star_S S.UP\ \eta)) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $S.UP\ \mathrm{a}^{-1}[f,\ g,\ f] =$
$S.cmp_{UP}\ (f \star g,\ f) \cdot_S (S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}$
$(g,\ f))) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (f,\ g \star f))$
using *ide-left ide-right antipar UP.image-of-associator* **by** *simp*
moreover have $S.UP\ (\varepsilon \star f) =$
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \star_S S.UP\ f) \cdot_S S.inv\ (S.cmp_{UP}\ (f \star g,\ f))$
proof −
have $S.seq\ (S.cmp_{UP}\ (trg\ f,\ f))\ (S.UP\ \varepsilon \star_S S.UP\ f)$
using *antipar UP.FF-def VV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *UP.cmp-in-hom* [*of trg*
f f]
apply (*intro S.seqI*) **by** *auto*
moreover have
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \star_S S.UP\ f) = S.UP\ (\varepsilon \star f) \cdot_S S.cmp_{UP}\ (f \star g,\ f)$
using *antipar UP.*Φ*.naturality* [*of* $(\varepsilon,\ f)$] *UP.FF-def VV.arr-char*$_{SbC}$
VV.dom-simp VV.cod-simp
by *simp*

moreover have $S.iso\ (S.cmp_{UP}\ (f \star g,\ f))$
using *antipar* **by** *simp*
ultimately show *?thesis*
using *antipar S.comp-assoc*
$S.invert\text{-}side\text{-}of\text{-}triangle(2)$
$[of\ S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \star_S S.UP\ f)\ S.UP\ (\varepsilon \star f)$
$S.cmp_{UP}\ (f \star g,\ f)]$
by *metis*
qed
moreover have $S.UP\ (f \star \eta) =$
$(S.cmp_{UP}\ (f,\ g \star f) \cdot_S (S.UP\ f \star_S S.UP\ \eta)) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $S.cmp_{UP}\ (f,\ g \star f) \cdot_S (S.UP\ f \star_S S.UP\ \eta) =$
$S.UP\ (f \star \eta) \cdot_S S.cmp_{UP}\ (f,\ src\ f)$
using *antipar* $UP.\Phi.naturality\ [of\ (f,\ \eta)]$ *UP.FF-def* $VV.arr\text{-}char_{SbC}$
$VV.dom\text{-}simp\ VV.cod\text{-}simp$
by *simp*
moreover have $S.seq\ (S.cmp_{UP}\ (f,\ g \star f))\ (S.UP\ f \star_S S.UP\ \eta)$
using *antipar* **by** (*intro* $S.seqI$, *auto*)
ultimately show *?thesis*
using *antipar* $S.invert\text{-}side\text{-}of\text{-}triangle(2)$ **by** *auto*
qed
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have $... = S.cmp_{UP}\ (trg\ f,\ f) \cdot_S$
$((S.UP\ \varepsilon \star_S S.UP\ f) \cdot_S (S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f)) \cdot_S$
$((S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.UP\ f \star_S S.UP\ \eta)) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $(S.inv\ (S.cmp_{UP}\ (f \star g,\ f)) \cdot_S S.cmp_{UP}\ (f \star g,\ f)) \cdot_S (S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) =$
$(S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f)$
using *antipar* $S.comp\text{-}cod\text{-}arr\ VV.arr\text{-}char_{SbC}\ S.comp\text{-}inv\text{-}arr'$ **by** *auto*
moreover have $(S.inv\ (S.cmp_{UP}\ (f,\ g \star f)) \cdot_S S.cmp_{UP}\ (f,\ g \star f)) \cdot_S$
$(S.UP\ f \star_S S.UP\ \eta)$
$= (S.UP\ f \star_S S.UP\ \eta)$
using *antipar* $S.comp\text{-}inv\text{-}arr'\ S.comp\text{-}cod\text{-}arr$ **by** *auto*
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have $... = S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S$
$S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
using *antipar* $VV.arr\text{-}char_{SbC}\ S.whisker\text{-}left\ S.whisker\text{-}right$ **by** *auto*
finally show *?thesis* **by** *simp*
qed
show $S.UP\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]) =$
$(S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ (trg\ f) \star_S S.UP\ f)) \cdot_S$

$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
using *UP.image-of-unitor* **by** *simp*
show $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \Longrightarrow$
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f)) =$
$(S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ (trg\ f) \star_S S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
assume *A*: $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
have *B*: $(S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g)) \cdot_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ (trg\ g) \star_S S.UP\ g) = S.UP\ g$
proof −
show $(S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g)) \cdot_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ (trg\ g) \star_S S.UP\ g) = S.UP\ g$
proof −
have *2*: $S.cmp_{UP}\ (g,\ src\ g) \cdot_S (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
$= (S.cmp_{UP}\ (g,\ src\ g) \cdot_S (S.UP\ g \star_S UP.unit\ (src\ g))) \cdot_S$
$(S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g) \cdot_S S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
using *A T1 T2* **by** *simp*
show *?thesis*
proof −
have *8*: $S.seq\ (S.cmp_{UP}\ (g,\ src\ g))$
$((S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)))$
using *antipar VV.arr-char*$_{SbC}$ *S.hcomp-assoc*
by (*metis* (*no-types*, *lifting*) *S.arr-UP T1 arrI triangle-in-hom*(*2*))
have *7*: $S.seq\ (S.cmp_{UP}\ (g,\ src\ g))$
$((S.UP\ g \star_S UP.unit\ (src\ g)) \cdot_S (S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)))$
using *antipar 2 8 S.comp-assoc* **by** *auto*
have *5*: $(S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g) =$
$(S.UP\ g \star_S UP.unit\ (src\ g)) \cdot_S (S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g)$
proof −
have $((S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S (S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S$
$S.UP\ \eta \star_S S.UP\ g)) \cdot_S S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)) =$
$((S.UP\ g \star_S UP.unit\ (src\ g)) \cdot_S (S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g)) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (trg\ g,\ g))$
proof −
have $S.mono\ (S.cmp_{UP}\ (g,\ src\ g))$
using *antipar S.iso-is-section S.section-is-mono* **by** *simp*
thus *?thesis*
using *2 8 7 S.mono-cancel S.comp-assoc* **by** *presburger*
qed
moreover have $S.epi\ (S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)))$
using *antipar S.iso-is-retraction S.retraction-is-epi* **by** *simp*

moreover have $S.seq\ ((S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S$
$S.UP\ \eta \star_S S.UP\ g))$
$(S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)))$
using *S.comp-assoc S.seq-char 8* **by** *presburger*
moreover have
$S.seq\ ((S.UP\ g \star_S UP.unit\ (src\ g)) \cdot_S (S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g))$
$(S.inv\ (S.cmp_{UP}\ (trg\ g,\ g)))$
using *antipar calculation(1,3)* **by** *presburger*
ultimately show *?thesis*
using *2 S.epi-cancel* **by** *blast*
qed
have *6*: $S.seq\ (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g))$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)$
using *antipar* $VV.arr\text{-}char_{SbC}$ *S.hcomp-assoc* **by** *auto*
have *3*: $(S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g))) \cdot_S (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,$
$g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g) =$
$(S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g)$
proof −
have $S.seq\ (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g))$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)$
using *6* **by** *simp*
moreover have $(S.UP\ g \star_S UP.unit\ (src\ g)) \cdot_S (S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP$
$g) =$
$(S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)$
using *5* **by** *argo*
moreover have $S.iso\ (S.UP\ g \star_S UP.unit\ (src\ g))$
using *antipar UP.unit-char* $S.UP\text{-}map_0\text{-}obj$ **by** *simp*
moreover have $S.inv\ (S.UP\ g \star_S UP.unit\ (src\ g)) =$
$S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g))$
using *antipar UP.unit-char* $S.UP\text{-}map_0\text{-}obj$ **by** *simp*
ultimately show *?thesis*
using *S.invert-side-of-triangle(1)*
$[of\ (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g)) \cdot_S$
$(S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)$
$S.UP\ g \star_S UP.unit\ (src\ g)\ S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g]$
by *presburger*
qed
have *4*: $((S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g))) \cdot_S$
$(S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g))) \cdot_S$
$((S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)) \cdot_S$
$(UP.unit\ (trg\ g) \star_S S.UP\ g)$
$= S.UP\ g$
proof −
have $(((S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g))) \cdot_S (S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,$
$g))) \cdot_S$
$((S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g)) \cdot_S$

$(UP.unit\ (trg\ g) \star_S S.UP\ g)) =$
$(((S.UP\ g \star_S S.inv\ (UP.unit\ (src\ g))) \cdot_S$
$(S.UP\ g \star_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g))) \cdot_S$
$((S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \star_S S.UP\ g))) \cdot_S$
$(UP.unit\ (trg\ g) \star_S S.UP\ g)$
using *S.comp-assoc* **by** *simp*
also have ... =
$(S.inv\ (UP.unit\ (trg\ g)) \star_S S.UP\ g) \cdot_S (UP.unit\ (trg\ g) \star_S S.UP\ g)$
using *3 S.comp-assoc* **by** *auto*
also have ... = $S.inv\ (UP.unit\ (trg\ g)) \cdot_S UP.unit\ (trg\ g) \star_S S.UP\ g$
using *UP.unit-char*(*2*) *S.whisker-right* **by** *auto*
also have ... = $UP.map_0\ (trg\ g) \star_S S.UP\ g$
using *UP.unit-char* [*of trg g*] *S.comp-inv-arr S.inv-is-inverse* **by** *simp*
also have ... = $S.UP\ g$
using $S.UP\text{-}map_0\text{-}obj$ **by** (*simp add*: *S.hcomp-obj-arr*)
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *antipar S.whisker-left S.whisker-right UP.unit-char S.comp-assoc* **by** *simp*
qed
qed
qed
show $S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f)) =$
$(S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ (trg\ f) \star_S S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $(S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S$
$S.UP\ \eta) =$
$(UP.unit\ (trg\ f) \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f)))$
proof −
have *2*: $(S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f) \cdot_S$
$((S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta)) \cdot_S$
$(S.UP\ f \star_S UP.unit\ (src\ f)) =$
$S.UP\ f$
proof −
have $S.UP\ f = (S.inv\ (UP.unit\ (trg\ f)) \cdot_S S.UP\ \varepsilon \cdot_S$
$S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ (src\ f))$
using *B antipar E.triangle-right-implies-left* **by** *argo*
also have ... = $(S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f) \cdot_S$
$((S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta)) \cdot_S$
$(S.UP\ f \star_S UP.unit\ (src\ f))$
proof −
have $S.inv\ (UP.unit\ (trg\ f)) \cdot_S S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f =$
$(S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f) \cdot_S$
$(S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f)$

using *UP.unit-char S.whisker-right* **by** *simp*
moreover have $S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ (src\ f) =$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S$
$(S.UP\ f \star_S UP.unit\ (src\ f))$
using *antipar UP.unit-char S.whisker-left S.comp-assoc* **by** *simp*
ultimately show *?thesis*
using *S.comp-assoc* **by** *presburger*
qed
finally show *?thesis* **by** *argo*
qed
show *?thesis*
proof −
have $((S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)) \cdot_S$
$(S.UP\ f \star_S UP.unit\ (src\ f)) =$
$(UP.unit\ (trg\ f) \star_S S.UP\ f)$
proof −
have $S.inv\ (S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f) \cdot_S S.UP\ f =$
$UP.unit\ (trg\ f) \star_S S.UP\ f$
using *UP.unit-char S.comp-arr-dom S.UP-map*$_0$*-obj*
by (*simp add*: *S.hcomp-obj-arr*)
moreover have $S.arr\ (S.UP\ f)$
by *simp*
moreover have $S.iso\ (S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f)$
using *S.UP-map*$_0$*-obj* **by** (*simp add*: *UP.unit-char*(*2*))
ultimately show *?thesis*
using *2 S.invert-side-of-triangle*(*1*)
[*of* $S.UP\ f\ S.inv\ (UP.unit\ (trg\ f)) \star_S S.UP\ f$
$((S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)) \cdot_S$
$(S.UP\ f \star_S UP.unit\ (src\ f))$]
by *presburger*
qed
moreover have $S.hseq\ (UP.unit\ (trg\ f))\ (S.UP\ f) \wedge$
$S.iso\ (S.UP\ f \star_S UP.unit\ (src\ f)) \wedge$
$S.inv\ (S.UP\ f \star_S UP.unit\ (src\ f)) = S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))$
using *UP.unit-char S.UP-map*$_0$*-obj* **by** *simp*
ultimately show *?thesis*
using *S.invert-side-of-triangle*(*2*)
[*of* $UP.unit\ (trg\ f) \star_S S.UP\ f$
$(S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)$
$S.UP\ f \star_S UP.unit\ (src\ f)$]
by *presburger*
qed
qed
hence $S.cmp_{UP}\ (trg\ f, f) \cdot_S ((S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)) \cdot_S S.inv\ (S.cmp_{UP}\ (f, src\ f)) =$
$S.cmp_{UP}\ (trg\ f, f) \cdot_S ((UP.unit\ (trg\ f) \star_S S.UP\ f) \cdot_S$

$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f)))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
by *simp*
thus *?thesis*
using *S.comp-assoc* **by** *simp*
qed
qed
qed

lemma *triangle-right-implies-left*:
assumes $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
shows $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
proof −
have *par* $((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta))\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f])$
using *antipar* **by** *simp*
moreover have $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) = S.UP\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f])$
using *assms UP-triangle*(*3*−*5*) **by** *simp*
ultimately show $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
using *UP.is-faithful* **by** *blast*
qed

We *really* don't want to go through the ordeal of proving a dual version of *UP-triangle*(*5*), do we? So let's be smart and dualize via the opposite bicategory.

lemma *triangle-left-implies-right*:
assumes $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
shows $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
proof −
interpret *Cop*: *op-bicategory V H* a i *src trg* **..**
interpret *Eop*: *equivalence-in-bicategory V Cop.H Cop*.a i *Cop.src Cop.trg g f* η ε
using *antipar unit-in-hom counit-in-hom*
by (*unfold-locales*) *simp-all*
have $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \Longrightarrow$
$(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
using *antipar Cop.lunit-ide-simp Cop.runit-ide-simp Cop.assoc-ide-simp*
VVV.ide-char VVV.arr-char$_{SbC}$ *VV.arr-char*$_{SbC}$ *Eop.triangle-right-implies-left*
by *simp*
thus *?thesis*
using *assms* **by** *simp*
qed

lemma *triangle-left-iff-right*:
shows $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \longleftrightarrow$
$(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
using *triangle-left-implies-right triangle-right-implies-left* **by** *auto*

end

We might as well specialize the dual result back to the strict case while we are at it.

context *equivalence-in-strict-bicategory*
begin

lemma *triangle-left-iff-right*:
shows $(\varepsilon \star f) \cdot (f \star \eta) = f \longleftrightarrow (g \star \varepsilon) \cdot (\eta \star g) = g$
proof −
have $(\varepsilon \star f) \cdot (f \star \eta) = f \longleftrightarrow (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
proof −
have $\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] = f$
using *strict-lunit strict-runit* **by** *simp*
moreover have $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = (\varepsilon \star f) \cdot (f \star \eta)$
using *antipar strict-assoc assoc′-in-hom*(*2*) [*of f g f*] *comp-cod-arr*
by *auto*
ultimately show *?thesis* **by** *simp*
qed
also have ... $\longleftrightarrow (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
using *triangle-left-iff-right* **by** *blast*
also have ... $\longleftrightarrow (g \star \varepsilon) \cdot (\eta \star g) = g$
proof −
have $\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] = g$
using *strict-lunit strict-runit* **by** *simp*
moreover have $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = (g \star \varepsilon) \cdot (\eta \star g)$
using *antipar strict-assoc assoc-in-hom*(*2*) [*of g f g*] *comp-cod-arr*
by *auto*
ultimately show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed

end

end

1.12 Bicategory of Categories

In this section we construct a bicategory whose objects correspond to categories having arrows in a given type, whose 1-cells correspond to functors between such categories, and whose 2-cells correspond to natural transformations between such functors. We show that the bicategory that results from the construction is strict.

theory *CatBicat*
imports *Bicategory.Strictness ConcreteBicategory*
begin

locale *catbicat*
begin

abbreviation *ARR*
where *ARR A B* ≡ *partial-composition.arr* (*functor-category.comp A B*)

abbreviation *MKARR*

where *MKARR* ≡ *concrete-category.MkArr*

abbreviation *MAP*
where *MAP* ≡ *concrete-category.Map*

abbreviation *DOM*
where *DOM* ≡ *concrete-category.Dom*

abbreviation *COD*
where *COD* ≡ *concrete-category.Cod*

abbreviation *NULL*
where *NULL* ≡ *concrete-category.Null*

abbreviation *OBJ*
where *OBJ* ≡ *Collect category*

abbreviation *HOM*
where *HOM* ≡ *functor-category.comp*

abbreviation *COMP*
where *COMP C B A* μ ν ≡ *if ARR B C* μ ∧ *ARR A B* ν
then MKARR (*DOM* μ *o DOM* ν) (*COD* μ *o COD* ν) (*MAP* μ *o MAP* ν)
else NULL

abbreviation *ID*
where *ID A* ≡ *MKARR* (*identity-functor.map A*) (*identity-functor.map A*)
(*identity-functor.map A*)

abbreviation *ASSOC*
where *ASSOC D C B A* τ μ ν ≡
if ARR C D τ ∧ *ARR B C* μ ∧ *ARR A B* ν *then*
MKARR (*DOM* τ *o DOM* μ *o DOM* ν) (*COD* τ *o COD* μ *o COD* ν)
(*MAP* τ *o MAP* μ *o MAP* ν)
else NULL

Although we are using the *concrete-bicategory* construction to take care of some of the details, the proof is still awkward, because the locale assumptions we need to verify are all conditioned on universally quantified entities being in the set *OBJ*, and we cannot create interpretations to unpack these conditions until we are within a proof context where these entities have been fixed and the conditions have been introduced as assumptions. So, for example, to prove that *COMP* has the required functoriality property, we have to fix A, B, and C and introduce assumptions $A \in OBJ$, $B \in OBJ$, and $C \in OBJ$, and only then can we use these assumptions to interpret A, B, and C as categories and *HOM A B*, *HOM B C*, and *HOM A C* as functor categories. We have to go into a still deeper proof context before we can fix particular arguments μ and ν to *COMP C B A*, introduce the assumptions that they are arrows of their respective hom-categories, and finally use those assumptions to interpret them as natural transformations. At that point, we are finally

in a position to apply the already-proved interchange law for natural transformations, which is the essential core of the functoriality property we need to show.

```
sublocale concrete-bicategory OBJ HOM ID COMP ID ASSOC
proof (unfold concrete-bicategory-def, intro conjI allI impI)
 show 1: ⋀(A :: 'a comp) (B :: 'a comp). ⟦ A ∈ OBJ; B ∈ OBJ ⟧ ⟹ category (HOM A B)
   using functor-category-def functor-category.is-category by auto
 show COMP:
     ⋀(A :: 'a comp) (B :: 'a comp) (C :: 'a comp).
       ⟦ A ∈ OBJ; B ∈ OBJ; C ∈ OBJ ⟧
         ⟹ binary-functor (HOM B C) (HOM A B) (HOM A C)
                          (λ(μ, ν). COMP C B A μ ν)
 proof −
   fix A :: 'a comp and B :: 'a comp and C :: 'a comp
   assume A: A ∈ OBJ and B: B ∈ OBJ and C: C ∈ OBJ
   let ?COMP = λ(μ, ν). COMP C B A μ ν
   interpret A: category A using A by simp
   interpret B: category B using B by simp
   interpret C: category C using C by simp
   interpret BC: functor-category B C ..
   interpret AB: functor-category A B ..
   interpret AC: functor-category A C ..
   interpret BC-AB: product-category BC.comp AB.comp ..
   show binary-functor (HOM B C) (HOM A B) (HOM A C) (λ(μ, ν). COMP C B A μ ν)
   proof
     show ⋀μν. ¬ BC-AB.arr μν ⟹ ?COMP μν = AC.null
       using BC-AB.arr-char AC.null-char AC.arr-MkArr by auto
     fix μν
     assume μν: BC-AB.arr μν
     interpret μ: natural-transformation B C
                 ‹AC.Dom (fst μν)› ‹AC.Cod (fst μν)› ‹MAP (fst μν)›
       using μν BC.arr-char [of fst μν] by simp
     interpret ν: natural-transformation A B
                 ‹AC.Dom (snd μν)› ‹AC.Cod (snd μν)› ‹MAP (snd μν)›
       using μν AB.arr-char [of snd μν] by simp
     interpret μν: natural-transformation A C
                 ‹AC.Dom (fst μν) o AC.Dom (snd μν)›
                 ‹AC.Cod (fst μν) o AC.Cod (snd μν)›
                 ‹MAP (fst μν) o MAP (snd μν)›
       using μ.natural-transformation-axioms ν.natural-transformation-axioms
             horizontal-composite
       by blast
     show AC.arr (?COMP μν)
       using μν BC-AB.arr-char μν.natural-transformation-axioms
             μν.F.functor-axioms μν.G.functor-axioms AC.arr-MkArr
       by (cases μν) simp
     show AC.dom (?COMP μν) = ?COMP (BC-AB.dom μν)
       using μν BC-AB.arr-char μν.natural-transformation-axioms
             μν.F.functor-axioms μν.G.functor-axioms AC.arr-MkArr
       by (cases μν) simp
```

show *AC.cod* (*?COMP* $\mu\nu$) = *?COMP* (*BC-AB.cod* $\mu\nu$)
  using $\mu\nu$ *BC-AB.arr-char* $\mu\nu$*.natural-transformation-axioms*
    $\mu\nu$*.F.functor-axioms* $\mu\nu$*.G.functor-axioms AC.arr-MkArr*
  by (*cases* $\mu\nu$) *simp*
next
fix $\mu\nu$ $\sigma\tau$
assume *seq*: *BC-AB.seq* $\mu\nu$ $\sigma\tau$
interpret μ: *natural-transformation B C*
    ‹*BC.Dom* (*fst* $\mu\nu$)› ‹*BC.Cod* (*fst* $\mu\nu$)› ‹*MAP* (*fst* $\mu\nu$)›
  using *seq BC-AB.seq-char BC-AB.arr-char BC.arr-char* [*of fst* $\mu\nu$] *AB.arr-char*
  by *auto*
interpret ν: *natural-transformation A B*
    ‹*AB.Dom* (*snd* $\mu\nu$)› ‹*AB.Cod* (*snd* $\mu\nu$)› ‹*MAP* (*snd* $\mu\nu$)›
  using *seq BC-AB.seq-char BC-AB.arr-char BC.arr-char AB.arr-char* [*of snd* $\mu\nu$]
  by *auto*
interpret $\mu\nu$: *natural-transformation A C*
    ‹*AC.Dom* (*fst* $\mu\nu$) *o AC.Dom* (*snd* $\mu\nu$)›
    ‹*AC.Cod* (*fst* $\mu\nu$) *o AC.Cod* (*snd* $\mu\nu$)›
    ‹*MAP* (*fst* $\mu\nu$) *o MAP* (*snd* $\mu\nu$)›
  using μ*.natural-transformation-axioms* ν*.natural-transformation-axioms*
    horizontal-composite
  by *blast*
interpret σ: *natural-transformation B C*
    ‹*BC.Dom* (*fst* $\sigma\tau$)› ‹*BC.Cod* (*fst* $\sigma\tau$)› ‹*MAP* (*fst* $\sigma\tau$)›
  using *seq BC-AB.seq-char BC-AB.arr-char BC.arr-char* [*of fst* $\sigma\tau$] *AB.arr-char*
  by *auto*
interpret τ: *natural-transformation A B*
    ‹*AB.Dom* (*snd* $\sigma\tau$)› ‹*AB.Cod* (*snd* $\sigma\tau$)› ‹*MAP* (*snd* $\sigma\tau$)›
  using *seq BC-AB.seq-char BC-AB.arr-char BC.arr-char AB.arr-char* [*of snd* $\sigma\tau$]
  by *auto*
interpret $\sigma\tau$: *natural-transformation A C*
    ‹*AC.Dom* (*fst* $\sigma\tau$) *o AC.Dom* (*snd* $\sigma\tau$)›
    ‹*AC.Cod* (*fst* $\sigma\tau$) *o AC.Cod* (*snd* $\sigma\tau$)›
    ‹*MAP* (*fst* $\sigma\tau$) *o MAP* (*snd* $\sigma\tau$)›
  using σ*.natural-transformation-axioms* τ*.natural-transformation-axioms*
    horizontal-composite
  by *blast*
interpret $\mu o\sigma$: *vertical-composite B C*
    ‹*BC.Dom* (*fst* $\sigma\tau$)› ‹*BC.Cod* (*fst* $\sigma\tau$)› ‹*BC.Cod* (*fst* $\mu\nu$)›
    ‹*MAP* (*fst* $\sigma\tau$)› ‹*MAP* (*fst* $\mu\nu$)›
  using *seq BC-AB.seq-char BC.seq-char* μ*.natural-transformation-axioms*
  by *intro-locales* (*simp add*: *natural-transformation-def*)
interpret $\nu o\tau$: *vertical-composite A B*
    ‹*AB.Dom* (*snd* $\sigma\tau$)› ‹*AB.Cod* (*snd* $\sigma\tau$)› ‹*AB.Cod* (*snd* $\mu\nu$)›
    ‹*MAP* (*snd* $\sigma\tau$)› ‹*MAP* (*snd* $\mu\nu$)›
  using *seq BC-AB.seq-char AB.seq-char* ν*.natural-transformation-axioms*
  by *intro-locales* (*simp add*: *natural-transformation-def*)

show *?COMP* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$) = *AC.comp* (*?COMP* $\mu\nu$) (*?COMP* $\sigma\tau$)

proof −
have *?COMP* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$) =
AC.MkArr
(*BC.Dom* (*fst* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)) ∘ *AB.Dom* (*snd* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)))
(*BC.Cod* (*fst* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)) ∘ *AB.Cod* (*snd* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)))
(*BC.Map* (*fst* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)) ∘ *AB.Map* (*snd* (*BC-AB.comp* $\mu\nu$ $\sigma\tau$)))
using *seq BC-AB.arr-char* **by** (*cases BC-AB.comp* $\mu\nu$ $\sigma\tau$, *simp*)
also have
... =
AC.MkArr
(*BC.Dom* (*fst* $\sigma\tau$) ∘ *AB.Dom* (*snd* $\sigma\tau$)) (*BC.Cod* (*fst* $\mu\nu$) ∘ *AB.Cod* (*snd* $\mu\nu$))
(*vertical-composite.map B C* (*BC.Map* (*fst* $\sigma\tau$)) (*AB.Map* (*fst* $\mu\nu$)) ∘
vertical-composite.map A B (*BC.Map* (*snd* $\sigma\tau$)) (*AB.Map* (*snd* $\mu\nu$)))
using *seq BC-AB.comp-char BC-AB.seq-char BC-AB.arr-char BC.seq-char AB.seq-char*
BC.Dom-comp BC.Cod-comp BC.Map-comp
AB.Dom-comp AB.Cod-comp AB.Map-comp
by *auto*
also have ... = *AC.MkArr*
(*BC.Dom* (*fst* $\sigma\tau$) ∘ *AB.Dom* (*snd* $\sigma\tau$))
(*BC.Cod* (*fst* $\mu\nu$) ∘ *AB.Cod* (*snd* $\mu\nu$))
(*vertical-composite.map A C*
(*AC.Map* (*fst* $\sigma\tau$) ∘ *AC.Map* (*snd* $\sigma\tau$))
(*AC.Map* (*fst* $\mu\nu$) ∘ *AC.Map* (*snd* $\mu\nu$)))
using σ*.natural-transformation-axioms* τ*.natural-transformation-axioms*
μ*.natural-transformation-axioms* ν*.natural-transformation-axioms*
$\mu o\sigma$*.*τ*.natural-transformation-axioms* $\nu o\tau$*.*τ*.natural-transformation-axioms*
interchange
by *blast*
also have ... = *AC.comp* (*AC.MkArr* (*AC.Dom* (*fst* $\mu\nu$) ∘ *AC.Dom* (*snd* $\mu\nu$))
(*AC.Cod* (*fst* $\mu\nu$) ∘ *AC.Cod* (*snd* $\mu\nu$))
(*AC.Map* (*fst* $\mu\nu$) ∘ *AC.Map* (*snd* $\mu\nu$)))
(*AC.MkArr* (*AC.Dom* (*fst* $\sigma\tau$) ∘ *AC.Dom* (*snd* $\sigma\tau$))
(*AC.Cod* (*fst* $\sigma\tau$) ∘ *AC.Cod* (*snd* $\sigma\tau$))
(*AC.Map* (*fst* $\sigma\tau$) ∘ *AC.Map* (*snd* $\sigma\tau$)))
using *seq BC-AB.seq-char BC.seq-char AB.seq-char AC.comp-MkArr*
$\sigma\tau$*.natural-transformation-axioms* $\mu\nu$*.natural-transformation-axioms*
by *simp*
also have ... = *AC.comp* (*?COMP* $\mu\nu$) (*?COMP* $\sigma\tau$)
using *seq BC-AB.seq-char BC-AB.arr-char BC.seq-char AB.seq-char*
BC.Dom-comp BC.Cod-comp BC.Map-comp
AB.Dom-comp AB.Cod-comp AB.Map-comp
by (*cases* $\mu\nu$, *cases* $\sigma\tau$) *simp*
finally show *?thesis* **by** *blast*
qed
qed
qed
show *2*: $\bigwedge$(*A* :: *'a comp*). *A* ∈ *OBJ* ⟹ *partial-composition.ide* (*HOM A A*) (*ID A*)
using *concrete-category.ide-MkIde functor-category.is-concrete-category*

functor-category-def identity-functor.intro identity-functor.is-functor
by *fastforce*
show $\bigwedge$(*A* :: *'a comp*). *A* ∈ *OBJ*
⟹ *partial-composition.in-hom* (*HOM A A*) (*ID A*)
(*COMP A A A* (*ID A*) (*ID A*)) (*ID A*)
proof −
fix *A* :: *'a comp*
assume *A*: *A* ∈ *OBJ*
interpret *A*: *category A*
using *A* **by** *simp*
interpret *AA*: *functor-category A A* **..**
show *AA.in-hom* (*ID A*) (*COMP A A A* (*ID A*) (*ID A*)) (*ID A*)
using *A.is-functor* **by** *force*
qed
show $\bigwedge$(*A* :: *'a comp*). *A* ∈ *OBJ* ⟹ *category.iso* (*HOM A A*) (*ID A*)
by (*simp add*: *1 2 category.ide-is-iso*)
show $\bigwedge$(*A* :: *'a comp*) (*B* :: *'a comp*) (*C* :: *'a comp*) (*D* :: *'a comp*).
⟦ *A* ∈ *OBJ*; *B* ∈ *OBJ*; *C* ∈ *OBJ*; *D* ∈ *OBJ* ⟧
⟹ *natural-isomorphism*
(*product-category.comp*
(*HOM C D*) (*product-category.comp* (*HOM B C*) (*HOM A B*)))
(*HOM A D*)
(λ(*f*, *g*, *h*). *COMP D B A* (*COMP D C B f g*) *h*)
(λ(*f*, *g*, *h*). *COMP D C A f* (*COMP C B A g h*))
(λ(*f*, *g*, *h*). *ASSOC D C B A f g h*)
proof −
fix *A* :: *'a comp* **and** *B* :: *'a comp* **and** *C* :: *'a comp* **and** *D* :: *'a comp*
assume *A*: *A* ∈ *OBJ* **and** *B*: *B* ∈ *OBJ* **and** *C*: *C* ∈ *OBJ* **and** *D*: *D* ∈ *OBJ*
interpret *A*: *category A* **using** *A* **by** *simp*
interpret *B*: *category B* **using** *B* **by** *simp*
interpret *C*: *category C* **using** *C* **by** *simp*
interpret *D*: *category D* **using** *D* **by** *simp*
interpret *AB*: *functor-category A B* **..**
interpret *BC*: *functor-category B C* **..**
interpret *CD*: *functor-category C D* **..**
interpret *AD*: *functor-category A D* **..**
interpret *BD*: *functor-category B D* **..**
interpret *AC*: *functor-category A C* **..**
interpret *BC-AB*: *product-category BC.comp AB.comp* **..**
interpret *CD-BC-AB*: *product-category CD.comp BC-AB.comp* **..**

let *?L* = λ(*f*, *g*, *h*). *COMP D B A* (*COMP D C B f g*) *h*
let *?R* = λ(*f*, *g*, *h*). *COMP D C A f* (*COMP C B A g h*)
let *?A* = λ(*f*, *g*, *h*). *ASSOC D C B A f g h*

have ∗:
$\bigwedge$*fgh fgh'*. *CD-BC-AB.seq fgh fgh'* ⟹
AD.comp
(*AD.MkArr*

```
        (CD.Dom (fst fgh) ∘ BC.Dom (fst (snd fgh)) ∘ AB.Dom (snd (snd fgh)))
        (CD.Cod (fst fgh) ∘ BC.Cod (fst (snd fgh)) ∘ AB.Cod (snd (snd fgh)))
        (CD.Map (fst fgh) ∘ BC.Map (fst (snd fgh)) ∘ AB.Map (snd (snd fgh))))
      (AD.MkArr
        (CD.Dom (fst fgh') ∘ BC.Dom (fst (snd fgh')) ∘ AB.Dom (snd (snd fgh')))
        (CD.Cod (fst fgh') ∘ BC.Cod (fst (snd fgh')) ∘ AB.Cod (snd (snd fgh')))
        (CD.Map (fst fgh') ∘ BC.Map (fst (snd fgh')) ∘ AB.Map (snd (snd fgh')))) =
    AD.MkArr (AD.Dom (fst fgh') ∘ AD.Dom (BC.comp (fst (snd fgh)) (fst (snd fgh')))
              ∘ AD.Dom (AB.comp (snd (snd fgh)) (snd (snd fgh'))))
            (AD.Cod (fst fgh) ∘ AD.Cod (BC.comp (fst (snd fgh)) (fst (snd fgh')))
              ∘ AD.Cod (AB.comp (snd (snd fgh)) (snd (snd fgh'))))
            (vertical-composite.map C D (AD.Map (fst fgh')) (AD.Map (fst fgh))
              ∘ AD.Map (BC.comp (fst (snd fgh)) (fst (snd fgh')))
              ∘ AD.Map (AB.comp (snd (snd fgh)) (snd (snd fgh'))))
proof −
  fix fgh fgh'
  assume seq: CD-BC-AB.seq fgh fgh'
  have fgh: CD-BC-AB.arr fgh
    using seq by blast
  have fgh': CD-BC-AB.arr fgh'
    using seq by blast

  let ?f = fst (fgh) and ?g = fst (snd (fgh)) and ?h = snd (snd fgh)
  let ?f' = fst (fgh') and ?g' = fst (snd (fgh')) and ?h' = snd (snd fgh')
  let ?LHS = AD.comp
            (AD.MkArr (CD.Dom ?f ∘ BC.Dom ?g ∘ AB.Dom ?h)
                      (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
                      (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h))
            (AD.MkArr (CD.Dom ?f' ∘ BC.Dom ?g' ∘ AB.Dom ?h')
                      (CD.Cod ?f' ∘ BC.Cod ?g' ∘ AB.Cod ?h')
                      (CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h'))
  let ?RHS = AD.MkArr
            (AD.Dom ?f' ∘ AD.Dom (BC.comp ?g ?g') ∘ AD.Dom (AB.comp ?h ?h'))
            (AD.Cod ?f ∘ AD.Cod (BC.comp ?g ?g') ∘ AD.Cod (AB.comp ?h ?h'))
            (vertical-composite.map C D (AD.Map ?f') (AD.Map ?f)
              ∘ AD.Map (BC.comp ?g ?g') ∘ AD.Map (AB.comp ?h ?h'))

  interpret f: natural-transformation C D ‹CD.Dom ?f› ‹CD.Cod ?f› ‹MAP ?f›
    using fgh CD-BC-AB.arr-char CD.arr-char by simp
  interpret g: natural-transformation B C ‹BC.Dom ?g› ‹BC.Cod ?g› ‹MAP ?g›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char by simp
  interpret h: natural-transformation A B ‹AB.Dom ?h› ‹AB.Cod ?h› ‹MAP ?h›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char by simp
  interpret fg: natural-transformation B D
            ‹CD.Dom ?f o BC.Dom ?g› ‹CD.Cod ?f o BC.Cod ?g› ‹MAP ?f o MAP ?g›
    using f.natural-transformation-axioms g.natural-transformation-axioms
          horizontal-composite
    by blast
  interpret fgh: natural-transformation A D
```

‹CD.Dom ?f o BC.Dom ?g o AB.Dom ?h›
‹CD.Cod ?f o BC.Cod ?g o AB.Cod ?h›
‹MAP ?f o MAP ?g o MAP ?h›
using *f.natural-transformation-axioms g.natural-transformation-axioms h.natural-transformation-axioms horizontal-composite*
by *blast*
interpret *f'*: *natural-transformation C D* ‹CD.Dom ?f'› ‹CD.Cod ?f'› ‹MAP ?f'›
using *fgh' CD-BC-AB.arr-char CD.arr-char* **by** *simp*
interpret *g'*: *natural-transformation B C* ‹BC.Dom ?g'› ‹BC.Cod ?g'› ‹MAP ?g'›
using *fgh' CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char* **by** *simp*
interpret *h'*: *natural-transformation A B* ‹AB.Dom ?h'› ‹AB.Cod ?h'› ‹MAP ?h'›
using *fgh' CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char* **by** *simp*
interpret *fg'*: *natural-transformation B D*
‹CD.Dom ?f' o BC.Dom ?g'› ‹CD.Cod ?f' o BC.Cod ?g'›
‹MAP ?f' o MAP ?g'›
using *f'.natural-transformation-axioms g'.natural-transformation-axioms horizontal-composite*
by *blast*
interpret *fgh'*: *natural-transformation A D*
‹CD.Dom ?f' o BC.Dom ?g' o AB.Dom ?h'›
‹CD.Cod ?f' o BC.Cod ?g' o AB.Cod ?h'›
‹MAP ?f' o MAP ?g' o MAP ?h'›
using *f'.natural-transformation-axioms g'.natural-transformation-axioms h'.natural-transformation-axioms horizontal-composite*
by *blast*
interpret *ff'*: *vertical-composite C D* ‹CD.Dom ?f'› ‹CD.Cod ?f'› ‹CD.Cod ?f›
‹MAP ?f'› ‹MAP ?f›
using *seq CD-BC-AB.seq-char* [*of fgh fgh'*] *CD.seq-char* [*of ?f ?f'*]
f.natural-transformation-axioms f'.natural-transformation-axioms
C.category-axioms D.category-axioms
f'.F.functor-axioms f'.G.functor-axioms f.G.functor-axioms
by (*metis vertical-composite.intro*)
interpret *gg'*: *vertical-composite B C* ‹BC.Dom ?g'› ‹BC.Cod ?g'› ‹BC.Cod ?g›
‹MAP ?g'› ‹MAP ?g›
using *seq CD-BC-AB.seq-char* [*of fgh fgh'*] *BC-AB.seq-char BC.seq-char* [*of ?g ?g'*]
g.natural-transformation-axioms g'.natural-transformation-axioms
B.category-axioms C.category-axioms
g'.F.functor-axioms g'.G.functor-axioms g.G.functor-axioms
by (*metis vertical-composite.intro*)
interpret *hh'*: *vertical-composite A B* ‹AB.Dom ?h'› ‹AB.Cod ?h'› ‹AB.Cod ?h›
‹MAP ?h'› ‹MAP ?h›
using *seq CD-BC-AB.seq-char* [*of fgh fgh'*] *BC-AB.seq-char AB.seq-char* [*of ?h ?h'*]
h.natural-transformation-axioms h'.natural-transformation-axioms
A.category-axioms B.category-axioms
h'.F.functor-axioms h'.G.functor-axioms h.G.functor-axioms
by (*metis vertical-composite.intro*)

have *?LHS* = *AD.MkArr*
(*CD.Dom ?f'* ∘ *BC.Dom ?g'* ∘ *AB.Dom ?h'*)

```
            (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
            (vertical-composite.map A D
              (CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h')
              (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h))
  using AD.comp-MkArr [of CD.Dom ?f' ∘ BC.Dom ?g' ∘ AB.Dom ?h'
                          CD.Cod ?f' ∘ BC.Cod ?g' ∘ AB.Cod ?h'
                          CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h'
                          CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h
                          CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h]
  using seq CD-BC-AB.seq-char CD.seq-char BC-AB.seq-char AB.arr-char
        BD.seq-char AB.seq-char BC.seq-char fg.natural-transformation-axioms
        fg'.natural-transformation-axioms fgh'.natural-transformation-axioms
        fgh.natural-transformation-axioms
  by auto
also have ... = AD.MkArr
                (CD.Dom ?f' ∘ BC.Dom ?g' ∘ AB.Dom ?h')
                (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
                (vertical-composite.map C D (CD.Map ?f') (CD.Map ?f)
                  ∘ vertical-composite.map B C (BC.Map ?g') (BC.Map ?g)
                  o vertical-composite.map A B (AB.Map ?h') (AB.Map ?h))
proof −
  have vertical-composite.map A D
          (CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h')
          (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h) =
        vertical-composite.map C D (CD.Map ?f') (CD.Map ?f)
          ∘ vertical-composite.map B C (BC.Map ?g') (BC.Map ?g)
          o vertical-composite.map A B (AB.Map ?h') (AB.Map ?h)
    using interchange
            [of B C BC.Dom ?g' BC.Cod ?g' BC.Map ?g'
                BC.Cod ?g BC.Map ?g D
                CD.Dom ?f' CD.Cod ?f' CD.Map ?f'
                CD.Cod ?f  CD.Map ?f]
          interchange
          f'.natural-transformation-axioms ff'.τ.natural-transformation-axioms
          g'.natural-transformation-axioms gg'.τ.natural-transformation-axioms
          fg'.natural-transformation-axioms
          gg'.τ.natural-transformation-axioms h'.natural-transformation-axioms
          hh'.τ.natural-transformation-axioms horizontal-composite
    by (metis (no-types, lifting))
  thus ?thesis by simp
qed
also have ... = ?RHS
  by (metis (no-types, lifting) AB.Cod-comp AB.Dom-comp AB.Map-comp'
      AB.seq-char BC.Cod-comp BC.Dom-comp BC.Map-comp' BC.seq-char
      BC-AB.seq-char CD-BC-AB.seqE_PC seq)
finally show ?LHS = ?RHS by fastforce
qed

interpret L: functor CD-BC-AB.comp AD.comp ?L
```

```
proof
  interpret CD-BC: product-category CD.comp BC.comp ..
  interpret BD: functor-category B D ..
  fix fgh
  show ¬ CD-BC-AB.arr fgh ⟹ ?L fgh = AD.null
    using AD.null-char BD.null-char by (cases fgh, auto)
  assume fgh: CD-BC-AB.arr fgh
  let ?f = fst (fgh) and ?g = fst (snd (fgh)) and ?h = snd (snd fgh)
  interpret f: natural-transformation C D ‹CD.Dom ?f› ‹CD.Cod ?f› ‹MAP ?f›
    using fgh CD-BC-AB.arr-char CD.arr-char by simp
  interpret g: natural-transformation B C ‹BC.Dom ?g› ‹BC.Cod ?g› ‹MAP ?g›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char by simp
  interpret h: natural-transformation A B ‹AB.Dom ?h› ‹AB.Cod ?h› ‹MAP ?h›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char by simp
  interpret fg: natural-transformation B D
              ‹CD.Dom ?f o BC.Dom ?g› ‹CD.Cod ?f o BC.Cod ?g› ‹MAP ?f o MAP ?g›
    using f.natural-transformation-axioms g.natural-transformation-axioms
          horizontal-composite
    by blast
  interpret fgh: natural-transformation A D
                ‹CD.Dom ?f o BC.Dom ?g o AB.Dom ?h›
                ‹CD.Cod ?f o BC.Cod ?g o AB.Cod ?h›
                ‹MAP ?f o MAP ?g o MAP ?h›
    using f.natural-transformation-axioms g.natural-transformation-axioms
          h.natural-transformation-axioms horizontal-composite
    by blast
  show 1: AD.arr (?L fgh)
   using fgh CD-BC-AB.arr-char CD.arr-char BC-AB.arr-char BC.arr-char AB.arr-char
          fg.natural-transformation-axioms fgh.natural-transformation-axioms
          fgh.F.functor-axioms fgh.G.functor-axioms AD.arr-char
    by (cases fgh, auto)
  show AD.dom (?L fgh) = ?L (CD-BC-AB.dom fgh)
    using fgh 1 fg.natural-transformation-axioms fg.F.functor-axioms
    by (cases fgh, auto)
  show AD.cod (?L fgh) = ?L (CD-BC-AB.cod fgh)
    using fgh 1 fg.natural-transformation-axioms fg.G.functor-axioms
    by (cases fgh, auto)
  next

  fix fgh fgh'
  assume seq: CD-BC-AB.seq fgh fgh'
  have fgh: CD-BC-AB.arr fgh
    using seq by blast
  have fgh': CD-BC-AB.arr fgh'
    using seq by blast
  let ?f = fst (fgh) and ?g = fst (snd (fgh)) and ?h = snd (snd fgh)
  interpret f: natural-transformation C D ‹CD.Dom ?f› ‹CD.Cod ?f› ‹MAP ?f›
    using fgh CD-BC-AB.arr-char CD.arr-char by simp
  interpret g: natural-transformation B C ‹BC.Dom ?g› ‹BC.Cod ?g› ‹MAP ?g›
```

using *fgh CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char* **by** *simp*
interpret *h*: *natural-transformation A B* ‹*AB.Dom ?h*› ‹*AB.Cod ?h*› ‹*MAP ?h*›
using *fgh CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char* **by** *simp*
interpret *fg*: *natural-transformation B D*
‹*CD.Dom ?f o BC.Dom ?g*› ‹*CD.Cod ?f o BC.Cod ?g*› ‹*MAP ?f o MAP ?g*›
using *f.natural-transformation-axioms g.natural-transformation-axioms*
horizontal-composite
by *blast*
let *?f′* = *fst* (*fgh′*) **and** *?g′* = *fst* (*snd* (*fgh′*)) **and** *?h′* = *snd* (*snd fgh′*)
interpret *f′*: *natural-transformation C D* ‹*CD.Dom ?f′*› ‹*CD.Cod ?f′*› ‹*MAP ?f′*›
using *fgh′ CD-BC-AB.arr-char CD.arr-char* **by** *simp*
interpret *g′*: *natural-transformation B C* ‹*BC.Dom ?g′*› ‹*BC.Cod ?g′*› ‹*MAP ?g′*›
using *fgh′ CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char* **by** *simp*
interpret *h′*: *natural-transformation A B* ‹*AB.Dom ?h′*› ‹*AB.Cod ?h′*› ‹*MAP ?h′*›
using *fgh′ CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char* **by** *simp*
interpret *fg′*: *natural-transformation B D*
‹*CD.Dom ?f′ o BC.Dom ?g′*› ‹*CD.Cod ?f′ o BC.Cod ?g′*›
‹*MAP ?f′ o MAP ?g′*›
using *f′.natural-transformation-axioms g′.natural-transformation-axioms*
horizontal-composite
by *blast*
interpret *ff′*: *vertical-composite C D* ‹*CD.Dom ?f′*› ‹*CD.Cod ?f′*› ‹*CD.Cod ?f*›
‹*MAP ?f′*› ‹*MAP ?f*›
using *seq CD-BC-AB.seq-char* [*of fgh fgh′*] *CD.seq-char* [*of ?f ?f′*]
f.natural-transformation-axioms f′.natural-transformation-axioms
C.category-axioms D.category-axioms
f′.F.functor-axioms f′.G.functor-axioms f.G.functor-axioms
by (*unfold vertical-composite-def*) *presburger*
interpret *gg′*: *vertical-composite B C* ‹*CD.Dom ?g′*› ‹*CD.Cod ?g′*› ‹*CD.Cod ?g*›
‹*MAP ?g′*› ‹*MAP ?g*›
using *seq CD-BC-AB.seq-char* [*of fgh fgh′*] *CD.seq-char* [*of ?g ?g′*]
g.natural-transformation-axioms g′.natural-transformation-axioms
C.category-axioms D.category-axioms
by (*unfold vertical-composite-def*)
(*metis BC.seq-char BC-AB.seqE$_{PC}$ natural-transformation-def*)
interpret *gg′off′*: *natural-transformation B D*
‹*AC.Dom* (*fst fgh′*) ∘ *AC.Dom* (*fst* (*snd fgh′*))›
‹*AC.Cod* (*fst fgh*) ∘ *AC.Cod* (*fst* (*snd fgh*))› ‹*ff′.map* ∘ *gg′.map*›
by (*meson ff′.natural-transformation-axioms gg′.natural-transformation-axioms*
horizontal-composite)

show *?L* (*CD-BC-AB.comp fgh fgh′*) = *AD.comp* (*?L fgh*) (*?L fgh′*)
proof −
have *AD.comp* (*?L fgh*) (*?L fgh′*) =
AD.comp
(*AD.MkArr* (*CD.Dom ?f* ∘ *BC.Dom ?g* ∘ *AB.Dom ?h*)
(*CD.Cod ?f* ∘ *BC.Cod ?g* ∘ *AB.Cod ?h*)
(*CD.Map ?f* ∘ *BC.Map ?g* ∘ *AB.Map ?h*))
(*AD.MkArr* (*CD.Dom ?f′* ∘ *BC.Dom ?g′* ∘ *AB.Dom ?h′*)

```
                (CD.Cod ?f' ∘ BC.Cod ?g' ∘ AB.Cod ?h')
                (CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h'))
      using seq CD-BC-AB.seq-char CD.seq-char BC-AB.seq-char AB.arr-char
          BD.seq-char AB.seq-char BC.seq-char fg.natural-transformation-axioms
          fg'.natural-transformation-axioms
      by (cases fgh, cases fgh') auto
    also have ... =
            AD.MkArr
              (AD.Dom ?f' ∘ AD.Dom (BC.comp ?g ?g') ∘ AD.Dom (AB.comp ?h ?h'))
              (AD.Cod ?f ∘ AD.Cod (BC.comp ?g ?g') ∘ AD.Cod (AB.comp ?h ?h'))
              (vertical-composite.map C D (AD.Map ?f') (AD.Map ?f)
                ∘ AD.Map (BC.comp ?g ?g')
                ∘ AD.Map (AB.comp ?h ?h'))
      using seq * [of fgh fgh'] by blast
    also have ... = ?L (CD-BC-AB.comp fgh fgh')
      using fgh fgh' CD-BC-AB.comp-char CD.seq-char BC-AB.comp-char
          BC.seq-char AB.seq-char gg'off'.natural-transformation-axioms
      apply simp
      by (metis (no-types, lifting) BC-AB.seqE_PC CD-BC-AB.seqE_PC seq)
    finally show ?thesis by argo
  qed
qed

interpret R: functor CD-BC-AB.comp AD.comp ?R
proof
  interpret CD-BC: product-category CD.comp BC.comp ..
  interpret AC: functor-category A C ..
  fix fgh
  show ¬ CD-BC-AB.arr fgh ⟹ ?R fgh = AD.null
    using AD.null-char AC.null-char by (cases fgh, auto)
  assume fgh: CD-BC-AB.arr fgh
  let ?f = fst (fgh) and ?g = fst (snd (fgh)) and ?h = snd (snd fgh)
  interpret f: natural-transformation C D ‹CD.Dom ?f› ‹CD.Cod ?f› ‹MAP ?f›
    using fgh CD-BC-AB.arr-char CD.arr-char by simp
  interpret g: natural-transformation B C ‹BC.Dom ?g› ‹BC.Cod ?g› ‹MAP ?g›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char by simp
  interpret h: natural-transformation A B ‹AB.Dom ?h› ‹AB.Cod ?h› ‹MAP ?h›
    using fgh CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char by simp
  interpret fg: natural-transformation B D
              ‹CD.Dom ?f o BC.Dom ?g› ‹CD.Cod ?f o BC.Cod ?g› ‹MAP ?f o MAP ?g›
    using f.natural-transformation-axioms g.natural-transformation-axioms
        horizontal-composite
    by blast
  interpret gh: natural-transformation A C
                ‹BC.Dom ?g o AB.Dom ?h› ‹BC.Cod ?g o AB.Cod ?h›
                ‹MAP ?g o MAP ?h›
    using g.natural-transformation-axioms h.natural-transformation-axioms
        horizontal-composite
    by blast
```

interpret *fgh*: *natural-transformation A D*
‹*CD.Dom ?f o* (*BC.Dom ?g o AB.Dom ?h*)›
‹*CD.Cod ?f o* (*BC.Cod ?g o AB.Cod ?h*)›
‹*MAP ?f o* (*MAP ?g o MAP ?h*)›
using *f.natural-transformation-axioms g.natural-transformation-axioms h.natural-transformation-axioms horizontal-composite*
by *blast*
show *1*: *AD.arr* (*?R fgh*)
using *fgh CD-BC-AB.arr-char AB.arr-char CD-BC.arr-char CD.arr-char BC.arr-char gh.natural-transformation-axioms fgh.natural-transformation-axioms fgh.F.functor-axioms fgh.G.functor-axioms AD.arr-char AC.arr-char*
by (*cases fgh*) (*simp add*: *natural-transformation-def*)
show *AD.dom* (*?R fgh*) = *?R* (*CD-BC-AB.dom fgh*)
using *fgh 1 gh.natural-transformation-axioms gh.F.functor-axioms*
by (*cases fgh*) *auto*
show *AD.cod* (*?R fgh*) = *?R* (*CD-BC-AB.cod fgh*)
using *fgh 1 gh.natural-transformation-axioms gh.G.functor-axioms*
by (*cases fgh*) *auto*
next
fix *fgh fgh′*
assume *seq*: *CD-BC-AB.seq fgh fgh′*
have *fgh*: *CD-BC-AB.arr fgh*
using *seq* **by** *blast*
have *fgh′*: *CD-BC-AB.arr fgh′*
using *seq* **by** *blast*
let *?f* = *fst* (*fgh*) **and** *?g* = *fst* (*snd* (*fgh*)) **and** *?h* = *snd* (*snd fgh*)
have *f*: *CD.arr ?f* **and** *g*: *BC.arr ?g* **and** *h*: *AB.arr ?h*
using *fgh* **by** *simp-all*
interpret *f*: *natural-transformation C D* ‹*CD.Dom ?f*› ‹*CD.Cod ?f*› ‹*MAP ?f*›
using *fgh CD-BC-AB.arr-char CD.arr-char* **by** *simp*
interpret *g*: *natural-transformation B C* ‹*BC.Dom ?g*› ‹*BC.Cod ?g*› ‹*MAP ?g*›
using *fgh CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char* **by** *simp*
interpret *h*: *natural-transformation A B* ‹*AB.Dom ?h*› ‹*AB.Cod ?h*› ‹*MAP ?h*›
using *fgh CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char* **by** *simp*
interpret *gh*: *natural-transformation A C*
‹*BC.Dom ?g o AB.Dom ?h*› ‹*BC.Cod ?g o AB.Cod ?h*›
‹*MAP ?g o MAP ?h*›
using *g.natural-transformation-axioms h.natural-transformation-axioms horizontal-composite*
by *blast*
let *?f′* = *fst* (*fgh′*) **and** *?g′* = *fst* (*snd* (*fgh′*)) **and** *?h′* = *snd* (*snd fgh′*)
have *f′*: *CD.arr ?f′* **and** *g′*: *BC.arr ?g′* **and** *h′*: *AB.arr ?h′*
using *fgh′* **by** *simp-all*
interpret *f′*: *natural-transformation C D* ‹*CD.Dom ?f′*› ‹*CD.Cod ?f′*› ‹*MAP ?f′*›
using *seq CD-BC-AB.arr-char CD.arr-char* **by** *blast*
interpret *g′*: *natural-transformation B C* ‹*BC.Dom ?g′*› ‹*BC.Cod ?g′*› ‹*MAP ?g′*›
using *seq CD-BC-AB.arr-char BC-AB.arr-char BC.arr-char* **by** *blast*
interpret *h′*: *natural-transformation A B* ‹*AB.Dom ?h′*› ‹*AB.Cod ?h′*› ‹*MAP ?h′*›
using *seq CD-BC-AB.arr-char BC-AB.arr-char AB.arr-char* **by** *blast*

```
interpret gh': natural-transformation A C
            ‹BC.Dom ?g' o AB.Dom ?h'› ‹BC.Cod ?g' o AB.Cod ?h'›
            ‹MAP ?g' o MAP ?h'›
  using g'.natural-transformation-axioms h'.natural-transformation-axioms
      horizontal-composite
  by blast
interpret fg': natural-transformation B D
            ‹CD.Dom ?f' o BC.Dom ?g'› ‹CD.Cod ?f' o BC.Cod ?g'›
            ‹MAP ?f' o MAP ?g'›
  using f'.natural-transformation-axioms g'.natural-transformation-axioms
      horizontal-composite
  by blast
interpret ff': vertical-composite C D ‹CD.Dom ?f'› ‹CD.Cod ?f'› ‹CD.Cod ?f›
             ‹MAP ?f'› ‹MAP ?f›
  using seq CD-BC-AB.seq-char [of fgh fgh'] CD.seq-char [of ?f ?f']
      f.natural-transformation-axioms f'.natural-transformation-axioms
      C.category-axioms D.category-axioms
      f'.F.functor-axioms f'.G.functor-axioms f.G.functor-axioms
  by (metis vertical-composite.intro)
interpret gg': vertical-composite B C ‹BC.Dom ?g'› ‹BC.Cod ?g'› ‹BC.Cod ?g›
            ‹MAP ?g'› ‹MAP ?g›
  using seq CD-BC-AB.seq-char [of fgh fgh'] BC-AB.seq-char BC.seq-char [of ?g ?g']
      g.natural-transformation-axioms g'.natural-transformation-axioms
      B.category-axioms C.category-axioms
      g'.F.functor-axioms g'.G.functor-axioms g.G.functor-axioms
  by (metis vertical-composite.intro)
interpret hh': vertical-composite A B ‹AB.Dom ?h'› ‹AB.Cod ?h'› ‹AB.Cod ?h›
            ‹MAP ?h'› ‹MAP ?h›
  using seq CD-BC-AB.seq-char [of fgh fgh'] BC-AB.seq-char AB.seq-char [of ?h ?h']
      h.natural-transformation-axioms h'.natural-transformation-axioms
      A.category-axioms B.category-axioms
      h'.F.functor-axioms h'.G.functor-axioms h.G.functor-axioms
  by (metis vertical-composite.intro)

show ?R (CD-BC-AB.comp fgh fgh') = AD.comp (?R fgh) (?R fgh')
proof −
  have AD.comp (?R fgh) (?R fgh') =
      AD.comp
        (AD.MkArr (CD.Dom ?f ∘ BC.Dom ?g ∘ AB.Dom ?h)
                (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
                (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h))
        (AD.MkArr (CD.Dom ?f' ∘ BC.Dom ?g' ∘ AB.Dom ?h')
                (CD.Cod ?f' ∘ BC.Cod ?g' ∘ AB.Cod ?h')
                (CD.Map ?f' ∘ BC.Map ?g' ∘ AB.Map ?h'))
    using fgh fgh' gh.natural-transformation-axioms gh'.natural-transformation-axioms
    apply (cases fgh, cases fgh')
    by (simp add: o-assoc)
  also have ... =
          AD.MkArr
```

(AD.Dom ?f′ ∘ AD.Dom (BC.comp ?g ?g′) ∘ AD.Dom (AB.comp ?h ?h′))
(AD.Cod ?f ∘ AD.Cod (BC.comp ?g ?g′) ∘ AD.Cod (AB.comp ?h ?h′))
(vertical-composite.map C D (AD.Map ?f′) (AD.Map ?f)
∘ AD.Map (BC.comp ?g ?g′)
∘ AD.Map (AB.comp ?h ?h′))
using *seq* ∗ [*of fgh fgh′*] **by** *blast*
also have ... = *AD.MkArr*
(AD.Dom ?f′ ∘ BC.Dom ?g′ ∘ AB.Dom ?h′)
(AD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
(vertical-composite.map C D (AD.Map ?f′) (AD.Map ?f)
∘ AD.Map (BC.comp ?g ?g′)
∘ AD.Map (AB.comp ?h ?h′))
using *seq CD-BC-AB.seq-char BC-AB.seq-char BC.Dom-comp AB.Dom-comp BC.Cod-comp AB.Cod-comp BC.seq-char AB.seq-char*
by (*simp add*: *o-assoc*)
also have ... = *AD.MkArr*
(AD.Dom ?f′ ∘ BC.Dom ?g′ ∘ AB.Dom ?h′)
(AD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
(vertical-composite.map C D (AD.Map ?f′) (AD.Map ?f) o
vertical-composite.map B C (BC.Map ?g′) (BC.Map ?g) o
vertical-composite.map A B (AB.Map ?h′) (AB.Map ?h))
using *seq CD-BC-AB.seq-char BC-AB.seq-char BC.Map-comp AB.Map-comp BC.seq-char AB.seq-char*
by (*cases fgh*) *simp*
also have ... = *AD.MkArr*
(AD.Dom ?f′ ∘ BC.Dom ?g′ ∘ AB.Dom ?h′)
(AD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
(CD.Map (CD.comp ?f ?f′) o BC.Map (BC.comp ?g ?g′)
o AB.Map (AB.comp ?h ?h′))
using *seq CD-BC-AB.seq-char BC-AB.seq-char CD.Map-comp BC.Map-comp AB.Map-comp CD.seq-char BC.seq-char AB.seq-char*
by *simp*
also have
... =
AD.MkArr
(CD.Dom (CD.comp ?f ?f′) ∘ (BC.Dom (BC.comp ?g ?g′) ∘ AB.Dom (AB.comp ?h ?h′)))
(CD.Cod (CD.comp ?f ?f′) ∘ (BC.Cod (BC.comp ?g ?g′) ∘ AB.Cod (AB.comp ?h ?h′)))
(CD.Map (CD.comp ?f ?f′) ∘ (BC.Map (BC.comp ?g ?g′) ∘ AB.Map (AB.comp ?h ?h′)))
using *seq CD-BC-AB.seq-char BC-AB.seq-char BC.Dom-comp AB.Dom-comp BC.Cod-comp AB.Cod-comp CD.seq-char BC.seq-char AB.seq-char*
by (*cases fgh*) (*simp add*: *o-assoc*)
also have ... = *?R* (*CD.comp ?f ?f′*, *BC.comp ?g ?g′*, *AB.comp ?h ?h′*)
proof −
have *CD.seq ?f ?f′* ∧ *BC.seq ?g ?g′* ∧ *AB.seq ?h ?h′*
using *seq CD-BC-AB.seq-char BC-AB.seq-char CD.seq-char BC.seq-char AB.seq-char*
by *simp*

```
      moreover have natural-transformation A C
                    (AC.Dom (BC.comp ?g ?g') ∘ AC.Dom (AB.comp ?h ?h'))
                    (AC.Cod (BC.comp ?g ?g') ∘ AC.Cod (AB.comp ?h ?h'))
                    (AC.Map (BC.comp ?g ?g') ∘ AC.Map (AB.comp ?h ?h'))
        using horizontal-composite calculation by blast
      ultimately show ?thesis by simp
    qed
    also have ... = ?R (CD-BC-AB.comp fgh fgh')
      using seq fgh fgh' CD-BC-AB.comp-simp BC-AB.comp-simp CD-BC-AB.seq-char
            BC-AB.seq-char
      by (cases fgh) simp
    finally show ?thesis by argo
  qed
qed

interpret A: natural-transformation CD-BC-AB.comp AD.comp ?L ?R ?A
proof
  interpret CD-BC: product-category CD.comp BC.comp ..
  fix fgh
  show ¬ CD-BC-AB.arr fgh ⟹ ?A fgh = AD.null
    using AD.null-char BD.null-char by (cases fgh, auto)
  assume fgh: CD-BC-AB.arr fgh
  show AD.arr (?A fgh)
    using fgh AB.arr-char BC.arr-char CD.arr-char
    apply (cases fgh)
    apply simp
    by (metis (no-types, lifting) horizontal-composite)
  show AD.comp (?A (CD-BC-AB.cod fgh)) (?L fgh) = ?A fgh
  proof −
    let ?f = fst fgh and ?g = fst (snd fgh) and ?h = snd (snd fgh)
    have 1: natural-transformation B D
              (CD.Dom ?f ∘ BC.Dom ?g) (CD.Cod ?f ∘ BC.Cod ?g)
              (CD.Map ?f ∘ BC.Map ?g)
      using fgh BC.arr-char CD.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis (no-types, lifting) horizontal-composite)
    have 2: natural-transformation A D (AC.Dom ?f ∘ AC.Dom ?g ∘ AC.Dom ?h)
              (AC.Cod ?f ∘ AC.Cod ?g ∘ AC.Cod ?h)
              (AC.Map ?f ∘ AC.Map ?g ∘ AC.Map ?h)
      using fgh AB.arr-char BC.arr-char
            CD.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis (no-types, lifting) horizontal-composite)
    have AD.comp (?A (CD-BC-AB.cod fgh)) (?L fgh) =
          AD.comp (AD.MkIde (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h))
                  (AD.MkArr (CD.Dom ?f ∘ BC.Dom ?g ∘ AB.Dom ?h)
                            (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
```

```
                    (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h))
      using 1 2 fgh AB.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis AB.cod-char CD.arr.simps(2))
    also have ... = ?A fgh
      using 2 fgh
      by (cases fgh) (simp add: o-assoc AD.comp-cod-arr)
    finally show ?thesis by simp
  qed
  show AD.comp (?R fgh) (?A (CD-BC-AB.dom fgh)) = ?A fgh
  proof −
    let ?f = fst fgh and ?g = fst (snd fgh) and ?h = snd (snd fgh)
    have 1: natural-transformation A C
              (AC.Dom ?g ∘ AC.Dom ?h) (AC.Cod ?g ∘ AC.Cod ?h)
              (AC.Map ?g ∘ AC.Map ?h)
      using fgh AB.arr-char BC.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis (no-types, lifting) horizontal-composite)
    have 2: natural-transformation A D (AC.Dom ?f ∘ AC.Dom ?g ∘ AC.Dom ?h)
              (AC.Cod ?f ∘ AC.Cod ?g ∘ AC.Cod ?h)
              (AC.Map ?f ∘ AC.Map ?g ∘ AC.Map ?h)
      using fgh AB.arr-char BC.arr-char CD.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis (no-types, lifting) horizontal-composite)
    have AD.comp (?R fgh) (?A (CD-BC-AB.dom fgh)) =
          AD.comp (AD.MkArr (CD.Dom ?f ∘ BC.Dom ?g ∘ AB.Dom ?h)
                            (CD.Cod ?f ∘ BC.Cod ?g ∘ AB.Cod ?h)
                            (CD.Map ?f ∘ BC.Map ?g ∘ AB.Map ?h))
                  (AD.MkIde (CD.Dom ?f ∘ BC.Dom ?g ∘ AB.Dom ?h))
      using 1 2 fgh CD.arr-char
      apply (cases fgh)
      apply (simp add: o-assoc)
      by (metis CD.arr-dom-iff-arr)
    also have ... = ?A fgh
      using 2 fgh
      by (cases fgh) (simp add: o-assoc AD.comp-arr-dom)
    finally show ?thesis by simp
  qed
qed
show natural-isomorphism CD-BC-AB.comp AD.comp ?L ?R ?A
proof
  fix abc
  assume abc: CD-BC-AB.ide abc
  show AD.iso (?A abc)
  proof −
    interpret A-abc: natural-transformation A D
```

```
                    ‹BD.Dom (?A abc)› ‹BD.Cod (?A abc)› ‹BD.Map (?A abc)›
      using abc
      apply (cases abc)
      apply (simp add: o-assoc)
      by (metis functor-is-transformation horizontal-composite)
    interpret A-abc: natural-isomorphism A D
                    ‹BD.Dom (?A abc)› ‹BD.Cod (?A abc)› ‹BD.Map (?A abc)›
    proof −
      have ⋀a′. A.ide a′ ⟹ D.iso (CD.Map (fst abc)
                                   (BC.Map (fst (snd abc))
                                        (AB.Map (snd (snd abc)) a′)))
      proof −
        fix a′
        assume a′: A.ide a′
        have BC.ide (fst (snd abc)) ∧ AB.ide (snd (snd abc))
          using abc by force
        hence C.iso (AC.Map (fst (snd abc)) (AC.Map (snd (snd abc)) a′))
          using a′ by (metis (no-types) A.ide-is-iso AB.ide-char BC.ide-char
              functor.preserves-iso)
        thus D.iso (AC.Map (fst abc) (AC.Map (fst (snd abc))
                 (AC.Map (snd (snd abc)) a′)))
          by (meson CD.ide-char CD-BC-AB.ide-char_PC abc functor.preserves-iso)
      qed
      thus natural-isomorphism A D
                    (BD.Dom (?A abc)) (BD.Cod (?A abc)) (BD.Map (?A abc))
        apply unfold-locales
        using abc CD-BC-AB.iso-char BC-AB.iso-char A.ide-is-iso functor.preserves-iso
        by (cases abc) simp
    qed
    have ?A abc ≠ AD.null
      using abc
      by (metis (no-types, lifting) A.preserves-reflects-arr AD.not-arr-null
          CD-BC-AB.ideD(1))
    moreover have natural-isomorphism A D
                    (BD.Dom (?A abc)) (BD.Cod (?A abc)) (BD.Map (?A abc))
      ..
    ultimately show ?thesis by simp
  qed
 qed
qed
show ⋀(A :: 'a comp) (B :: 'a comp). ⟦ A ∈ OBJ; B ∈ OBJ ⟧
        ⟹ fully-faithful-functor (HOM A B) (HOM A B)
                (λf. if partial-composition.arr (HOM A B) f
                     then COMP B B A (ID B) f
                     else partial-magma.null (HOM A B))
proof −
 fix A :: 'a comp and B :: 'a comp
 assume A: A ∈ OBJ and B: B ∈ OBJ
 interpret A: category A using A by simp
```

```
  interpret B: category B using B by simp
  interpret AB: functor-category A B ..
  interpret BB: functor-category B B ..
  let ?L = λf. if partial-composition.arr (HOM A B) f
               then COMP B B A (ID B) f
               else partial-magma.null (HOM A B)
  have ?L = AB.map
    using AB.arr-char B.functor-axioms AB.MkArr-Map AB.extensionality by force
  thus fully-faithful-functor AB.comp AB.comp ?L
    by (simp add: AB.is-fully-faithful)
qed
show ⋀(A :: 'a comp) (B :: 'a comp). ⟦ A ∈ OBJ; B ∈ OBJ ⟧
          ⟹ fully-faithful-functor (HOM A B) (HOM A B)
               (λf. if partial-composition.arr (HOM A B) f
                    then COMP B A A f (ID A)
                    else partial-magma.null (HOM A B))
proof −
  fix A :: 'a comp and B :: 'a comp
  assume A: A ∈ OBJ and B: B ∈ OBJ
  interpret A: category A using A by simp
  interpret B: category B using B by simp
  interpret AB: functor-category A B ..
  interpret AA: functor-category A A ..
  interpret BB: functor-category B B ..
  let ?R = λf. if partial-composition.arr (HOM A B) f
               then COMP B A A f (ID A)
               else partial-magma.null (HOM A B)
  have ?R = AB.map
  proof
    fix f
    have ¬ AB.arr f ⟹ ?R f = AB.map f
      using AB.extensionality by simp
    moreover have AB.arr f ⟹ ?R f = AB.map f
    proof −
      assume f: AB.arr f
      have ?R f =
            BB.MkArr (BB.Dom f ∘ A.map) (BB.Cod f ∘ A.map) (BB.Map f ∘ A.map)
        using f AB.arr-char BB.ide-MkIde BB.arr-char A.functor-axioms by simp
      also have ... = AB.map f
        using f AB.arr-char AB.MkArr-Map by force
      finally show ?R f = AB.map f by blast
    qed
    ultimately show ?R f = AB.map f by blast
  qed
  thus fully-faithful-functor AB.comp AB.comp ?R
    by (simp add: AB.is-fully-faithful)
qed
show ⋀(A :: 'a comp) (B :: 'a comp) (C :: 'a comp) (D :: 'a comp) (E :: 'a comp) f g h k.
     ⟦ A ∈ OBJ; B ∈ OBJ; C ∈ OBJ; D ∈ OBJ; E ∈ OBJ;
```

partial-composition.ide (*HOM D E*) *f*; *partial-composition.ide* (*HOM C D*) *g*;
partial-composition.ide (*HOM B C*) *h*; *partial-composition.ide* (*HOM A B*) *k* ⟧ ⟹
HOM A E (*COMP E D A f* (*ASSOC D C B A g h k*))
(*HOM A E* (*ASSOC E D B A f* (*COMP D C B g h*) *k*)
(*COMP E B A* (*ASSOC E D C B f g h*) *k*)) =
HOM A E (*ASSOC E D C A f g* (*COMP C B A h k*))
(*ASSOC E C B A* (*COMP E D C f g*) *h k*)
proof −
fix *A* :: *'a comp* **and** *B* :: *'a comp* **and** *C* :: *'a comp*
and *D* :: *'a comp* **and** *E* :: *'a comp*
fix *f g h k*
assume *A*: *A* ∈ *OBJ* **and** *B*: *B* ∈ *OBJ* **and** *C*: *C* ∈ *OBJ*
and *D*: *D* ∈ *OBJ* **and** *E*: *E* ∈ *OBJ*
assume *f*: *partial-composition.ide* (*HOM D E*) *f*
assume *g*: *partial-composition.ide* (*HOM C D*) *g*
assume *h*: *partial-composition.ide* (*HOM B C*) *h*
assume *k*: *partial-composition.ide* (*HOM A B*) *k*
interpret *A*: *category A* **using** *A* **by** *simp*
interpret *B*: *category B* **using** *B* **by** *simp*
interpret *C*: *category C* **using** *C* **by** *simp*
interpret *D*: *category D* **using** *D* **by** *simp*
interpret *E*: *category E* **using** *E* **by** *simp*
interpret *AB*: *functor-category A B* **..**
interpret *BC*: *functor-category B C* **..**
interpret *CD*: *functor-category C D* **..**
interpret *DE*: *functor-category D E* **..**
interpret *BD*: *functor-category B D* **..**
interpret *AD*: *functor-category A D* **..**
interpret *CE*: *functor-category C E* **..**
interpret *AC*: *functor-category A C* **..**
interpret *BE*: *functor-category B E* **..**
interpret *AE*: *functor-category A E* **..**
interpret *f*: *natural-transformation D E* ‹*DE.Dom f*› ‹*DE.Cod f*› ‹*MAP f*›
using *f DE.arr-char* **by** *simp*
interpret *g*: *natural-transformation C D* ‹*CD.Dom g*› ‹*CD.Cod g*› ‹*MAP g*›
using *g CD.arr-char* **by** *simp*
interpret *h*: *natural-transformation B C* ‹*BC.Dom h*› ‹*BC.Cod h*› ‹*MAP h*›
using *h BC.arr-char* **by** *simp*
interpret *k*: *natural-transformation A B* ‹*AB.Dom k*› ‹*AB.Cod k*› ‹*MAP k*›
using *k AB.arr-char* **by** *simp*
have *0*: *natural-transformation A E*
(*DE.Map f* ∘ *CD.Map g* ∘ *BC.Map h* ∘ *AB.Map k*)
(*DE.Map f* ∘ *CD.Map g* ∘ *BC.Map h* ∘ *AB.Map k*)
(*DE.Map f* ∘ *CD.Map g* ∘ *BC.Map h* ∘ *AB.Map k*)
using *f.natural-transformation-axioms g.natural-transformation-axioms*
h.natural-transformation-axioms k.natural-transformation-axioms
f g h k AB.ide-char BC.ide-char CD.ide-char DE.ide-char
horizontal-composite
by (*metis* (*no-types*, *lifting*))

```
    have 1: natural-transformation B D
              (CD.Map g ∘ BC.Map h) (CD.Map g ∘ BC.Map h) (CD.Map g ∘ BC.Map h)
      using g.natural-transformation-axioms h.natural-transformation-axioms
            g h BC.ide-char CD.ide-char horizontal-composite
      by (metis (no-types, lifting))
    have 2: natural-transformation B E
              (DE.Map f ∘ CD.Map g ∘ BC.Map h)
              (DE.Map f ∘ CD.Map g ∘ BC.Map h)
              (DE.Map f ∘ CD.Map g ∘ BC.Map h)
      using 1 f.natural-transformation-axioms f DE.ide-char horizontal-composite o-assoc
      by (metis (no-types, lifting))
    have 3: natural-transformation A C
              (BC.Map h ∘ AB.Map k) (BC.Map h ∘ AB.Map k) (BC.Map h ∘ AB.Map k)
      using k.natural-transformation-axioms h.natural-transformation-axioms
            k h AB.ide-char BC.ide-char horizontal-composite
      by metis
    have 4: natural-transformation C E
              (DE.Map f ∘ CD.Map g) (DE.Map f ∘ CD.Map g) (DE.Map f ∘ CD.Map g)
      using f g f.natural-transformation-axioms g.natural-transformation-axioms
            CD.ide-char DE.ide-char horizontal-composite
      by metis

    have HOM A E (COMP E D A f (ASSOC D C B A g h k))
                 (HOM A E (ASSOC E D B A f (COMP D C B g h) k)
                          (COMP E B A (ASSOC E D C B f g h) k)) =
          AE.MkIde (DE.Map f ∘ CD.Map g ∘ BC.Map h ∘ AB.Map k)
    proof −
      have functor B E (DE.Map f ∘ CD.Map g ∘ BC.Map h)
        using 2 natural-transformation-def by blast
      moreover have functor A D (CD.Map g ∘ BC.Map h ∘ AB.Map k)
        using k 1 AB.ide-char horizontal-composite k.natural-transformation-axioms
              natural-transformation-def
        by (metis (no-types, lifting))
      ultimately show ?thesis
        using f g h k 0 1 2 by (simp add: o-assoc)
    qed
    also have ... = HOM A E (ASSOC E D C A f g (COMP C B A h k))
                            (ASSOC E C B A (COMP E D C f g) h k)
      using f g h k 0 3 4 AC.ide-MkIde CE.ide-MkIde
      by (simp add: o-assoc)
    finally show HOM A E (COMP E D A f (ASSOC D C B A g h k))
                         (HOM A E (ASSOC E D B A f (COMP D C B g h) k)
                                  (COMP E B A (ASSOC E D C B f g h) k)) =
                 HOM A E (ASSOC E D C A f g (COMP C B A h k))
                         (ASSOC E C B A (COMP E D C f g) h k)
      by blast
  qed
qed
```

lemma *is-concrete-bicategory*:
shows *concrete-bicategory OBJ HOM ID COMP ID ASSOC*
  ..

lemma *unit-simp*:
assumes *obj a*
shows i *a* = *a*
  by (*metis* (*mono-tags*, *lifting*) i-*def assms obj-char*)

lemma *assoc-simp*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows a *f g h* = *hcomp f* (*hcomp g h*)
proof −
  let *?A* = *Src h* **and** *?B* = *Trg h* **and** *?C* = *Trg g* **and** *?D* = *Trg f*
  have *1*: *Src f* = *Trg g* ∧ *Src g* = *Trg h*
    using *assms* **by** (*simp add*: *src-def trg-def*)
  interpret *A*: *category ?A* **using** *assms Src-in-Obj* [*of h*] **by** *simp*
  interpret *B*: *category ?B* **using** *assms Trg-in-Obj* [*of h*] **by** *simp*
  interpret *C*: *category ?C* **using** *assms Trg-in-Obj* [*of g*] **by** *simp*
  interpret *D*: *category ?D* **using** *assms Trg-in-Obj* [*of f*] **by** *simp*
  interpret *AB*: *functor-category ?A ?B* **..**
  interpret *BC*: *functor-category ?B ?C* **..**
  interpret *CD*: *functor-category ?C ?D* **..**
  interpret *AD*: *functor-category ?A ?D* **..**
  interpret *BC-AB*: *product-category BC.comp AB.comp* **..**
  interpret *CD-BC-AB*: *product-category CD.comp BC-AB.comp* **..**
  interpret *f*: *functor ?C ?D* ‹*CD.Map* (*Map f*)›
    using *assms 1 ide-char''* [*of f*] *arr-char* [*of f*] *CD.ide-char* **by** *simp*
  interpret *g*: *functor ?B ?C* ‹*BC.Map* (*Map g*)›
    using *assms 1 ide-char''* [*of g*] *arr-char* [*of g*] *CD.ide-char* **by** *simp*
  interpret *h*: *functor ?A ?B* ‹*AB.Map* (*Map h*)›
    using *assms 1 ide-char''* [*of h*] *arr-char* [*of h*] *CD.ide-char* **by** *simp*
  have a *f g h* = *MkCell ?A ?D*
                (*AD.MkArr* (*AD.Dom* (*Map f*) ∘ *AD.Dom* (*Map g*) ∘ *AD.Dom* (*Map h*))
                (*AD.Cod* (*Map f*) ∘ *AD.Cod* (*Map g*) ∘ *AD.Cod* (*Map h*))
                (*AD.Map* (*Map f*) ∘ *AD.Map* (*Map g*) ∘ *AD.Map* (*Map h*)))
    using *assms 1 ide-char''* a-*simp-ide* [*of f g h*]
        arr-char [*of f*] *arr-char* [*of g*] *arr-char* [*of h*]
    by *simp*
  moreover have *AD.ide* (*AD.MkArr*
                (*CD.Dom* (*Map f*) ∘ *BC.Dom* (*Map g*) ∘ *AB.Dom* (*Map h*))
                (*CD.Cod* (*Map f*) ∘ *BC.Cod* (*Map g*) ∘ *AB.Cod* (*Map h*))
                (*CD.Map* (*Map f*) ∘ *BC.Map* (*Map g*) ∘ *AB.Map* (*Map h*)))
  proof −
    have *functor ?A ?D* (*CD.Map* (*Map f*) ∘ *BC.Map* (*Map g*) ∘ *AB.Map* (*Map h*))
      using *f.functor-axioms g.functor-axioms h.functor-axioms functor-comp* **by** *blast*
    hence *AD.ide* (*AD.MkArr* (*CD.Map* (*Map f*) ∘ *BC.Map* (*Map g*) ∘ *AB.Map* (*Map h*))
                (*CD.Map* (*Map f*) ∘ *BC.Map* (*Map g*) ∘ *AB.Map* (*Map h*))
                (*CD.Map* (*Map f*) ∘ *BC.Map* (*Map g*) ∘ *AB.Map* (*Map h*)))

```
      using AD.ide-char AD.null-char by simp
    thus ?thesis
      using assms 1
            ide-char'' [of f] CD.ide-char [of Map f]
            ide-char'' [of g] BC.ide-char [of Map g]
            ide-char'' [of h] AB.ide-char [of Map h]
      by presburger
  qed
  moreover have arr (a f g h)
    using assms by simp
  ultimately have ide (a f g h)
    using ide-char''' [of a f g h] by auto
  hence a f g h = cod (a f g h)
    by simp
  thus a f g h = hcomp f (hcomp g h)
    using assms by simp
qed

lemma is-strict-bicategory:
shows strict-bicategory vcomp hcomp a i src trg
proof −
  have ⋀f :: ('a comp, ('a, 'a) functor-category.arr) cell. ide f ⟹ hcomp f (src f) = f
  proof −
    fix f :: ('a comp, ('a, 'a) functor-category.arr) cell
    assume f: ide f
    let ?A = Src f and ?B = Trg f
    interpret A: category ?A
      using f Src-in-Obj [of f] by simp
    interpret B: category ?B
      using f Trg-in-Obj [of f] by simp
    interpret AA: functor-category ?A ?A ..
    interpret AB: functor-category ?A ?B ..
    have 1: functor ?A ?B (AB.Cod (Map f)) ∧ functor ?A ?B (AB.Dom (Map f)) ∧
            functor ?A ?B (AB.Map (Map f))
      using f ide-char''' AB.ide-char [of Map f] ide-char'' [of f] arr-char [of f]
      by auto
    have hcomp f (src f) = MkCell ?A ?B (Map f)
      unfolding hcomp-def src-def
      using f 1 AB.arr-char AB.null-char AB.ide-char
            obj-MkObj [of Src f] obj-def functor-is-transformation ide-char'''
            AB.ide-MkIde Src-in-Obj [of f] obj-MkObj obj-def AB.null-char arr-Map [of f]
            A.functor-axioms AB.MkArr-Map arr-Map
      by auto
    also have ... = f
      using f MkCell-Map [of f] not-arr-null ideD(1) [of f] by fastforce
    finally show hcomp f (src f) = f by blast
  qed
  moreover have
    ⋀f :: ('a comp, ('a, 'a) functor-category.arr) cell. ide f ⟹ hcomp (trg f) f = f
```

```
    proof −
      fix f :: ('a comp, ('a, 'a) functor-category.arr) cell
      assume f: ide f
      let ?A = Src f and ?B = Trg f
      interpret A: category ?A
        using f Src-in-Obj [of f] by simp
      interpret B: category ?B
        using f Trg-in-Obj [of f] by simp
      interpret BB: functor-category ?B ?B ..
      interpret AB: functor-category ?A ?B ..
      have 1: functor ?A ?B (AB.Cod (Map f)) ∧ functor ?A ?B (AB.Dom (Map f)) ∧
              functor ?A ?B (AB.Map (Map f))
        using f ide-char''' AB.ide-char [of Map f] ide-char'' [of f] arr-char [of f]
        by auto
      have hcomp (trg f) f = MkCell ?A ?B (Map f)
        unfolding hcomp-def trg-def
        using f 1 AB.arr-char AB.null-char AB.ide-char
              obj-MkObj [of Trg f] obj-def functor-is-transformation ide-char'''
              AB.ide-MkIde Trg-in-Obj [of f] obj-MkObj obj-def AB.null-char arr-Map [of f]
              B.functor-axioms AB.MkArr-Map arr-Map
        by auto
      also have ... = f
        using f MkCell-Map [of f] not-arr-null ideD(1) [of f] by fastforce
      finally show hcomp (trg f) f = f by blast
    qed
    moreover have ⋀a. obj a ⟹ ide (i a)
      using i-def by (metis (mono-tags, lifting) obj-MkObj-Src objE obj-def)
    moreover have ⋀(f :: ('a comp, ('a, 'a) functor-category.arr) cell) g h.
                    ⟦ ide f; ide g; ide h; src f = trg g; src g = trg h ⟧ ⟹ ide (a f g h)
      by (simp add: assoc-simp)
    ultimately show ?thesis
      using is-strict-if by blast
  qed

  sublocale strict-bicategory vcomp hcomp a i src trg
    using is-strict-bicategory by simp

 end

end
```

1.13 Adjunctions in a Bicategory

theory *InternalAdjunction*
imports *CanonicalIsos Strictness*
begin

An *internal adjunction* in a bicategory is a four-tuple $(f, g, \eta, \varepsilon)$, where f and g are antiparallel 1-cells and «η : *src* $f \Rightarrow g \star f$» and «ε : $f \star g \Rightarrow$ *src* g» are 2-cells, such

that the familiar "triangle" (or "zig-zag") identities are satisfied. We state the triangle identities in two equivalent forms, each of which is convenient in certain situations.

locale *adjunction-in-bicategory* =
  adjunction-data-in-bicategory +
assumes *triangle-left*: $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
and *triangle-right*: $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
begin

  lemma *triangle-left'*:
  shows $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
    using *triangle-left triangle-equiv-form* **by** *simp*

  lemma *triangle-right'*:
  shows $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] = g$
    using *triangle-right triangle-equiv-form* **by** *simp*

end

Internal adjunctions have a number of properties, which we now develop, that generalize those of ordinary adjunctions involving functors and natural transformations.

context *bicategory*
begin

  lemma *adjunction-unit-determines-counit*:
  assumes *adjunction-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f g* η ε
  and *adjunction-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f g* η ε'
  shows $\varepsilon = \varepsilon'$
  proof −
    interpret *E*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
      using *assms*(*1*) **by** *auto*
    interpret *E'*: *adjunction-in-bicategory V H* a i *src trg f g* η ε'
      using *assms*(*2*) **by** *auto*

Note that since we want to prove the the result for an arbitrary bicategory, not just in for a strict bicategory, the calculation is a little more involved than one might expect from a treatment that suppresses canonical isomorphisms.

    have $\varepsilon = \varepsilon \cdot (f \star \mathrm{r}[g] \cdot (g \star \varepsilon') \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g])$
      using *E'.triangle-right' comp-arr-dom* **by** *simp*
    also have ... $= \varepsilon \cdot (f \star \mathrm{r}[g]) \cdot (f \star g \star \varepsilon') \cdot (f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
      using *E.antipar whisker-left* **by** *simp*
    also have ... $= \varepsilon \cdot ((f \star \mathrm{r}[g]) \cdot (f \star g \star \varepsilon')) \cdot (f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
      using *comp-assoc* **by** *simp*
    also have ... $= \varepsilon \cdot \mathrm{r}[f \star g] \cdot (\mathrm{a}^{-1}[f, g, src\ g] \cdot (f \star g \star \varepsilon')) \cdot$
      $(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
    proof −
      have $f \star \mathrm{r}[g] = \mathrm{r}[f \star g] \cdot \mathrm{a}^{-1}[f, g, src\ g]$
        using *E.antipar*(*1*) *runit-hcomp*(*3*) **by** *auto*
      thus *?thesis*

using *comp-assoc* **by** *simp*
qed
also have ... $= (\varepsilon \cdot \mathrm{r}[f \star g]) \cdot ((f \star g) \star \varepsilon') \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar E'.counit-in-hom assoc'-naturality* [*of f g ε'*] *comp-assoc* **by** *simp*
also have ... $= \mathrm{r}[trg\ f] \cdot ((\varepsilon \star trg\ f) \cdot ((f \star g) \star \varepsilon')) \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar E.counit-in-hom runit-naturality* [*of ε*] *comp-assoc* **by** *simp*
also have ... $= (\mathrm{l}[src\ g] \cdot (src\ g \star \varepsilon')) \cdot (\varepsilon \star f \star g) \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
proof −
have $(\varepsilon \star trg\ f) \cdot ((f \star g) \star \varepsilon') = (src\ g \star \varepsilon') \cdot (\varepsilon \star f \star g)$
using *E.antipar interchange E.counit-in-hom comp-arr-dom comp-cod-arr*
by (*metis E'.counit-simps*(*1*−*3*) *E.counit-simps*(*1*−*3*))
thus *?thesis*
using *E.antipar comp-assoc unitor-coincidence* **by** *simp*
qed
also have ... $= \varepsilon' \cdot \mathrm{l}[f \star g] \cdot (\varepsilon \star f \star g) \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
proof −
have $\mathrm{l}[src\ g] \cdot (src\ g \star \varepsilon') = \varepsilon' \cdot \mathrm{l}[f \star g]$
using *E.antipar lunit-naturality* [*of ε'*] **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \star g) \cdot (\mathrm{a}^{-1}[trg\ f, f, g] \cdot (\varepsilon \star f \star g)) \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g]) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar lunit-hcomp comp-assoc* **by** *simp*
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \star g) \cdot ((\varepsilon \star f) \star g) \cdot (\mathrm{a}^{-1}[f \star g, f, g] \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot$
$(f \star \mathrm{a}[g, f, g])) \cdot (f \star \eta \star g) \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar assoc'-naturality* [*of ε f g*] *comp-assoc* **by** *simp*
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \star g) \cdot ((\varepsilon \star f) \star g) \cdot (\mathrm{a}^{-1}[f, g, f] \star g) \cdot$
$(\mathrm{a}^{-1}[f, g \star f, g] \cdot (f \star \eta \star g)) \cdot (f \star \mathrm{l}^{-1}[g])$
proof −
have $\mathrm{a}^{-1}[f \star g, f, g] \cdot \mathrm{a}^{-1}[f, g, f \star g] \cdot (f \star \mathrm{a}[g, f, g]) =$
$(\mathrm{a}^{-1}[f, g, f] \star g) \cdot \mathrm{a}^{-1}[f, g \star f, g]$
using *E.antipar iso-assoc' pentagon' comp-assoc*
invert-side-of-triangle(*2*)
[*of* $\mathrm{a}^{-1}[f \star g, f, g] \cdot \mathrm{a}^{-1}[f, g, f \star g]$
$(\mathrm{a}^{-1}[f, g, f] \star g) \cdot \mathrm{a}^{-1}[f, g \star f, g]$ $f \star \mathrm{a}^{-1}[g, f, g]$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \star g) \cdot ((\varepsilon \star f) \star g) \cdot (\mathrm{a}^{-1}[f, g, f] \star g) \cdot$
$((f \star \eta) \star g) \cdot \mathrm{a}^{-1}[f, trg\ g, g] \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar assoc'-naturality* [*of f η g*] *comp-assoc* **by** *simp*
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \star g) \cdot ((\varepsilon \star f) \star g) \cdot (\mathrm{a}^{-1}[f, g, f] \star g) \cdot$
$((f \star \eta) \star g) \cdot (\mathrm{r}^{-1}[f] \star g)$

proof −
have $\mathrm{a}^{-1}[f,\ trg\ g,\ g] \cdot (f \star \mathrm{l}^{-1}[g]) = \mathrm{r}^{-1}[f] \star g$
proof −
have $\mathrm{r}^{-1}[f] \star g = inv\ (\mathrm{r}[f] \star g)$
using *E.antipar* **by** *simp*
also have ... $= inv\ ((f \star \mathrm{l}[g]) \cdot \mathrm{a}[f,\ trg\ g,\ g])$
using *E.antipar* **by** (*simp add*: *triangle*)
also have ... $= \mathrm{a}^{-1}[f,\ trg\ g,\ g] \cdot (f \star \mathrm{l}^{-1}[g])$
using *E.antipar inv-comp* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= \varepsilon' \cdot (\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] \star g)$
using *E.antipar whisker-right* **by** *simp*
also have ... $= \varepsilon'$
using *E.triangle-left′ comp-arr-dom* **by** *simp*
finally show *?thesis* **by** *simp*
qed

end

1.13.1 Adjoint Transpose

context *adjunction-in-bicategory*
begin

interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
notation *E.eval* (‹⦃-⦄›)

Just as for an ordinary adjunction between categories, an adjunction in a bicategory determines bijections between hom-sets. There are two versions of this relationship: depending on whether the transposition is occurring on the left (*i.e.* "output") side or the right (*i.e.* "input") side.

definition $trnl_\eta$
where $trnl_\eta\ v\ \mu \equiv (g \star \mu) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v]$

definition $trnl_\varepsilon$
where $trnl_\varepsilon\ u\ \nu \equiv \mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \nu)$

lemma *adjoint-transpose-left*:
assumes *ide u* **and** *ide v* **and** *src f* = *trg v* **and** *src g* = *trg u*
shows $trnl_\eta\ v \in hom\ (f \star v)\ u \rightarrow hom\ v\ (g \star u)$
and $trnl_\varepsilon\ u \in hom\ v\ (g \star u) \rightarrow hom\ (f \star v)\ u$
and $«\mu : f \star v \Rightarrow u» \Longrightarrow trnl_\varepsilon\ u\ (trnl_\eta\ v\ \mu) = \mu$
and $«\nu : v \Rightarrow g \star u» \Longrightarrow trnl_\eta\ v\ (trnl_\varepsilon\ u\ \nu) = \nu$
and $bij\text{-}betw\ (trnl_\eta\ v)\ (hom\ (f \star v)\ u)\ (hom\ v\ (g \star u))$
and $bij\text{-}betw\ (trnl_\varepsilon\ u)\ (hom\ v\ (g \star u))\ (hom\ (f \star v)\ u)$
proof −

show A: $trnl_\eta\ v \in hom\ (f \star v)\ u \to hom\ v\ (g \star u)$
  using *assms antipar* $trnl_\eta$*-def* **by** *fastforce*
show B: $trnl_\varepsilon\ u \in hom\ v\ (g \star u) \to hom\ (f \star v)\ u$
  using *assms antipar* $trnl_\varepsilon$*-def* **by** *fastforce*
show C: $\bigwedge\mu.\ «\mu : f \star v \Rightarrow u» \Longrightarrow trnl_\varepsilon\ u\ (trnl_\eta\ v\ \mu) = \mu$
proof −
  fix μ
  assume μ: $«\mu : f \star v \Rightarrow u»$
  have $trnl_\varepsilon\ u\ (trnl_\eta\ v\ \mu) =$
      $\mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star (g \star \mu) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v])$
    using $trnl_\eta$*-def* $trnl_\varepsilon$*-def* **by** *simp*
  also have ... $= \mathrm{l}[u] \cdot (\varepsilon \star u) \cdot (\mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star g \star \mu)) \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot$
      $(f \star \eta \star v) \cdot (f \star \mathrm{l}^{-1}[v])$
    using *assms* μ *antipar whisker-left comp-assoc* **by** *auto*
  also have ... $= \mathrm{l}[u] \cdot ((\varepsilon \star u) \cdot ((f \star g) \star \mu)) \cdot \mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot$
      $(f \star \eta \star v) \cdot (f \star \mathrm{l}^{-1}[v])$
    using *assms* μ *antipar assoc′-naturality* [*of f g* μ] *comp-assoc* **by** *fastforce*
  also have ... $= \mathrm{l}[u] \cdot (trg\ u \star \mu) \cdot$
      $(\varepsilon \star f \star v) \cdot \mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot$
        $(f \star \eta \star v) \cdot (f \star \mathrm{l}^{-1}[v])$
  proof −
    have $(\varepsilon \star u) \cdot ((f \star g) \star \mu) = (trg\ u \star \mu) \cdot (\varepsilon \star f \star v)$
      using *assms* μ *antipar comp-cod-arr comp-arr-dom*
        interchange [*of trg u* ε μ $f \star v$] *interchange* [*of* ε $f \star g$ u μ]
      by *auto*
    thus *?thesis*
      using *comp-assoc* **by** *simp*
  qed
  also have ... $= \mathrm{l}[u] \cdot (trg\ u \star \mu) \cdot \mathrm{a}[trg\ f,\ f,\ v] \cdot$
      $((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \star v) \cdot$
        $\mathrm{a}^{-1}[f,\ trg\ v,\ v] \cdot (f \star \mathrm{l}^{-1}[v])$
  proof −
    have *1*: $(\varepsilon \star f \star v) \cdot \mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot (f \star \eta \star v) =$
        $\mathrm{a}[trg\ f,\ f,\ v] \cdot ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \star v) \cdot \mathrm{a}^{-1}[f,\ trg\ v,\ v]$
    proof −
      have $(\varepsilon \star f \star v) \cdot \mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot (f \star \eta \star v) =$
          $(\varepsilon \star f \star v) \cdot$
            $\mathrm{a}[f \star g,\ f,\ v] \cdot (\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot \mathrm{a}^{-1}[f,\ g \star f,\ v] \cdot$
              $(f \star \eta \star v)$
      proof −
        have $(\varepsilon \star f \star v) \cdot \mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) \cdot (f \star \eta \star v) =$
            $(\varepsilon \star f \star v) \cdot (\mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v])) \cdot (f \star \eta \star v)$
          using *comp-assoc* **by** *simp*
        also have ... $= (\varepsilon \star f \star v) \cdot$
            $\mathrm{a}[f \star g,\ f,\ v] \cdot (\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot \mathrm{a}^{-1}[f,\ g \star f,\ v] \cdot$
              $(f \star \eta \star v)$
        proof −
          have $\mathrm{a}^{-1}[f,\ g,\ f \star v] \cdot (f \star \mathrm{a}[g,\ f,\ v]) =$
              $\mathrm{a}[f \star g,\ f,\ v] \cdot (\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot \mathrm{a}^{-1}[f,\ g \star f,\ v]$

using *assms antipar canI-associator-0 whisker-can-left-0 whisker-can-right-0 canI-associator-hcomp(1−3)*
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
also have ... $= ((\varepsilon \star f \star v) \cdot \mathrm{a}[f \star g,\ f,\ v]) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot ((f \star \eta) \star v) \cdot$
$\mathrm{a}^{-1}[f,\ trg\ v,\ v]$
using *assms* μ *antipar assoc′-naturality* [*of f* η *v*] *comp-assoc* **by** *simp*
also have ... $= (\mathrm{a}[trg\ f,\ f,\ v] \cdot ((\varepsilon \star f) \star v)) \cdot (\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot ((f \star \eta) \star v) \cdot$
$\mathrm{a}^{-1}[f,\ trg\ v,\ v]$
using *assms* μ *antipar assoc-naturality* [*of* ε *f v*] **by** *simp*
also have ... $= \mathrm{a}[trg\ f,\ f,\ v] \cdot$
$(((\varepsilon \star f) \star v) \cdot (\mathrm{a}^{-1}[f,\ g,\ f] \star v) \cdot ((f \star \eta) \star v)) \cdot$
$\mathrm{a}^{-1}[f,\ trg\ v,\ v]$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}[trg\ f,\ f,\ v] \cdot ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \star v) \cdot \mathrm{a}^{-1}[f,\ trg\ v,\ v]$
using *assms* μ *antipar whisker-right* **by** *simp*
finally show *?thesis* **by** *simp*
qed
show *?thesis*
using *1 comp-assoc* **by** *metis*
qed
also have ... $= \mathrm{l}[u] \cdot (trg\ u \star \mu) \cdot$
$\mathrm{a}[trg\ f,\ f,\ v] \cdot (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star v) \cdot \mathrm{a}^{-1}[f,\ trg\ v,\ v] \cdot (f \star \mathrm{l}^{-1}[v])$
using *assms* μ *antipar triangle-left* **by** *simp*
also have ... $= \mathrm{l}[u] \cdot (trg\ u \star \mu) \cdot can\ (\langle trg\ u\rangle_0 \star \langle f\rangle \star \langle v\rangle)\ (\langle f\rangle \star \langle v\rangle)$
using *assms* μ *antipar canI-unitor-0 canI-associator-1 canI-associator-1(1−2)* [*of f v*] *whisker-can-right-0 whisker-can-left-0*
by *simp*
also have ... $= \mathrm{l}[u] \cdot (trg\ u \star \mu) \cdot \mathrm{l}^{-1}[f \star v]$
unfolding *can-def* **using** *assms antipar comp-arr-dom comp-cod-arr* ‹*-ide-simp*
by *simp*
also have ... $= (\mathrm{l}[u] \cdot \mathrm{l}^{-1}[u]) \cdot \mu$
using *assms* μ *lunit′-naturality* [*of* μ] *comp-assoc* **by** *auto*
also have ... $= \mu$
using *assms* μ *comp-cod-arr iso-lunit comp-arr-inv inv-is-inverse* **by** *auto*
finally show $trnl_\varepsilon\ u\ (trnl_\eta\ v\ \mu) = \mu$ **by** *simp*
qed
show *D*: $\bigwedge\nu.$ «$\nu : v \Rightarrow g \star u$» $\Longrightarrow trnl_\eta\ v\ (trnl_\varepsilon\ u\ \nu) = \nu$
proof −
fix ν
assume ν: «$\nu : v \Rightarrow g \star u$»
have $trnl_\eta\ v\ (trnl_\varepsilon\ u\ \nu) =$
$(g \star \mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \nu)) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v]$
using $trnl_\eta$*-def* $trnl_\varepsilon$*-def* **by** *simp*

also have ... $= (g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot (g \star f \star \nu) \cdot$
$\mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v]$
using *assms* ν *antipar interchange* [*of* $g\ g \cdot g \cdot g$] *comp-assoc* **by** *auto*
also have ... $= ((g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot$
$\mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u)) \cdot (trg\ v \star \nu) \cdot \mathrm{l}^{-1}[v]$
proof −
have $(g \star f \star \nu) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v] =$
$\mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u) \cdot (trg\ v \star \nu) \cdot \mathrm{l}^{-1}[v]$
proof −
have $(g \star f \star \nu) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v] =$
$\mathrm{a}[g,\ f,\ g \star u] \cdot ((g \star f) \star \nu) \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v]$
proof −
have $(g \star f \star \nu) \cdot \mathrm{a}[g,\ f,\ v] = \mathrm{a}[g,\ f,\ g \star u] \cdot ((g \star f) \star \nu)$
using *assms* ν *antipar assoc-naturality* [*of* $g\ f\ \nu$] **by** *auto*
thus *?thesis*
using *assms comp-assoc* **by** *metis*
qed
also have ... $= \mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u) \cdot (trg\ v \star \nu) \cdot \mathrm{l}^{-1}[v]$
proof −
have $((g \star f) \star \nu) \cdot (\eta \star v) = (\eta \star g \star u) \cdot (trg\ v \star \nu)$
using *assms* ν *antipar comp-arr-dom comp-cod-arr*
interchange [*of* $g \star f\ \eta\ \nu\ v$] *interchange* [*of* $\eta\ trg\ v\ g \star u\ \nu$]
by *auto*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **using** *comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{l}[g \star u] \cdot (trg\ v \star \nu) \cdot \mathrm{l}^{-1}[v]$
proof −
have $(g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u) =$
$\mathrm{l}[g \star u]$
proof −
have $(g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u) =$
$(g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot$
$((g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) \star u) \cdot$
$\mathrm{a}^{-1}[trg\ v,\ g,\ u]$
proof −
have $(g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot (\eta \star g \star u) =$
$(g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot$
$((\eta \star g \star u) \cdot \mathrm{a}[trg\ v,\ g,\ u]) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
using *assms antipar comp-arr-dom comp-assoc comp-assoc-assoc′*(*1*) **by** *simp*
also have ... $= (g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot$
$(\mathrm{a}[g \star f,\ g,\ u] \cdot ((\eta \star g) \star u)) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
using *assms antipar assoc-naturality* [*of* $\eta\ g\ u$] **by** *simp*
also have ... $= (g \star \mathrm{l}[u]) \cdot (g \star \varepsilon \star u) \cdot$
$((g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u]) \cdot$

$((\eta \star g) \star u) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
using *comp-assoc* **by** *simp*
also have ... $= (g \star \mathrm{l}[u]) \cdot ((\mathrm{a}[g,\ trg\ u,\ u] \cdot \mathrm{a}^{-1}[g,\ trg\ u,\ u]) \cdot (g \star \varepsilon \star u)) \cdot$
$((g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u]) \cdot$
$((\eta \star g) \star u) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
proof −
have $(\mathrm{a}[g,\ trg\ u,\ u] \cdot \mathrm{a}^{-1}[g,\ trg\ u,\ u]) \cdot (g \star \varepsilon \star u) = g \star \varepsilon \star u$
using *assms antipar comp-cod-arr comp-assoc-assoc′(1)* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot (\mathrm{a}^{-1}[g,\ trg\ u,\ u] \cdot (g \star \varepsilon \star u)) \cdot$
$(g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u] \cdot$
$((\eta \star g) \star u) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
using *comp-assoc* **by** *simp*
also have ... $= (g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot (((g \star \varepsilon) \star u) \cdot (\mathrm{a}^{-1}[g,\ f \star g,\ u] \cdot$
$(g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u]) \cdot$
$((\eta \star g) \star u)) \cdot \mathrm{a}^{-1}[trg\ v,\ g,\ u]$
using *assms antipar assoc′-naturality* [*of g ε u*] *comp-assoc* **by** *simp*
also have ... $= (g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot$
$((g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) \star u) \cdot$
$\mathrm{a}^{-1}[trg\ v,\ g,\ u]$
proof −
have $\mathrm{a}^{-1}[g,\ f \star g,\ u] \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u] =$
$\mathrm{a}[g,\ f,\ g] \star u$
using *assms antipar canI-associator-0 whisker-can-left-0 whisker-can-right-0 canI-associator-hcomp*
by *simp*
hence $((g \star \varepsilon) \star u) \cdot$
$(\mathrm{a}^{-1}[g,\ f \star g,\ u] \cdot (g \star \mathrm{a}^{-1}[f,\ g,\ u]) \cdot \mathrm{a}[g,\ f,\ g \star u] \cdot \mathrm{a}[g \star f,\ g,\ u]) \cdot$
$((\eta \star g) \star u) =$
$(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) \star u$
using *assms antipar whisker-right* **by** *simp*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
also have ... $= (g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot (\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \star u) \cdot \mathrm{a}^{-1}[trg\ g,\ g,\ u]$
using *assms antipar triangle-right* **by** *simp*
also have ... $= can\ (\langle g \rangle \star \langle u \rangle)\ (\langle trg\ g \rangle_0 \star \langle g \rangle \star \langle u \rangle)$
proof −
have $(g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot (\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \star u) \cdot \mathrm{a}^{-1}[trg\ g,\ g,\ u] =$
$((g \star \mathrm{l}[u]) \cdot \mathrm{a}[g,\ trg\ u,\ u] \cdot (\mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \star u) \cdot \mathrm{a}^{-1}[trg\ g,\ g,\ u])$
using *comp-assoc* **by** *simp*
moreover have ... $= can\ (\langle g \rangle \star \langle u \rangle)\ (\langle trg\ g \rangle_0 \star \langle g \rangle \star \langle u \rangle)$
using *assms antipar canI-unitor-0 canI-associator-1* [*of g u*] *inv-can whisker-can-left-0 whisker-can-right-0*
by *simp*
ultimately show *?thesis* **by** *simp*

qed
also have ... $= \mathrm{l}[g \star u]$
unfolding *can-def* **using** *assms comp-arr-dom comp-cod-arr* ‹*-ide-simp* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (\mathrm{l}[g \star u] \cdot \mathrm{l}^{-1}[g \star u]) \cdot \nu$
using *assms* ν *lunit′-naturality comp-assoc* **by** *auto*
also have ... $= \nu$
using *assms* ν *comp-cod-arr iso-lunit comp-arr-inv inv-is-inverse* **by** *auto*
finally show $trnl_\eta\ v\ (trnl_\varepsilon\ u\ \nu) = \nu$ **by** *simp*
qed
show *bij-betw* $(trnl_\eta\ v)$ $(hom\ (f \star v)\ u)$ $(hom\ v\ (g \star u))$
using *A B C D* **by** (*intro bij-betwI*) *auto*
show *bij-betw* $(trnl_\varepsilon\ u)$ $(hom\ v\ (g \star u))$ $(hom\ (f \star v)\ u)$
using *A B C D* **by** (*intro bij-betwI*) *auto*
qed

lemma $trnl_\varepsilon$*-comp*:
assumes *ide u* **and** *seq* $\mu\ \nu$ **and** *src f* = *trg* μ
shows $trnl_\varepsilon\ u\ (\mu \cdot \nu) = trnl_\varepsilon\ u\ \mu \cdot (f \star \nu)$
using *assms* $trnl_\varepsilon$*-def whisker-left* [*of f* $\mu\ \nu$] *comp-assoc* **by** *auto*

definition $trnr_\eta$
where $trnr_\eta\ v\ \mu \equiv (\mu \star f) \cdot \mathrm{a}^{-1}[v,\ g,\ f] \cdot (v \star \eta) \cdot \mathrm{r}^{-1}[v]$

definition $trnr_\varepsilon$
where $trnr_\varepsilon\ u\ \nu \equiv \mathrm{r}[u] \cdot (u \star \varepsilon) \cdot \mathrm{a}[u,\ f,\ g] \cdot (\nu \star g)$

lemma *adjoint-transpose-right*:
assumes *ide u* **and** *ide v* **and** *src v* = *trg g* **and** *src u* = *trg f*
shows $trnr_\eta\ v \in hom\ (v \star g)\ u \rightarrow hom\ v\ (u \star f)$
and $trnr_\varepsilon\ u \in hom\ v\ (u \star f) \rightarrow hom\ (v \star g)\ u$
and «$\mu : v \star g \Rightarrow u$» $\Longrightarrow trnr_\varepsilon\ u\ (trnr_\eta\ v\ \mu) = \mu$
and «$\nu : v \Rightarrow u \star f$» $\Longrightarrow trnr_\eta\ v\ (trnr_\varepsilon\ u\ \nu) = \nu$
and *bij-betw* $(trnr_\eta\ v)$ $(hom\ (v \star g)\ u)$ $(hom\ v\ (u \star f))$
and *bij-betw* $(trnr_\varepsilon\ u)$ $(hom\ v\ (u \star f))$ $(hom\ (v \star g)\ u)$
proof −
show *A*: $trnr_\eta\ v \in hom\ (v \star g)\ u \rightarrow hom\ v\ (u \star f)$
using *assms antipar* $trnr_\eta$*-def* **by** *fastforce*
show *B*: $trnr_\varepsilon\ u \in hom\ v\ (u \star f) \rightarrow hom\ (v \star g)\ u$
using *assms antipar* $trnr_\varepsilon$*-def* **by** *fastforce*
show *C*: $\bigwedge\mu.$ «$\mu : v \star g \Rightarrow u$» $\Longrightarrow trnr_\varepsilon\ u\ (trnr_\eta\ v\ \mu) = \mu$
proof −
fix μ
assume μ: «$\mu : v \star g \Rightarrow u$»
have $trnr_\varepsilon\ u\ (trnr_\eta\ v\ \mu) =$
$\mathrm{r}[u] \cdot (u \star \varepsilon) \cdot \mathrm{a}[u,\ f,\ g] \cdot ((\mu \star f) \cdot \mathrm{a}^{-1}[v,\ g,\ f] \cdot (v \star \eta) \cdot \mathrm{r}^{-1}[v] \star g)$

unfolding *trnr*$_\varepsilon$*-def trnr*$_\eta$*-def* **by** *simp*
also have ... $= \mathrm{r}[u] \cdot (u \star \varepsilon) \cdot (\mathrm{a}[u, f, g] \cdot ((\mu \star f) \star g)) \cdot$
$(\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) \cdot (\mathrm{r}^{-1}[v] \star g)$
using *assms* μ *antipar whisker-right comp-assoc* **by** *auto*
also have ... $= \mathrm{r}[u] \cdot (u \star \varepsilon) \cdot ((\mu \star f \star g) \cdot \mathrm{a}[v \star g, f, g]) \cdot$
$(\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) \cdot (\mathrm{r}^{-1}[v] \star g)$
using *assms* μ *antipar assoc-naturality* [*of* μ f g] **by** *auto*
also have ... $= \mathrm{r}[u] \cdot ((u \star \varepsilon) \cdot (\mu \star f \star g)) \cdot \mathrm{a}[v \star g, f, g] \cdot$
$(\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) \cdot (\mathrm{r}^{-1}[v] \star g)$
using *comp-assoc* **by** *auto*
also have ... $= \mathrm{r}[u] \cdot (\mu \star src\ u) \cdot ((v \star g) \star \varepsilon) \cdot \mathrm{a}[v \star g, f, g] \cdot$
$(\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) \cdot (\mathrm{r}^{-1}[v] \star g)$
proof −
have $(u \star \varepsilon) \cdot (\mu \star f \star g) = (\mu \star src\ u) \cdot ((v \star g) \star \varepsilon)$
using *assms* μ *antipar comp-arr-dom comp-cod-arr*
interchange [*of* μ $v \star g$ *src* u ε] *interchange* [*of* u μ ε $f \star g$]
by *auto*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}[u] \cdot (\mu \star src\ u) \cdot$
$(((v \star g) \star \varepsilon) \cdot \mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g)) \cdot$
$(\mathrm{r}^{-1}[v] \star g)$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{r}[u] \cdot (\mu \star src\ u) \cdot$
$(\mathrm{a}^{-1}[v, g, src\ u] \cdot (v \star (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)) \cdot$
$\mathrm{a}[v, src\ v, g]) \cdot (\mathrm{r}^{-1}[v] \star g)$
proof −
have $((v \star g) \star \varepsilon) \cdot \mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) =$
$\mathrm{a}^{-1}[v, g, src\ u] \cdot (v \star (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)) \cdot \mathrm{a}[v, src\ v, g]$
proof −
have $((v \star g) \star \varepsilon) \cdot \mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g) =$
$((\mathrm{a}^{-1}[v, g, src\ u] \cdot \mathrm{a}[v, g, src\ u]) \cdot ((v \star g) \star \varepsilon)) \cdot$
$\mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g)$
proof −
have $arr\ v \wedge dom\ v = v \wedge cod\ v = v$
using *assms*(*2*) *ide-char* **by** *blast*
moreover have $arr\ g \wedge dom\ g = g \wedge cod\ g = g$
using *ide-right ide-char* **by** *blast*
ultimately show *?thesis*
by (*metis* (*no-types*) *antipar*(*2*) *assms*(*3*−*4*) *assoc-naturality*
counit-simps(*1*,*3*,*5*) *hcomp-reassoc*(*1*) *comp-assoc*)
qed
also have ... $= \mathrm{a}^{-1}[v, g, src\ u] \cdot (\mathrm{a}[v, g, src\ u] \cdot ((v \star g) \star \varepsilon)) \cdot$
$\mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot ((v \star \eta) \star g)$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}^{-1}[v, g, src\ u] \cdot ((v \star g \star \varepsilon) \cdot \mathrm{a}[v, g, f \star g]) \cdot$
$\mathrm{a}[v \star g, f, g] \cdot (\mathrm{a}^{-1}[v, g, f] \star g) \cdot$
$(\mathrm{a}^{-1}[v, g \star f, g] \cdot \mathrm{a}[v, g \star f, g]) \cdot ((v \star \eta) \star g)$

proof −
have $\mathrm{a}[v,\ g,\ src\ u] \cdot ((v \star g) \star \varepsilon) = (v \star g \star \varepsilon) \cdot \mathrm{a}[v,\ g,\ f \star g]$
using *assms antipar assoc-naturality* [*of v g* ε] **by** *simp*
moreover have $(\mathrm{a}^{-1}[v,\ g \star f,\ g] \cdot \mathrm{a}[v,\ g \star f,\ g]) \cdot ((v \star \eta) \star g) = (v \star \eta) \star g$
using *assms antipar comp-cod-arr comp-assoc-assoc′*(*2*) **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot (v \star g \star \varepsilon) \cdot$
$\mathrm{a}[v,\ g,\ f \star g] \cdot \mathrm{a}[v \star g,\ f,\ g] \cdot (\mathrm{a}^{-1}[v,\ g,\ f] \star g) \cdot$
$\mathrm{a}^{-1}[v,\ g \star f,\ g] \cdot \mathrm{a}[v,\ g \star f,\ g] \cdot ((v \star \eta) \star g)$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot ((v \star g \star \varepsilon) \cdot$
$(\mathrm{a}[v,\ g,\ f \star g] \cdot \mathrm{a}[v \star g,\ f,\ g] \cdot (\mathrm{a}^{-1}[v,\ g,\ f] \star g) \cdot$
$\mathrm{a}^{-1}[v,\ g \star f,\ g]) \cdot (v \star \eta \star g)) \cdot \mathrm{a}[v,\ src\ v,\ g]$
using *assms antipar assoc-naturality* [*of v* η *g*] *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot$
$((v \star g \star \varepsilon) \cdot (v \star \mathrm{a}[g,\ f,\ g]) \cdot (v \star \eta \star g)) \cdot$
$\mathrm{a}[v,\ src\ v,\ g]$
proof −
have $\mathrm{a}[v,\ g,\ f \star g] \cdot \mathrm{a}[v \star g,\ f,\ g] \cdot (\mathrm{a}^{-1}[v,\ g,\ f] \star g) \cdot \mathrm{a}^{-1}[v,\ g \star f,\ g] =$
$v \star \mathrm{a}[g,\ f,\ g]$
using *assms antipar canI-associator-0 canI-associator-hcomp*
whisker-can-left-0 whisker-can-right-0
by *simp*
thus *?thesis*
using *assms antipar whisker-left* **by** *simp*
qed
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot$
$(v \star (g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g)) \cdot$
$\mathrm{a}[v,\ src\ v,\ g]$
using *assms antipar whisker-left* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **by** *auto*
qed
also have ... $= \mathrm{r}[u] \cdot (\mu \star src\ u) \cdot$
$\mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot (v \star \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]) \cdot$
$\mathrm{a}[v,\ src\ v,\ g] \cdot (\mathrm{r}^{-1}[v] \star g)$
using *triangle-right comp-assoc* **by** *simp*
also have ... $= \mathrm{r}[u] \cdot (\mu \star src\ u) \cdot \mathrm{r}^{-1}[v \star g]$
proof −
have $\mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot (v \star \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]) \cdot \mathrm{a}[v,\ src\ v,\ g] \cdot (\mathrm{r}^{-1}[v] \star g) = \mathrm{r}^{-1}[v \star g]$
proof −
have $\mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot (v \star \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]) \cdot \mathrm{a}[v,\ src\ v,\ g] \cdot (\mathrm{r}^{-1}[v] \star g) =$
$\mathrm{a}^{-1}[v,\ g,\ trg\ f] \cdot can\ (\langle v \rangle \star \langle g \rangle \star \langle src\ g \rangle_0)\ (\langle v \rangle \star \langle g \rangle)$
using *assms canI-unitor-0 canI-associator-1*(*2*−*3*) *whisker-can-left-0*(*1*)
whisker-can-right-0
by *simp*
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ g] \cdot can\ (\langle v \rangle \star \langle g \rangle \star \langle src\ g \rangle_0)\ (\langle v \rangle \star \langle g \rangle)$

using *antipar* **by** *simp*

also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ g] \cdot (v \star can\ (\langle g\rangle \star \langle src\ g\rangle_0)\ \langle g\rangle)$
using *assms canI-unitor-0(2) whisker-can-left-0* **by** *simp*
also have ... $= \mathrm{a}^{-1}[v,\ g,\ src\ g] \cdot (v \star \mathrm{r}^{-1}[g])$
using *assms canI-unitor-0(2)* **by** *simp*
also have ... $= \mathrm{r}^{-1}[v \star g]$
using *assms runit-hcomp(2)* **by** *simp*
finally have $\mathrm{a}^{-1}[v,\ g,\ src\ u] \cdot (v \star \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]) \cdot \mathrm{a}[v,\ src\ v,\ g] \cdot (\mathrm{r}^{-1}[v] \star g) =$
$\mathrm{r}^{-1}[v \star g]$
by *simp*
thus *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (\mathrm{r}[u] \cdot \mathrm{r}^{-1}[u]) \cdot \mu$
using *assms* μ *runit′-naturality* [*of* μ] *comp-assoc* **by** *auto*
also have ... $= \mu$
using μ *comp-cod-arr iso-runit inv-is-inverse comp-arr-inv* **by** *auto*
finally show $trnr_\varepsilon\ u\ (trnr_\eta\ v\ \mu) = \mu$ **by** *simp*
qed
show *D*: $\bigwedge\nu.\ «\nu : v \Rightarrow u \star f» \Longrightarrow trnr_\eta\ v\ (trnr_\varepsilon\ u\ \nu) = \nu$
proof −
fix ν
assume ν: $«\nu : v \Rightarrow u \star f»$
have $trnr_\eta\ v\ (trnr_\varepsilon\ u\ \nu) =$
$(\mathrm{r}[u] \cdot (u \star \varepsilon) \cdot \mathrm{a}[u,\ f,\ g] \cdot (\nu \star g) \star f) \cdot \mathrm{a}^{-1}[v,\ g,\ f] \cdot (v \star \eta) \cdot \mathrm{r}^{-1}[v]$
unfolding $trnr_\eta$*-def* $trnr_\varepsilon$*-def* **by** *simp*
also have ... $= (\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot (\mathrm{a}[u,\ f,\ g] \star f) \cdot$
$(((\nu \star g) \star f) \cdot \mathrm{a}^{-1}[v,\ g,\ f]) \cdot (v \star \eta) \cdot \mathrm{r}^{-1}[v]$
using *assms* ν *antipar whisker-right comp-assoc* **by** *auto*
also have ... $= (\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot (\mathrm{a}[u,\ f,\ g] \star f) \cdot$
$(\mathrm{a}^{-1}[u \star f,\ g,\ f] \cdot (\nu \star g \star f)) \cdot (v \star \eta) \cdot \mathrm{r}^{-1}[v]$
using *assms* ν *antipar assoc′-naturality* [*of* ν *g f*] **by** *auto*
also have ... $= (\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot (\mathrm{a}[u,\ f,\ g] \star f) \cdot$
$\mathrm{a}^{-1}[u \star f,\ g,\ f] \cdot ((\nu \star g \star f) \cdot (v \star \eta)) \cdot \mathrm{r}^{-1}[v]$
using *comp-assoc* **by** *simp*
also have ... $= (\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot (\mathrm{a}[u,\ f,\ g] \star f) \cdot$
$\mathrm{a}^{-1}[u \star f,\ g,\ f] \cdot (((u \star f) \star \eta) \cdot (\nu \star src\ v)) \cdot \mathrm{r}^{-1}[v]$
proof −
have $(\nu \star g \star f) \cdot (v \star \eta) = ((u \star f) \star \eta) \cdot (\nu \star src\ v)$
using *assms* ν *antipar interchange* [*of* $u \star f\ \nu\ \eta\ src\ v$]
interchange [*of* $\nu\ v\ g \star f\ \eta$] *comp-arr-dom comp-cod-arr*
by *auto*
thus *?thesis* **by** *simp*
qed
also have ... $= ((\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot$
$((\mathrm{a}[u,\ f,\ g] \star f) \cdot \mathrm{a}^{-1}[u \star f,\ g,\ f]) \cdot$
$((u \star f) \star \eta)) \cdot (\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$

using *comp-assoc* **by** *simp*
also have ... $= ((\mathrm{r}[u] \star f) \cdot ((u \star \varepsilon) \star f) \cdot$
$(\mathrm{a}^{-1}[u, f \star g, f] \cdot (u \star \mathrm{a}^{-1}[f, g, f]) \cdot \mathrm{a}[u, f, g \star f]) \cdot$
$((u \star f) \star \eta)) \cdot (\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *assms antipar canI-associator-hcomp canI-associator-0 whisker-can-left-0 whisker-can-right-0*
by *simp*
also have ... $= ((\mathrm{r}[u] \star f) \cdot (((u \star \varepsilon) \star f) \cdot$
$\mathrm{a}^{-1}[u, f \star g, f]) \cdot (u \star \mathrm{a}^{-1}[f, g, f]) \cdot (\mathrm{a}[u, f, g \star f]) \cdot$
$((u \star f) \star \eta)) \cdot (\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *comp-assoc* **by** *simp*
also have ... $= ((\mathrm{r}[u] \star f) \cdot (\mathrm{a}^{-1}[u, src\ u, f] \cdot (u \star \varepsilon \star f)) \cdot$
$(u \star \mathrm{a}^{-1}[f, g, f]) \cdot ((u \star f \star \eta) \cdot \mathrm{a}[u, f, src\ f])) \cdot$
$(\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *assms antipar assoc′-naturality* [*of u ε f*] *assoc-naturality* [*of u f η*]
by *auto*
also have ... $= (\mathrm{r}[u] \star f) \cdot \mathrm{a}^{-1}[u, src\ u, f] \cdot$
$((u \star \varepsilon \star f) \cdot (u \star \mathrm{a}^{-1}[f, g, f]) \cdot (u \star f \star \eta)) \cdot \mathrm{a}[u, f, src\ f] \cdot$
$(\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *comp-assoc* **by** *simp*
also have ... $= (\mathrm{r}[u] \star f) \cdot \mathrm{a}^{-1}[u, src\ u, f] \cdot$
$(u \star (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)) \cdot \mathrm{a}[u, f, src\ f] \cdot$
$(\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *assms antipar whisker-left* **by** *auto*
also have ... $= ((\mathrm{r}[u] \star f) \cdot (\mathrm{a}^{-1}[u, src\ u, f] \cdot (u \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]) \cdot \mathrm{a}[u, f, src\ f])) \cdot$
$(\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
using *assms antipar triangle-left comp-assoc* **by** *simp*
also have ... $= \mathrm{r}[u \star f] \cdot (\nu \star src\ v) \cdot \mathrm{r}^{-1}[v]$
proof −
have $(\mathrm{r}[u] \star f) \cdot \mathrm{a}^{-1}[u, src\ u, f] \cdot (u \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]) \cdot \mathrm{a}[u, f, src\ f] =$
$((u \star \mathrm{l}[f]) \cdot (\mathrm{a}[u, src\ u, f] \cdot \mathrm{a}^{-1}[u, src\ u, f])) \cdot$
$(u \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]) \cdot \mathrm{a}[u, f, src\ f]$
using *assms ide-left ide-right antipar triangle comp-assoc* **by** *metis*
also have ... $= (u \star \mathrm{r}[f]) \cdot \mathrm{a}[u, f, src\ f]$
using *assms antipar canI-associator-1 canI-unitor-0 whisker-can-left-0 whisker-can-right-0 canI-associator-1*
by *simp*
also have ... $= \mathrm{r}[u \star f]$
using *assms antipar runit-hcomp* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= (\mathrm{r}[u \star f] \cdot \mathrm{r}^{-1}[u \star f]) \cdot \nu$
using *assms ν runit′-naturality* [*of ν*] *comp-assoc* **by** *auto*
also have ... $= \nu$
using *assms ν comp-cod-arr comp-arr-inv inv-is-inverse iso-runit* **by** *auto*
finally show $trnr_\eta\ v\ (trnr_\varepsilon\ u\ \nu) = \nu$ **by** *auto*
qed
show *bij-betw* $(trnr_\eta\ v)$ (*hom* $(v \star g)\ u$) (*hom* $v\ (u \star f)$)
using *A B C D* **by** (*intro bij-betwI*, *auto*)

show *bij-betw* ($trnr_\varepsilon$ u) (*hom* v ($u \star f$)) (*hom* ($v \star g$) u)
using A B C D **by** (*intro bij-betwI*, *auto*)
qed

lemma $trnr_\eta$*-comp*:
assumes *ide* v **and** *seq* μ ν **and** *src* μ = *trg* f
shows $trnr_\eta$ v ($\mu \cdot \nu$) = ($\mu \star f$) $\cdot$ $trnr_\eta$ v ν
using *assms* $trnr_\eta$*-def whisker-right comp-assoc* **by** *auto*

end

It is useful to have at hand the simpler versions of the preceding results that hold in a normal bicategory and in a strict bicategory.

locale *adjunction-in-normal-bicategory* =
normal-bicategory +
adjunction-in-bicategory
begin

lemma *triangle-left*:
shows ($\varepsilon \star f$) $\cdot$ $\mathrm{a}^{-1}[f, g, f]$ $\cdot$ ($f \star \eta$) = f
using *triangle-left strict-lunit strict-runit* **by** *simp*

lemma *triangle-right*:
shows ($g \star \varepsilon$) $\cdot$ $\mathrm{a}[g, f, g]$ $\cdot$ ($\eta \star g$) = g
using *triangle-right strict-lunit strict-runit* **by** *simp*

lemma $trnr_\eta$*-eq*:
assumes *ide* u **and** *ide* v
and *src* v = *trg* g **and** *src* u = *trg* f
and $\mu \in$ *hom* ($v \star g$) u
shows $trnr_\eta$ v μ = ($\mu \star f$) $\cdot$ $\mathrm{a}^{-1}[v, g, f]$ $\cdot$ ($v \star \eta$)
unfolding $trnr_\eta$*-def*
using *assms antipar strict-runit′ comp-arr-ide* [*of* $\mathrm{r}^{-1}[v]$ $v \star \eta$] *hcomp-arr-obj*
by *auto*

lemma $trnr_\varepsilon$*-eq*:
assumes *ide* u **and** *ide* v
and *src* v = *trg* g **and** *src* u = *trg* f
and $\nu \in$ *hom* v ($u \star f$)
shows $trnr_\varepsilon$ u ν = ($u \star \varepsilon$) $\cdot$ $\mathrm{a}[u, f, g]$ $\cdot$ ($\nu \star g$)
unfolding $trnr_\varepsilon$*-def*
using *assms antipar strict-runit comp-ide-arr hcomp-arr-obj* **by** *auto*

lemma $trnl_\eta$*-eq*:
assumes *ide* u **and** *ide* v
and *src* f = *trg* v **and** *src* g = *trg* u
and $\mu \in$ *hom* ($f \star v$) u
shows $trnl_\eta$ v μ = ($g \star \mu$) $\cdot$ $\mathrm{a}[g, f, v]$ $\cdot$ ($\eta \star v$)
using *assms* $trnl_\eta$*-def antipar strict-lunit comp-arr-dom hcomp-obj-arr* **by** *auto*

lemma $trnl_\varepsilon$-*eq*:
assumes *ide u* **and** *ide v*
and *src f* = *trg v* **and** *src g* = *trg u*
and $\nu \in hom\ v\ (g \star u)$
shows $trnl_\varepsilon\ u\ \nu = (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \nu)$
using *assms* $trnl_\varepsilon$-*def antipar strict-lunit comp-cod-arr hcomp-obj-arr* **by** *auto*

end

locale *adjunction-in-strict-bicategory* =
strict-bicategory +
adjunction-in-normal-bicategory
begin

lemma *triangle-left*:
shows $(\varepsilon \star f) \cdot (f \star \eta) = f$
using *ide-left ide-right antipar triangle-left strict-assoc′ comp-cod-arr*
by (*metis dom-eqI ideD*(*1*) *seqE*)

lemma *triangle-right*:
shows $(g \star \varepsilon) \cdot (\eta \star g) = g$
using *ide-left ide-right antipar triangle-right strict-assoc comp-cod-arr*
by (*metis ideD*(*1*) *ideD*(*2*) *seqE*)

lemma $trnr_\eta$-*eq*:
assumes *ide u* **and** *ide v*
and *src v* = *trg g* **and** *src u* = *trg f*
and $\mu \in hom\ (v \star g)\ u$
shows $trnr_\eta\ v\ \mu = (\mu \star f) \cdot (v \star \eta)$
using *assms antipar* $trnr_\eta$-*eq strict-assoc′ comp-ide-arr* [*of* $\mathrm{a}^{-1}[v,\ g,\ f]\ v \star \eta$]
by *force*

lemma $trnr_\varepsilon$-*eq*:
assumes *ide u* **and** *ide v*
and *src v* = *trg g* **and** *src u* = *trg f*
and $\nu \in hom\ v\ (u \star f)$
shows $trnr_\varepsilon\ u\ \nu = (u \star \varepsilon) \cdot (\nu \star g)$
using *assms antipar* $trnr_\varepsilon$-*eq strict-assoc comp-ide-arr* [*of* $\mathrm{a}[u,\ f,\ g]\ \nu \star g$]
by *force*

lemma $trnl_\eta$-*eq*:
assumes *ide u* **and** *ide v*
and *src f* = *trg v* **and** *src g* = *trg u*
and $\mu \in hom\ (f \star v)\ u$
shows $trnl_\eta\ v\ \mu = (g \star \mu) \cdot (\eta \star v)$
using *assms antipar* $trnl_\eta$-*eq strict-assoc comp-ide-arr* [*of* $\mathrm{a}[g,\ f,\ v]\ \eta \star v$]
by *force*

lemma $trnl_\varepsilon$-*eq*:
assumes *ide u* **and** *ide v*
and *src f* = *trg v* **and** *src g* = *trg u*
and $\nu \in hom\ v\ (g \star u)$
shows $trnl_\varepsilon\ u\ \nu = (\varepsilon \star u) \cdot (f \star \nu)$
  using *assms antipar* $trnl_\varepsilon$-*eq strict-assoc′ comp-ide-arr* [*of* $\mathrm{a}^{-1}[f, g, u]\ f \star \nu$]
  by *fastforce*

end

1.13.2 Preservation Properties for Adjunctions

Here we show that adjunctions are preserved under isomorphisms of the left and right adjoints.

context *bicategory*
begin

interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
notation *E.eval* (‹⦃-⦄›)

definition *adjoint-pair*
where *adjoint-pair f g* ≡ ∃ η ε. *adjunction-in-bicategory V H* a i *src trg f g η ε*

abbreviation *is-left-adjoint*
where *is-left-adjoint f* ≡ ∃ *g*. *adjoint-pair f g*

abbreviation *is-right-adjoint*
where *is-right-adjoint g* ≡ ∃ *f*. *adjoint-pair f g*

lemma *adjoint-pair-antipar*:
assumes *adjoint-pair f g*
shows *ide f* **and** *ide g* **and** *src f* = *trg g* **and** *src g* = *trg f*
proof −
  obtain η ε **where** *A*: *adjunction-in-bicategory V H* a i *src trg f g η ε*
    using *assms adjoint-pair-def* **by** *auto*
  interpret *A*: *adjunction-in-bicategory V H* a i *src trg f g η ε*
    using *A* **by** *auto*
  show *ide f* **by** *simp*
  show *ide g* **by** *simp*
  show *src f* = *trg g* **using** *A.antipar* **by** *simp*
  show *src g* = *trg f* **using** *A.antipar* **by** *simp*
qed

lemma *left-adjoint-is-ide*:
assumes *is-left-adjoint f*
shows *ide f*
  using *assms adjoint-pair-antipar* **by** *auto*

lemma *right-adjoint-is-ide*:
assumes *is-right-adjoint f*
shows *ide f*
  using *assms adjoint-pair-antipar* **by** *auto*

lemma *adjunction-preserved-by-iso-right*:
assumes *adjunction-in-bicategory V H* a i *src trg f g* η ε
and «φ : $g \Rightarrow g'$» **and** *iso* φ
shows *adjunction-in-bicategory V H* a i *src trg f g'* $((\varphi \star f) \cdot \eta)$ $(\varepsilon \cdot (f \star inv\ \varphi))$
proof
  interpret *A*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
    using *assms* **by** *auto*
  show *ide f* **by** *simp*
  show *ide g'*
    using *assms*(*2*) *isomorphic-def* **by** *auto*
  show «$(\varphi \star f) \cdot \eta$: *src f* $\Rightarrow$ $g' \star f$»
    using *assms A.antipar* **by** *fastforce*
  show «$\varepsilon \cdot (f \star inv\ \varphi)$: $f \star g' \Rightarrow$ *src g'*»
    using *assms A.antipar A.counit-in-hom* **by** *auto*
  show $(\varepsilon \cdot (f \star inv\ \varphi) \star f) \cdot \mathrm{a}^{-1}[f, g', f] \cdot (f \star (\varphi \star f) \cdot \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
  proof −
    have $(\varepsilon \cdot (f \star inv\ \varphi) \star f) \cdot \mathrm{a}^{-1}[f, g', f] \cdot (f \star (\varphi \star f) \cdot \eta) =$
      $(\varepsilon \star f) \cdot (((f \star inv\ \varphi) \star f) \cdot \mathrm{a}^{-1}[f, g', f]) \cdot (f \star \varphi \star f) \cdot (f \star \eta)$
      using *assms A.antipar whisker-right whisker-left comp-assoc* **by** *auto*
    also have ... $= (\varepsilon \star f) \cdot (\mathrm{a}^{-1}[f, g, f] \cdot (f \star inv\ \varphi \star f)) \cdot (f \star \varphi \star f) \cdot (f \star \eta)$
      using *assms A.antipar assoc′-naturality* [*of f inv* φ *f*] **by** *auto*
    also have ... $= (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot ((f \star inv\ \varphi \star f) \cdot (f \star \varphi \star f)) \cdot (f \star \eta)$
      using *comp-assoc* **by** *simp*
    also have ... $= (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star g \star f) \cdot (f \star \eta)$
      using *assms A.antipar comp-inv-arr inv-is-inverse whisker-left*
        whisker-right [*of f inv* φ φ]
      by *auto*
    also have ... $= (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)$
      using *assms A.antipar comp-cod-arr* **by** *simp*
    also have ... $= \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
      using *A.triangle-left* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
  show $(g' \star \varepsilon \cdot (f \star inv\ \varphi)) \cdot \mathrm{a}[g', f, g'] \cdot ((\varphi \star f) \cdot \eta \star g') = \mathrm{r}^{-1}[g'] \cdot \mathrm{l}[g']$
  proof −
    have $(g' \star \varepsilon \cdot (f \star inv\ \varphi)) \cdot \mathrm{a}[g', f, g'] \cdot ((\varphi \star f) \cdot \eta \star g') =$
      $(g' \star \varepsilon) \cdot ((g' \star f \star inv\ \varphi) \cdot \mathrm{a}[g', f, g']) \cdot ((\varphi \star f) \star g') \cdot (\eta \star g')$
      using *assms A.antipar whisker-left whisker-right comp-assoc* **by** *auto*
    also have ... $= (g' \star \varepsilon) \cdot (\mathrm{a}[g', f, g] \cdot ((g' \star f) \star inv\ \varphi)) \cdot ((\varphi \star f) \star g') \cdot (\eta \star g')$
      using *assms A.antipar assoc-naturality* [*of g′ f inv* φ] **by** *auto*
    also have ... $= (g' \star \varepsilon) \cdot \mathrm{a}[g', f, g] \cdot (((g' \star f) \star inv\ \varphi) \cdot ((\varphi \star f) \star g')) \cdot (\eta \star g')$
      using *comp-assoc* **by** *simp*
    also have ... $= (g' \star \varepsilon) \cdot (\mathrm{a}[g', f, g] \cdot ((\varphi \star f) \star g)) \cdot ((g \star f) \star inv\ \varphi) \cdot (\eta \star g')$
    proof −

have $((g' \star f) \star inv\ \varphi) \cdot ((\varphi \star f) \star g') = (\varphi \star f) \star inv\ \varphi$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of* $g' \star f\ \varphi \star f\ inv\ \varphi\ g'$]
by *auto*
also have ... $= ((\varphi \star f) \star g) \cdot ((g \star f) \star inv\ \varphi)$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of* $\varphi \star f\ g \star f\ g\ inv\ \varphi$]
by *auto*
finally show *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((g' \star \varepsilon) \cdot (\varphi \star f \star g)) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot (trg\ g \star inv\ \varphi)$
proof −
have $\mathrm{a}[g', f, g] \cdot ((\varphi \star f) \star g) = (\varphi \star f \star g) \cdot \mathrm{a}[g, f, g]$
using *assms A.antipar assoc-naturality* [*of* $\varphi\ f\ g$] **by** *auto*
moreover have $((g \star f) \star inv\ \varphi) \cdot (\eta \star g') = (\eta \star g) \cdot (trg\ g \star inv\ \varphi)$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of* $g \star f\ \eta\ inv\ \varphi\ g'$] *interchange* [*of* $\eta\ trg\ g\ g\ inv\ \varphi$]
by *auto*
ultimately show *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((\varphi \star src\ g) \cdot (g \star \varepsilon)) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot (trg\ g \star inv\ \varphi)$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of* $g'\ \varphi\ \varepsilon\ f \star g$] *interchange* [*of* $\varphi\ g\ src\ g\ \varepsilon$]
by (*metis A.counit-simps*(*1*) *A.counit-simps*(*2*) *A.counit-simps*(*3*) *in-homE*)
also have ... $= (\varphi \star src\ g) \cdot ((g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)) \cdot (trg\ g \star inv\ \varphi)$
using *comp-assoc* **by** *simp*
also have ... $= ((\varphi \star src\ g) \cdot \mathrm{r}^{-1}[g]) \cdot \mathrm{l}[g] \cdot (trg\ g \star inv\ \varphi)$
using *assms A.antipar A.triangle-right comp-cod-arr comp-assoc*
by *simp*
also have ... $= (\mathrm{r}^{-1}[g'] \cdot \varphi) \cdot inv\ \varphi \cdot \mathrm{l}[g']$
using *assms A.antipar runit′-naturality* [*of* φ] *lunit-naturality* [*of* $inv\ \varphi$]
by *auto*
also have ... $= \mathrm{r}^{-1}[g'] \cdot (\varphi \cdot inv\ \varphi) \cdot \mathrm{l}[g']$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{r}^{-1}[g'] \cdot \mathrm{l}[g']$
using *assms comp-cod-arr comp-arr-inv′* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed

lemma *adjunction-preserved-by-iso-left*:
assumes *adjunction-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
and «$\varphi : f \Rightarrow f'$» **and** *iso* φ
shows *adjunction-in-bicategory V H* a i *src trg* $f'\ g\ ((g \star \varphi) \cdot \eta)\ (\varepsilon \cdot (inv\ \varphi \star g))$
proof
interpret *A*: *adjunction-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
using *assms* **by** *auto*

show *ide g* **by** *simp*
show *ide f′*
  using *assms(2) isomorphic-def* **by** *auto*
show «$(g \star \varphi) \cdot \eta$: *src f′* ⇒ $g \star f'$»
  using *assms A.antipar A.unit-in-hom* **by** *force*
show «$\varepsilon \cdot (inv\ \varphi \star g)$: $f' \star g$ ⇒ *src g*»
  using *assms A.antipar* **by** *force*
show $(g \star \varepsilon \cdot (inv\ \varphi \star g)) \cdot \mathrm{a}[g, f', g] \cdot ((g \star \varphi) \cdot \eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
proof −
  have $(g \star \varepsilon \cdot (inv\ \varphi \star g)) \cdot \mathrm{a}[g, f', g] \cdot ((g \star \varphi) \cdot \eta \star g) =$
      $(g \star \varepsilon) \cdot ((g \star inv\ \varphi \star g) \cdot \mathrm{a}[g, f', g]) \cdot ((g \star \varphi) \star g) \cdot (\eta \star g)$
    using *assms A.antipar whisker-left whisker-right comp-assoc* **by** *auto*
  also have ... $= (g \star \varepsilon) \cdot (\mathrm{a}[g, f, g] \cdot ((g \star inv\ \varphi) \star g)) \cdot ((g \star \varphi) \star g) \cdot (\eta \star g)$
    using *assms A.antipar assoc-naturality* [*of g inv φ g*] **by** *auto*
  also have ... $= (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (((g \star inv\ \varphi) \star g) \cdot ((g \star \varphi) \star g)) \cdot (\eta \star g)$
    using *comp-assoc* **by** *simp*
  also have ... $= (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot ((g \star f) \star g) \cdot (\eta \star g)$
    using *assms A.antipar comp-inv-arr inv-is-inverse whisker-right*
      whisker-left [*of g inv φ φ*]
    by *auto*
  also have ... $= (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)$
    using *assms A.antipar comp-cod-arr* **by** *simp*
  also have ... $= \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
    using *A.triangle-right* **by** *simp*
  finally show *?thesis* **by** *simp*
qed
show $(\varepsilon \cdot (inv\ \varphi \star g) \star f') \cdot \mathrm{a}^{-1}[f', g, f'] \cdot (f' \star (g \star \varphi) \cdot \eta) = \mathrm{l}^{-1}[f'] \cdot \mathrm{r}[f']$
proof −
  have $(\varepsilon \cdot (inv\ \varphi \star g) \star f') \cdot \mathrm{a}^{-1}[f', g, f'] \cdot (f' \star (g \star \varphi) \cdot \eta) =$
      $(\varepsilon \star f') \cdot (((inv\ \varphi \star g) \star f') \cdot \mathrm{a}^{-1}[f', g, f']) \cdot (f' \star g \star \varphi) \cdot (f' \star \eta)$
    using *assms A.antipar whisker-right whisker-left comp-assoc*
    by *auto*
  also have ... $= (\varepsilon \star f') \cdot (\mathrm{a}^{-1}[f, g, f'] \cdot (inv\ \varphi \star g \star f')) \cdot (f' \star g \star \varphi) \cdot (f' \star \eta)$
    using *assms A.antipar assoc′-naturality* [*of inv φ g f′*] **by** *auto*
  also have ... $= (\varepsilon \star f') \cdot \mathrm{a}^{-1}[f, g, f'] \cdot ((inv\ \varphi \star g \star f') \cdot (f' \star g \star \varphi)) \cdot (f' \star \eta)$
    using *comp-assoc* **by** *simp*
  also have ... $= (\varepsilon \star f') \cdot (\mathrm{a}^{-1}[f, g, f'] \cdot (f \star g \star \varphi)) \cdot (inv\ \varphi \star g \star f) \cdot (f' \star \eta)$
  proof −
    have $(inv\ \varphi \star g \star f') \cdot (f' \star g \star \varphi) = (f \star g \star \varphi) \cdot (inv\ \varphi \star g \star f)$
      using *assms(2−3) A.antipar comp-arr-dom comp-cod-arr*
        interchange [*of inv φ f′ g ⋆ f′ g ⋆ φ*]
        interchange [*of f inv φ g ⋆ φ g ⋆ f*]
      by *auto*
    thus *?thesis*
      using *comp-assoc* **by** *simp*
  qed
  also have ... $= ((\varepsilon \star f') \cdot ((f \star g) \star \varphi)) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot (inv\ \varphi \star src\ f)$
  proof −
    have $\mathrm{a}^{-1}[f, g, f'] \cdot (f \star g \star \varphi) = ((f \star g) \star \varphi) \cdot \mathrm{a}^{-1}[f, g, f]$

using *assms A.antipar assoc′-naturality* [*of f g φ*] **by** *auto*
moreover have $(inv\ \varphi \star g \star f) \cdot (f' \star \eta) = (f \star \eta) \cdot (inv\ \varphi \star src\ f)$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of inv φ f′ g ⋆ f η*] *interchange* [*of f inv φ η src f*]
by *auto*
ultimately show *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((trg\ f \star \varphi) \cdot (\varepsilon \star f)) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \cdot (inv\ \varphi \star src\ f)$
using *assms A.antipar comp-arr-dom comp-cod-arr*
interchange [*of ε f ⋆ g f′ φ*] *interchange* [*of trg f ε φ f*]
by (*metis A.counit-simps*(*1*) *A.counit-simps*(*2*) *A.counit-simps*(*3*) *in-homE*)
also have ... $= (trg\ f \star \varphi) \cdot ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) \cdot (inv\ \varphi \star src\ f)$
using *comp-assoc* **by** *simp*
also have ... $= ((trg\ f \star \varphi) \cdot \mathrm{l}^{-1}[f]) \cdot \mathrm{r}[f] \cdot (inv\ \varphi \star src\ f)$
using *assms A.antipar A.triangle-left comp-cod-arr comp-assoc*
by *simp*
also have ... $= (\mathrm{l}^{-1}[f'] \cdot \varphi) \cdot inv\ \varphi \cdot \mathrm{r}[f']$
using *assms A.antipar lunit′-naturality runit-naturality* [*of inv φ*] **by** *auto*
also have ... $= \mathrm{l}^{-1}[f'] \cdot (\varphi \cdot inv\ \varphi) \cdot \mathrm{r}[f']$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{l}^{-1}[f'] \cdot \mathrm{r}[f']$
using *assms comp-cod-arr comp-arr-inv inv-is-inverse* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed

lemma *adjoint-pair-preserved-by-iso*:
assumes *adjoint-pair f g*
and «$\varphi : f \Rightarrow f'$» **and** *iso φ*
and «$\psi : g \Rightarrow g'$» **and** *iso ψ*
shows *adjoint-pair f′ g′*
using *assms adjoint-pair-def adjunction-preserved-by-iso-left*
adjunction-preserved-by-iso-right
by *metis*

lemma *left-adjoint-preserved-by-iso*:
assumes *is-left-adjoint f*
and «$\varphi : f \Rightarrow f'$» **and** *iso φ*
shows *is-left-adjoint f′*
proof −
obtain *g* **where** *g*: *adjoint-pair f g*
using *assms* **by** *auto*
have *adjoint-pair f′ g*
using *assms g adjoint-pair-preserved-by-iso* [*of f g φ f′ g g*]
adjoint-pair-antipar [*of f g*]
by *auto*
thus *?thesis* **by** *auto*
qed

lemma *right-adjoint-preserved-by-iso*:
assumes *is-right-adjoint g*
and «φ : $g \Rightarrow g'$» **and** *iso* φ
shows *is-right-adjoint* g'
proof −
  obtain f **where** f: *adjoint-pair f g*
    using *assms* **by** *auto*
  have *adjoint-pair f* g'
    using *assms f adjoint-pair-preserved-by-iso* [*of f g f f* φ g']
      adjoint-pair-antipar [*of f g*]
    by *auto*
  thus *?thesis* **by** *auto*
qed

lemma *left-adjoint-preserved-by-iso'*:
assumes *is-left-adjoint f* **and** $f \cong f'$
shows *is-left-adjoint* f'
  using *assms isomorphic-def left-adjoint-preserved-by-iso* **by** *blast*

lemma *right-adjoint-preserved-by-iso'*:
assumes *is-right-adjoint g* **and** $g \cong g'$
shows *is-right-adjoint* g'
  using *assms isomorphic-def right-adjoint-preserved-by-iso* **by** *blast*

lemma *obj-self-adjunction*:
assumes *obj a*
shows *adjunction-in-bicategory V H* a i *src trg a a* $\mathrm{l}^{-1}[a]$ $\mathrm{r}[a]$
proof
  show *1*: *ide a*
    using *assms* **by** *auto*
  show «$\mathrm{l}^{-1}[a]$: *src a* $\Rightarrow$ $a \star a$»
    using *assms 1* **by** *auto*
  show «$\mathrm{r}[a]$: $a \star a \Rightarrow$ *src a*»
    using *assms 1* **by** *fastforce*
  show $(\mathrm{r}[a] \star a) \cdot \mathrm{a}^{-1}[a, a, a] \cdot (a \star \mathrm{l}^{-1}[a]) = \mathrm{l}^{-1}[a] \cdot \mathrm{r}[a]$
    using *assms 1 canI-unitor-1 canI-associator-1*(*2*) *canI-associator-3*
      whisker-can-right-1 whisker-can-left-1 can-Ide-self obj-simps
    by *simp*
  show $(a \star \mathrm{r}[a]) \cdot \mathrm{a}[a, a, a] \cdot (\mathrm{l}^{-1}[a] \star a) = \mathrm{r}^{-1}[a] \cdot \mathrm{l}[a]$
    using *assms 1 canI-unitor-1 canI-associator-1*(*2*) *canI-associator-3*
      whisker-can-right-1 whisker-can-left-1 can-Ide-self
    by *simp*
qed

lemma *obj-is-self-adjoint*:
assumes *obj a*
shows *adjoint-pair a a* **and** *is-left-adjoint a* **and** *is-right-adjoint a*
  using *assms obj-self-adjunction adjoint-pair-def* **by** *auto*

end

1.13.3 Pseudofunctors and Adjunctions

context *pseudofunctor*
begin

lemma *preserves-adjunction*:
assumes *adjunction-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
shows *adjunction-in-bicategory* V_D H_D a_D i_D src_D trg_D $(F\ f)$ $(F\ g)$
$(D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g))$
proof −
interpret *adjunction-in-bicategory* V_C H_C a_C i_C src_C trg_C f g η ε
using *assms* **by** *auto*
interpret *A*: *adjunction-data-in-bicategory* V_D H_D a_D i_D src_D trg_D
‹$F\ f$› ‹$F\ g$› ‹$D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)$›
‹$D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g)$›
using *adjunction-data-in-bicategory-axioms preserves-adjunction-data* **by** *auto*
show *adjunction-in-bicategory* V_D H_D a_D i_D src_D trg_D $(F\ f)$ $(F\ g)$
$(D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g))$
proof
show $(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)) =$
$D.lunit'\ (F\ f) \cdot_D \mathrm{r}_D[F\ f]$
proof −
have *1*: $D.iso\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)))$
using *antipar* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *D.iso-is-arr FF-def*
by (*intro D.isos-compose D.seqI*, *simp-all*)
have $(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)) =$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g) \star_D F\ f) \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f)) \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f] \cdot_D$
$\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)) \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
proof −
have $\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] =$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f)) \cdot_D F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f] \cdot_D$
$\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f))$
proof −
have $\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D D.inv\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f))) =$
$D.inv\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f))$
proof −
have $D.inv\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ f]) =$
$D.inv\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f))$
using *antipar assoc-coherence* **by** *simp*

moreover
have $D.inv\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)) \cdot_D \mathrm{a}_D[F\ f,\ F\ g,\ F\ f]) =$
$\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D D.inv\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)))$
proof −
have $D.seq\ (\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)))\ \mathrm{a}_D[F\ f,\ F\ g,\ F\ f]$
using *antipar* **by** *fastforce*
thus *?thesis*
using *1 antipar D.comp-assoc*
D.inv-comp [*of* $\mathrm{a}_D[F\ f,\ F\ g,\ F\ f]\ \Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f))$]
by *auto*
qed
ultimately show *?thesis* **by** *simp*
qed
moreover have *2*: $D.iso\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f))$
using *antipar D.isos-compose* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *cmp-simps*(*4*)
by *simp*
ultimately have $\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] =$
$D.inv\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f)) \cdot_D$
$(\Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D \Phi\ (g,\ f)))$
using *1 2 antipar D.invert-side-of-triangle*(*2*) *D.inv-inv D.iso-inv-iso D.arr-inv*
by *metis*
moreover have $D.inv\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f)) =$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f)) \cdot_D F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
proof −
have $D.inv\ (F\ \mathrm{a}_C[f,\ g,\ f] \cdot_D \Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f)) =$
$D.inv\ (\Phi\ (f \star_C g,\ f) \cdot_D (\Phi\ (f,\ g) \star_D F\ f)) \cdot_D F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
using *antipar D.isos-compose* $C.VV.arr\text{-}char_{SbC}$ *cmp-simps*(*4*)
preserves-inv D.inv-comp $C.VV.cod\text{-}char_{SbC}$
by *simp*
also have ... $= (D.inv\ (\Phi\ (f,\ g) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f))) \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
using *antipar D.inv-comp* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *cmp-simps*(*4*)
by *simp*
also have ... $= ((D.inv\ (\Phi\ (f,\ g)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f))) \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
using *antipar* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ **by** *simp*
also have ... $= (D.inv\ (\Phi\ (f,\ g)) \star_D F\ f) \cdot_D D.inv\ (\Phi\ (f \star_C g,\ f)) \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
using *D.comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \star_D F\ f) \cdot_D$
$D.inv\ (\Phi\ (f \star_C g,\ f)) \cdot_D$

$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D$
$\Phi\ (f,\ g \star_C f)\ \cdot_D$
$(F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f))$

proof −
have ... = $((D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon \star_D F\ f)\ \cdot_D\ (\Phi\ (f,\ g) \star_D F\ f))\ \cdot_D$
$(D.inv\ (\Phi\ (f,\ g)) \star_D F\ f)\ \cdot_D\ D.inv\ (\Phi\ (f \star_C g,\ f))\ \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D$
$\Phi\ (f,\ g \star_C f)\ \cdot_D\ (F\ f \star_D \Phi\ (g,\ f))\ \cdot_D$
$((F\ f \star_D D.inv\ (\Phi\ (g,\ f)))\ \cdot_D\ (F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f)))$
proof −
have $D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon\ \cdot_D\ \Phi\ (f,\ g) \star_D F\ f =$
$(D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon \star_D F\ f)\ \cdot_D\ (\Phi\ (f,\ g) \star_D F\ f)$
using *ide-left ide-right antipar D.whisker-right unit-char*(*2*)
by (*metis A.counit-simps*(*1*) *A.ide-left D.comp-assoc*)
moreover have $F\ f \star_D D.inv\ (\Phi\ (g,\ f))\ \cdot_D\ F\ \eta\ \cdot_D\ unit\ (src_C\ f) =$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ f)))\ \cdot_D\ (F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f))$
using *antipar unit-char*(*2*) *D.whisker-left* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also have ... = $(D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon \star_D F\ f)\ \cdot_D$
$(((\Phi\ (f,\ g) \star_D F\ f)\ \cdot_D\ (D.inv\ (\Phi\ (f,\ g)) \star_D F\ f))\ \cdot_D$
$D.inv\ (\Phi\ (f \star_C g,\ f)))\ \cdot_D\ F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D\ \Phi\ (f,\ g \star_C f)\ \cdot_D$
$(((F\ f \star_D \Phi\ (g,\ f))\ \cdot_D\ (F\ f \star_D D.inv\ (\Phi\ (g,\ f))))\ \cdot_D$
$(F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f)))$
using *D.comp-assoc* **by** *simp*
also have ... = $(D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon \star_D F\ f)\ \cdot_D$
$D.inv\ (\Phi\ (f \star_C g,\ f))\ \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D$
$\Phi\ (f,\ g \star_C f)\ \cdot_D$
$(F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f))$
proof −
have $((F\ f \star_D \Phi\ (g,\ f))\ \cdot_D\ (F\ f \star_D D.inv\ (\Phi\ (g,\ f))))\ \cdot_D$
$(F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f)) =$
$F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f)$
proof −
have $(F\ f \star_D \Phi\ (g,\ f))\ \cdot_D\ (F\ f \star_D D.inv\ (\Phi\ (g,\ f))) = F\ f \star_D F\ (g \star_C f)$
using *antipar unit-char*(*2*) *D.comp-arr-inv D.inv-is-inverse*
D.whisker-left [*of* $F\ f\ \Phi\ (g,\ f)\ D.inv\ (\Phi\ (g,\ f))$]
by *simp*
moreover have $D.seq\ (F\ f \star_D F\ (g \star_C f))\ (F\ f \star_D F\ \eta\ \cdot_D\ unit\ (src_C\ f))$
using *antipar* **by** *fastforce*
ultimately show *?thesis*
using *D.comp-cod-arr* **by** *auto*
qed
moreover have $((\Phi\ (f,\ g) \star_D F\ f)\ \cdot_D\ (D.inv\ (\Phi\ (f,\ g)) \star_D F\ f))\ \cdot_D$
$D.inv\ (\Phi\ (f \star_C g,\ f)) =$
$D.inv\ (\Phi\ (f \star_C g,\ f))$
using *antipar D.comp-arr-inv D.inv-is-inverse D.comp-cod-arr*
D.whisker-right [*of* $F\ f\ \Phi\ (f,\ g)\ D.inv\ (\Phi\ (f,\ g))$]

by *simp*
ultimately show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
also have ... = $(D.inv\ (unit\ (trg_C\ f))\ \star_D\ F\ f)\ \cdot_D$
$D.inv\ (\Phi\ (trg_C\ f,\ f))\ \cdot_D\ F\ (\varepsilon\ \star_C\ f)\ \cdot_D$
$((\Phi\ (f\ \star_C\ g,\ f)\ \cdot_D\ D.inv\ (\Phi\ (f\ \star_C\ g,\ f)))\ \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f])\ \cdot_D$
$((\Phi\ (f,\ g\ \star_C\ f)\ \cdot_D\ D.inv\ (\Phi\ (f,\ g\ \star_C\ f)))\ \cdot_D\ F\ (f\ \star_C\ \eta))\ \cdot_D$
$\Phi\ (f,\ src_C\ f)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f))$
proof −
have $(D.inv\ (unit\ (trg_C\ f))\ \cdot_D\ F\ \varepsilon\ \star_D\ F\ f)\ \cdot_D$
$D.inv\ (\Phi\ (f\ \star_C\ g,\ f))\ \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D$
$\Phi\ (f,\ g\ \star_C\ f)\ \cdot_D$
$(F\ f\ \star_D\ F\ \eta\ \cdot_D\ unit\ (src_C\ f)) =$
$((D.inv\ (unit\ (trg_C\ f))\ \star_D\ F\ f)\ \cdot_D\ (F\ \varepsilon\ \star_D\ F\ f))\ \cdot_D$
$D.inv\ (\Phi\ (f\ \star_C\ g,\ f))\ \cdot_D$
$F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D$
$\Phi\ (f,\ g\ \star_C\ f)\ \cdot_D$
$((F\ f\ \star_D\ F\ \eta)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f)))$
using *antipar D.comp-assoc D.whisker-left D.whisker-right unit-char*(*2*)
by *simp*
moreover have $F\ \varepsilon\ \star_D\ F\ f = D.inv\ (\Phi\ (trg_C\ f,\ f))\ \cdot_D\ F\ (\varepsilon\ \star_C\ f)\ \cdot_D\ \Phi\ (f\ \star_C\ g,\ f)$
using *antipar* Φ*.naturality* [*of* $(\varepsilon,\ f)$] $C.VV.arr\text{-}char_{SbC}$ *FF-def*
$D.invert\text{-}side\text{-}of\text{-}triangle(1)$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
moreover have $F\ f\ \star_D\ F\ \eta = D.inv\ (\Phi\ (f,\ g\ \star_C\ f))\ \cdot_D\ F\ (f\ \star_C\ \eta)\ \cdot_D\ \Phi\ (f,\ src_C\ f)$
using *antipar* Φ*.naturality* [*of* $(f,\ \eta)$] $C.VV.arr\text{-}char_{SbC}$ *FF-def*
$D.invert\text{-}side\text{-}of\text{-}triangle(1)$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $((D.inv\ (unit\ (trg_C\ f))\ \star_D\ F\ f)\ \cdot_D\ D.inv\ (\Phi\ (trg_C\ f,\ f)))\ \cdot_D$
$(F\ (\varepsilon\ \star_C\ f)\ \cdot_D$
$(F\ ((f\ \star_C\ g)\ \star_C\ f)\ \cdot_D\ F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D\ F\ (f\ \star_C\ g\ \star_C\ f))\ \cdot_D$
$F\ (f\ \star_C\ \eta))\ \cdot_D$
$\Phi\ (f,\ src_C\ f)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f))$
using *antipar D.comp-arr-inv′ D.comp-assoc* **by** *simp*
also have ... = $((D.inv\ (unit\ (trg_C\ f))\ \star_D\ F\ f)\ \cdot_D\ D.inv\ (\Phi\ (trg_C\ f,\ f)))\ \cdot_D$
$(F\ (\varepsilon\ \star_C\ f)\ \cdot_D\ F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D\ F\ (f\ \star_C\ \eta))\ \cdot_D$
$\Phi\ (f,\ src_C\ f)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f))$
proof −
have $F\ ((f\ \star_C\ g)\ \star_C\ f)\ \cdot_D\ F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D\ F\ (f\ \star_C\ g\ \star_C\ f) = F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]$
using *antipar D.comp-arr-dom D.comp-cod-arr* **by** *simp*
thus *?thesis* **by** *simp*
qed

also have ... = $D.inv\ (\Phi\ (trg_C\ f,\ f)\ \cdot_D\ (unit\ (trg_C\ f)\ \star_D\ F\ f))\ \cdot_D$
$F\ ((\varepsilon\ \star_C\ f)\ \cdot_C\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_C\ (f\ \star_C\ \eta))\ \cdot_D$
$\Phi\ (f,\ src_C\ f)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f))$
proof −
have $(D.inv\ (unit\ (trg_C\ f))\ \star_D\ F\ f)\ \cdot_D\ D.inv\ (\Phi\ (trg_C\ f,\ f)) =$
$D.inv\ (\Phi\ (trg_C\ f,\ f)\ \cdot_D\ (unit\ (trg_C\ f)\ \star_D\ F\ f))$
proof −
have $D.iso\ (\Phi\ (trg_C\ f,\ f))$
using *antipar* **by** *simp*
moreover have $D.iso\ (unit\ (trg_C\ f)\ \star_D\ F\ f)$
using *antipar unit-char*(*2*) **by** *simp*
moreover have $D.seq\ (\Phi\ (trg_C\ f,\ f))\ (unit\ (trg_C\ f)\ \star_D\ F\ f)$
using *antipar D.iso-is-arr calculation*(*2*)
apply (*intro D.seqI D.hseqI*) **by** *auto*
ultimately show *?thesis*
using *antipar D.inv-comp unit-char*(*2*) **by** *simp*
qed
moreover have $F\ (\varepsilon\ \star_C\ f)\ \cdot_D\ F\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_D\ F\ (f\ \star_C\ \eta) =$
$F\ ((\varepsilon\ \star_C\ f)\ \cdot_C\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_C\ (f\ \star_C\ \eta))$
using *antipar* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also have ... = $(D.lunit'\ (F\ f)\ \cdot_D\ F\ \mathrm{l}_C[f])\ \cdot_D$
$F\ (C.lunit'\ f\ \cdot_C\ \mathrm{r}_C[f])\ \cdot_D$
$(D.inv\ (F\ \mathrm{r}_C[f])\ \cdot_D\ \mathrm{r}_D[F\ f])$
proof −
have $F\ ((\varepsilon\ \star_C\ f)\ \cdot_C\ \mathrm{a}_C{}^{-1}[f,\ g,\ f]\ \cdot_C\ (f\ \star_C\ \eta)) = F\ (C.lunit'\ f\ \cdot_C\ \mathrm{r}_C[f])$
using *triangle-left* **by** *simp*
moreover have $D.inv\ (\Phi\ (trg_C\ f,\ f)\ \cdot_D\ (unit\ (trg_C\ f)\ \star_D\ F\ f)) =$
$D.lunit'\ (F\ f)\ \cdot_D\ F\ \mathrm{l}_C[f]$
proof −
have *0*: $D.iso\ (\Phi\ (trg_C\ f,\ f)\ \cdot_D\ (unit\ (trg_C\ f)\ \star_D\ F\ f))$
using *unit-char*(*2*)
apply (*intro D.isos-compose D.seqI*) **by** *auto*
show *?thesis*
proof −
have *1*: $D.iso\ (F\ \mathrm{l}_C[f])$
using *C.iso-lunit preserves-iso* **by** *auto*
moreover have $D.iso\ (F\ \mathrm{l}_C[f]\ \cdot_D\ \Phi\ (trg_C\ f,\ f)\ \cdot_D\ (unit\ (trg_C\ f)\ \star_D\ F\ f))$
by (*metis* (*no-types*) *A.ide-left D.iso-lunit ide-left lunit-coherence*)
moreover have $D.inv\ (D.inv\ (F\ \mathrm{l}_C[f])) = F\ \mathrm{l}_C[f]$
using *1 D.inv-inv* **by** *blast*
ultimately show *?thesis*
by (*metis 0 D.inv-comp D.invert-side-of-triangle*(*2*) *D.iso-inv-iso*
D.iso-is-arr ide-left lunit-coherence)
qed
qed
moreover have $\Phi\ (f,\ src_C\ f)\ \cdot_D\ (F\ f\ \star_D\ unit\ (src_C\ f)) = D.inv\ (F\ \mathrm{r}_C[f])\ \cdot_D\ \mathrm{r}_D[F\ f]$
using *ide-left runit-coherence preserves-iso C.iso-runit D.invert-side-of-triangle*(*1*)

by (*metis A.ide-left D.runit-simps*(*1*))
ultimately show *?thesis* **by** *simp*
qed
also have ... = $D.lunit'\ (F\ f) \cdot_D$
$((F\ \mathrm{l}_C[f] \cdot_D F\ (C.lunit'\ f)) \cdot_D (F\ \mathrm{r}_C[f] \cdot_D D.inv\ (F\ \mathrm{r}_C[f]))) \cdot_D$
$\mathrm{r}_D[F\ f]$
using *D.comp-assoc* **by** *simp*
also have ... = $D.lunit'\ (F\ f) \cdot_D \mathrm{r}_D[F\ f]$
using *D.comp-cod-arr C.iso-runit C.iso-lunit preserves-iso D.comp-arr-inv′ preserves-inv*
by *force*
finally show *?thesis* **by** *blast*
qed
show $(F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ F\ g] \cdot_D (D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g) =$
$D.runit'\ (F\ g) \cdot_D \mathrm{l}_D[F\ g]$
proof −
have $\mathrm{a}_D[F\ g,\ F\ f,\ F\ g] =$
$D.inv\ (\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D \Phi\ (g \star_C f,\ g) \cdot_D (\Phi\ (g,\ f) \star_D F\ g)$
proof −
have $D.iso\ (\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g)))$
using *antipar D.iso-is-arr*
apply (*intro D.isos-compose*, *auto*)
by (*metis C.iso-assoc D.comp-assoc D.seqE ide-left ide-right preserves-assoc*(*1*) *preserves-iso*)
moreover have $F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D \Phi\ (g \star_C f,\ g) \cdot_D (\Phi\ (g,\ f) \star_D F\ g) =$
$\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g)) \cdot_D \mathrm{a}_D[F\ g,\ F\ f,\ F\ g]$
using *antipar assoc-coherence* **by** *simp*
moreover have $D.seq\ (F\ \mathrm{a}_C[g,\ f,\ g])\ (\Phi\ (g \star_C f,\ g) \cdot_D (\Phi\ (g,\ f) \star_D F\ g))$
using *antipar* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def*
by (*intro D.seqI D.hseqI′*) *auto*
ultimately show *?thesis*
using *D.invert-side-of-triangle*(*1*) *D.comp-assoc* **by** *auto*
qed
hence $(F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ F\ g] \cdot_D$
$(D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g) =$
$(F\ g \star_D (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D \Phi\ (f,\ g)) \cdot_D$
$D.inv\ (\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D$
$\Phi\ (g \star_C f,\ g) \cdot_D (\Phi\ (g,\ f) \star_D F\ g) \cdot_D$
$(D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g)$
using *D.comp-assoc* **by** *simp*
also have ... = $((F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$D.inv\ (\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D \Phi\ (g \star_C f,\ g) \cdot_D$
$(\Phi\ (g,\ f) \star_D F\ g) \cdot_D ((D.inv\ (\Phi\ (g,\ f)) \star_D F\ g) \cdot_D$
$(F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g))$

proof −
have $F\ g \star_D (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D \Phi\ (f,\ g) =$
$(F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D (F\ g \star_D \Phi\ (f,\ g))$
proof −
have $D.seq\ (D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon)\ (\Phi\ (f,\ g))$
using *antipar D.comp-assoc* **by** *simp*
thus *?thesis*
using *antipar D.whisker-left* **by** *simp*
qed
moreover have $D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g =$
$(D.inv\ (\Phi\ (g,\ f)) \star_D F\ g) \cdot_D (F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g)$
using *antipar D.whisker-right* **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D$
$(((F\ g \star_D \Phi\ (f,\ g)) \cdot_D D.inv\ (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$D.inv\ (\Phi\ (g,\ f \star_C g))) \cdot_D F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D \Phi\ (g \star_C f,\ g) \cdot_D$
$((\Phi\ (g,\ f) \star_D F\ g) \cdot_D (D.inv\ (\Phi\ (g,\ f)) \star_D F\ g)) \cdot_D$
$(F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g)$
proof −
have $D.inv\ (\Phi\ (g,\ f \star_C g) \cdot_D (F\ g \star_D \Phi\ (f,\ g))) =$
$D.inv\ (F\ g \star_D \Phi\ (f,\ g)) \cdot_D D.inv\ (\Phi\ (g,\ f \star_C g))$
proof −
have $D.iso\ (\Phi\ (g,\ f \star_C g))$
using *antipar* **by** *simp*
moreover have $D.iso\ (F\ g \star_D \Phi\ (f,\ g))$
using *antipar* **by** *simp*
moreover have $D.seq\ (\Phi\ (g,\ f \star_C g))\ (F\ g \star_D \Phi\ (f,\ g))$
using *antipar cmp-in-hom A.ide-right D.iso-is-arr*
by (*intro D.seqI*) *auto*
ultimately show *?thesis*
using *antipar D.inv-comp* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ g \star_D D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon) \cdot_D$
$D.inv\ (\Phi\ (g,\ f \star_C g)) \cdot_D F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D \Phi\ (g \star_C f,\ g) \cdot_D$
$(F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g)$
proof −
have $((\Phi\ (g,\ f) \star_D F\ g) \cdot_D (D.inv\ (\Phi\ (g,\ f)) \star_D F\ g)) \cdot_D$
$(F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g) =$
$(F\ \eta \cdot_D unit\ (src_C\ f) \star_D F\ g)$
proof −
have $(\Phi\ (g,\ f) \star_D F\ g) \cdot_D (D.inv\ (\Phi\ (g,\ f)) \star_D F\ g) = F\ (g \star_C f) \star_D F\ g$
using *antipar D.comp-arr-inv′*
D.whisker-right [*of* $F\ g\ \Phi\ (g,\ f)\ D.inv\ (\Phi\ (g,\ f))$]
by *simp*

thus *?thesis*
using *antipar D.comp-cod-arr D.whisker-right* **by** *simp*
qed
moreover have $((F\ g \star_D \Phi\ (f,\ g)) \cdot_D D.inv\ (F\ g \star_D \Phi\ (f,\ g))) \cdot_D$
$D.inv\ (\Phi\ (g,\ f \star_C g)) =$
$D.inv\ (\Phi\ (g,\ f \star_C g))$
using *antipar D.comp-arr-inv′ D.comp-cod-arr*
D.whisker-left [*of* $F\ g\ \Phi\ (f,\ g)\ D.inv\ (\Phi\ (f,\ g))$]
by *simp*
ultimately show *?thesis* **by** *simp*
qed
also have ... $= (F\ g \star_D D.inv\ (unit\ (trg_C\ f))) \cdot_D$
$((F\ g \star_D F\ \varepsilon) \cdot_D D.inv\ (\Phi\ (g,\ f \star_C g))) \cdot_D$
$F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D$
$(\Phi\ (g \star_C f,\ g) \cdot_D (F\ \eta \star_D F\ g)) \cdot_D$
$(unit\ (src_C\ f) \star_D F\ g)$
using *antipar D.whisker-left D.whisker-right unit-char*(*2*) *D.comp-assoc* **by** *simp*
also have ... $= (F\ g \star_D D.inv\ (unit\ (trg_C\ f))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) \cdot_D$
$(F\ (g \star_C \varepsilon) \cdot_D F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D F\ (\eta \star_C g)) \cdot_D$
$\Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (src_C\ f) \star_D F\ g)$
proof −
have $(F\ g \star_D F\ \varepsilon) \cdot_D D.inv\ (\Phi\ (g,\ f \star_C g)) = D.inv\ (\Phi\ (g,\ src_C\ g)) \cdot_D F\ (g \star_C \varepsilon)$
using *antipar* $C.VV.arr\text{-}char_{SbC}$ $\Phi.naturality$ [*of* $(g,\ \varepsilon)$] *FF-def*
D.invert-opposite-sides-of-square $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
moreover have $\Phi\ (g \star_C f,\ g) \cdot_D (F\ \eta \star_D F\ g) = F\ (\eta \star_C g) \cdot_D \Phi\ (trg_C\ g,\ g)$
using *antipar* $C.VV.arr\text{-}char_{SbC}$ $\Phi.naturality$ [*of* $(\eta,\ g)$] *FF-def*
$C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((F\ g \star_D D.inv\ (unit\ (trg_C\ f))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) \cdot_D$
$F\ (C.runit'\ g)) \cdot_D (F\ \mathrm{l}_C[g] \cdot_D \Phi\ (trg_C\ g,\ g) \cdot_D (unit\ (src_C\ f) \star_D F\ g))$
proof −
have $F\ (g \star_C \varepsilon) \cdot_D F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D F\ (\eta \star_C g) = F\ (C.runit'\ g) \cdot_D F\ \mathrm{l}_C[g]$
using *ide-left ide-right antipar triangle-right*
by (*metis C.comp-in-homE C.seqI′ preserves-comp triangle-in-hom*(*2*))
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= D.runit'\ (F\ g) \cdot_D \mathrm{l}_D[F\ g]$
proof −
have $D.inv\ \mathrm{r}_D[F\ g] =$
$(F\ g \star_D D.inv\ (unit\ (trg_C\ f))) \cdot_D D.inv\ (\Phi\ (g,\ src_C\ g)) \cdot_D F\ (C.runit'\ g)$
proof −
have $D.runit'\ (F\ g) = D.inv\ (F\ \mathrm{r}_C[g] \cdot_D \Phi\ (g,\ src_C\ g) \cdot_D (F\ g \star_D unit\ (src_C\ g)))$
using *runit-coherence* **by** *simp*
also have

```
        ... = (F g ⋆D D.inv (unit (trgC f))) ·D D.inv (Φ (g, srcC g)) ·D F (C.runit' g)
      proof −
        have D.inv (F rC[g] ·D Φ (g, srcC g) ·D (F g ⋆D unit (srcC g))) =
              D.inv (F g ⋆D unit (srcC g)) ·D D.inv (Φ (g, srcC g)) ·D F (C.runit' g)
        proof −
          have D.iso (F rC[g])
            using preserves-iso by simp
          moreover have 1: D.iso (Φ (g, srcC g) ·D (F g ⋆D unit (srcC g)))
            using preserves-iso unit-char(2) D.arrI D.seqE ide-right runit-coherence
            by (intro D.isos-compose D.seqI, auto)
          moreover have D.seq (F rC[g]) (Φ (g, srcC g) ·D (F g ⋆D unit (srcC g)))
            using ide-right A.ide-right D.runit-simps(1) runit-coherence by metis
          ultimately have D.inv (F rC[g] ·D Φ (g, srcC g) ·D (F g ⋆D unit (srcC g))) =
                          D.inv (Φ (g, srcC g) ·D (F g ⋆D unit (srcC g))) ·D F (C.runit' g)
            using C.iso-runit preserves-inv D.inv-comp by simp
          moreover have D.inv (Φ (g, srcC g) ·D (F g ⋆D unit (srcC g))) =
                        D.inv (F g ⋆D unit (srcC g)) ·D D.inv (Φ (g, srcC g))
          proof −
            have D.seq (Φ (g, srcC g)) (F g ⋆D unit (srcC g))
              using 1 antipar preserves-iso unit-char(2) by fast

            thus ?thesis
              using 1 antipar preserves-iso unit-char(2) D.inv-comp by auto
          qed
          ultimately show ?thesis
            using D.comp-assoc by simp
        qed
        thus ?thesis
          using antipar unit-char(2) preserves-iso by simp
      qed
      finally show ?thesis by simp
    qed
    thus ?thesis
      using antipar lunit-coherence by simp
  qed
  finally show ?thesis by simp
 qed
qed
qed

lemma preserves-adjoint-pair:
assumes C.adjoint-pair f g
shows D.adjoint-pair (F f) (F g)
  using assms C.adjoint-pair-def D.adjoint-pair-def preserves-adjunction by blast

lemma preserves-left-adjoint:
assumes C.is-left-adjoint f
shows D.is-left-adjoint (F f)
  using assms preserves-adjoint-pair by auto
```

lemma *preserves-right-adjoint*:
assumes $C.$*is-right-adjoint* g
shows $D.$*is-right-adjoint* $(F\ g)$
using *assms preserves-adjoint-pair* **by** *auto*

end

context *equivalence-pseudofunctor*
begin

lemma *reflects-adjunction*:
assumes $C.$*ide* f **and** $C.$*ide* g
and «$\eta : src_C\ f \Rightarrow_C g \star_C f$» **and** «$\varepsilon : f \star_C g \Rightarrow_C src_C\ g$»
and *adjunction-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ f)\ (F\ g)$
$(D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g))$
shows *adjunction-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
proof −
let $?\eta' = D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)$
let $?\varepsilon' = D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)$
interpret A': *adjunction-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$F\ f$› ‹$F\ g$› $?\eta'\ ?\varepsilon'$
using *assms*(*5*) **by** *auto*
interpret A: *adjunction-data-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
using *assms*(*1*−*4*) **by** (*unfold-locales*, *auto*)
show *?thesis*
proof
show $(\varepsilon \star_C f) \cdot_C \mathrm{a}_C{}^{-1}[f, g, f] \cdot_C (f \star_C \eta) = \mathrm{l}_C{}^{-1}[f] \cdot_C \mathrm{r}_C[f]$
proof −
have *1*: $C.par\ ((\varepsilon \star_C f) \cdot_C \mathrm{a}_C{}^{-1}[f, g, f] \cdot_C (f \star_C \eta))\ (\mathrm{l}_C{}^{-1}[f] \cdot_C \mathrm{r}_C[f])$
using *assms A.antipar* **by** *simp*
moreover have $F\ ((\varepsilon \star_C f) \cdot_C \mathrm{a}_C{}^{-1}[f, g, f] \cdot_C (f \star_C \eta)) = F\ (\mathrm{l}_C{}^{-1}[f] \cdot_C \mathrm{r}_C[f])$
proof −
have $F\ ((\varepsilon \star_C f) \cdot_C \mathrm{a}_C{}^{-1}[f, g, f] \cdot_C (f \star_C \eta)) =$
$F\ (\varepsilon \star_C f) \cdot_D F\ \mathrm{a}_C{}^{-1}[f, g, f] \cdot_D F\ (f \star_C \eta)$
using *1* **by** (*metis C.seqE preserves-comp*)
also have ... =
$(F\ (\varepsilon \star_C f) \cdot_D \Phi\ (f \star_C g, f)) \cdot_D$
$(\Phi\ (f, g) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ f, F\ g, F\ f] \cdot_D (F\ f \star_D D.inv\ (\Phi\ (g, f))) \cdot_D$
$(D.inv\ (\Phi\ (f, g \star_C f)) \cdot_D F\ (f \star_C \eta))$
using *assms A.antipar preserves-assoc*(*2*) *D.comp-assoc* **by** *auto*
also have ... = $\Phi\ (trg_C\ f, f) \cdot_D ((F\ \varepsilon \star_D F\ f) \cdot_D (\Phi\ (f, g) \star_D F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f, F\ g, F\ f] \cdot_D$
$((F\ f \star_D D.inv\ (\Phi\ (g, f))) \cdot_D (F\ f \star_D F\ \eta)) \cdot_D$
$D.inv\ (\Phi\ (f, src_C\ f))$
proof −
have $F\ (\varepsilon \star_C f) \cdot_D \Phi\ (f \star_C g, f) = \Phi\ (trg_C\ f, f) \cdot_D (F\ \varepsilon \star_D F\ f)$
using *assms* $\Phi.$*naturality* [*of* (ε, f)] *FF-def* $C.VV.arr\text{-}char_{SbC}$
$C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$

by *simp*
moreover have $D.inv\ (\Phi\ (f,\ g \star_C f)) \cdot_D F\ (f \star_C \eta) =$
$(F\ f \star_D F\ \eta) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
proof −
have $F\ (f \star_C \eta) \cdot_D \Phi\ (f,\ src_C\ f) = \Phi\ (f,\ g \star_C f) \cdot_D (F\ f \star_D F\ \eta)$
using *assms* Φ*.naturality* [*of* $(f,\ \eta)$] *FF-def* $C.VV.arr\text{-}char_{SbC}$ *A.antipar*
$C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
thus *?thesis*
using *assms A.antipar cmp-components-are-iso* $C.VV.arr\text{-}char_{SbC}$ *cmp-in-hom*
FF-def $C.VV.dom\text{-}simp$ $C.VV.cod\text{-}simp$
D.invert-opposite-sides-of-square
[*of* $\Phi\ (f,\ g \star_C f)\ F\ f \star_D F\ \eta\ F\ (f \star_C \eta)\ \Phi\ (f,\ src_C\ f)$]
by *fastforce*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (trg_C\ f,\ f) \cdot_D (F\ \varepsilon \cdot_D \Phi\ (f,\ g) \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D$
$(F\ f \star_D D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
using *assms A.antipar cmp-in-hom A.ide-left A.ide-right A′.ide-left A′.ide-right*
D.whisker-left [*of F f D.inv* $(\Phi\ (g,\ f))\ F\ \eta$]
D.whisker-right [*of F f F* $\varepsilon\ \Phi\ (f,\ g)$]
by (*metis A′.counit-in-vhom A′.unit-simps*(*1*)*D.arrI D.comp-assoc*
D.src.preserves-reflects-arr D.src-vcomp D.vseq-implies-hpar(*1*) *cmp-simps*(*2*))
also have ... $= \Phi\ (trg_C\ f,\ f) \cdot_D (unit\ (trg_C\ f) \cdot_D ?\varepsilon' \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D$
$(F\ f \star_D ?\eta' \cdot_D D.inv\ (unit\ (src_C\ f))) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
proof −
have $F\ \varepsilon \cdot_D \Phi\ (f,\ g) = unit\ (trg_C\ f) \cdot_D ?\varepsilon'$
proof −
have $D.iso\ (unit\ (trg_C\ f))$
using *A.ide-left C.ideD*(*1*) *unit-char*(*2*) **by** *blast*
thus *?thesis*
by (*metis A′.counit-simps*(*1*) *D.comp-assoc D.comp-cod-arr D.inv-is-inverse*
D.seqE D.comp-arr-inv)
qed
moreover have $D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta = ?\eta' \cdot_D D.inv\ (unit\ (src_C\ f))$
using *assms*(*2*) *unit-char D.comp-arr-inv D.inv-is-inverse D.comp-assoc D.comp-cod-arr*
by (*metis A′.unit-simps*(*1*) *A.antipar*(*1*) *C.ideD*(*1*) *C.obj-trg*
D.invert-side-of-triangle(*2*))
ultimately show *?thesis* **by** *simp*
qed
also have ... $= \Phi\ (trg_C\ f,\ f) \cdot_D ((unit\ (trg_C\ f) \star_D F\ f) \cdot_D$
$(?\varepsilon' \star_D F\ f)) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D ((F\ f \star_D ?\eta') \cdot_D$
$(F\ f \star_D D.inv\ (unit\ (src_C\ f)))) \cdot_D D.inv\ (\Phi\ (f,\ src_C\ f))$
using *assms A.antipar A′.antipar unit-char D.whisker-left D.whisker-right*
by *simp*

also have ... $= \Phi\ (trg_C\ f,\ f) \cdot_D\ (unit\ (trg_C\ f) \star_D\ F\ f) \cdot_D$
$((?\varepsilon' \star_D\ F\ f) \cdot_D\ \mathrm{a}_D{}^{-1}[F\ f,\ F\ g,\ F\ f] \cdot_D\ (F\ f \star_D\ ?\eta')) \cdot_D$
$(F\ f \star_D\ D.inv\ (unit\ (src_C\ f))) \cdot_D\ D.inv\ (\Phi\ (f,\ src_C\ f))$
using *D.comp-assoc* **by** *simp*
also have ... $= (\Phi\ (trg_C\ f,\ f) \cdot_D\ (unit\ (trg_C\ f) \star_D\ F\ f) \cdot_D\ \mathrm{l}_D{}^{-1}[F\ f]) \cdot_D$
$\mathrm{r}_D[F\ f] \cdot_D\ (F\ f \star_D\ D.inv\ (unit\ (src_C\ f))) \cdot_D\ D.inv\ (\Phi\ (f,\ src_C\ f))$
using *A′.triangle-left D.comp-assoc* **by** *simp*
also have ... $= F\ \mathrm{l}_C{}^{-1}[f] \cdot_D\ F\ \mathrm{r}_C[f]$
using *assms A.antipar preserves-lunit(2) preserves-runit(1)* **by** *simp*
also have ... $= F\ (\mathrm{l}_C{}^{-1}[f] \cdot_C\ \mathrm{r}_C[f])$
using *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
using *is-faithful* **by** *blast*
qed
show $(g \star_C\ \varepsilon) \cdot_C\ \mathrm{a}_C[g,\ f,\ g] \cdot_C\ (\eta \star_C\ g) = \mathrm{r}_C{}^{-1}[g] \cdot_C\ \mathrm{l}_C[g]$
proof −
have *1*: $C.par\ ((g \star_C\ \varepsilon) \cdot_C\ \mathrm{a}_C\ g\ f\ g \cdot_C\ (\eta \star_C\ g))\ (\mathrm{r}_C{}^{-1}[g] \cdot_C\ \mathrm{l}_C[g])$
using *assms A.antipar* **by** *auto*
moreover have $F\ ((g \star_C\ \varepsilon) \cdot_C\ \mathrm{a}_C[g,\ f,\ g] \cdot_C\ (\eta \star_C\ g)) = F\ (\mathrm{r}_C{}^{-1}[g] \cdot_C\ \mathrm{l}_C[g])$
proof −
have $F\ ((g \star_C\ \varepsilon) \cdot_C\ \mathrm{a}_C\ g\ f\ g \cdot_C\ (\eta \star_C\ g)) =$
$F\ (g \star_C\ \varepsilon) \cdot_D\ F\ \mathrm{a}_C[g,\ f,\ g] \cdot_D\ F\ (\eta \star_C\ g)$
using *1* **by** *auto*
also have ... $= (F\ (g \star_C\ \varepsilon) \cdot_D\ \Phi\ (g,\ f \star_C\ g)) \cdot_D\ (F\ g \star_D\ \Phi\ (f,\ g)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ F\ g] \cdot_D$
$(D.inv\ (\Phi\ (g,\ f)) \star_D\ F\ g) \cdot_D\ (D.inv\ (\Phi\ (g \star_C\ f,\ g)) \cdot_D\ F\ (\eta \star_C\ g))$
using *assms A.antipar preserves-assoc(1) [of g f g] D.comp-assoc* **by** *auto*
also have ... $= \Phi\ (g,\ src_C\ g) \cdot_D\ ((F\ g \star_D\ F\ \varepsilon) \cdot_D\ (F\ g \star_D\ \Phi\ (f,\ g))) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ F\ g] \cdot_D$
$((D.inv\ (\Phi\ (g,\ f)) \star_D\ F\ g) \cdot_D\ (F\ \eta \star_D\ F\ g)) \cdot_D\ D.inv\ (\Phi\ (trg_C\ g,\ g))$
proof −
have $F\ (g \star_C\ \varepsilon) \cdot_D\ \Phi\ (g,\ f \star_C\ g) = \Phi\ (g,\ src_C\ g) \cdot_D\ (F\ g \star_D\ F\ \varepsilon)$
using *assms* Φ*.naturality [of* $(g,\ \varepsilon)$*] FF-def* $C.VV.arr\text{-}char_{SbC}$
$C.VV.dom\text{-}simp\ C.VV.cod\text{-}simp$
by *auto*
moreover have $D.inv\ (\Phi\ (g \star_C\ f,\ g)) \cdot_D\ F\ (\eta \star_C\ g) =$
$(F\ \eta \star_D\ F\ g) \cdot_D\ D.inv\ (\Phi\ (trg_C\ g,\ g))$
proof −
have $F\ (\eta \star_C\ g) \cdot_D\ \Phi\ (trg_C\ g,\ g) = \Phi\ (g \star_C\ f,\ g) \cdot_D\ (F\ \eta \star_D\ F\ g)$
using *assms* Φ*.naturality [of* $(\eta,\ g)$*] FF-def* $C.VV.arr\text{-}char_{SbC}$ *A.antipar*
$C.VV.dom\text{-}simp\ C.VV.cod\text{-}simp$
by *auto*
thus *?thesis*
using *assms A.antipar cmp-components-are-iso* $C.VV.arr\text{-}char_{SbC}$ *FF-def*
$C.VV.dom\text{-}simp\ C.VV.cod\text{-}simp$
D.invert-opposite-sides-of-square
$[of\ \Phi\ (g \star_C\ f,\ g)\ F\ \eta \star_D\ F\ g\ F\ (\eta \star_C\ g)\ \Phi\ (trg_C\ g,\ g)]$

```
        by fastforce
      qed
      ultimately show ?thesis
        using D.comp-assoc by simp
    qed
    also have ... = Φ (g, src_C g) ·_D (F g ⋆_D F ε ·_D Φ (f, g)) ·_D
                    a_D[F g, F f, F g] ·_D
                    (D.inv (Φ (g, f)) ·_D F η ⋆_D F g) ·_D D.inv (Φ (trg_C g, g))
    proof −
      have (F g ⋆_D F ε) ·_D (F g ⋆_D Φ (f, g)) = F g ⋆_D F ε ·_D Φ (f, g)
        using assms A.antipar D.whisker-left
        by (metis A′.counit-simps(1) A′.ide-right D.seqE)
      moreover have (D.inv (Φ (g, f)) ⋆_D F g) ·_D (F η ⋆_D F g) =
                    D.inv (Φ (g, f)) ·_D F η ⋆_D F g
        using assms A.antipar D.whisker-right by simp
      ultimately show ?thesis by simp
    qed
    also have ... = Φ (g, src_C g) ·_D (F g ⋆_D unit (trg_C f) ·_D ?ε′) ·_D
                    a_D[F g, F f, F g] ·_D
                    (?η′ ·_D D.inv (unit (src_C f)) ⋆_D F g) ·_D D.inv (Φ (trg_C g, g))
    proof −
      have F ε ·_D Φ (f, g) = unit (trg_C f) ·_D ?ε′
        using unit-char D.comp-arr-inv D.inv-is-inverse D.comp-assoc D.comp-cod-arr
        by (metis A′.counit-simps(1) C.ideD(1) C.obj-trg D.seqE assms(1))
      moreover have D.inv (Φ (g, f)) ·_D F η = ?η′ ·_D D.inv (unit (src_C f))
        using unit-char D.comp-arr-inv D.inv-is-inverse D.comp-assoc D.comp-cod-arr
        by (metis A′.unit-simps(1) A.unit-simps(1) A.unit-simps(5)
            C.obj-trg D.invert-side-of-triangle(2))
      ultimately show ?thesis by simp
    qed
    also have ... = Φ (g, src_C g) ·_D (F g ⋆_D unit (trg_C f)) ·_D
                    ((F g ⋆_D ?ε′) ·_D a_D[F g, F f, F g] ·_D (?η′ ⋆_D F g)) ·_D
                    (D.inv (unit (src_C f)) ⋆_D F g) ·_D D.inv (Φ (trg_C g, g))
      using assms A.antipar unit-char D.whisker-left D.whisker-right D.comp-assoc
      by simp
    also have ... = Φ (g, src_C g) ·_D (F g ⋆_D unit (trg_C f)) ·_D r_D^-1[F g] ·_D
                    l_D[F g] ·_D (D.inv (unit (src_C f)) ⋆_D F g) ·_D D.inv (Φ (trg_C g, g))
      using A′.triangle-right D.comp-assoc by simp
    also have ... = F r_C^-1[g] ·_D F l_C[g]
        using assms A.antipar preserves-lunit(1) preserves-runit(2) D.comp-assoc
        by simp
    also have ... = F (r_C^-1[g] ·_C l_C[g])
      using assms by simp
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using is-faithful by blast
 qed
qed
```

qed

lemma *reflects-adjoint-pair*:
assumes $C.ide\ f$ **and** $C.ide\ g$
and $src_C\ f = trg_C\ g$ **and** $src_C\ g = trg_C\ f$
and $D.adjoint\text{-}pair\ (F\ f)\ (F\ g)$
shows $C.adjoint\text{-}pair\ f\ g$
proof −
 obtain $\eta'\ \varepsilon'$ **where** A': $adjunction\text{-}in\text{-}bicategory\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ f)\ (F\ g)\ \eta'\ \varepsilon'$
  using *assms D.adjoint-pair-def* **by** *auto*
 interpret A': $adjunction\text{-}in\text{-}bicategory\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ \langle F\ f\rangle\ \langle F\ g\rangle\ \eta'\ \varepsilon'$
  using A' **by** *auto*
 have *1*: «$\Phi\ (g, f) \cdot_D \eta' \cdot_D D.inv\ (unit\ (src_C\ f)) : F\ (src_C\ f) \Rightarrow_D F\ (g \star_C f)$»
  using *assms unit-char* [*of* $src_C\ f$] A'*.unit-in-hom*
  by (*intro D.comp-in-homI*, *auto*)
 have *2*: «$unit\ (trg_C\ f) \cdot_D \varepsilon' \cdot_D D.inv\ (\Phi\ (f, g)): F\ (f \star_C g) \Rightarrow_D F\ (trg_C\ f)$»
  using *assms cmp-in-hom* [*of f g*] *unit-char* [*of* $trg_C\ f$] A'*.counit-in-hom*
  by (*intro D.comp-in-homI*, *auto*)
 obtain η **where** η: «$\eta : src_C\ f \Rightarrow_C g \star_C f$» $\wedge$
  $F\ \eta = \Phi\ (g, f) \cdot_D \eta' \cdot_D D.inv\ (unit\ (src_C\ f))$
  using *assms 1* A'*.unit-in-hom cmp-in-hom locally-full* **by** *fastforce*
 have η': $\eta' = D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f)$
  using *assms 1* η *cmp-in-hom D.iso-inv-iso cmp-components-are-iso unit-char*(*2*)
   D.invert-side-of-triangle(*1*) [*of* $F\ \eta\ \Phi\ (g, f)\ \eta' \cdot_D D.inv\ (unit\ (src_C\ f))$]
   D.invert-side-of-triangle(*2*) [*of* $D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta\ \eta'\ D.inv\ (unit\ (src_C\ f))$]
  by (*metis* (*no-types*, *lifting*) *C.ideD*(*1*) *C.obj-trg D.arrI D.comp-assoc D.inv-inv*)
 obtain ε **where** ε: «$\varepsilon : f \star_C g \Rightarrow_C trg_C\ f$» $\wedge$
  $F\ \varepsilon = unit\ (trg_C\ f) \cdot_D \varepsilon' \cdot_D D.inv\ (\Phi\ (f, g))$
  using *assms 2* A'*.counit-in-hom cmp-in-hom locally-full* **by** *fastforce*
 have ε': $\varepsilon' = D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g)$
  using *assms 2* ε *cmp-in-hom D.iso-inv-iso unit-char*(*2*) *D.comp-assoc*
   D.invert-side-of-triangle(*1*) [*of* $F\ \varepsilon\ unit\ (trg_C\ f)\ \varepsilon' \cdot_D D.inv\ (\Phi\ (f, g))$]
   D.invert-side-of-triangle(*2*) [*of* $D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon\ \varepsilon'\ D.inv\ (\Phi\ (f, g))$]
 by (*metis* (*no-types*, *lifting*) *C.arrI C.ideD*(*1*) *C.obj-trg D.inv-inv cmp-components-are-iso preserves-arr*)
 have $adjunction\text{-}in\text{-}bicategory\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ f)\ (F\ g)$
   $(D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
   $(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g))$
  using A'*.adjunction-in-bicategory-axioms* $\eta'\ \varepsilon'$ **by** *simp*
 hence $adjunction\text{-}in\text{-}bicategory\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
  using *assms* $\eta\ \varepsilon$ *reflects-adjunction* **by** *simp*
 thus *?thesis*
  using *C.adjoint-pair-def* **by** *auto*
qed

lemma *reflects-left-adjoint*:
assumes $C.ide\ f$ **and** $D.is\text{-}left\text{-}adjoint\ (F\ f)$
shows $C.is\text{-}left\text{-}adjoint\ f$
proof −

obtain g' **where** g': *D.adjoint-pair* $(F\ f)\ g'$
using *assms D.adjoint-pair-def* **by** *auto*
obtain g **where** g: «g : $trg_C\ f \rightarrow_C src_C\ f$» $\wedge$ *C.ide* g $\wedge$ *D.isomorphic* $(F\ g)\ g'$
using *assms* g' *locally-essentially-surjective* [*of* $trg_C\ f\ src_C\ f\ g'$]
D.adjoint-pair-antipar [*of* $F\ f\ g'$]
by *auto*
obtain φ **where** φ: «$\varphi : g' \Rightarrow_D F\ g$» $\wedge$ *D.iso* φ
using g *D.isomorphic-def D.isomorphic-symmetric* **by** *metis*
have *D.adjoint-pair* $(F\ f)\ (F\ g)$
using *assms* $g\ g'\ \varphi$ *D.adjoint-pair-preserved-by-iso* [*of* $F\ f\ g'\ F\ f\ F\ f\ \varphi\ F\ g$]
by *auto*
thus *?thesis*
using *assms* g *reflects-adjoint-pair* [*of* $f\ g$] *D.adjoint-pair-antipar C.in-hhom-def*
by *auto*
qed

lemma *reflects-right-adjoint*:
assumes *C.ide* g **and** *D.is-right-adjoint* $(F\ g)$
shows *C.is-right-adjoint* g
proof −
obtain f' **where** f': *D.adjoint-pair* $f'\ (F\ g)$
using *assms D.adjoint-pair-def* **by** *auto*
obtain f **where** f: «f : $trg_C\ g \rightarrow_C src_C\ g$» $\wedge$ *C.ide* f $\wedge$ *D.isomorphic* $(F\ f)\ f'$
using *assms* f' *locally-essentially-surjective* [*of* $trg_C\ g\ src_C\ g\ f'$]
D.adjoint-pair-antipar [*of* $f'\ F\ g$]
by *auto*
obtain φ **where** φ: «$\varphi : f' \Rightarrow_D F\ f$» $\wedge$ *D.iso* φ
using f *D.isomorphic-def D.isomorphic-symmetric* **by** *metis*
have *D.adjoint-pair* $(F\ f)\ (F\ g)$
using *assms* $f\ f'\ \varphi$ *D.adjoint-pair-preserved-by-iso* [*of* $f'\ F\ g\ \varphi\ F\ f\ F\ g\ F\ g$]
by *auto*
thus *?thesis*
using *assms* f *reflects-adjoint-pair* [*of* $f\ g$] *D.adjoint-pair-antipar C.in-hhom-def*
by *auto*
qed

end

1.13.4 Composition of Adjunctions

We first consider the strict case, then extend to all bicategories using strictification.

locale *composite-adjunction-in-strict-bicategory* =
strict-bicategory $V\ H$ a i *src trg* +
fg: *adjunction-in-strict-bicategory* $V\ H$ a i *src trg* $f\ g\ \zeta\ \xi$ +
hk: *adjunction-in-strict-bicategory* $V\ H$ a i *src trg* $h\ k\ \sigma\ \tau$
for V :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)

```
and src :: 'a ⇒ 'a
and trg :: 'a ⇒ 'a
and f :: 'a
and g :: 'a
and ζ :: 'a
and ξ :: 'a
and h :: 'a
and k :: 'a
and σ :: 'a
and τ :: 'a +
assumes composable: src h = trg f
begin

  abbreviation η
  where η ≡ (g ⋆ σ ⋆ f) · ζ

  abbreviation ε
  where ε ≡ τ · (h ⋆ ξ ⋆ k)

  interpretation adjunction-data-in-bicategory V H a i src trg ‹h ⋆ f› ‹g ⋆ k› η ε
  proof
    show ide (h ⋆ f)
      using composable by simp
    show ide (g ⋆ k)
      using fg.antipar hk.antipar composable by simp
    show «η : src (h ⋆ f) ⇒ (g ⋆ k) ⋆ h ⋆ f»
    proof
      show «ζ : src (h ⋆ f) ⇒ g ⋆ f»
        using fg.antipar hk.antipar composable ‹ide (h ⋆ f)› by auto
      show «g ⋆ σ ⋆ f : g ⋆ f ⇒ (g ⋆ k) ⋆ h ⋆ f»
      proof −
        have «g ⋆ σ ⋆ f : g ⋆ trg f ⋆ f ⇒ g ⋆ (k ⋆ h) ⋆ f»
          using fg.antipar hk.antipar composable hk.unit-in-hom
          apply (intro hcomp-in-vhom) by auto
        thus ?thesis
          using hcomp-obj-arr hcomp-assoc by fastforce
      qed
    qed
    show «ε : (h ⋆ f) ⋆ g ⋆ k ⇒ src (g ⋆ k)»
    proof
      show «h ⋆ ξ ⋆ k : (h ⋆ f) ⋆ g ⋆ k ⇒ h ⋆ k»
      proof −
        have «h ⋆ ξ ⋆ k : h ⋆ (f ⋆ g) ⋆ k ⇒ h ⋆ trg f ⋆ k»
          using composable fg.antipar(1−2) hk.antipar(1) by fastforce
        thus ?thesis
          using fg.antipar hk.antipar composable hk.unit-in-hom hcomp-obj-arr hcomp-assoc
          by simp
      qed
      show «τ : h ⋆ k ⇒ src (g ⋆ k)»
```

using *fg.antipar hk.antipar composable hk.unit-in-hom* **by** *auto*
qed
qed

sublocale *adjunction-in-strict-bicategory V H* a i *src trg* ‹$h \star f$› ‹$g \star k$› η ε
proof
show $(\varepsilon \star h \star f) \cdot \mathrm{a}^{-1}[h \star f,\ g \star k,\ h \star f] \cdot ((h \star f) \star \eta) = \mathrm{l}^{-1}[h \star f] \cdot \mathrm{r}[h \star f]$
proof −
have $(\varepsilon \star h \star f) \cdot \mathrm{a}^{-1}[h \star f,\ g \star k,\ h \star f] \cdot ((h \star f) \star \eta) =$
$(\tau \cdot (h \star \xi \star k) \star h \star f) \cdot ((h \star f) \star (g \star \sigma \star f) \cdot \zeta)$
using *fg.antipar hk.antipar composable strict-assoc comp-ide-arr ide-left ide-right antipar(1) antipar(2)*
by (*metis arrI seqE strict-assoc′ triangle-in-hom(1)*)
also have ... $= (\tau \star h \star f) \cdot ((h \star \xi \star (k \star h) \star f) \cdot (h \star (f \star g) \star \sigma \star f)) \cdot (h \star f \star \zeta)$
using *fg.antipar hk.antipar composable whisker-left* [*of* $h \star f$] *whisker-right comp-assoc hcomp-assoc*
by *simp*
also have ... $= (\tau \star h \star f) \cdot (h \star (\xi \star (k \star h)) \cdot ((f \star g) \star \sigma) \star f) \cdot (h \star f \star \zeta)$
using *fg.antipar hk.antipar composable whisker-left whisker-right hcomp-assoc*
by *simp*
also have ... $= (\tau \star h \star f) \cdot (h \star (\mathit{trg}\ f \star \sigma) \cdot (\xi \star \mathit{trg}\ f) \star f) \cdot (h \star f \star \zeta)$
using *fg.antipar hk.antipar composable comp-arr-dom comp-cod-arr interchange* [*of* $\xi\ f \star g\ k \star h\ \sigma$] *interchange* [*of trg f* $\xi\ \sigma$ *trg f*]
by (*metis fg.counit-simps(1) fg.counit-simps(2) fg.counit-simps(3) hk.unit-simps(1) hk.unit-simps(2) hk.unit-simps(3)*)
also have ... $= (\tau \star h \star f) \cdot (h \star \sigma \cdot \xi \star f) \cdot (h \star f \star \zeta)$
using *fg.antipar hk.antipar composable hcomp-obj-arr hcomp-arr-obj*
by (*metis fg.counit-simps(1) fg.counit-simps(4) hk.unit-simps(1) hk.unit-simps(5) obj-src*)
also have ... $= ((\tau \star h \star f) \cdot (h \star \sigma \star f)) \cdot ((h \star \xi \star f) \cdot (h \star f \star \zeta))$
using *fg.antipar hk.antipar composable whisker-left whisker-right comp-assoc*
by *simp*
also have ... $= ((\tau \star h) \cdot (h \star \sigma) \star f) \cdot (h \star (\xi \star f) \cdot (f \star \zeta))$
using *fg.antipar hk.antipar composable whisker-left whisker-right hcomp-assoc*
by *simp*
also have ... $= h \star f$
using *fg.antipar hk.antipar composable fg.triangle-left hk.triangle-left*
by *simp*
also have ... $= \mathrm{l}^{-1}[h \star f] \cdot \mathrm{r}[h \star f]$
using *fg.antipar hk.antipar composable strict-lunit′ strict-runit* **by** *simp*
finally show *?thesis* **by** *simp*
qed
show $((g \star k) \star \varepsilon) \cdot \mathrm{a}[g \star k,\ h \star f,\ g \star k] \cdot (\eta \star g \star k) = \mathrm{r}^{-1}[g \star k] \cdot \mathrm{l}[g \star k]$
proof −
have $((g \star k) \star \varepsilon) \cdot \mathrm{a}[g \star k,\ h \star f,\ g \star k] \cdot (\eta \star g \star k) =$
$((g \star k) \star \tau \cdot (h \star \xi \star k)) \cdot ((g \star \sigma \star f) \cdot \zeta \star g \star k)$
using *fg.antipar hk.antipar composable strict-assoc comp-ide-arr ide-left ide-right*
by (*metis antipar(1) antipar(2) arrI seqE triangle-in-hom(2)*)

also have ... $= (g \star k \star \tau) \cdot ((g \star (k \star h) \star \xi \star k) \cdot (g \star \sigma \star (f \star g) \star k)) \cdot (\zeta \star g \star k)$
using *fg.antipar hk.antipar composable whisker-left* [*of* $g \star k$] *whisker-right comp-assoc hcomp-assoc*
by *simp*
also have ... $= (g \star k \star \tau) \cdot (g \star ((k \star h) \star \xi) \cdot (\sigma \star f \star g) \star k) \cdot (\zeta \star g \star k)$
using *fg.antipar hk.antipar composable whisker-left whisker-right hcomp-assoc*
by *simp*
also have ... $= (g \star k \star \tau) \cdot (g \star (\sigma \star src\ g) \cdot (src\ g \star \xi) \star k) \cdot (\zeta \star g \star k)$
using *fg.antipar hk.antipar composable interchange* [*of* $k \star h\ \sigma\ \xi\ f \star g$] *interchange* [*of* σ *src g src g* ξ] *comp-arr-dom comp-cod-arr*
by (*metis fg.counit-simps*(*1*) *fg.counit-simps*(*2*) *fg.counit-simps*(*3*) *hk.unit-simps*(*1*) *hk.unit-simps*(*2*) *hk.unit-simps*(*3*))
also have ... $= (g \star k \star \tau) \cdot (g \star \sigma \cdot \xi \star k) \cdot (\zeta \star g \star k)$
using *fg.antipar hk.antipar composable hcomp-obj-arr* [*of src g* ξ] *hcomp-arr-obj* [*of* σ *src g*]
by *simp*
also have ... $= ((g \star k \star \tau) \cdot (g \star \sigma \star k)) \cdot (g \star \xi \star k) \cdot (\zeta \star g \star k)$
using *fg.antipar hk.antipar composable whisker-left whisker-right comp-assoc*
by *simp*
also have ... $= (g \star (k \star \tau) \cdot (\sigma \star k)) \cdot ((g \star \xi) \cdot (\zeta \star g) \star k)$
using *fg.antipar hk.antipar composable whisker-left whisker-right hcomp-assoc*
by *simp*
also have ... $= g \star k$
using *fg.antipar hk.antipar composable fg.triangle-right hk.triangle-right*
by *simp*
also have ... $= \mathrm{r}^{-1}[g \star k] \cdot \mathrm{l}[g \star k]$
using *fg.antipar hk.antipar composable strict-lunit strict-runit'* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *is-adjunction-in-strict-bicategory*:
shows *adjunction-in-strict-bicategory V H* a i *src trg* $(h \star f)$ $(g \star k)$ η ε
..

end

context *strict-bicategory*
begin

lemma *left-adjoints-compose*:
assumes *is-left-adjoint f* **and** *is-left-adjoint f'* **and** *src f' = trg f*
shows *is-left-adjoint* $(f' \star f)$
proof −
obtain *g* η ε **where** *fg*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
using *assms adjoint-pair-def* **by** *auto*
interpret *fg*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
using *fg* **by** *auto*
obtain *g'* η' ε' **where** *f'g'*: *adjunction-in-bicategory V H* a i *src trg f' g'* η' ε'

using *assms adjoint-pair-def* **by** *auto*
interpret $f'g'$: *adjunction-in-bicategory V H* a i *src trg* f' g' η' ε'
using $f'g'$ **by** *auto*
interpret $f'fgg'$: *composite-adjunction-in-strict-bicategory V H* a i *src trg*
f g η ε f' g' η' ε'
using *assms* **apply** *unfold-locales* **by** *simp*
have *adjoint-pair* $(f' \star f)$ $(g \star g')$
using *adjoint-pair-def* $f'fgg'$*.adjunction-in-bicategory-axioms* **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *right-adjoints-compose*:
assumes *is-right-adjoint* g **and** *is-right-adjoint* g' **and** *src* g = *trg* g'
shows *is-right-adjoint* $(g \star g')$
proof −
obtain f η ε **where** *fg*: *adjunction-in-bicategory V H* a i *src trg* f g η ε
using *assms adjoint-pair-def* **by** *auto*
interpret *fg*: *adjunction-in-bicategory V H* a i *src trg* f g η ε
using *fg* **by** *auto*
obtain f' η' ε' **where** $f'g'$: *adjunction-in-bicategory V H* a i *src trg* f' g' η' ε'
using *assms adjoint-pair-def* **by** *auto*
interpret $f'g'$: *adjunction-in-bicategory V H* a i *src trg* f' g' η' ε'
using $f'g'$ **by** *auto*
interpret $f'fgg'$: *composite-adjunction-in-strict-bicategory V H* a i *src trg*
f g η ε f' g' η' ε'
using *assms* fg*.antipar* $f'g'$*.antipar* **apply** *unfold-locales* **by** *simp*
have *adjoint-pair* $(f' \star f)$ $(g \star g')$
using *adjoint-pair-def* $f'fgg'$*.adjunction-in-bicategory-axioms* **by** *auto*
thus *?thesis* **by** *auto*
qed

end

We now use strictification to extend the preceding results to an arbitrary bicategory. We only prove that "left adjoints compose" and "right adjoints compose"; I did not work out formulas for the unit and counit of the composite adjunction in the non-strict case.

context *bicategory*
begin

interpretation S: *strictified-bicategory V H* a i *src trg* **..**

notation S*.vcomp* (**infixr** ‹$\cdot_S$› *55*)
notation S*.hcomp* (**infixr** ‹$\star_S$› *53*)
notation S*.in-hom* (‹«- : - $\Rightarrow_S$ -»›)
notation S*.in-hhom* (‹«- : - $\rightarrow_S$ -»›)

interpretation *UP*: *fully-faithful-functor V* S*.vcomp* S*.UP*
using S*.UP-is-fully-faithful-functor* **by** *auto*
interpretation *UP*: *equivalence-pseudofunctor V H* a i *src trg*

$S.vcomp\ S.hcomp\ S.\mathrm{a}\ S.\mathrm{i}\ S.src\ S.trg\ S.UP\ S.cmp_{UP}$
using *S.UP-is-equivalence-pseudofunctor* **by** *auto*

lemma *left-adjoints-compose*:
assumes *is-left-adjoint f* **and** *is-left-adjoint f′* **and** *src f = trg f′*
shows *is-left-adjoint* $(f \star f')$
proof −
have $S.is\text{-}left\text{-}adjoint\ (S.UP\ f) \wedge S.is\text{-}left\text{-}adjoint\ (S.UP\ f')$
using *assms UP.preserves-left-adjoint* **by** *simp*
moreover have $S.src\ (S.UP\ f) = S.trg\ (S.UP\ f')$
using *assms left-adjoint-is-ide* **by** *simp*
ultimately have $S.is\text{-}left\text{-}adjoint\ (S.hcomp\ (S.UP\ f)\ (S.UP\ f'))$
using *S.left-adjoints-compose* **by** *simp*
moreover have $S.isomorphic\ (S.hcomp\ (S.UP\ f)\ (S.UP\ f'))\ (S.UP\ (f \star f'))$
proof −
have «$S.cmp_{UP}\ (f, f') : S.hcomp\ (S.UP\ f)\ (S.UP\ f') \Rightarrow_S S.UP\ (f \star f')$»
using *assms left-adjoint-is-ide UP.cmp-in-hom* **by** *simp*
moreover have $S.iso\ (S.cmp_{UP}\ (f, f'))$
using *assms left-adjoint-is-ide* **by** *simp*
ultimately show *?thesis*
using *S.isomorphic-def* **by** *blast*
qed
ultimately have $S.is\text{-}left\text{-}adjoint\ (S.UP\ (f \star f'))$
using *S.left-adjoint-preserved-by-iso S.isomorphic-def* **by** *blast*
thus *is-left-adjoint* $(f \star f')$
using *assms left-adjoint-is-ide UP.reflects-left-adjoint* **by** *simp*
qed

lemma *right-adjoints-compose*:
assumes *is-right-adjoint g* **and** *is-right-adjoint g′* **and** *src g′ = trg g*
shows *is-right-adjoint* $(g' \star g)$
proof −
have $S.is\text{-}right\text{-}adjoint\ (S.UP\ g) \wedge S.is\text{-}right\text{-}adjoint\ (S.UP\ g')$
using *assms UP.preserves-right-adjoint* **by** *simp*
moreover have $S.src\ (S.UP\ g') = S.trg\ (S.UP\ g)$
using *assms right-adjoint-is-ide* **by** *simp*
ultimately have $S.is\text{-}right\text{-}adjoint\ (S.hcomp\ (S.UP\ g')\ (S.UP\ g))$
using *S.right-adjoints-compose* **by** *simp*
moreover have $S.isomorphic\ (S.hcomp\ (S.UP\ g')\ (S.UP\ g))\ (S.UP\ (g' \star g))$
proof −
have «$S.cmp_{UP}\ (g', g) : S.hcomp\ (S.UP\ g')\ (S.UP\ g) \Rightarrow_S S.UP\ (g' \star g)$»
using *assms right-adjoint-is-ide UP.cmp-in-hom* **by** *simp*
moreover have $S.iso\ (S.cmp_{UP}\ (g', g))$
using *assms right-adjoint-is-ide* **by** *simp*
ultimately show *?thesis*
using *S.isomorphic-def* **by** *blast*
qed
ultimately have $S.is\text{-}right\text{-}adjoint\ (S.UP\ (g' \star g))$
using *S.right-adjoint-preserved-by-iso S.isomorphic-def* **by** *blast*

thus *is-right-adjoint* $(g' \star g)$
using *assms right-adjoint-is-ide UP.reflects-right-adjoint* **by** *simp*
qed

end

1.13.5 Choosing Right Adjoints

It will be useful in various situations to suppose that we have made a choice of right adjoint for each left adjoint (*i.e.* each "map") in a bicategory.

locale *chosen-right-adjoints* =
bicategory
begin

unbundle *no rtrancl-syntax*

definition *some-right-adjoint* (‹-*› [*1000*] *1000*)
where $f^* \equiv$ *SOME g. adjoint-pair f g*

definition *some-unit*
where *some-unit f* $\equiv$ *SOME* η. $\exists\varepsilon$. *adjunction-in-bicategory V H* a i *src trg f* f^* η ε

definition *some-counit*
where *some-counit f* $\equiv$
SOME ε. *adjunction-in-bicategory V H* a i *src trg f* f^* (*some-unit f*) ε

lemma *left-adjoint-extends-to-adjunction*:
assumes *is-left-adjoint f*
shows *adjunction-in-bicategory V H* a i *src trg f* f^* (*some-unit f*) (*some-counit f*)
using *assms some-right-adjoint-def adjoint-pair-def some-unit-def some-counit-def*
someI-ex [*of* λg. *adjoint-pair f g*]
someI-ex [*of* $\lambda\eta$. $\exists\varepsilon$. *adjunction-in-bicategory V H* a i *src trg f* f^* η ε]
someI-ex [*of* $\lambda\varepsilon$. *adjunction-in-bicategory V H* a i *src trg f* f^* (*some-unit f*) ε]
by *auto*

lemma *left-adjoint-extends-to-adjoint-pair*:
assumes *is-left-adjoint f*
shows *adjoint-pair f* f^*
using *assms adjoint-pair-def left-adjoint-extends-to-adjunction* **by** *blast*

lemma *right-adjoint-in-hom* [*intro*]:
assumes *is-left-adjoint f*
shows «f^* : *trg f* $\rightarrow$ *src f*»
and «f^* : $f^* \Rightarrow f^*$»
using *assms left-adjoint-extends-to-adjoint-pair adjoint-pair-antipar* [*of f* f^*]
by *auto*

lemma *right-adjoint-simps* [*simp*]:
assumes *is-left-adjoint f*

shows *ide* f^*
and *src* f^* = *trg* f **and** *trg* f^* = *src* f
and *dom* f^* = f^* **and** *cod* f^* = f^*
using *assms right-adjoint-in-hom left-adjoint-extends-to-adjoint-pair* **apply** *auto*
using *assms right-adjoint-is-ide* [*of* f^*] **by** *blast*

end

locale *map-in-bicategory* =
bicategory + *chosen-right-adjoints* +
fixes f :: $'a$
assumes *is-map*: *is-left-adjoint* f
begin

abbreviation η
where $\eta \equiv$ *some-unit* f

abbreviation ε
where $\varepsilon \equiv$ *some-counit* f

sublocale *adjunction-in-bicategory* V H a i *src trg* f ‹f^*› η ε
using *is-map left-adjoint-extends-to-adjunction* **by** *simp*

end

1.13.6 Equivalences Refine to Adjoint Equivalences

In this section, we show that, just as an equivalence between categories can always be refined to an adjoint equivalence, an internal equivalence in a bicategory can also always be so refined. The proof, which follows that of Theorem 3.3 from [9], makes use of the fact that if an internal equivalence satisfies one of the triangle identities, then it also satisfies the other.

locale *adjoint-equivalence-in-bicategory* =
equivalence-in-bicategory +
adjunction-in-bicategory
begin

lemma *dual-adjoint-equivalence*:
shows *adjoint-equivalence-in-bicategory* V H a i *src trg* g f (*inv* ε) (*inv* η)
proof −
interpret *gf*: *equivalence-in-bicategory* V H a i *src trg* g f ‹*inv* ε› ‹*inv* η›
using *dual-equivalence* **by** *simp*
show *?thesis*
proof
show (*inv* $\eta \star g$) $\cdot$ $\mathrm{a}^{-1}[g, f, g]$ $\cdot$ ($g \star$ *inv* ε) = $\mathrm{l}^{-1}[g]$ $\cdot$ $\mathrm{r}[g]$
proof −
have (*inv* $\eta \star g$) $\cdot$ $\mathrm{a}^{-1}[g, f, g]$ $\cdot$ ($g \star$ *inv* ε) =
inv (($g \star \varepsilon$) $\cdot$ $\mathrm{a}[g, f, g]$ $\cdot$ ($\eta \star g$))

```
        using antipar inv-comp isos-compose comp-assoc by simp
      also have ... = inv (r⁻¹[g] · l[g])
        using triangle-right by simp
      also have ... = l⁻¹[g] · r[g]
        using inv-comp by simp
      finally show ?thesis
        by blast
    qed
    show (f ⋆ inv η) · a[f, g, f] · (inv ε ⋆ f) = r⁻¹[f] · l[f]
    proof −
      have (f ⋆ inv η) · a[f, g, f] · (inv ε ⋆ f) =
            inv ((ε ⋆ f) · a⁻¹[f, g, f] · (f ⋆ η))
        using antipar inv-comp isos-compose comp-assoc by simp
      also have ... = inv (l⁻¹[f] · r[f])
        using triangle-left by simp
      also have ... = r⁻¹[f] · l[f]
        using inv-comp by simp
      finally show ?thesis by blast
    qed
  qed
qed

end

context bicategory
begin

  lemma adjoint-equivalence-preserved-by-iso-right:
  assumes adjoint-equivalence-in-bicategory V H a i src trg f g η ε
  and «φ : g ⇒ g'» and iso φ
  shows adjoint-equivalence-in-bicategory V H a i src trg f g' ((φ ⋆ f) · η) (ε · (f ⋆ inv φ))
  proof −
    interpret fg: adjoint-equivalence-in-bicategory V H a i src trg f g η ε
      using assms by simp
    interpret fg': adjunction-in-bicategory V H a i src trg f g' ‹(φ ⋆ f) · η› ‹ε · (f ⋆ inv φ)›
      using assms fg.adjunction-in-bicategory-axioms adjunction-preserved-by-iso-right
      by simp
    interpret fg': equivalence-in-bicategory V H a i src trg f g' ‹(φ ⋆ f) · η› ‹ε · (f ⋆ inv φ)›
      using assms fg.equivalence-in-bicategory-axioms equivalence-preserved-by-iso-right
      by simp
    show ?thesis ..
  qed

  lemma adjoint-equivalence-preserved-by-iso-left:
  assumes adjoint-equivalence-in-bicategory V H a i src trg f g η ε
  and «φ : f ⇒ f'» and iso φ
  shows adjoint-equivalence-in-bicategory V H a i src trg f' g ((g ⋆ φ) · η) (ε · (inv φ ⋆ g))
  proof −
    interpret fg: adjoint-equivalence-in-bicategory V H a i src trg f g η ε
```

```
    using assms by simp
  interpret fg': adjunction-in-bicategory V H a i src trg f' g ‹(g ⋆ φ) · η› ‹ε · (inv φ ⋆ g)›
    using assms fg.adjunction-in-bicategory-axioms adjunction-preserved-by-iso-left
    by simp
  interpret fg': equivalence-in-bicategory V H a i src trg f' g ‹(g ⋆ φ) · η› ‹ε · (inv φ ⋆ g)›
    using assms fg.equivalence-in-bicategory-axioms equivalence-preserved-by-iso-left
    by simp
  show ?thesis ..
qed

end

context strict-bicategory
begin

  notation isomorphic (infix ‹≅› 50)

  lemma equivalence-refines-to-adjoint-equivalence:
  assumes equivalence-map f and «g : trg f → src f» and ide g
  and «η : src f ⇒ g ⋆ f» and iso η
  shows ∃!ε. adjoint-equivalence-in-bicategory V H a i src trg f g η ε
  proof −
    obtain g' η' ε' where E': equivalence-in-bicategory V H a i src trg f g' η' ε'
      using assms equivalence-map-def by auto
    interpret E': equivalence-in-bicategory V H a i src trg f g' η' ε'
      using E' by auto
    let ?a = src f and ?b = trg f

    have f-in-hhom: «f : ?a → ?b» and ide-f: ide f
      using assms equivalence-map-def by auto
    have g-in-hhom: «g : ?b → ?a» and ide-g: ide g
      using assms by auto
    have g'-in-hhom: «g' : ?b → ?a» and ide-g': ide g'
      using assms f-in-hhom E'.antipar by auto
    have η-in-hom: «η : ?a ⇒ g ⋆ f» and iso-η: iso η
      using assms by auto
    have a: obj ?a and b: obj ?b
      using f-in-hhom by auto
    have η-in-hhom: «η : ?a → ?a»
      using a η-in-hom
      by (metis arrI in-hhom-def obj-simps(2−3) vconn-implies-hpar(1−2))
```

The following is quoted from [9]:

> "Since $g \cong gfg' \cong g'$, the isomorphism $fg' \cong 1$ also induces an isomorphism $fg \cong 1$, which we denote ξ. Now η and ξ may not satisfy the zigzag identities, but if we define ε by $\xi \cdot (f \star \eta^{-1}) \cdot (f \star g \star \xi^{-1}) : f \star g \Rightarrow 1$, then we can verify, using string diagram notation as above, that ε satisfies one zigzag identity, and hence (by the previous lemma) also the other.

Finally, if ε': $fg \Rightarrow 1$ is any other isomorphism satisfying the zigzag identities with η, then we have:

$$\varepsilon' = \varepsilon' \cdot (\varepsilon\ f\ g) \cdot (f\ \eta\ g) = \varepsilon \cdot (f\ g\ \varepsilon') \cdot (f\ \eta\ g) = \varepsilon$$

using the interchange law and two zigzag identities. This shows uniqueness."

```
have 1: g ≅ g'
proof −
  have g ≅ g ⋆ ?b
    using assms hcomp-arr-obj isomorphic-reflexive by auto
  also have ... ≅ g ⋆ f ⋆ g'
    using assms f-in-hhom g-in-hhom g'-in-hhom E'.counit-in-vhom E'.counit-is-iso
          isomorphic-def hcomp-ide-isomorphic isomorphic-symmetric
    by (metis E'.counit-simps(5) in-hhomE trg-trg)
  also have ... ≅ ?a ⋆ g'
    using assms f-in-hhom g-in-hhom g'-in-hhom ide-g' E'.unit-in-vhom E'.unit-is-iso
          isomorphic-def hcomp-isomorphic-ide isomorphic-symmetric
    by (metis hcomp-assoc hcomp-isomorphic-ide in-hhomE src-src)
  also have ... ≅ g'
    using assms
    by (simp add: E'.antipar(1) hcomp-obj-arr isomorphic-reflexive)
  finally show ?thesis by blast
qed
have f ⋆ g' ≅ ?b
  using E'.counit-is-iso isomorphicI [of ε'] by auto
hence 2: f ⋆ g ≅ ?b
  using assms 1 ide-f hcomp-ide-isomorphic [of f g g'] isomorphic-transitive
        isomorphic-symmetric
  by (metis in-hhomE)
obtain ξ where ξ: «ξ : f ⋆ g ⇒ ?b» ∧ iso ξ
  using 2 by auto
have ξ-in-hom: «ξ : f ⋆ g ⇒ ?b» and iso-ξ: iso ξ
  using ξ by auto
have ξ-in-hhom: «ξ : ?b → ?b»
  using b ξ-in-hom
  by (metis ξ in-hhom-def iso-is-arr obj-simps(2−3) vconn-implies-hpar(1−4))
```

At the time of this writing, the definition of ε given on nLab [9] had an apparent typo: the expression $f \star g \star \xi^{-1}$ should read $\xi^{-1} \star f \star g$, as we have used here.

```
let ?ε = ξ · (f ⋆ inv η ⋆ g) · (inv ξ ⋆ f ⋆ g)
have ε-in-hom: «?ε : f ⋆ g ⇒ ?b»
proof (intro comp-in-homI)
  show «f ⋆ inv η ⋆ g : f ⋆ g ⋆ f ⋆ g ⇒ f ⋆ g»
  proof −
    have «f ⋆ inv η ⋆ g : f ⋆ (g ⋆ f) ⋆ g ⇒ f ⋆ g»
    proof −
      have «f ⋆ inv η ⋆ g : f ⋆ (g ⋆ f) ⋆ g ⇒ f ⋆ ?a ⋆ g»
        using assms η-in-hom iso-η by (intro hcomp-in-vhom) auto
```

```
      thus ?thesis
        using assms f-in-hhom hcomp-obj-arr by (metis in-hhomE)
    qed
    moreover have f ⋆ (g ⋆ f) ⋆ g = f ⋆ g ⋆ f ⋆ g
      using hcomp-assoc by simp
    ultimately show ?thesis by simp
  qed
  show «inv ξ ⋆ f ⋆ g : f ⋆ g ⇒ f ⋆ g ⋆ f ⋆ g»
  proof −
    have «inv ξ ⋆ f ⋆ g : ?b ⋆ f ⋆ g ⇒ (f ⋆ g) ⋆ f ⋆ g»
      using assms ξ-in-hom iso-ξ by (intro hcomp-in-vhom, auto)
    thus ?thesis
      using hcomp-assoc f-in-hhom g-in-hhom b hcomp-obj-arr [of ?b f ⋆ g]
      by fastforce
  qed
  show «ξ : f ⋆ g ⇒ ?b»
    using ξ-in-hom by blast
qed
have iso ?ε
 using f-in-hhom g-in-hhom η-in-hhom ide-f ide-g η-in-hom iso-η ξ-in-hhom ξ-in-hom iso-ξ
       iso-inv-iso isos-compose
  by (metis ε-in-hom arrI hseqE ide-is-iso iso-hcomp seqE)
have 4: «inv ξ ⋆ f : ?b ⋆ f ⇒ f ⋆ g ⋆ f»
proof −
  have «inv ξ ⋆ f : ?b ⋆ f ⇒ (f ⋆ g) ⋆ f»
    using ξ-in-hom iso-ξ f-in-hhom
    by (intro hcomp-in-vhom, auto)
  thus ?thesis
    using hcomp-assoc by simp
qed
```

First show *?ε* and η satisfy the "left" triangle identity.

```
have triangle-left: (?ε ⋆ f) · (f ⋆ η) = f
proof −
  have (?ε ⋆ f) · (f ⋆ η) = (ξ ⋆ f) · (f ⋆ inv η ⋆ g ⋆ f) · (inv ξ ⋆ f ⋆ g ⋆ f) · (?b ⋆ f ⋆ η)
  proof −
    have f ⋆ η = ?b ⋆ f ⋆ η
      using b η-in-hhom hcomp-obj-arr [of ?b f ⋆ η] by fastforce
    moreover have ξ · (f ⋆ inv η ⋆ g) · (inv ξ ⋆ f ⋆ g) ⋆ f =
                  (ξ ⋆ f) · ((f ⋆ inv η ⋆ g) ⋆ f) · ((inv ξ ⋆ f ⋆ g) ⋆ f)
      using ide-f ide-g ξ-in-hhom ξ-in-hom iso-ξ η-in-hhom η-in-hom iso-η whisker-right
      by (metis ε-in-hom arrI seqE)
    moreover have ... = (ξ ⋆ f) · (f ⋆ inv η ⋆ g ⋆ f) · (inv ξ ⋆ f ⋆ g ⋆ f)
      using hcomp-assoc by simp
    ultimately show ?thesis
      using comp-assoc by simp
  qed
  also have ... = (ξ ⋆ f) · ((f ⋆ inv η ⋆ g ⋆ f) · (f ⋆ g ⋆ f ⋆ η)) · (inv ξ ⋆ f)
  proof −
```

have $(inv\ \xi \star f \star g \star f) \cdot (?b \star f \star \eta) = ((inv\ \xi \star f) \star (g \star f)) \cdot ((?b \star f) \star \eta)$
  using *hcomp-assoc* **by** *simp*
also have ... $= (inv\ \xi \star f) \cdot (?b \star f) \star (g \star f) \cdot \eta$
proof −
  have *seq* $(inv\ \xi \star f)\ (?b \star f)$
    using *a b 4 ide-f ide-g ξ-in-hhom ξ-in-hom iso-ξ* **by** *blast*
  moreover have *seq* $(g \star f)\ \eta$
    using *f-in-hhom g-in-hhom ide-g ide-f η-in-hom* **by** *fast*
  ultimately show *?thesis*
    using *interchange* [*of inv* $\xi \star f$ *?b* $\star f\ g \star f\ \eta$] **by** *simp*
qed
also have ... $= inv\ \xi \star f \star \eta$
  using *4 comp-arr-dom comp-cod-arr η-in-hom hcomp-assoc* **by** (*metis in-homE*)
also have ... $= (f \star g) \cdot inv\ \xi \star (f \star \eta) \cdot (f \star ?a)$
proof −
  have $(f \star g) \cdot inv\ \xi = inv\ \xi$
    using *ξ-in-hom iso-ξ comp-cod-arr* **by** *auto*
  moreover have $(f \star \eta) \cdot (f \star ?a) = f \star \eta$
  proof −
    have «$f \star \eta : f \star ?a \Rightarrow f \star g \star f$»
      using *η-in-hom* **by** *fastforce*
    thus *?thesis*
      using *comp-arr-dom* **by** *blast*
  qed
  ultimately show *?thesis* **by** *argo*
qed
also have ... $= ((f \star g) \star (f \star \eta)) \cdot (inv\ \xi \star (f \star ?a))$
proof −
  have *seq* $(f \star g)\ (inv\ \xi)$
    using *ξ-in-hom iso-ξ comp-cod-arr* **by** *auto*
  moreover have *seq* $(f \star \eta)\ (f \star ?a)$
    using *f-in-hhom η-in-hom* **by** *force*
  ultimately show *?thesis*
    using *interchange* **by** *simp*
qed
also have ... $= (f \star g \star f \star \eta) \cdot (inv\ \xi \star f)$
  using *hcomp-arr-obj hcomp-assoc* **by** *auto*
finally have $(inv\ \xi \star f \star g \star f) \cdot (?b \star f \star \eta) = (f \star g \star f \star \eta) \cdot (inv\ \xi \star f)$
  by *simp*
thus *?thesis*
  using *comp-assoc* **by** *simp*
qed
also have ... $= (\xi \star f) \cdot ((f \star ?a \star \eta) \cdot (f \star inv\ \eta \star ?a)) \cdot (inv\ \xi \star f)$
proof −
  have $(f \star inv\ \eta \star g \star f) \cdot (f \star (g \star f) \star \eta) = f \star (inv\ \eta \star g \star f) \cdot ((g \star f) \star \eta)$
  proof −
    have $(f \star (inv\ \eta \star g) \star f) \cdot (f \star (g \star f) \star \eta) = f \star ((inv\ \eta \star g) \star f) \cdot ((g \star f) \star \eta)$
    proof −
      have *seq* $((inv\ \eta \star g) \star f)\ ((g \star f) \star \eta)$

```
      proof −
        have seq (inv η ⋆ g ⋆ f) ((g ⋆ f) ⋆ η)
          using f-in-hhom ide-f g-in-hhom ide-g η-in-hhom η-in-hom iso-η
          apply (intro seqI hseqI′)
                  apply auto
            by fastforce+
        thus ?thesis
          using hcomp-assoc by simp
      qed
      thus ?thesis
        using whisker-left by simp
    qed
    thus ?thesis
      using hcomp-assoc by simp
  qed
  also have ... = f ⋆ (?a ⋆ η) · (inv η ⋆ ?a)
  proof −
    have (inv η ⋆ g ⋆ f) · ((g ⋆ f) ⋆ η) = (?a ⋆ η) · (inv η ⋆ ?a)
    proof −
      have (inv η ⋆ g ⋆ f) · ((g ⋆ f) ⋆ η) = inv η · (g ⋆ f) ⋆ (g ⋆ f) · η
        using g-in-hhom ide-g η-in-hom iso-η
              interchange [of inv η g ⋆ f g ⋆ f η]
        by force
      also have ... = inv η ⋆ η
        using η-in-hom iso-η comp-arr-dom comp-cod-arr by auto
      also have ... = ?a · inv η ⋆ η · ?a
        using η-in-hom iso-η comp-arr-dom comp-cod-arr by auto
      also have ... = (?a ⋆ η) · (inv η ⋆ ?a)
        using a η-in-hom iso-η interchange [of ?a inv η η ?a] by blast
      finally show ?thesis by simp
    qed
    thus ?thesis by argo
  qed
  also have ... = (f ⋆ ?a ⋆ η) · (f ⋆ inv η ⋆ ?a)
  proof −
    have seq (?a ⋆ η) (inv η ⋆ ?a)
    proof (intro seqI′)
      show «inv η ⋆ ?a : (g ⋆ f) ⋆ ?a ⇒ ?a ⋆ ?a»
        using a g-in-hhom η-in-hom iso-η hseqI ide-f ide-g
        by (intro hcomp-in-vhom) auto
      show «?a ⋆ η : ?a ⋆ ?a ⇒ ?a ⋆ g ⋆ f»
        using a η-in-hom hseqI by (intro hcomp-in-vhom) auto
    qed
    thus ?thesis
      using whisker-left by simp
  qed
  finally show ?thesis
    using hcomp-assoc by simp
qed
```

```
    also have ... = (ξ ⋆ f) · ((f ⋆ η) · (f ⋆ inv η)) · (inv ξ ⋆ f)
      using a η-in-hhom iso-η hcomp-obj-arr [of ?a η] hcomp-arr-obj [of inv η ?a] by auto
    also have ... = (ξ ⋆ f) · (inv ξ ⋆ f)
    proof −
      have ((f ⋆ η) · (f ⋆ inv η)) · (inv ξ ⋆ f) = (f ⋆ η · inv η) · (inv ξ ⋆ f)
        using η-in-hhom iso-η whisker-left inv-in-hom by auto
      also have ... = (f ⋆ g ⋆ f) · (inv ξ ⋆ f)
        using η-in-hom iso-η comp-arr-inv inv-is-inverse by auto
      also have ... = inv ξ ⋆ f
        using 4 comp-cod-arr by blast
      ultimately show ?thesis by simp
    qed
    also have ... = f
    proof −
      have (ξ ⋆ f) · (inv ξ ⋆ f) = ξ · inv ξ ⋆ f
        using ξ-in-hhom iso-ξ whisker-right by auto
      also have ... = ?b ⋆ f
        using ξ-in-hom iso-ξ comp-arr-inv' by auto
      also have ... = f
        using hcomp-obj-arr by auto
      finally show ?thesis by blast
    qed
    finally show ?thesis by blast
  qed


  interpret E: equivalence-in-strict-bicategory V H a i src trg f g η ?ε
    using ide-g η-in-hom ε-in-hom g-in-hhom ‹iso η› ‹iso ?ε›
    by (unfold-locales, auto)
```

Apply "triangle left if and only iff right" to show the "right" triangle identity.

```
  have triangle-right: ((g ⋆ ξ · (f ⋆ inv η ⋆ g) · (inv ξ ⋆ f ⋆ g)) · (η ⋆ g) = g)
    using triangle-left E.triangle-left-iff-right by simp
```

Use the two triangle identities to establish an adjoint equivalence and show that there is only one choice for the counit.

```
  show ∃!ε. adjoint-equivalence-in-bicategory V H a i src trg f g η ε
  proof −
    have adjoint-equivalence-in-bicategory V H a i src trg f g η ?ε
    proof
      show (?ε ⋆ f) · a⁻¹[f, g, f] · (f ⋆ η) = l⁻¹[f] · r[f]
      proof −
        have (?ε ⋆ f) · a⁻¹[f, g, f] · (f ⋆ η) = (?ε ⋆ f) · (f ⋆ η)
        proof −
          have seq a⁻¹[f, g, f] (f ⋆ η)
            using E.antipar
            by (intro seqI, auto)
          hence a⁻¹[f, g, f] · (f ⋆ η) = f ⋆ η
            using ide-f ide-g E.antipar triangle-right strict-assoc' comp-ide-arr
```

```
          by presburger
        thus ?thesis by simp
      qed
      also have ... = f
        using triangle-left by simp
      also have ... = l⁻¹[f] · r[f]
        using strict-lunit strict-runit by simp
      finally show ?thesis by simp
    qed
    show (g ⋆ ?ε) · a[g, f, g] · (η ⋆ g) = r⁻¹[g] · l[g]
    proof −
      have (g ⋆ ?ε) · a[g, f, g] · (η ⋆ g) = (g ⋆ ?ε) · (η ⋆ g)
      proof −
        have seq a[g, f, g] (η ⋆ g)
          using E.antipar
          by (intro seqI, auto)
        hence a[g, f, g] · (η ⋆ g) = η ⋆ g
          using ide-f ide-g E.antipar triangle-right strict-assoc comp-ide-arr
          by presburger
        thus ?thesis by simp
      qed
      also have ... = g
        using triangle-right by simp
      also have ... = r⁻¹[g] · l[g]
        using strict-lunit strict-runit by simp
      finally show ?thesis by blast
    qed
  qed
  moreover have ⋀ε ε'. ⟦ adjoint-equivalence-in-bicategory (·) (⋆) a i src trg f g η ε;
                         adjoint-equivalence-in-bicategory (·) (⋆) a i src trg f g η ε' ⟧
                           ⟹ ε = ε'
    using adjunction-unit-determines-counit
    by (meson adjoint-equivalence-in-bicategory.axioms(2))
  ultimately show ?thesis by auto
 qed
qed

end
```

We now apply strictification to generalize the preceding result to an arbitrary bicategory.

```
context bicategory
begin

  interpretation S: strictified-bicategory V H a i src trg ..

  notation S.vcomp (infixr ‹·S› 55)
  notation S.hcomp (infixr ‹⋆S› 53)
  notation S.in-hom (‹«- : - ⇒S -»›)
```

notation *S.in-hhom* (‹«- : - $\rightarrow_S$ -»›)

interpretation *UP*: *fully-faithful-functor V S.vcomp S.UP*
  using *S.UP-is-fully-faithful-functor* **by** *auto*
interpretation *UP*: *equivalence-pseudofunctor V H* a i *src trg*
  *S.vcomp S.hcomp S.*a *S.*i *S.src S.trg S.UP S.cmp$_{UP}$*
  using *S.UP-is-equivalence-pseudofunctor* **by** *auto*
interpretation *UP*: *pseudofunctor-into-strict-bicategory V H* a i *src trg*
  *S.vcomp S.hcomp S.*a *S.*i *S.src S.trg S.UP S.cmp$_{UP}$*
  ..

lemma *equivalence-refines-to-adjoint-equivalence*:
assumes *equivalence-map f* **and** «*g* : *trg f* $\rightarrow$ *src f*» **and** *ide g*
and «η : *src f* $\Rightarrow$ *g* $\star$ *f*» **and** *iso* η
shows $\exists!\varepsilon$. *adjoint-equivalence-in-bicategory V H* a i *src trg f g* η ε
proof −

To unpack the consequences of the assumptions, we need to obtain an interpretation of *equivalence-in-bicategory*, even though we don't need the associated data other than *f*, *a*, and *b*.

  obtain g' φ ψ **where** *E*: *equivalence-in-bicategory V H* a i *src trg f* g' φ ψ
    using *assms equivalence-map-def* **by** *auto*
  interpret *E*: *equivalence-in-bicategory V H* a i *src trg f* g' φ ψ
    using *E* **by** *auto*
  let *?a* = *src f* **and** *?b* = *trg f*
  have *ide-f*: *ide f* **by** *simp*
  have *f-in-hhom*: «*f* : *?a* $\rightarrow$ *?b*» **by** *simp*
  have *a*: *obj ?a* **and** *b*: *obj ?b* **by** *auto*
  have *1*: *S.equivalence-map* (*S.UP f*)
    using *assms UP.preserves-equivalence-maps* **by** *simp*
  let $?\eta'$ = *S.inv* (*S.cmp$_{UP}$* (*g*, *f*)) $\cdot_S$ *S.UP* η $\cdot_S$ *UP.unit ?a*
  have *2*: «*S.UP* η : *S.UP ?a* $\Rightarrow_S$ *S.UP* (*g* $\star$ *f*)»
    using *assms UP.preserves-hom* [*of* η *src f g* $\star$ *f*] **by** *auto*
  have *3*: «$?\eta'$: *UP.map*$_0$ *?a* $\Rightarrow_S$ *S.UP g* $\star_S$ *S.UP f*» $\wedge$ *S.iso* $?\eta'$
  proof (*intro S.comp-in-homI conjI*)
    show «*S.inv* (*S.cmp$_{UP}$* (*g*, *f*)) : *S.UP* (*g* $\star$ *f*) $\Rightarrow_S$ *S.UP g* $\star_S$ *S.UP f*»
      using *assms UP.cmp-in-hom*(*2*) **by** *auto*
    moreover show «*UP.unit ?a* : *UP.map*$_0$ *?a* $\Rightarrow_S$ *S.UP ?a*» **by** *auto*
    moreover show «*S.UP* η : *S.UP ?a* $\Rightarrow_S$ *S.UP* (*g* $\star$ *f*)»
      using *2* **by** *simp*
    ultimately show *S.iso* (*S.inv* (*S.cmp$_{UP}$* (*g*, *f*)) $\cdot_S$ *S.UP* η $\cdot_S$ *UP.unit ?a*)
      using *assms UP.cmp-components-are-iso UP.unit-char*(*2*)
      by (*intro S.isos-compose*) *auto*
  qed
  have *ex-un-*ξ': $\exists!\xi'$. *adjoint-equivalence-in-bicategory S.vcomp S.hcomp S.*a *S.*i *S.src S.trg*
    (*S.UP f*) (*S.UP g*) $?\eta'$ ξ'
  proof −
    have «*S.UP g* : *S.trg* (*S.UP f*) $\rightarrow_S$ *S.src* (*S.UP f*)»
      using *assms*(*2*) **by** *auto*

moreover have $S.ide\ (S.UP\ g)$
by (*simp add*: *assms*(*3*))
ultimately show *?thesis*
using *1 3 S.equivalence-refines-to-adjoint-equivalence S.UP-map*$_0$*-obj* **by** *simp*
qed
obtain ξ' **where** ξ': *adjoint-equivalence-in-bicategory S.vcomp S.hcomp S*.a *S*.i *S.src S.trg*
$(S.UP\ f)\ (S.UP\ g)\ ?\eta'\ \xi'$
using *ex-un-*ξ' **by** *auto*
interpret E': *adjoint-equivalence-in-bicategory S.vcomp S.hcomp S*.a *S*.i *S.src S.trg*
$\langle S.UP\ f\rangle\ \langle S.UP\ g\rangle\ ?\eta'\ \xi'$
using ξ' **by** *auto*
let $?\varepsilon' = UP.unit\ ?b \cdot_S \xi' \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ g))$
have ε': «$?\varepsilon' : S.UP\ (f \star g) \Rightarrow_S S.UP\ ?b$»
using *assms*(*2*−*3*) **by** *auto*
have *ex-un-*ε: $\exists!\varepsilon.$ «$\varepsilon : f \star g \Rightarrow ?b$» $\wedge$ $S.UP\ \varepsilon = ?\varepsilon'$
proof −
have $\exists\varepsilon.$ «$\varepsilon : f \star g \Rightarrow ?b$» $\wedge$ $S.UP\ \varepsilon = ?\varepsilon'$
proof −
have $src\ (f \star g) = src\ ?b \wedge trg\ (f \star g) = trg\ ?b$
using *assms*(*2*) *f-in-hhom* **by** *auto*
moreover have $ide\ (f \star g)$
using *assms*(*2*−*3*) **by** *auto*
ultimately show *?thesis*
using ε' *UP.locally-full* **by** *auto*
qed
moreover have
$\bigwedge\mu\ \nu.\ [\![$ «$\mu : f \star g \Rightarrow ?b$»; $S.UP\ \mu = ?\varepsilon'$; «$\nu : f \star g \Rightarrow ?b$»; $S.UP\ \nu = ?\varepsilon'$ $]\!]$
$\Longrightarrow \mu = \nu$
proof −
fix $\mu\ \nu$
assume μ: «$\mu : f \star g \Rightarrow ?b$» **and** ν: «$\nu : f \star g \Rightarrow ?b$»
and *1*: $S.UP\ \mu = ?\varepsilon'$ **and** *2*: $S.UP\ \nu = ?\varepsilon'$
have $par\ \mu\ \nu$
using $\mu\ \nu$ **by** *fastforce*
thus $\mu = \nu$
using *1 2 UP.is-faithful* [*of* $\mu\ \nu$] **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed
have *iso-*ε': $S.iso\ ?\varepsilon'$
proof (*intro S.isos-compose*)
show $S.iso\ (S.inv\ (S.cmp_{UP}\ (f,\ g)))$
using *assms UP.cmp-components-are-iso* **by** *auto*
show $S.iso\ \xi'$
using E'*.counit-is-iso* **by** *blast*
show $S.iso\ (UP.unit\ ?b)$
using *b UP.unit-char*(*2*) **by** *simp*
show $S.seq\ (UP.unit\ ?b)\ (\xi' \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ g)))$
proof (*intro S.seqI*$'$)

show «$UP.unit\ ?b : UP.map_0\ ?b \Rightarrow_S S.UP\ ?b$»
using b $UP.unit\text{-}char$ **by** $simp$
show «$\xi' \cdot_S S.inv\ (S.cmp_{UP}\ (f, g)) : S.UP\ (f \star g) \Rightarrow_S UP.map_0\ ?b$»
using $assms$ **by** $auto$
qed
thus $S.seq\ \xi'\ (S.inv\ (S.cmp_{UP}\ (f, g)))$ **by** $auto$
qed
obtain ε **where** ε: «$\varepsilon : f \star g \Rightarrow ?b$» $\wedge$ $S.UP\ \varepsilon = ?\varepsilon'$
using $ex\text{-}un\text{-}\varepsilon$ **by** $auto$
interpret E'': $equivalence\text{-}in\text{-}bicategory\ V\ H\ \mathrm{a}\ \mathrm{i}\ src\ trg\ f\ g\ \eta\ \varepsilon$
using $assms(1,3-5)$
apply $unfold\text{-}locales$
apply $simp\text{-}all$
using $assms(2)\ \varepsilon$
apply $auto[1]$
using ε $iso\text{-}\varepsilon'$ $UP.reflects\text{-}iso$ $[of\ \varepsilon]$
by $auto$
interpret E'': $adjoint\text{-}equivalence\text{-}in\text{-}bicategory\ V\ H\ \mathrm{a}\ \mathrm{i}\ src\ trg\ f\ g\ \eta\ \varepsilon$
proof
show $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
proof −
have $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta)) =$
$S.cmp_{UP}\ (trg\ f, f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f, src\ f))$
using $E''.UP\text{-}triangle(3)$ **by** $simp$
also have ... $= S.cmp_{UP}\ (trg\ f, f) \cdot_S$
$(UP.unit\ ?b \cdot_S \xi' \cdot_S S.inv\ (S.cmp_{UP}\ (f, g)) \cdot_S S.cmp_{UP}\ (f, g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f, src\ f))$
using ε $S.comp\text{-}assoc$ **by** $simp$
also have ... $= S.cmp_{UP}\ (trg\ f, f) \cdot_S (UP.unit\ ?b \cdot_S \xi' \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (f, src\ f))$
proof −
have $\xi' \cdot_S S.inv\ (S.cmp_{UP}\ (f, g)) \cdot_S S.cmp_{UP}\ (f, g) = \xi'$
proof −
have $S.iso\ (S.cmp_{UP}\ (f, g))$
using $assms$ **by** $auto$
moreover have $S.dom\ (S.cmp_{UP}\ (f, g)) = S.UP\ f \star_S S.UP\ g$
using $assms$ **by** $auto$
ultimately have $S.inv\ (S.cmp_{UP}\ (f, g)) \cdot_S S.cmp_{UP}\ (f, g) = S.UP\ f \star_S S.UP\ g$
using $S.comp\text{-}inv\text{-}arr'$ **by** $simp$
thus $?thesis$
using $S.comp\text{-}arr\text{-}dom$ $E'.counit\text{-}in\text{-}hom(2)$ **by** $simp$
qed
thus $?thesis$ **by** $argo$
qed
also have

$... = S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ ?b \star_S S.UP\ f) \cdot_S$
$((\xi' \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S$
$(S.UP\ f \star_S S.UP\ \eta)) \cdot_S$
$S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$

proof −
have $UP.unit\ ?b \cdot_S \xi' \star_S S.UP\ f = (UP.unit\ ?b \star_S S.UP\ f) \cdot_S (\xi' \star_S S.UP\ f)$
using *assms b UP.unit-char S.whisker-right* $S.UP\text{-}map_0\text{-}obj$ **by** *auto*
moreover have $S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta =$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.UP\ f \star_S S.UP\ \eta)$
using *assms S.whisker-left* **by** *auto*
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have $... = (S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ ?b \star_S S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
proof −
have $(\xi' \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.UP\ f \star_S S.UP\ \eta)$
$= (S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f)))$
proof −
have $(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.UP\ f \star_S S.UP\ \eta) =$
$S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta$
using *assms S.whisker-left* **by** *auto*
hence $((\xi' \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f))) \cdot_S (S.UP\ f \star_S S.UP\ \eta))$
$= ((\xi' \star_S S.UP\ f) \cdot_S (S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta))$
by *simp*
also have $... = ((\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta)$
proof −
have $(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) = \xi' \star_S S.UP\ f$
proof −
have «$\xi' \star_S S.UP\ f$:
$(S.UP\ f \star_S S.UP\ g) \star_S S.UP\ f \Rightarrow_S S.trg\ (S.UP\ f) \star_S S.UP\ f$»
using *assms* **by** *auto*
moreover have «$S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f)$:
$S.UP\ f \star_S S.UP\ g \star_S S.UP\ f \Rightarrow_S (S.UP\ f \star_S S.UP\ g) \star_S S.UP\ f$»
using *assms S.assoc′-in-hom* **by** *auto*
ultimately show *?thesis*
using *assms S.strict-assoc′ S.iso-assoc S.hcomp-assoc E′.antipar*
S.comp-arr-ide S.seqI′
by (*metis* (*no-types*, *lifting*) *E′.ide-left E′.ide-right*)
qed
thus *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have $... = ((\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta))$
using *S.comp-assoc* **by** *simp*
also have $... = (S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f)))$

proof −
have $(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) =$
$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f)))$
proof −
have $(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S$
$(S.UP\ f \star_S UP.unit\ ?a) =$
$S.lunit'\ (S.UP\ f) \cdot_S S.runit\ (S.UP\ f)$
proof −
have $(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S$
$(S.UP\ f \star_S UP.unit\ ?a) =$
$(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ ?a)$
proof −
have $S.seq\ (S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)\ (UP.unit\ ?a)$
using *assms UP.unit-char UP.cmp-components-are-iso*
E′.unit-simps(1) S.comp-assoc
by *presburger*
hence $(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) \cdot_S$
$(S.UP\ f \star_S UP.unit\ ?a) =$
$S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta \cdot_S UP.unit\ ?a$
using *assms UP.unit-char UP.cmp-components-are-iso S.comp-assoc*
S.whisker-left [*of* $S.UP\ f\ S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta\ UP.unit\ ?a$]
by *simp*
thus *?thesis* **by** *simp*
qed
thus *?thesis*
using *assms E′.triangle-left UP.cmp-components-are-iso UP.unit-char*
by *argo*
qed
also have ... $= S.UP\ f$
using *S.strict-lunit′ S.strict-runit* **by** *simp*
finally have *1*: $((\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta)) \cdot_S$
$(S.UP\ f \star_S UP.unit\ ?a) = S.UP\ f$
using *S.comp-assoc* **by** *simp*
have $(\xi' \star_S S.UP\ f) \cdot_S S.\mathrm{a}'\ (S.UP\ f)\ (S.UP\ g)\ (S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g, f)) \cdot_S S.UP\ \eta) =$
$S.UP\ f \cdot_S (S.UP\ f \star_S S.inv\ (UP.unit\ ?a))$
proof −
have $S.arr\ (S.UP\ f)$
using *assms* **by** *simp*
moreover have $S.iso\ (S.UP\ f \star_S UP.unit\ ?a)$
using *assms UP.unit-char* $S.UP\text{-}map_0\text{-}obj$ **by** *auto*
moreover have $S.inv\ (S.UP\ f \star_S UP.unit\ ?a) =$
$S.UP\ f \star_S S.inv\ (UP.unit\ ?a)$
using *assms a UP.unit-char* $S.UP\text{-}map_0\text{-}obj$ **by** *auto*

```
        ultimately show ?thesis
          using assms 1 UP.unit-char UP.cmp-components-are-iso
                S.invert-side-of-triangle(2)
                 [of S.UP f (ξ' ⋆S S.UP f) ·S S.a' (S.UP f) (S.UP g) (S.UP f) ·S
                            (S.UP f ⋆S S.inv (S.cmpUP (g, f)) ·S S.UP η)
                     S.UP f ⋆S UP.unit ?a]
          by presburger
      qed
      also have ... = S.UP f ⋆S S.inv (UP.unit ?a)
      proof −
        have «S.UP f ⋆S S.inv (UP.unit ?a) :
                S.UP f ⋆S S.UP ?a ⇒S S.UP f ⋆S UP.map0 ?a»
          using assms ide-f f-in-hhom UP.unit-char [of ?a] S.inv-in-hom
          apply (intro S.hcomp-in-vhom)
            apply auto[1]
           apply blast
          by auto
        moreover have S.UP f ⋆S UP.map0 ?a = S.UP f
          using a S.hcomp-arr-obj S.UP-map0-obj by auto
        finally show ?thesis
          using S.comp-cod-arr by blast
      qed
      finally show ?thesis by auto
    qed
    thus ?thesis
      using S.comp-assoc by simp
  qed
  finally show ?thesis by simp
 qed
 thus ?thesis
   using S.comp-assoc by simp
qed
also have ... = S.UP l⁻¹[f] ·S S.UP r[f]
proof −
  have S.cmpUP (trg f, f) ·S (UP.unit ?b ⋆S S.UP f) = S.UP l⁻¹[f]
  proof −
    have S.UP f = S.UP l[f] ·S S.cmpUP (trg f, f) ·S (UP.unit (trg f) ⋆S S.UP f)
      using UP.lunit-coherence iso-lunit S.strict-lunit by simp
    thus ?thesis
      using UP.image-of-unitor(3) ide-f by presburger
  qed
  moreover have (S.UP f ⋆S S.inv (UP.unit (src f))) ·S S.inv (S.cmpUP (f, src f))
                = S.UP r[f]
  proof −
    have S.UP r[f] ·S S.cmpUP (f, src f) ·S (S.UP f ⋆S UP.unit (src f)) = S.UP f
      using UP.runit-coherence [of f] S.strict-runit by simp
    moreover have S.iso (S.cmpUP (f, src f) ·S (S.UP f ⋆S UP.unit (src f)))
      using UP.unit-char UP.cmp-components-are-iso VV.arr-charSbC S.UP-map0-obj
      by (intro S.isos-compose) auto
```

ultimately have
$S.UP\ \mathrm{r}[f] = S.UP\ f \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f) \cdot_S (S.UP\ f \star_S UP.unit\ (src\ f)))$
using *S.invert-side-of-triangle*(*2*)
[*of* $S.UP\ f\ S.UP\ \mathrm{r}[f]\ S.cmp_{UP}\ (f,\ src\ f) \cdot_S (S.UP\ f \star_S UP.unit\ (src\ f))$]
ideD(*1*) *ide-f* **by** *blast*
thus *?thesis*
using *ide-f UP.image-of-unitor*(*2*) [*of f*] **by** *argo*
qed
ultimately show *?thesis*
using *S.comp-assoc* **by** *simp*
qed
also have ... $= S.UP\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f])$
by *simp*
finally have $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) = S.UP\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f])$
by *simp*
moreover have $par\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta))\ (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f])$
proof −
have «$(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) : f \star src\ f \Rightarrow trg\ f \star f$»
using *E''.triangle-in-hom*(*1*) **by** *simp*
moreover have «$\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] : f \star src\ f \Rightarrow trg\ f \star f$» **by** *auto*
ultimately show *?thesis*
by (*metis in-homE*)
qed
ultimately show *?thesis*
using *UP.is-faithful* **by** *blast*
qed
thus $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
using *E''.triangle-left-implies-right* **by** *simp*
qed
show *?thesis*
using *E''.adjoint-equivalence-in-bicategory-axioms E''.adjunction-in-bicategory-axioms*
adjunction-unit-determines-counit adjoint-equivalence-in-bicategory-def
by *metis*
qed

lemma *equivalence-map-extends-to-adjoint-equivalence*:
assumes *equivalence-map f*
shows $\exists\, g\ \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
proof −
obtain $g\ \eta\ \varepsilon'$ **where** *E*: *equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon'$
using *assms equivalence-map-def* **by** *auto*
interpret *E*: *equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon'$
using *E* **by** *auto*
obtain ε **where** *A*: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
using *assms equivalence-refines-to-adjoint-equivalence* [*of f g* η]
E.antipar E.unit-is-iso E.unit-in-hom **by** *auto*
show *?thesis*
using *E A* **by** *blast*
qed

end

1.13.7 Uniqueness of Adjoints

Left and right adjoints determine each other up to isomorphism.

context *strict-bicategory*
begin

lemma *left-adjoint-determines-right-up-to-iso*:
assumes *adjoint-pair f g* **and** *adjoint-pair f g′*
shows $g \cong g'$
proof −
obtain η ε **where** *A*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
using *assms adjoint-pair-def* **by** *auto*
interpret *A*: *adjunction-in-bicategory V H* a i *src trg f g* η ε
using *A* **by** *auto*
interpret *A*: *adjunction-in-strict-bicategory V H* a i *src trg f g* η ε **..**
obtain η' ε' **where** *A′*: *adjunction-in-bicategory V H* a i *src trg f g′* η' ε'
using *assms adjoint-pair-def* **by** *auto*
interpret *A′*: *adjunction-in-bicategory V H* a i *src trg f g′* η' ε'
using *A′* **by** *auto*
interpret *A′*: *adjunction-in-strict-bicategory V H* a i *src trg f g′* η' ε' **..**
let $?\varphi = A'.trnl_\eta\ g\ \varepsilon$
have «$?\varphi$: $g \Rightarrow g'$»
using $A'.trnl_\eta$*-eq A′.adjoint-transpose-left*(*1*) [*of trg f g*] *A.antipar A′.antipar hcomp-arr-obj*
by *auto*
moreover have *iso* $?\varphi$
proof (*intro isoI*)
let $?\psi = A.trnl_\eta\ g'\ \varepsilon'$
show *inverse-arrows* $?\varphi$ $?\psi$
proof
show *ide* $(?\varphi \cdot ?\psi)$
proof −
have *1*: *ide* (*trg f*) ∧ *trg* (*trg f*) = *trg f*
by *simp*
have $?\varphi \cdot ?\psi = (g' \star \varepsilon) \cdot ((\eta' \star g) \cdot (g \star \varepsilon')) \cdot (\eta \star g')$
using *1 A.antipar A′.antipar* $A.trnl_\eta$*-eq* [*of trg f g′* ε'] $A'.trnl_\eta$*-eq* [*of trg f g* ε] *comp-assoc A.counit-in-hom A′.counit-in-hom*
by *simp*
also have ... $= ((g' \star \varepsilon) \cdot (g' \star f \star g \star \varepsilon')) \cdot ((\eta' \star g \star f \star g') \cdot (\eta \star g'))$
proof −
have $(\eta' \star g) \cdot (g \star \varepsilon') = (\eta' \star g \star trg\ f) \cdot (src\ f \star g \star \varepsilon')$
using *A.antipar A′.antipar hcomp-arr-obj hcomp-obj-arr* [*of src f g* $\star$ ε'] *hseqI′*
by (*metis A′.counit-simps*(*1*) *A′.counit-simps*(*5*) *A.ide-right ideD*(*1*) *obj-trg trg-hcomp*)
also have ... $= \eta' \star g \star \varepsilon'$

```
    using A.antipar A'.antipar interchange [of η' src f g ⋆ trg f g ⋆ ε']
          whisker-left comp-arr-dom comp-cod-arr
    by simp
  also have ... = ((g' ⋆ f) ⋆ g ⋆ ε') · (η' ⋆ g ⋆ (f ⋆ g'))
    using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar
          A'.unit-in-hom A'.counit-in-hom interchange whisker-left
          comp-arr-dom comp-cod-arr
    by (metis A'.counit-simps(1−2,5) A'.unit-simps(1,3) hseqI' ide-char)
  also have ... = (g' ⋆ f ⋆ g ⋆ ε') · (η' ⋆ g ⋆ f ⋆ g')
    using hcomp-assoc by simp
  finally show ?thesis
    using comp-assoc by simp
qed
also have ... = (g' ⋆ ε') · ((g' ⋆ (ε ⋆ f) ⋆ g') · (g' ⋆ (f ⋆ η) ⋆ g')) · (η' ⋆ g')
proof −
  have (g' ⋆ ε) · (g' ⋆ f ⋆ g ⋆ ε') = (g' ⋆ ε') · (g' ⋆ ε ⋆ f ⋆ g')
  proof −
    have (g' ⋆ ε) · (g' ⋆ f ⋆ g ⋆ ε') = g' ⋆ ε ⋆ ε'
    proof −
      have ε · (f ⋆ g ⋆ ε') = ε ⋆ ε'
        using A.ide-left A.ide-right A.antipar A'.antipar hcomp-arr-obj comp-arr-dom
          comp-cod-arr interchange obj-src trg-src
        by (metis A'.counit-simps(1,3) A.counit-simps(1−2,4) hcomp-assoc)
      thus ?thesis
        using A.antipar A'.antipar whisker-left [of g' ε f ⋆ g ⋆ ε']
        by (simp add: hcomp-assoc)
    qed
    also have ... = (g' ⋆ ε') · (g' ⋆ ε ⋆ f ⋆ g')
    proof −
      have ε ⋆ ε' = ε' · (ε ⋆ f ⋆ g')
        using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar
              hcomp-obj-arr hcomp-arr-obj comp-arr-dom comp-cod-arr interchange
              obj-src trg-src
        by (metis A'.counit-simps(1−2,5) A.counit-simps(1,3−4) arr-cod
            not-arr-null seq-if-composable)
      thus ?thesis
        using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar
              whisker-left
        by (metis A'.counit-simps(1,5) A.counit-simps(1,4) hseqI')
    qed
    finally show ?thesis by simp
  qed
  moreover have (η' ⋆ g ⋆ f ⋆ g') · (η ⋆ g') = (g' ⋆ f ⋆ η ⋆ g') · (η' ⋆ g')
  proof −
    have (η' ⋆ g ⋆ f ⋆ g') · (η ⋆ g') = η' ⋆ η ⋆ g'
    proof −
      have (η' ⋆ g ⋆ f) · η = η' ⋆ η
        using A.ide-left A.ide-right A.antipar A'.antipar A'.unit-in-hom hcomp-arr-obj
              interchange comp-arr-dom comp-cod-arr
```

```
        by (metis A'.unit-simps(1−2,4) A.unit-simps(1,3,5) hcomp-obj-arr obj-trg)
      thus ?thesis
        using A.antipar A'.antipar whisker-right [of g' η' ⋆ g ⋆ f η]
        by (simp add: hcomp-assoc)
    qed
    also have ... = (g' ⋆ f ⋆ η ⋆ g') · (η' ⋆ g')
    proof −
      have η' ⋆ η = (g' ⋆ f ⋆ η) · η'
        using A.ide-left A.ide-right A.antipar A'.antipar A'.unit-in-hom hcomp-arr-obj
              comp-arr-dom comp-cod-arr hcomp-assoc interchange
        by (metis A'.unit-simps(1,3−4) A.unit-simps(1−2) obj-src)
      thus ?thesis
        using A.ide-left A.ide-right A.antipar A'.antipar A'.unit-in-hom hcomp-arr-obj
              whisker-right [of g' g' ⋆ f ⋆ η η']
        by (metis A'.ide-right A'.unit-simps(1,4) A.unit-simps(1,5)
            hseqI' hcomp-assoc)
    qed
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using comp-assoc hcomp-assoc by simp
qed
also have ... = (g' ⋆ ε') · ((g' ⋆ f) ⋆ g') · (η' ⋆ g')
proof −
  have (g' ⋆ (ε ⋆ f) ⋆ g') · (g' ⋆ (f ⋆ η) ⋆ g') = g' ⋆ f ⋆ g'
  proof −
    have (g' ⋆ (ε ⋆ f) ⋆ g') · (g' ⋆ (f ⋆ η) ⋆ g') =
          g' ⋆ ((ε ⋆ f) ⋆ g') · ((f ⋆ η) ⋆ g')
      using A.ide-left A.ide-right A.antipar A'.antipar A'.unit-in-hom
            A'.counit-in-hom whisker-left [of g' (ε ⋆ f) ⋆ g' (f ⋆ η) ⋆ g']
      by (metis A'.ide-right A.triangle-left hseqI' ideD(1) whisker-right)
    also have ... = g' ⋆ (ε ⋆ f) · (f ⋆ η) ⋆ g'
      using A.antipar A'.antipar whisker-right [of g' ε ⋆ f f ⋆ η]
      by (simp add: A.triangle-left)
    also have ... = g' ⋆ f ⋆ g'
      using A.triangle-left by simp
    finally show ?thesis by simp
  qed
  thus ?thesis
    using hcomp-assoc by simp
qed
also have ... = (g' ⋆ ε') · (η' ⋆ g')
  using A.antipar A'.antipar A'.unit-in-hom A'.counit-in-hom comp-cod-arr
  by (metis A'.ide-right A'.triangle-in-hom(2) A.ide-left arrI assoc-naturality2
      ide-char seqE strict-assoc)
also have ... = g'
  using A'.triangle-right by simp
finally have ?φ · ?ψ = g' by simp
thus ?thesis by simp
```

```
qed
show ide (?ψ · ?φ)
proof −
  have 1: ide (trg f) ∧ trg (trg f) = trg f
    by simp
  have ?ψ · ?φ = (g ⋆ ε′) · ((η ⋆ g′) · (g′ ⋆ ε)) · (η′ ⋆ g)
    using A.antipar A′.antipar A′.trnl_η-eq [of trg f g ε]
          A.trnl_η-eq [of trg f g′ ε′] comp-assoc A.counit-in-hom A′.counit-in-hom
    by simp
  also have ... = ((g ⋆ ε′) · (g ⋆ f ⋆ g′ ⋆ ε)) · ((η ⋆ g′ ⋆ f ⋆ g) · (η′ ⋆ g))
  proof −
    have (η ⋆ g′) · (g′ ⋆ ε) = (η ⋆ g′ ⋆ trg f) · (src f ⋆ g′ ⋆ ε)
      using A.antipar A′.antipar hcomp-arr-obj hcomp-obj-arr hseqI′
      by (metis A′.ide-right A.unit-simps(1,4) hcomp-assoc hcomp-obj-arr
          ideD(1) obj-src)
    also have ... = η ⋆ g′ ⋆ ε
      using A.ide-left A.ide-right A′.ide-right A.antipar A′.antipar A.unit-in-hom
            A.counit-in-hom interchange
      by (metis 1 A.counit-simps(5) A.unit-simps(4) hseqI′ ide-def ide-in-hom(2)
          not-arr-null seqI′ src.preserves-ide)
    also have ... = ((g ⋆ f) ⋆ g′ ⋆ ε) · (η ⋆ g′ ⋆ (f ⋆ g))
    using A′.ide-right A′.antipar interchange ide-char comp-arr-dom comp-cod-arr hseqI′
      by (metis A.counit-simps(1−2,5) A.unit-simps(1,3))
    also have ... = (g ⋆ f ⋆ g′ ⋆ ε) · (η ⋆ g′ ⋆ f ⋆ g)
      using hcomp-assoc by simp
    finally show ?thesis
      using comp-assoc by simp
  qed
  also have ... = (g ⋆ ε) · ((g ⋆ (ε′ ⋆ f) ⋆ g) · (g ⋆ (f ⋆ η′) ⋆ g)) · (η ⋆ g)
  proof −
    have (g ⋆ ε′) · (g ⋆ f ⋆ g′ ⋆ ε) = (g ⋆ ε) · (g ⋆ ε′ ⋆ f ⋆ g)
    proof −
      have (g ⋆ ε′) · (g ⋆ f ⋆ g′ ⋆ ε) = g ⋆ ε′ ⋆ ε
      proof −
        have ε′ · (f ⋆ g′ ⋆ ε) = ε′ ⋆ ε
          using A.ide-left A.ide-right A′.ide-right A.antipar A′.antipar hcomp-arr-obj
                comp-arr-dom comp-cod-arr interchange obj-src trg-src hcomp-assoc
          by (metis A.counit-simps(1,3) A′.counit-simps(1−2,4))
        thus ?thesis
          using A.antipar A′.antipar whisker-left [of g ε′ f ⋆ g′ ⋆ ε]
          by (simp add: hcomp-assoc)
      qed
      also have ... = (g ⋆ ε) · (g ⋆ ε′ ⋆ f ⋆ g)
      proof −
        have ε′ ⋆ ε = ε · (ε′ ⋆ f ⋆ g)
          using A.ide-left A.ide-right A′.ide-right A.antipar A′.antipar hcomp-obj-arr
                hcomp-arr-obj comp-arr-dom comp-cod-arr interchange obj-src trg-src
          by (metis A.counit-simps(1−2,5) A′.counit-simps(1,3−4)
              arr-cod not-arr-null seq-if-composable)
```

```
      thus ?thesis
        using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar
              whisker-left
        by (metis A.counit-simps(1,5) A'.counit-simps(1,4) hseqI')
    qed
    finally show ?thesis by simp
  qed
  moreover have (η ⋆ g' ⋆ f ⋆ g) · (η' ⋆ g) = (g ⋆ f ⋆ η' ⋆ g) · (η ⋆ g)
  proof −
    have (η ⋆ g' ⋆ f ⋆ g) · (η' ⋆ g) = η ⋆ η' ⋆ g
    proof −
      have (η ⋆ g' ⋆ f) · η' = η ⋆ η'
        using A.antipar A'.antipar A.unit-in-hom hcomp-arr-obj
              comp-arr-dom comp-cod-arr hcomp-obj-arr interchange
        by (metis A'.unit-simps(1,3,5) A.unit-simps(1−2,4) obj-trg)
      thus ?thesis
        using A.antipar A'.antipar whisker-right [of g η ⋆ g' ⋆ f η']
        by (simp add: hcomp-assoc)
    qed
    also have ... = ((g ⋆ f) ⋆ η' ⋆ g) · (η ⋆ src f ⋆ g)
      using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar A.unit-in-hom
            A'.unit-in-hom comp-arr-dom comp-cod-arr interchange
      by (metis A'.unit-simps(1−2,4) A.unit-simps(1,3) hseqI' ide-char)
    also have ... = (g ⋆ f ⋆ η' ⋆ g) · (η ⋆ g)
      using A.antipar A'.antipar hcomp-assoc
      by (simp add: hcomp-obj-arr)
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using comp-assoc hcomp-assoc by simp
qed
also have ... = (g ⋆ ε) · ((g ⋆ f) ⋆ g) · (η ⋆ g)
proof −
  have (g ⋆ (ε' ⋆ f) ⋆ g) · (g ⋆ (f ⋆ η') ⋆ g) = g ⋆ f ⋆ g
  proof −
    have (g ⋆ (ε' ⋆ f) ⋆ g) · (g ⋆ (f ⋆ η') ⋆ g) =
          g ⋆ ((ε' ⋆ f) ⋆ g) · ((f ⋆ η') ⋆ g)
      using A.ide-left A.ide-right A'.ide-right A.antipar A'.antipar A.unit-in-hom
            A.counit-in-hom whisker-left
      by (metis A'.triangle-left hseqI' ideD(1) whisker-right)
    also have ... = g ⋆ (ε' ⋆ f) · (f ⋆ η') ⋆ g
      using A.antipar A'.antipar whisker-right [of g ε' ⋆ f f ⋆ η']
      by (simp add: A'.triangle-left)
    also have ... = g ⋆ f ⋆ g
      using A'.triangle-left by simp
    finally show ?thesis by simp
  qed
  thus ?thesis
    using hcomp-assoc by simp
```

```
        qed
        also have ... = (g ⋆ ε) · (η ⋆ g)
          using A.antipar A'.antipar A.unit-in-hom A.counit-in-hom comp-cod-arr
          by (metis A.ide-left A.ide-right A.triangle-in-hom(2) arrI assoc-naturality2
              ide-char seqE strict-assoc)
        also have ... = g
          using A.triangle-right by simp
        finally have ?ψ · ?φ = g by simp
        moreover have ide g
          by simp
        ultimately show ?thesis by simp
      qed
    qed
  qed
  ultimately show ?thesis
    using isomorphic-def by auto
qed

end
```

We now use strictification to extend to arbitrary bicategories.

```
context bicategory
begin

  interpretation S: strictified-bicategory V H a i src trg ..

  notation S.vcomp (infixr ‹·S› 55)
  notation S.hcomp (infixr ‹⋆S› 53)
  notation S.in-hom (‹«- : - ⇒S -»›)
  notation S.in-hhom (‹«- : - →S -»›)

  interpretation UP: equivalence-pseudofunctor V H a i src trg
                     S.vcomp S.hcomp S.a S.i S.src S.trg S.UP S.cmpUP
    using S.UP-is-equivalence-pseudofunctor by auto
  interpretation UP: pseudofunctor-into-strict-bicategory V H a i src trg
                     S.vcomp S.hcomp S.a S.i S.src S.trg S.UP S.cmpUP
    ..
  interpretation UP: fully-faithful-functor V S.vcomp S.UP
    using S.UP-is-fully-faithful-functor by auto

  lemma left-adjoint-determines-right-up-to-iso:
  assumes adjoint-pair f g and adjoint-pair f g'
  shows g ≅ g'
  proof −
    have 0: ide g ∧ ide g'
      using assms adjoint-pair-def adjunction-in-bicategory-def
            adjunction-data-in-bicategory-def adjunction-data-in-bicategory-axioms-def
      by metis
    have 1: S.adjoint-pair (S.UP f) (S.UP g) ∧ S.adjoint-pair (S.UP f) (S.UP g')
```

```
    using assms UP.preserves-adjoint-pair by simp
  obtain ν where ν: «ν : S.UP g ⇒S S.UP g'» ∧ S.iso ν
    using 1 S.left-adjoint-determines-right-up-to-iso S.isomorphic-def by blast
  obtain μ where μ: «μ : g ⇒ g'» ∧ S.UP μ = ν
    using 0 ν UP.is-full [of g' g ν] by auto
  have «μ : g ⇒ g'» ∧ iso μ
    using μ ν UP.reflects-iso by auto
  thus ?thesis
    using isomorphic-def by auto
qed

lemma right-adjoint-determines-left-up-to-iso:
assumes adjoint-pair f g and adjoint-pair f' g
shows f ≅ f'
proof −
  obtain η ε where A: adjunction-in-bicategory V H a i src trg f g η ε
    using assms adjoint-pair-def by auto
  interpret A: adjunction-in-bicategory V H a i src trg f g η ε
    using A by auto
  obtain η' ε' where A': adjunction-in-bicategory V H a i src trg f' g η' ε'
    using assms adjoint-pair-def by auto
  interpret A': adjunction-in-bicategory V H a i src trg f' g η' ε'
    using A' by auto
  interpret Cop: op-bicategory V H a i src trg ..
  interpret Aop: adjunction-in-bicategory V Cop.H Cop.a i Cop.src Cop.trg g f η ε
    using A.antipar A.triangle-left A.triangle-right Cop.assoc-ide-simp
          Cop.lunit-ide-simp Cop.runit-ide-simp
    by (unfold-locales, auto)
  interpret Aop': adjunction-in-bicategory V Cop.H Cop.a i Cop.src Cop.trg g f' η' ε'
    using A'.antipar A'.triangle-left A'.triangle-right Cop.assoc-ide-simp
          Cop.lunit-ide-simp Cop.runit-ide-simp
    by (unfold-locales, auto)
  show ?thesis
    using Aop.adjunction-in-bicategory-axioms Aop'.adjunction-in-bicategory-axioms
          Cop.left-adjoint-determines-right-up-to-iso Cop.adjoint-pair-def
    by blast
qed

end

context chosen-right-adjoints
begin

lemma isomorphic-to-left-adjoint-implies-isomorphic-right-adjoint:
assumes is-left-adjoint f and f ≅ h
shows f* ≅ h*
proof −
  have 1: adjoint-pair f f*
    using assms left-adjoint-extends-to-adjoint-pair by blast
```

moreover have *adjoint-pair h f**
using *assms 1 adjoint-pair-preserved-by-iso isomorphic-symmetric isomorphic-reflexive*
by (*meson isomorphic-def right-adjoint-simps*(*1*))
thus *?thesis*
using *left-adjoint-determines-right-up-to-iso left-adjoint-extends-to-adjoint-pair*
by *blast*
qed

end

context *bicategory*
begin

lemma *equivalence-is-adjoint*:
assumes *equivalence-map f*
shows *equivalence-is-left-adjoint*: *is-left-adjoint f*
and *equivalence-is-right-adjoint*: *is-right-adjoint f*
proof −
obtain $g\ \eta\ \varepsilon$ **where** *fg*: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
using *assms equivalence-map-extends-to-adjoint-equivalence* **by** *blast*
interpret *fg*: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* $\eta\ \varepsilon$
using *fg* **by** *simp*
interpret *gf*: *adjoint-equivalence-in-bicategory V H* a i *src trg g f* ‹*inv* ε› ‹*inv* η›
using *fg.dual-adjoint-equivalence* **by** *simp*
show *is-left-adjoint f*
using *fg.adjunction-in-bicategory-axioms adjoint-pair-def* **by** *auto*
show *is-right-adjoint f*
using *gf.adjunction-in-bicategory-axioms adjoint-pair-def* **by** *auto*
qed

lemma *right-adjoint-to-equivalence-is-equivalence*:
assumes *equivalence-map f* **and** *adjoint-pair f g*
shows *equivalence-map g*
proof −
obtain $\eta\ \varepsilon$ **where** *fg*: *adjunction-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f g* $\eta\ \varepsilon$
using *assms adjoint-pair-def* **by** *auto*
interpret *fg*: *adjunction-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg f g* $\eta\ \varepsilon$
using *fg* **by** *simp*
obtain $g'\ \varphi\ \psi$ **where** *fg'*: *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f* $g'\ \varphi\ \psi$
using *assms equivalence-map-def* **by** *auto*
interpret *fg'*: *equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg f* $g'\ \varphi\ \psi$
using *fg'* **by** *auto*
obtain ψ' **where** ψ': *adjoint-equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f* $g'\ \varphi\ \psi'$
using *assms equivalence-refines-to-adjoint-equivalence* [*of f* $g'\ \varphi$]
fg'.antipar fg'.unit-in-hom fg'.unit-is-iso
by *auto*
interpret ψ': *adjoint-equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg f* $g'\ \varphi\ \psi'$
using ψ' **by** *simp*
have *1*: $g \cong g'$

using *fg.adjunction-in-bicategory-axioms* ψ'*.adjunction-in-bicategory-axioms*
left-adjoint-determines-right-up-to-iso adjoint-pair-def
by *blast*
obtain γ **where** γ: «$\gamma : g' \Rightarrow g$» $\wedge$ *iso* γ
using *1 isomorphic-def isomorphic-symmetric* **by** *metis*
have *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg f g* $((\gamma \star f) \cdot \varphi)$ $(\psi' \cdot (f \star inv\ \gamma))$
using γ *equivalence-preserved-by-iso-right* ψ'*.equivalence-in-bicategory-axioms* **by** *simp*
hence *quasi-inverses f g*
using *quasi-inverses-def* **by** *blast*
thus *?thesis*
using *equivalence-mapI quasi-inverses-symmetric* **by** *blast*
qed

lemma *left-adjoint-to-equivalence-is-equivalence*:
assumes *equivalence-map f* **and** *adjoint-pair g f*
shows *equivalence-map g*
proof −
obtain η ε **where** *gf*: *adjunction-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg g f* η ε
using *assms adjoint-pair-def* **by** *auto*
interpret *gf*: *adjunction-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg g f* η ε
using *gf* **by** *simp*
obtain g' **where** *1*: *quasi-inverses* g' *f*
using *assms equivalence-mapE quasi-inverses-symmetric* **by** *blast*
obtain φ ψ **where** *g'f*: *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* g' *f* φ ψ
using *assms 1 quasi-inverses-def* **by** *auto*
interpret *g'f*: *equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg* g' *f* φ ψ
using *g'f* **by** *auto*
obtain ψ' **where** ψ': *adjoint-equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* g' *f* φ ψ'
using *assms 1 equivalence-refines-to-adjoint-equivalence* [*of* g' *f* φ]
g'f.antipar g'f.unit-in-hom g'f.unit-is-iso quasi-inverses-def
equivalence-map-def
by *auto*
interpret ψ': *adjoint-equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg* g' *f* φ ψ'
using ψ' **by** *simp*
have *1*: $g \cong g'$
using *gf.adjunction-in-bicategory-axioms* ψ'*.adjunction-in-bicategory-axioms*
right-adjoint-determines-left-up-to-iso adjoint-pair-def
by *blast*
obtain γ **where** γ: «$\gamma : g' \Rightarrow g$» $\wedge$ *iso* γ
using *1 isomorphic-def isomorphic-symmetric* **by** *metis*
have *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg g f* $((f \star \gamma) \cdot \varphi)$ $(\psi' \cdot (inv\ \gamma \star f))$
using γ *equivalence-preserved-by-iso-left* ψ'*.equivalence-in-bicategory-axioms* **by** *simp*
hence *quasi-inverses g f*
using *quasi-inverses-def* **by** *auto*
thus *?thesis*
using *quasi-inverses-symmetric quasi-inverses-def equivalence-map-def* **by** *blast*
qed

lemma *quasi-inverses-are-adjoint-pair*:

assumes *quasi-inverses f g*
shows *adjoint-pair f g*
proof −
obtain η ε **where** $\eta\varepsilon$: *equivalence-in-bicategory V H* a i *src trg f g* η ε
using *assms quasi-inverses-def* **by** *auto*
interpret $\eta\varepsilon$: *equivalence-in-bicategory V H* a i *src trg f g* η ε
using $\eta\varepsilon$ **by** *auto*
obtain ε' **where** $\eta\varepsilon'$: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* η ε'
using $\eta\varepsilon$ *equivalence-map-def* $\eta\varepsilon$*.antipar* $\eta\varepsilon$*.unit-in-hom* $\eta\varepsilon$*.unit-is-iso*
$\eta\varepsilon$*.ide-right equivalence-refines-to-adjoint-equivalence* [*of f g* η]
by *force*
interpret $\eta\varepsilon'$: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* η ε'
using $\eta\varepsilon'$ **by** *auto*
show *?thesis*
using $\eta\varepsilon'$ *adjoint-pair-def* $\eta\varepsilon'$*.adjunction-in-bicategory-axioms* **by** *auto*
qed

lemma *quasi-inverses-isomorphic-right*:
assumes *quasi-inverses f g*
shows *quasi-inverses f* $g' \longleftrightarrow g \cong g'$
proof
show $g \cong g' \Longrightarrow$ *quasi-inverses f* g'
using *assms quasi-inverses-def isomorphic-def equivalence-preserved-by-iso-right*
by *metis*
assume g': *quasi-inverses f* g'
show $g \cong g'$
using *assms* g' *quasi-inverses-are-adjoint-pair left-adjoint-determines-right-up-to-iso*
by *blast*
qed

lemma *quasi-inverses-isomorphic-left*:
assumes *quasi-inverses f g*
shows *quasi-inverses* f' $g \longleftrightarrow f \cong f'$
proof
show $f \cong f' \Longrightarrow$ *quasi-inverses* f' g
using *assms quasi-inverses-def isomorphic-def equivalence-preserved-by-iso-left*
by *metis*
assume f': *quasi-inverses* f' g
show $f \cong f'$
using *assms* f' *quasi-inverses-are-adjoint-pair right-adjoint-determines-left-up-to-iso*
by *blast*
qed

end

end

1.14 Pseudonatural Transformations

theory *PseudonaturalTransformation*
imports *InternalEquivalence InternalAdjunction Pseudofunctor*
begin

1.14.1 Definition of Pseudonatural Transformation

Pseudonatural transformations are a generalization of natural transformations that is appropriate for pseudofunctors. The "components" of a pseudonatural transformation τ from a pseudofunctor F to a pseudofunctor G (both from bicategory C to D), are 1-cells «$\tau_0\ a : F_0\ a \rightarrow_D G_0\ a$» associated with the objects of C. Instead of "naturality squares" that commute on-the-nose, a pseudonatural transformation associates an invertible 2-cell «$\tau_1\ f : G\ f \star_D \tau_0\ a \Rightarrow_D \tau_0\ a' \star_D F\ f$» with each 1-cell «$f : a \rightarrow_C a'$» of C. The 1-cells $\tau_0\ a$ and 2-cells $\tau_1\ f$ are subject to coherence conditions which express that they transform naturally across 2-cells of C and behave sensibly with respect to objects and horizontal composition.

In formalizing ordinary natural transformations, we found it convenient to treat them similarly to functors in that a natural transformation $\tau : C \rightarrow D$ maps arrows of C to arrows of D; the components $\tau\ a$ at objects a being merely special cases. However, it is not possible to take the same approach for pseudofunctors, because it is not possible to treat the components $\tau_0\ a$ at objects a as a special case of the components $\tau_1\ f$ at 1-cells f. So we have to regard a pseudonatural transformation τ as consisting of two separate mappings: a mapping τ_0 from objects of C to 1-cells of D and a mapping τ_1 from 1-cells of C to invertible 2-cells of D.

Pseudonatural transformations are themselves a special case of the more general notion of "lax natural transformations" between pseudofunctors. For a lax natural transformation τ, the 2-cells $\tau_1\ f$ are not required to be invertible. This means that there is a distinction between a lax natural transformation with «$\tau_1\ f : G\ f \star_D \tau_0\ a \Rightarrow_D \tau_0\ a' \star_D F\ f$» and an "op-lax" natural transformation with «$\tau_1\ f : \tau_0\ a' \star_D F\ f \Rightarrow_D G\ f \star_D \tau_0\ a$». There is some variation in the literature on which direction is considered "lax" and which is "op-lax" and this variation extends as well to the special case of pseudofunctors, though in that case it does not result in any essential difference in the notion, due to the assumed invertibility. We have chosen the direction that agrees with Leinster [8], and with the "nLab" article [11] on lax natural transformations, but note that the "nLab" article [10] on pseudonatural transformations seems to make the opposite choice.

locale *pseudonatural-transformation* =
  C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
  D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
  F: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$ +
  G: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
for V_C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *'c comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: *'c* ⇒ *'c* ⇒ *'c* ⇒ *'c* (‹a_C[-, -, -]›)
and i_C :: *'c* ⇒ *'c* (‹i_C[-]›)
and src_C :: *'c* ⇒ *'c*

and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and τ_0 :: $'c \Rightarrow 'd$
and τ_1 :: $'c \Rightarrow 'd$ +
assumes map_0*-in-hhom*: $C.obj\ a \Longrightarrow$ «$\tau_0\ a : src_D\ (F\ a) \rightarrow_D src_D\ (G\ a)$»
and map_1*-in-vhom*: $C.ide\ f \Longrightarrow$ «$\tau_1\ f : G\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ f$»
and *ide-map*$_0$*-obj*: $C.obj\ a \Longrightarrow D.ide\ (\tau_0\ a)$
and *iso-map*$_1$*-ide*: $C.ide\ f \Longrightarrow D.iso\ (\tau_1\ f)$
and *naturality*: $C.arr\ \mu \Longrightarrow$
$\tau_1\ (C.cod\ \mu) \cdot_D (G\ \mu \star_D \tau_0\ (src_C\ \mu)) = (\tau_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D \tau_1\ (C.dom\ \mu)$
and *respects-unit*: $C.obj\ a \Longrightarrow$
$(\tau_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{l}_D[\tau_0\ a] = \tau_1\ a \cdot_D (G.unit\ a \star_D \tau_0\ a)$
and *respects-hcomp*:
$[\![\ C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f\ ]\!] \Longrightarrow$
$(\tau_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D (\tau_1\ g \star_D F\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[G\ g,\ \tau_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f) \cdot_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ (src_C\ f)]$
$= \tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ (src_C\ f))$
begin

lemma map_0*-in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\tau_0\ a : F.map_0\ a \rightarrow_D G.map_0\ a$»
and «$\tau_0\ a : \tau_0\ a \Rightarrow_D \tau_0\ a$»
using *assms* map_0*-in-hhom* [*of a*]
apply *fastforce*
using *assms* *ide-map*$_0$*-obj* [*of a*] **by** *fast*

lemma map_0*-simps* [*simp*]:
assumes $C.obj\ a$
shows $D.ide\ (\tau_0\ a)$
and $src_D\ (\tau_0\ a) = F.map_0\ a$ **and** $trg_D\ (\tau_0\ a) = G.map_0\ a$
using *assms* map_0*-in-hom* *iso-map*$_1$*-ide* *ide-map*$_0$*-obj* **by** *auto*

lemma map_1*-in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\tau_1\ f : F.map_0\ (src_C\ f) \rightarrow_D G.map_0\ (trg_C\ f)$»
and «$\tau_1\ f : G\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ f$»
using *assms* map_1*-in-vhom* [*of f*] *D.vconn-implies-hpar*(*1*−*2*) **by** *auto*

lemma map_1*-simps* [*simp*]:

assumes $C.ide\ f$
shows $D.arr\ (\tau_1\ f)$
and $src_D\ (\tau_1\ f) = F.map_0\ (src_C\ f)$
and $trg_D\ (\tau_1\ f) = G.map_0\ (trg_C\ f)$
and $D.dom\ (\tau_1\ f) = G\ f \star_D \tau_0\ (src_C\ f)$
and $D.cod\ (\tau_1\ f) = \tau_0\ (trg_C\ f) \star_D F\ f$
using *assms map*$_1$*-in-hom iso-map*$_1$*-ide* **by** *auto*

lemma *inv-naturality*:
assumes $C.arr\ \mu$
shows $(G\ \mu \star_D \tau_0\ (src_C\ \mu)) \cdot_D D.inv\ (\tau_1\ (C.dom\ \mu)) =$
$D.inv\ (\tau_1\ (C.cod\ \mu)) \cdot_D (\tau_0\ (trg_C\ \mu) \star_D F\ \mu)$
using *assms naturality iso-map*$_1$*-ide D.invert-opposite-sides-of-square* **by** *simp*

end

1.14.2 Identity Pseudonatural Transformation

locale *identity-pseudonatural-transformation* =
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
begin

abbreviation (*input*) map_0
where $map_0\ a \equiv F.map_0\ a$

abbreviation (*input*) map_1
where $map_1\ f \equiv \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]$

sublocale *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F\ map_0\ map_1$
proof
show *1*: $\bigwedge a.\ C.obj\ a \Longrightarrow D.ide\ (F.map_0\ a)$
using $F.map_0$*-simps*(*1*) **by** *blast*

show $\bigwedge f.\ C.ide\ f \Longrightarrow D.iso\ (map_1\ f)$
  by *auto*
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$F.map_0\ a : src_D\ (F\ a) \rightarrow_D src_D\ (F\ a)$»
  by (*metis C.obj-def′ D.in-hhomI D.src.preserves-arr* $F.map_0$*-def* $F.map_0$*-simps*(*2*)
    $F.map_0$*-simps*(*3*) *F.preserves-reflects-arr*)
show $\bigwedge f.\ C.ide\ f \Longrightarrow$ «$map_1\ f : F\ f \star_D F.map_0\ (src_C\ f) \Rightarrow_D F.map_0\ (trg_C\ f) \star_D F\ f$»
  by *simp*
show $\bigwedge \mu.\ C.arr\ \mu \Longrightarrow$
    $(map_1\ (C.cod\ \mu)) \cdot_D (F\ \mu \star_D F.map_0\ (src_C\ \mu))$
      $= (F.map_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D map_1\ (C.dom\ \mu)$
  by (*metis D.comp-assoc D.lunit′-naturality D.runit-naturality F.preserves-arr*
    F.preserves-cod F.preserves-dom F.preserves-src F.preserves-trg)
show $\bigwedge f\ g.\ [\![C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f]\!] \Longrightarrow$
    $(F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
    $\mathrm{a}_D[F.map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
    $(\mathrm{l}_D{}^{-1}[F\ g] \cdot_D \mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D$
    $D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D$
    $(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]) \cdot_D$
    $\mathrm{a}_D[F\ g,\ F\ f,\ F.map_0\ (src_C\ f)]$
      $= (\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \cdot_D \mathrm{r}_D[F\ (g \star_C f)]) \cdot_D (\Phi_F\ (g, f) \star_D F.map_0\ (src_C\ f))$
proof −
  fix *f g*
  assume *f*: $C.ide\ f$ **and** *g*: $C.ide\ g$ **and** *fg*: $src_C\ g = trg_C\ f$
  have $(F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
      $\mathrm{a}_D[F.map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
      $(\mathrm{l}_D{}^{-1}[F\ g] \cdot_D \mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D$
      $D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D$
      $(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]) \cdot_D$
      $\mathrm{a}_D[F\ g,\ F\ f,\ F.map_0\ (src_C\ f)]$
        $= (F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
          $(\mathrm{a}_D[F.map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
          $(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f)) \cdot_D$
          $(\mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D$
          $(D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D$
          $(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f])) \cdot_D$
          $(F\ g \star_D \mathrm{r}_D[F\ f]) \cdot_D$
          $\mathrm{a}_D[F\ g,\ F\ f,\ F.map_0\ (src_C\ f)]$
    using *f g fg D.whisker-right* [*of F f*] *D.whisker-left* [*of F g*] *D.comp-assoc*
    by *simp*
  also have ... $= (F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
      $\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \cdot_D$
      $((\mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D$
      $(\mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f)) \cdot_D$
      $\mathrm{r}_D[F\ g \star_D F\ f]$
  proof −
    have $D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D (F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f]) = \mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f$
    proof −
      have $D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D (F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f]) =$
        $D.inv\ ((F\ g \star_D \mathrm{l}_D[F\ f]) \cdot_D \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f])$

using *f g fg 1 D.inv-comp* **by** *auto*
also have ... $= D.inv\ (\mathrm{r}_D[F\ g] \star_D F\ f)$
using *f g fg D.triangle* [*of F f F g*] **by** *simp*
also have ... $= \mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f$
using *f g fg* **by** *simp*
finally show *?thesis* **by** *blast*
qed
moreover have $(F\ g \star_D \mathrm{r}_D[F\ f]) \cdot_D \mathrm{a}_D[F\ g,\ F\ f,\ F.map_0\ (src_C\ f)] = \mathrm{r}_D[F\ g \star_D F\ f]$
using *f g fg D.runit-hcomp*(*1*) **by** *simp*
moreover have $\mathrm{a}_D[F.map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D (\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f)$
$= \mathrm{l}_D{}^{-1}[F\ g \star_D F\ f]$
using *f g fg D.lunit-hcomp*(*4*) **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ g \star_D F\ f]) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f]$
proof −
have $((\mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D (\mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f)) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f] = \mathrm{r}_D[F\ g \star_D F\ f]$
proof −
have $((\mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D (\mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f)) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f] =$
$(\mathrm{r}_D[F\ g] \cdot_D \mathrm{r}_D{}^{-1}[F\ g] \star_D F\ f) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f]$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= (F\ g \star_D F\ f) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f]$
using *f g fg D.comp-arr-inv′* **by** *simp*
also have ... $= \mathrm{r}_D[F\ g \star_D F\ f]$
using *f g fg D.comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \cdot_D \Phi_F\ (g, f) \cdot_D \mathrm{r}_D[F\ g \star_D F\ f]$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *F.FF-def*
D.lunit′-naturality [*of* $\Phi_F\ (g, f)$] *D.comp-assoc*
by *simp*
also have ... $= (\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \cdot_D \mathrm{r}_D[F\ (g \star_C f)]) \cdot_D (\Phi_F\ (g, f) \star_D F.map_0\ (src_C\ f))$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *F.FF-def*
D.runit-naturality [*of* $\Phi_F\ (g, f)$] *D.comp-assoc*
by *simp*
finally show $(F.map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[F.map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \cdot_D \mathrm{r}_D[F\ g] \star_D F\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ F.map_0\ (src_C\ g),\ F\ f] \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ F.map_0\ (src_C\ f)]$
$= (\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \cdot_D \mathrm{r}_D[F\ (g \star_C f)]) \cdot_D (\Phi_F\ (g, f) \star_D F.map_0\ (src_C\ f))$
by *simp*
qed
show $\bigwedge a.\ C.obj\ a \Longrightarrow$

$(F.map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[F.map_0\ a] \cdot_D \mathrm{l}_D[F.map_0\ a] =$
$map_1\ a \cdot_D (F.unit\ a \star_D F.map_0\ a)$

proof −
fix a
assume a: $C.obj\ a$
have $(F.map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[F.map_0\ a] \cdot_D \mathrm{l}_D[F.map_0\ a] =$
$(F.map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{l}_D{}^{-1}[F.map_0\ a] \cdot_D \mathrm{r}_D[F.map_0\ a]$
using *a 1 D.unitor-coincidence* **by** *simp*
also have ... $= ((F.map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{l}_D{}^{-1}[F.map_0\ a]) \cdot_D \mathrm{r}_D[F.map_0\ a]$
using *D.comp-assoc* **by** *simp*
also have ... $= \mathrm{l}_D{}^{-1}[F\ a] \cdot_D F.unit\ a \cdot_D \mathrm{r}_D[F.map_0\ a]$
using *a 1 D.lunit′-naturality* [*of F.unit a*] *D.comp-assoc* **by** *simp*
also have ... $= map_1\ a \cdot_D (F.unit\ a \star_D F.map_0\ a)$
using *a 1 D.runit-naturality* [*of F.unit a*] *D.comp-assoc* **by** *simp*
finally show $(F.map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[F.map_0\ a] \cdot_D \mathrm{l}_D[F.map_0\ a] =$
$map_1\ a \cdot_D (F.unit\ a \star_D F.map_0\ a)$
by *blast*
qed
qed

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F\ map_0\ map_1$
..

end

1.14.3 Composite Pseudonatural Transformation

A pseudonatural transformation σ from F to G and a pseudonatural transformation ϱ from G to H can be composed to obtain a pseudonatural transformation τ from F to H. The components at objects are composed via horizontal composition. Composing the components at 1-cells is straightforward, but is formally complicated by the need for associativities. The additional complexity leads to somewhat lengthy proofs of the coherence conditions. This issue only gets worse as we consider additional constructions on pseudonatural transformations.

locale *composite-pseudonatural-transformation* =
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$ +
G: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$ +
H: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ H\ \Phi_H$ +
σ: *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \sigma_0\ \sigma_1$ +
ϱ: *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ H\ \Phi_H\ \varrho_0\ \varrho_1$
for V_C :: *′c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *′c comp* (**infixr** ‹$\star_C$› *53*)

and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and H :: $'c \Rightarrow 'd$
and Φ_H :: $'c * 'c \Rightarrow 'd$
and σ_0 :: $'c \Rightarrow 'd$
and σ_1 :: $'c \Rightarrow 'd$
and ϱ_0 :: $'c \Rightarrow 'd$
and ϱ_1 :: $'c \Rightarrow 'd$
begin

definition map_0
where $map_0\ a = \varrho_0\ a \star_D \sigma_0\ a$

definition map_1
where $map_1\ f = \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ f) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]$

sublocale *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ H\ \Phi_H\ map_0\ map_1$
proof
show $\bigwedge a.\ C.obj\ a \Longrightarrow D.ide\ (map_0\ a)$
unfolding map_0*-def* **by** *fastforce*
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$map_0\ a : src_D\ (F\ a) \rightarrow_D src_D\ (H\ a)$»
unfolding map_0*-def*
using $\varrho.map_0$*-in-hhom* $\sigma.map_0$*-in-hhom* **by** *blast*
show $\bigwedge f.\ C.ide\ f \Longrightarrow$ «$map_1\ f : H\ f \star_D map_0\ (src_C\ f) \Rightarrow_D map_0\ (trg_C\ f) \star_D F\ f$»
by (*unfold* map_1*-def* map_0*-def*, *intro D.comp-in-homI*) *auto*
show $\bigwedge f.\ C.ide\ f \Longrightarrow D.iso\ (map_1\ f)$
unfolding map_1*-def*
using $\varrho.ide$*-*map_0*-obj* $\varrho.iso$*-*map_1*-ide* $\sigma.ide$*-*map_0*-obj* $\sigma.iso$*-*map_1*-ide*
by (*intro D.isos-compose*) *auto*
show $\bigwedge \mu.\ C.arr\ \mu \Longrightarrow map_1\ (C.cod\ \mu) \cdot_D (H\ \mu \star_D map_0\ (src_C\ \mu))$
$= (map_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D map_1\ (C.dom\ \mu)$
proof −

fix μ
assume μ: $C.arr\ \mu$
have $map_1\ (C.cod\ \mu) \cdot_D (H\ \mu \star_D map_0\ (src_C\ \mu))$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu), \sigma_0\ (trg_C\ \mu), F\ (C.cod\ \mu)] \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ \mu), G\ (C.cod\ \mu), \sigma_0\ (src_C\ \mu)] \cdot_D$
$(\varrho_1\ (C.cod\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.cod\ \mu), \varrho_0\ (src_C\ \mu), \sigma_0\ (src_C\ \mu)] \cdot_D$
$(H\ \mu \star_D \varrho_0\ (src_C\ \mu) \star_D \sigma_0\ (src_C\ \mu))$
unfolding *map*$_1$*-def map*$_0$*-def*
using μ *D.comp-assoc* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu), \sigma_0\ (trg_C\ \mu), F\ (C.cod\ \mu)] \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ \mu), G\ (C.cod\ \mu), \sigma_0\ (src_C\ \mu)] \cdot_D$
$((\varrho_1\ (C.cod\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$((H\ \mu \star_D \varrho_0\ (src_C\ \mu)) \star_D \sigma_0\ (src_C\ \mu))) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.dom\ \mu), \varrho_0\ (src_C\ \mu), \sigma_0\ (src_C\ \mu)]$
using μ *D.assoc′-naturality* [*of* $H\ \mu\ \varrho_0\ (src_C\ \mu)\ \sigma_0\ (src_C\ \mu)$] *D.comp-assoc* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu), \sigma_0\ (trg_C\ \mu), F\ (C.cod\ \mu)] \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ \mu), G\ (C.cod\ \mu), \sigma_0\ (src_C\ \mu)] \cdot_D$
$((\varrho_0\ (trg_C\ \mu) \star_D G\ \mu) \star_D \sigma_0\ (src_C\ \mu))) \cdot_D$
$(\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.dom\ \mu), \varrho_0\ (src_C\ \mu), \sigma_0\ (src_C\ \mu)]$
proof −
have $(\varrho_1\ (C.cod\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D ((H\ \mu \star_D \varrho_0\ (src_C\ \mu)) \star_D \sigma_0\ (src_C\ \mu)) =$
$\varrho_1\ (C.cod\ \mu) \cdot_D (H\ \mu \star_D \varrho_0\ (src_C\ \mu)) \star_D \sigma_0\ (src_C\ \mu)$
using μ *D.whisker-right* **by** *simp*
also have ... $= (\varrho_0\ (trg_C\ \mu) \star_D G\ \mu) \cdot_D \varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu)$
using μ ϱ*.naturality* **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ \mu) \star_D G\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$(\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu))$
using μ *D.whisker-right* **by** *simp*
finally have
$(\varrho_1\ (C.cod\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D ((H\ \mu \star_D \varrho_0\ (src_C\ \mu)) \star_D \sigma_0\ (src_C\ \mu)) =$
$((\varrho_0\ (trg_C\ \mu) \star_D G\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D (\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu), \sigma_0\ (trg_C\ \mu), F\ (C.cod\ \mu)] \cdot_D$
$((\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \sigma_0\ (src_C\ \mu))) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ \mu), G\ (C.dom\ \mu), \sigma_0\ (src_C\ \mu)] \cdot_D$
$(\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.dom\ \mu), \varrho_0\ (src_C\ \mu), \sigma_0\ (src_C\ \mu)]$
using μ *D.assoc-naturality* [*of* $\varrho_0\ (trg_C\ \mu)\ G\ \mu\ \sigma_0\ (src_C\ \mu)$] *D.comp-assoc*
by *simp*
also have ... $= (\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu), \sigma_0\ (trg_C\ \mu), F\ (C.cod\ \mu)] \cdot_D$

$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_0\ (trg_C\ \mu) \star_D F\ \mu)) \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.dom\ \mu)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ \mu),\ G\ (C.dom\ \mu),\ \sigma_0\ (src_C\ \mu)] \cdot_D$
$(\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.dom\ \mu),\ \varrho_0\ (src_C\ \mu),\ \sigma_0\ (src_C\ \mu)]$
proof −
have $(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D (\varrho_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \sigma_0\ (src_C\ \mu)) =$
$\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu) \cdot_D (G\ \mu \star_D \sigma_0\ (src_C\ \mu))$
using μ *D.whisker-left* **by** *simp*
also have ... $= \varrho_0\ (trg_C\ \mu) \star_D (\sigma_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D \sigma_1\ (C.dom\ \mu)$
using μ σ*.naturality* **by** *simp*
also have ... $= (\varrho_0\ (trg_C\ \mu) \star_D \sigma_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D (\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.dom\ \mu))$
using μ *D.whisker-left* **by** *simp*
finally have $(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.cod\ \mu)) \cdot_D (\varrho_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \sigma_0\ (src_C\ \mu)) =$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D (\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.dom\ \mu))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ \mu) \star_D \sigma_0\ (trg_C\ \mu)) \star_D F\ \mu) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ \mu),\ \sigma_0\ (trg_C\ \mu),\ F\ (C.dom\ \mu)] \cdot_D$
$(\varrho_0\ (trg_C\ \mu) \star_D \sigma_1\ (C.dom\ \mu)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ \mu),\ G\ (C.dom\ \mu),\ \sigma_0\ (src_C\ \mu)] \cdot_D$
$(\varrho_1\ (C.dom\ \mu) \star_D \sigma_0\ (src_C\ \mu)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (C.dom\ \mu),\ \varrho_0\ (src_C\ \mu),\ \sigma_0\ (src_C\ \mu)]$
using μ *D.assoc′-naturality* [*of* $\varrho_0\ (trg_C\ \mu)\ \sigma_0\ (trg_C\ \mu)\ F\ \mu$] *D.comp-assoc*
by *simp*
also have ... $= (map_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D map_1\ (C.dom\ \mu)$
unfolding map_1*-def* map_0*-def*
using μ **by** *simp*
finally show $map_1\ (C.cod\ \mu) \cdot_D (H\ \mu \star_D map_0\ (src_C\ \mu)) =$
$(map_0\ (trg_C\ \mu) \star_D F\ \mu) \cdot_D map_1\ (C.dom\ \mu)$
by *blast*
qed
show $\bigwedge a.\ C.obj\ a \Longrightarrow (map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
$= map_1\ a \cdot_D (H.unit\ a \star_D map_0\ a)$
proof −
fix a
assume a: $C.obj\ a$
have $map_1\ a \cdot_D (H.unit\ a \star_D map_0\ a)$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F\ a] \cdot_D$
$(\varrho_0\ a \star_D \sigma_1\ a) \cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G\ a,\ \sigma_0\ a] \cdot_D$
$(\varrho_1\ a \star_D \sigma_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ a,\ \varrho_0\ a,\ \sigma_0\ a] \cdot_D$
$(H.unit\ a \star_D \varrho_0\ a \star_D \sigma_0\ a)$
unfolding map_1*-def* map_0*-def*
using a *C.obj-simps D.comp-assoc* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F\ a] \cdot_D$

$(\varrho_0\ a \star_D \sigma_1\ a) \cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G\ a,\ \sigma_0\ a] \cdot_D$
$((\varrho_1\ a \star_D \sigma_0\ a) \cdot_D$
$((H.unit\ a \star_D \varrho_0\ a) \star_D \sigma_0\ a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
using *a C.obj-simps D.assoc′-naturality* [*of H.unit a* ϱ_0 *a* σ_0 *a*] *D.comp-assoc*
by *auto*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F\ a] \cdot_D$
$(\varrho_0\ a \star_D \sigma_1\ a) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ a,\ G\ a,\ \sigma_0\ a] \cdot_D$
$((\varrho_0\ a \star_D G.unit\ a) \star_D \sigma_0\ a)) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
proof −
have $(\varrho_1\ a \star_D \sigma_0\ a) \cdot_D ((H.unit\ a \star_D \varrho_0\ a) \star_D \sigma_0\ a)$
$= \varrho_1\ a \cdot_D (H.unit\ a \star_D \varrho_0\ a) \star_D \sigma_0\ a$
using *a C.obj-simps D.whisker-right*
by (*metis C.objE D.hcomp-simps*(*4*) *D.hseqI′ D.ideD*(*1*) *D.ideD*(*3*) *D.seqI*
H.unit-simps(*1*) *H.unit-simps*(*2*) *H.unit-simps*(*5*)
ϱ*.ide-map*$_0$*-obj* ϱ*.map*$_0$*-simps*(*3*) ϱ*.map*$_1$*-simps*(*1*) ϱ*.map*$_1$*-simps*(*4*) σ*.ide-map*$_0$*-obj*)
also have ... $= (\varrho_0\ a \star_D G.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a$
using *a C.obj-simps* ϱ*.respects-unit D.comp-assoc* **by** *simp*
also have ... $= ((\varrho_0\ a \star_D G.unit\ a) \star_D \sigma_0\ a) \cdot_D (\mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a)$
using *a C.obj-simps D.whisker-right*
by (*metis C.objE D.hcomp-simps*(*4*) *D.hseqI′ D.ideD*(*1*) *D.ideD*(*3*) *D.seqI D.trg-cod*
H.unit-simps(*1*−*2*,*5*) *H*.i*-simps*(*3*,*5*) ϱ*.ide-map*$_0$*-obj* ϱ*.map*$_0$*-simps*(*3*)
ϱ*.map*$_1$*-simps*(*1*,*4*) ϱ*.respects-unit* σ*.ide-map*$_0$*-obj*)
finally have $(\varrho_1\ a \star_D \sigma_0\ a) \cdot_D ((H.unit\ a \star_D \varrho_0\ a) \star_D \sigma_0\ a) =$
$((\varrho_0\ a \star_D G.unit\ a) \star_D \sigma_0\ a) \cdot_D (\mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F\ a] \cdot_D$
$((\varrho_0\ a \star_D \sigma_1\ a) \cdot_D$
$(\varrho_0\ a \star_D G.unit\ a \star_D \sigma_0\ a)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a] \cdot_D$
$(\mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
using *a D.assoc-naturality* [*of* ϱ_0 *a G.unit a* σ_0 *a*] *D.comp-assoc* **by** *auto*
also have ... $= (\mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F\ a] \cdot_D$
$(\varrho_0\ a \star_D \sigma_0\ a \star_D F.unit\ a)) \cdot_D$
$(\varrho_0\ a \star_D \mathrm{r}_D{}^{-1}[\sigma_0\ a] \cdot_D \mathrm{l}_D[\sigma_0\ a]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a] \cdot_D$
$(\mathrm{r}_D{}^{-1}[\varrho_0\ a] \cdot_D \mathrm{l}_D[\varrho_0\ a] \star_D \sigma_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
proof −
have $(\varrho_0\ a \star_D \sigma_1\ a) \cdot_D (\varrho_0\ a \star_D G.unit\ a \star_D \sigma_0\ a)$
$= \varrho_0\ a \star_D \sigma_1\ a \cdot_D (G.unit\ a \star_D \sigma_0\ a)$

using *a D.whisker-left* [*of* ϱ_0 *a* σ_1 *a G.unit a* $\star_D$ σ_0 *a*] **by** *force*
also have ... = ϱ_0 *a* $\star_D$ (σ_0 *a* $\star_D$ *F.unit a*) $\cdot_D$ $\mathrm{r}_D{}^{-1}[\sigma_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\sigma_0\ a]$
using *a* σ*.respects-unit* **by** *simp*
also have ... = (ϱ_0 *a* $\star_D$ σ_0 *a* $\star_D$ *F.unit a*) $\cdot_D$ (ϱ_0 *a* $\star_D$ $\mathrm{r}_D{}^{-1}[\sigma_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\sigma_0\ a]$)
using *a D.whisker-left C.obj-simps* **by** *fastforce*
finally have (ϱ_0 *a* $\star_D$ σ_1 *a*) $\cdot_D$ (ϱ_0 *a* $\star_D$ *G.unit a* $\star_D$ σ_0 *a*) =
(ϱ_0 *a* $\star_D$ σ_0 *a* $\star_D$ *F.unit a*) $\cdot_D$ (ϱ_0 *a* $\star_D$ $\mathrm{r}_D{}^{-1}[\sigma_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\sigma_0\ a]$)
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = ((ϱ_0 *a* $\star_D$ σ_0 *a*) $\star_D$ *F.unit a*) $\cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F.map_0\ a]$ $\cdot_D$
(ϱ_0 *a* $\star_D$ $\mathrm{r}_D{}^{-1}[\sigma_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\sigma_0\ a]$) $\cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$ $\cdot_D$
($\mathrm{r}_D{}^{-1}[\varrho_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\varrho_0\ a]$ $\star_D$ σ_0 *a*) $\cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
using *a D.assoc′-naturality* [*of* ϱ_0 *a* σ_0 *a F.unit a*] *D.comp-assoc*
by *fastforce*
also have ... = ((ϱ_0 *a* $\star_D$ σ_0 *a*) $\star_D$ *F.unit a*) $\cdot_D$
($\mathrm{a}_D{}^{-1}[\varrho_0\ a,\ \sigma_0\ a,\ F.map_0\ a]$ $\cdot_D$
(ϱ_0 *a* $\star_D$ $\mathrm{r}_D{}^{-1}[\sigma_0\ a]$)) $\cdot_D$
(ϱ_0 *a* $\star_D$ $\mathrm{l}_D[\sigma_0\ a]$) $\cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$ $\cdot_D$
($\mathrm{r}_D{}^{-1}[\varrho_0\ a]$ $\star_D$ σ_0 *a*) $\cdot_D$
($\mathrm{l}_D[\varrho_0\ a]$ $\star_D$ σ_0 *a*) $\cdot_D$
$\mathrm{a}_D{}^{-1}[H.map_0\ a,\ \varrho_0\ a,\ \sigma_0\ a]$
using *a D.whisker-left D.whisker-right D.comp-assoc* **by** *simp*
also have ... = ((ϱ_0 *a* $\star_D$ σ_0 *a*) $\star_D$ *F.unit a*) $\cdot_D$
$\mathrm{r}_D{}^{-1}[\varrho_0\ a \star_D \sigma_0\ a]$ $\cdot_D$
($\mathrm{r}_D[\varrho_0\ a]$ $\star_D$ σ_0 *a*) $\cdot_D$
(($\mathrm{a}_D{}^{-1}[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$ $\cdot_D$
$\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$) $\cdot_D$
($\mathrm{r}_D{}^{-1}[\varrho_0\ a]$ $\star_D$ σ_0 *a*)) $\cdot_D$
$\mathrm{l}_D[\varrho_0\ a \star_D \sigma_0\ a]$
using *a D.lunit-hcomp*(*3*) [*of* ϱ_0 *a* σ_0 *a*] *D.runit-hcomp*(*2*) [*of* ϱ_0 *a* σ_0 *a*]
D.triangle′ [*of* ϱ_0 *a* σ_0 *a*] *D.comp-assoc*
by *auto*
also have ... = ((ϱ_0 *a* $\star_D$ σ_0 *a*) $\star_D$ *F.unit a*) $\cdot_D$ $\mathrm{r}_D{}^{-1}[\varrho_0\ a \star_D \sigma_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\varrho_0\ a \star_D \sigma_0\ a]$
proof −
have $\mathrm{a}_D{}^{-1}[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$ $\cdot_D$ $\mathrm{a}_D[\varrho_0\ a,\ G.map_0\ a,\ \sigma_0\ a]$
= (ϱ_0 *a* $\star_D$ $G.map_0$ *a*) $\star_D$ σ_0 *a*
using *a D.comp-inv-arr′*
by (*metis C.obj-def′ D.comp-assoc-assoc′*(*2*) $G.map_0$*-def* $G.map_0$*-simps*(*1*)
G.preserves-trg G.weak-arrow-of-homs-axioms ϱ*.ide-map*$_0$*-obj* ϱ*.map*$_0$*-simps*(*2*)
σ*.ide-map*$_0$*-obj* σ*.map*$_0$*-simps*(*3*) *horizontal-homs.objE weak-arrow-of-homs-def*)
moreover have ((ϱ_0 *a* $\star_D$ $G.map_0$ *a*) $\star_D$ σ_0 *a*) $\cdot_D$ ($\mathrm{r}_D{}^{-1}[\varrho_0\ a]$ $\star_D$ σ_0 *a*)
= ($\mathrm{r}_D{}^{-1}[\varrho_0\ a]$ $\star_D$ σ_0 *a*)
using *a D.comp-cod-arr D.whisker-right*

by (*metis D.runit′-simps(1) D.runit′-simps(5) G.map₀-def ϱ.ide-map₀-obj*
ϱ.map₀-simps(2) σ.ide-map₀-obj)
moreover have $(\mathrm{r}_D[\varrho_0\ a] \star_D \sigma_0\ a) \cdot_D (\mathrm{r}_D{}^{-1}[\varrho_0\ a] \star_D \sigma_0\ a) = \varrho_0\ a \star_D \sigma_0\ a$
using *a D.whisker-right D.comp-arr-inv′ D.R.as-nat-iso.components-are-iso*
by (*metis D.ideD(1) D.iso-runit D.runit-simps(5) ϱ.ide-map₀-obj σ.ide-map₀-obj*)
moreover have $(\varrho_0\ a \star_D \sigma_0\ a) \cdot_D \mathrm{l}_D[\varrho_0\ a \star_D \sigma_0\ a] = \mathrm{l}_D[\varrho_0\ a \star_D \sigma_0\ a]$
using *a D.comp-cod-arr* ‹$\bigwedge a.\ C.obj\ a \Longrightarrow D.ide\ (map_0\ a)$› *map₀-def* **by** *auto*
ultimately show *?thesis*
using *D.comp-assoc* **by** *metis*
qed
also have ... $= (map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
unfolding *map₀-def* **by** *simp*
finally show $(map_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
$= map_1\ a \cdot_D (H.unit\ a \star_D map_0\ a)$
by *simp*
qed
show $\bigwedge f\ g.\ [\![C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f]\!] \Longrightarrow$
$(map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(map_1\ g \star_D F\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[H\ g,\ map_0\ (src_C\ g),\ F\ f] \cdot_D$
$(H\ g \star_D map_1\ f) \cdot_D \mathrm{a}_D[H\ g,\ H\ f,\ map_0\ (src_C\ f)]$
$= map_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D map_0\ (src_C\ f))$
proof −
fix *f g*
assume *f*: *C.ide f* **and** *g*: *C.ide g* **and** *fg*: $src_C\ g = trg_C\ f$
have $(map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(map_1\ g \star_D F\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[H\ g,\ map_0\ (src_C\ g),\ F\ f] \cdot_D$
$(H\ g \star_D map_1\ f) \cdot_D \mathrm{a}_D[H\ g,\ H\ f,\ map_0\ (src_C\ f)]$
$= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \cdot_D$
$(\varrho_1\ g \star_D \sigma_0\ (src_C\ g)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g)]$
$\star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ f) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
unfolding *map₀-def map₁-def*
using *f g fg* **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$

$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$((\varrho_1\ g \star_D \sigma_0\ (src_C\ g)) \star_D F\ f) \cdot_D$
$((\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f]) \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f) \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
using *f g fg D.whisker-left D.whisker-right D.comp-assoc* **by** *fastforce*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$(((\varrho_1\ g \star_D \sigma_0\ (src_C\ g)) \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g \star_D \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g) \star_D F\ f] \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f]) \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f) \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
using *f g fg D.pentagon′ D.comp-assoc*
D.invert-side-of-triangle(2)
[*of* $\mathrm{a}_D{}^{-1}[H\ g \star_D \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g) \star_D F\ f]$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g),\ F\ f]$
$H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]$]
by *force*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$(\varrho_1\ g \star_D \sigma_0\ (src_C\ g) \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g) \star_D F\ f] \cdot_D$
$(((H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f])) \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$

$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$

proof −
have $((\varrho_1\ g \star_D \sigma_0\ (src_C\ g)) \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g \star_D \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (\varrho_1\ g \star_D \sigma_0\ (src_C\ g) \star_D F\ f)$
using *f g fg D.assoc′-naturality* [*of* $\varrho_1\ g\ \sigma_0\ (src_C\ g)\ F\ f$] **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$(\varrho_1\ g \star_D \sigma_0\ (src_C\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g) \star_D F\ f] \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$

proof −
have $((H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ f),\ \sigma_0\ (trg_C\ f),\ F\ f])) \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f)$
$= H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f$
proof −
have $(H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f])$
$= H\ g \star_D \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g) \star_D F\ f$
proof −
have $(H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f])$
$= H\ g \star_D \mathrm{a}_D[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (src_C\ g),\ \sigma_0\ (src_C\ g),\ F\ f]$
using *f g fg D.whisker-left* **by** *simp*
also have ... $= H\ g \star_D \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g) \star_D F\ f$
using *f g fg D.comp-arr-inv′* **by** *simp*
finally show *?thesis* **by** *simp*
qed
moreover have $(H\ g \star_D \varrho_0\ (src_C\ g) \star_D \sigma_0\ (src_C\ g) \star_D F\ f) \cdot_D$
$(H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f)$
$= (H\ g \star_D \varrho_0\ (trg_C\ f) \star_D \sigma_1\ f)$
using *f g fg D.comp-cod-arr* **by** *simp*
ultimately show *?thesis*
using *fg* **by** *simp*
qed

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g), F\ g, F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g), \sigma_0\ (trg_C\ g), F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g), G\ g, \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g, \sigma_0\ (src_C\ g), F\ f] \cdot_D$
$((\varrho_1\ g \star_D \sigma_0\ (src_C\ g) \star_D F\ f) \cdot_D$
$((H\ g \star_D \varrho_0\ (trg_C\ f)) \star_D \sigma_1\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g, \varrho_0\ (src_C\ g), G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f), G\ f, \sigma_0\ (src_C\ f)]) \cdot_D$
$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f, \varrho_0\ (src_C\ f), \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g, H\ f, \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
using *f g fg D.assoc′-naturality* [*of* $H\ g\ \varrho_0\ (src_C\ g)\ \sigma_1\ f$] *D.comp-assoc* **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g), F\ g, F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g), \sigma_0\ (trg_C\ g), F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g), G\ g, \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g, \sigma_0\ (src_C\ g), F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$(\varrho_1\ g \star_D G\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g, \varrho_0\ (src_C\ g), G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f), G\ f, \sigma_0\ (src_C\ f)])) \cdot_D$
$(H\ g \star_D \varrho_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f, \varrho_0\ (src_C\ f), \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g, H\ f, \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
proof −
have $(\varrho_1\ g \star_D \sigma_0\ (src_C\ g) \star_D F\ f) \cdot_D ((H\ g \star_D \varrho_0\ (trg_C\ f)) \star_D \sigma_1\ f) =$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D (\varrho_1\ g \star_D G\ f \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.interchange D.comp-arr-dom D.comp-cod-arr*
by (*metis C.ideD(1)* ϱ*.map*$_1$*-simps(1,5)* ϱ*.naturality* σ*.map*$_1$*-simps(1,4)* σ*.naturality C.ideD(2,3)*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g), F\ g, F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g), \sigma_0\ (trg_C\ g), F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g), G\ g, \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g, \sigma_0\ (src_C\ g), F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$(\varrho_1\ g \star_D G\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\mathrm{a}_D{}^{-1}[H\ g, \varrho_0\ (src_C\ g), G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D[\varrho_0\ (trg_C\ f), G\ f, \sigma_0\ (src_C\ f)])) \cdot_D$

$\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f) \star_D\ G\ f,\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$((H\ g \star_D\ \varrho_1\ f) \star_D\ \sigma_0\ (src_C\ f))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]\ \cdot_D$
$(H\ g \star_D\ \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D\ \sigma_0\ (src_C\ f)]$
using *f g fg D.comp-assoc D.hcomp-reassoc*(*2*) **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D\ \sigma_0\ (trg_C\ g)) \star_D\ \Phi_F\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D\ \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f]\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D\ F\ f)\ \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D\ \sigma_1\ g) \star_D\ F\ f)\ \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D\ F\ f)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D\ G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]\ \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D\ G\ g) \star_D\ \sigma_1\ f)\ \cdot_D$
$((\varrho_1\ g \star_D\ G\ f \star_D\ \sigma_0\ (src_C\ f))\ \cdot_D$
$\mathrm{a}_D[H\ g \star_D\ \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D\ \sigma_0\ (src_C\ f))\ \cdot_D$
$((H\ g \star_D\ \varrho_1\ f) \star_D\ \sigma_0\ (src_C\ f))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]\ \cdot_D$
$(H\ g \star_D\ \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D\ \sigma_0\ (src_C\ f)]$
proof −
have $(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f \star_D\ \sigma_0\ (src_C\ f)]\ \cdot_D$
$(H\ g \star_D\ \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]))\ \cdot_D$
$\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f) \star_D\ G\ f,\ \sigma_0\ (src_C\ f)]$
$= \mathrm{a}_D[H\ g \star_D\ \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D\ \sigma_0\ (src_C\ f))$
proof −
have $((H\ g \star_D\ \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f) \star_D\ G\ f,\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$(\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D\ \sigma_0\ (src_C\ f))$
$= \mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f \star_D\ \sigma_0\ (src_C\ f)]\ \cdot_D$
$\mathrm{a}_D[H\ g \star_D\ \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]$
using *f g fg D.pentagon D.comp-assoc* **by** *simp*
moreover have *D.seq* $\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f \star_D\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[H\ g \star_D\ \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]$
using *f g fg*
by (*intro D.seqI*) *auto*
ultimately show *?thesis*
using *f g fg D.comp-assoc*
D.invert-opposite-sides-of-square
[*of* $\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f \star_D\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[H\ g \star_D\ \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)]$
$(H\ g \star_D\ \mathrm{a}_D[\varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)])\ \cdot_D$
$\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f) \star_D\ G\ f,\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D\ \sigma_0\ (src_C\ f)]$
by *fastforce*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(((\varrho_1\ g \star_D G\ f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((H\ g \star_D \varrho_1\ f) \star_D \sigma_0\ (src_C\ f))) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
proof −
have $(\varrho_1\ g \star_D G\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D \mathrm{a}_D[H\ g \star_D \varrho_0\ (trg_C\ f),\ G\ f,\ \sigma_0\ (src_C\ f)] =$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D ((\varrho_1\ g \star_D G\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.assoc-naturality* [*of* $\varrho_1\ g\ G\ f\ \sigma_0\ (src_C\ f)$] **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)])$
proof −
have $((\varrho_1\ g \star_D G\ f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((H\ g \star_D \varrho_1\ f) \star_D \sigma_0\ (src_C\ f)) =$
$(\varrho_1\ g \star_D G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \cdot_D (H\ g \star_D \varrho_1\ f) \star_D \sigma_0\ (src_C\ f)$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D (\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D$
$\varrho_1\ (g \star_C f) \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f)) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
$\star_D \sigma_0\ (src_C\ f)$
proof −

have $((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \cdot_D \mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D$
$((\varrho_1\ g \star_D\ G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f))) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
$= \varrho_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))$
using *f g fg ϱ.respects-hcomp D.comp-assoc* **by** *simp*
moreover have $D.seq\ (\varrho_1\ (g \star_C f))\ (\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ **by** *auto*
ultimately
have $(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \cdot_D \mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D$
$((\varrho_1\ g \star_D\ G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f))$
$= (\varrho_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
using *f g fg D.invert-side-of-triangle(2)* **by** *simp*
moreover have $D.seq\ (\varrho_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f)))$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *H.FF-def*
by *auto*
ultimately
have $\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D$
$((\varrho_1\ g \star_D\ G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f))$
$= (\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D$
$(\varrho_1\ (g \star_C f) \cdot_D$
$(\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
using *f g fg*
D.invert-side-of-triangle(1)
$[of\ (\varrho_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
$\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D ((\varrho_1\ g \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f))]$
by *simp*
hence $(\varrho_1\ g \star_D\ G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f)$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D (\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D$
$(\varrho_1\ (g \star_C f) \cdot_D$
$(\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *H.FF-def*
D.invert-side-of-triangle(1)
by *simp*
hence $(\varrho_1\ g \star_D\ G\ f) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (src_C\ g),\ G\ f] \cdot_D (H\ g \star_D\ \varrho_1\ f)$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D (\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D$
$\varrho_1\ (g \star_C f) \cdot_D$
$(\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f)) \cdot_D \mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)]$
using *D.comp-assoc* **by** *simp*
thus *?thesis*
using *fg* **by** *simp*
qed
also have ... $= (\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\Phi_H\ (g, f) \star_D\ \varrho_0\ (src_C\ f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f))$

using *f g fg D.whisker-right C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ H.FF-def G.FF-def*
by *force*
finally have $((\varrho_1\ g \star_D G\ f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ \varrho_0\ (trg_C\ f),\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((H\ g \star_D \varrho_1\ f) \star_D \sigma_0\ (src_C\ f))$
$= (\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g \star_D H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]$
proof −
have $((\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)])) \cdot_D$
$\mathrm{a}_D[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
$= \mathrm{a}_D{}^{-1}[H\ g \star_D H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]$
proof −
have $(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)])$
$= \mathrm{a}_D{}^{-1}[H\ g \star_D H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]$
using *f g fg D.pentagon′ D.comp-assoc* **by** *simp*
moreover have *D.seq* $(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f))$
$(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]))$
using *f g fg* **by** *auto*
ultimately show *?thesis*
using *f g fg D.comp-assoc*
D.invert-side-of-triangle(2)
[*of* $(\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f)] \star_D \sigma_0\ (src_C\ f)) \cdot_D$

$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f \star_D \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(H\ g \star_D \mathrm{a}_D{}^{-1}[H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)])$
$\mathrm{a}_D{}^{-1}[H\ g \star_D H\ f,\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D{}^{-1}[H\ g,\ H\ f,\ \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f)]]$
by *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g) \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.assoc′-naturality* [*of* $\Phi_H\ (g, f)\ \varrho_0\ (src_C\ f)\ \sigma_0\ (src_C\ f)$]
by *fastforce*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g) \star_D F\ g,\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g \star_D F\ f) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.hcomp-reassoc(1) D.comp-assoc* **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g) \star_D F\ g,\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g \star_D F\ f) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g) \star_D F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D G\ g) \star_D \sigma_1\ f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$

$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.comp-assoc D.pentagon*
D.invert-opposite-sides-of-square
[*of* $\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D \sigma_0\ (src_C\ g),\ F\ f] \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g)] \star_D F\ f)$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ \sigma_0\ (src_C\ g) \star_D F\ f]$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]]$
by *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g) \star_D F\ g,\ F\ f] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g \star_D F\ f) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D G\ g \star_D \sigma_1\ f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.assoc-naturality* [*of* $\varrho_0\ (trg_C\ g)\ G\ g\ \sigma_1\ f$] *D.comp-assoc* **by** *simp*
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g) \star_D F\ g,\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f])) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
proof −
have $(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g \star_D F\ f) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D G\ g \star_D \sigma_1\ f)$
$= \varrho_0\ (trg_C\ g) \star_D (\sigma_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)$

using *f g fg D.whisker-left* **by** *simp*
also have ... $= \varrho_0\ (trg_C\ g) \star_D$
$\mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$\sigma_1\ (g \star_C f) \cdot_D$
$(\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
proof −
have $((\sigma_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\sigma_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)) \cdot_D$
$\mathrm{a}_D[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
$= \sigma_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg σ.respects-hcomp D.comp-assoc* **by** *simp*
hence $(\sigma_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\sigma_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)$
$= (\sigma_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ D.invert-side-of-triangle(2)*
by *simp*
hence $\mathrm{a}_D[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D$
$(\sigma_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)$
$= (\sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$(\sigma_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ G.FF-def*
D.invert-side-of-triangle(1)
$[of\ (\sigma_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
$\sigma_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)$
$\mathrm{a}_D[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D (\sigma_1\ g \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)]$
by *simp*
hence $(\sigma_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \sigma_1\ f)$
$= \mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f] \cdot_D (\sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$(\sigma_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f))) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ G.FF-def F.FF-def*
D.invert-side-of-triangle(1)
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)])$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ F.FF-def G.FF-def*
D.whisker-left

by *force*
finally have $(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ g \star_D F\ f) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \sigma_0\ (src_C\ g),\ F\ f]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D G\ g \star_D \sigma_1\ f)$
$= (\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f))) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g \star_D F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g) \star_D D.inv\ (\Phi_F\ (g, f)))) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.pentagon′ D.comp-assoc*
D.invert-side-of-triangle(1)
[*of* $(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g] \star_D F\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g) \star_D F\ g,\ F\ f] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[\sigma_0\ (trg_C\ g),\ F\ g,\ F\ f])$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g),\ F\ g,\ F\ f]$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ g \star_D F\ f]$]
by *simp*
also have ... $= ((((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D D.inv\ (\Phi_F\ (g, f)))) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)]) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.assoc′-naturality* [*of* $\varrho_0\ (trg_C\ g)\ \sigma_0\ (trg_C\ g)\ D.inv\ (\Phi_F\ (g, f))$]
D.comp-assoc

by *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f))) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
proof −
have $(((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D D.inv\ (\Phi_F\ (g, f)))) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)]$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)]$
proof −
have $((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D D.inv\ (\Phi_F\ (g, f)))$
$= (\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f) \cdot_D D.inv\ (\Phi_F\ (g, f))$
using *f g fg D.whisker-left C.VV.arr-char*$_{SbC}$ **by** *simp*
also have ... $= (\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D F\ (g \star_C f)$
using *f g fg D.comp-arr-inv′ F.cmp-components-are-iso* **by** *simp*
finally have $((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D \Phi_F\ (g, f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D D.inv\ (\Phi_F\ (g, f)))$
$= (\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D F\ (g \star_C f)$
by *blast*
moreover have $((\varrho_0\ (trg_C\ g) \star_D \sigma_0\ (trg_C\ g)) \star_D F\ (g \star_C f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)]$
$= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)]$
using *f g fg D.comp-cod-arr* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.pentagon D.comp-assoc*
D.invert-side-of-triangle(*1*)
[*of* $\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$

$\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D\ G\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f))]$
D.invert-side-of-triangle(2)
$[of\ (\varrho_0\ (trg_C\ g) \star_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f \star_D \sigma_0\ (src_C\ f)] \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g) \star_D\ G\ g,\ G\ f,\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D\ G\ f,\ \sigma_0\ (src_C\ f)]$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g,\ G\ f] \star_D \sigma_0\ (src_C\ f)]$
by *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ (g \star_C f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$((((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f))) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f))) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
proof −
have $(\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ g \star_D\ G\ f,\ \sigma_0\ (src_C\ f)]$
$= \mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ (g \star_C f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \star_D \sigma_0\ (src_C\ f))$
using *f g fg D.assoc-naturality* $[of\ \varrho_0\ (trg_C\ g)\ \Phi_G\ (g, f)\ \sigma_0\ (src_C\ f)]$
by *fastforce*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\varrho_0\ (trg_C\ g),\ \sigma_0\ (trg_C\ g),\ F\ (g \star_C f)] \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D \sigma_1\ (g \star_C f)) \cdot_D$
$\mathrm{a}_D[\varrho_0\ (trg_C\ g),\ G\ (g \star_C f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (g \star_C f),\ \varrho_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\Phi_H\ (g, f) \star_D \varrho_0\ (src_C\ f) \star_D \sigma_0\ (src_C\ f))$
proof −
have $(((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f))) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f))$
$= \varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)$
proof −
have $((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f))$
$= (\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \cdot_D$
$(\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f)))$
$\star_D \sigma_0\ (src_C\ f)$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ *D.whisker-right* **by** *simp*
also have ... $= (\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f) \cdot_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f)$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ *D.whisker-left* $[of\ \varrho_0\ (trg_C\ g)]$ **by** *simp*
also have ... $= (\varrho_0\ (trg_C\ g) \star_D G\ (g \star_C f)) \star_D \sigma_0\ (src_C\ f)$

using *f g fg D.comp-arr-inv′ G.cmp-components-are-iso G.cmp-simps*(*5*) **by** *auto*
finally have $((\varrho_0\ (trg_C\ g) \star_D \Phi_G\ (g, f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$((\varrho_0\ (trg_C\ g) \star_D D.inv\ (\Phi_G\ (g, f))) \star_D \sigma_0\ (src_C\ f))$
$= (\varrho_0\ (trg_C\ g) \star_D G\ (g \star_C f)) \star_D \sigma_0\ (src_C\ f)$
by *blast*
moreover have $((\varrho_0\ (trg_C\ g) \star_D G\ (g \star_C f)) \star_D \sigma_0\ (src_C\ f)) \cdot_D$
$(\varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f))$
$= \varrho_1\ (g \star_C f) \star_D \sigma_0\ (src_C\ f)$
using *f g fg D.comp-cod-arr* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= map_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D map_0\ (src_C\ f))$
unfolding map_0*-def* map_1*-def*
using *f g fg D.comp-assoc* **by** *simp*
finally show $(map_0\ (trg_C\ g) \star_D \Phi_F\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_C\ g), F\ g, F\ f] \cdot_D$
$(map_1\ g \star_D F\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[H\ g, map_0\ (src_C\ g), F\ f] \cdot_D$
$(H\ g \star_D map_1\ f) \cdot_D$
$\mathrm{a}_D[H\ g, H\ f, map_0\ (src_C\ f)]$
$= map_1\ (g \star_C f) \cdot_D (\Phi_H\ (g, f) \star_D map_0\ (src_C\ f))$
using *D.comp-assoc* **by** *simp*
qed
qed

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ H\ \Phi_H\ map_0\ map_1$
..

end

1.14.4 Whiskering of Pseudonatural Transformations

Similarly to ordinary natural transformations, pseudonatural transformations can be whiskered with pseudofunctors on the left and the right.

locale *pseudonatural-transformation-whisker-right* =
B: *bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ +
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
$\tau.F$: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$ +
$\tau.G$: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$ +
H: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ H\ \Phi_H$ +
τ: *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
for V_B :: *'b comp* (**infixr** ‹$\cdot_B$› *55*)

and H_B :: $'b$ *comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and i_B :: $'b \Rightarrow 'b$ (‹i_B[-]›)
and src_B :: $'b \Rightarrow 'b$
and trg_B :: $'b \Rightarrow 'b$
and V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and H :: $'b \Rightarrow 'c$
and Φ_H :: $'b * 'b \Rightarrow 'c$
and τ_0 :: $'c \Rightarrow 'd$
and τ_1 :: $'c \Rightarrow 'd$
begin

interpretation *FoH*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D H Φ_H F Φ_F
..
interpretation *GoH*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D H Φ_H G Φ_G
..

definition map_0
where $map_0\ a = \tau_0\ (H.map_0\ a)$

definition map_1
where $map_1\ f = \tau_1\ (H\ f)$

sublocale *pseudonatural-transformation* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
‹*F o H*› *FoH.cmp* ‹*G o H*› *GoH.cmp* map_0 map_1
proof
show $\bigwedge a.\ B.obj\ a \Longrightarrow D.ide\ (map_0\ a)$
using map_0*-def* **by** *simp*
show $\bigwedge a.\ B.obj\ a \Longrightarrow$ «$map_0\ a : src_D\ ((F \circ H)\ a) \rightarrow_D src_D\ ((G \circ H)\ a)$»
using map_0*-def* τ*.*map_0*-in-hhom B.obj-simps C.obj-simps* **by** *simp*
show $\bigwedge f.\ B.ide\ f \Longrightarrow D.iso\ (map_1\ f)$
using map_1*-def* τ*.iso-*map_1*-ide* **by** *simp*

show $\bigwedge f.\ B.ide\ f \Longrightarrow$
$\quad$ «$map_1\ f : (G \circ H)\ f \star_D map_0\ (src_B\ f) \Rightarrow_D map_0\ (trg_B\ f) \star_D (F \circ H)\ f$»
using map_0-*def* map_1-*def* **by** *auto*
show $\bigwedge \mu.\ B.arr\ \mu \Longrightarrow$
$\quad map_1\ (B.cod\ \mu) \cdot_D ((G \circ H)\ \mu \star_D map_0\ (src_B\ \mu)) =$
$\quad (map_0\ (trg_B\ \mu) \star_D (F \circ H)\ \mu) \cdot_D map_1\ (B.dom\ \mu)$
unfolding map_0-*def* map_1-*def*
using τ.*naturality* **by** *force*
show $\bigwedge a.\ B.obj\ a \Longrightarrow$
$\quad (map_0\ a \star_D FoH.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
$\quad\ = map_1\ a \cdot_D (GoH.unit\ a \star_D map_0\ a)$
proof −
fix a
assume a: $B.obj\ a$
have $map_1\ a \cdot_D (GoH.unit\ a \star_D map_0\ a) =$
$\quad \tau_1\ (H\ a) \cdot_D (G\ (H.unit\ a) \cdot_D \tau.G.unit\ (H.map_0\ a) \star_D \tau_0\ (H.map_0\ a))$
unfolding map_0-*def* map_1-*def*
using *a GoH.unit-char′* **by** *simp*
also have ... $= (\tau_1\ (H\ a) \cdot_D (G\ (H.unit\ a) \star_D \tau_0\ (H.map_0\ a))) \cdot_D$
$\quad (\tau.G.unit\ (H.map_0\ a) \star_D \tau_0\ (H.map_0\ a))$
using *a D.whisker-right D.comp-assoc* **by** *simp*
also have ... $= (\tau_0\ (H.map_0\ a) \star_D F\ (H.unit\ a)) \cdot_D \tau_1\ (H.map_0\ a) \cdot_D$
$\quad (\tau.G.unit\ (H.map_0\ a) \star_D \tau_0\ (H.map_0\ a))$
using *a* τ.*naturality* [*of H.unit a*] *D.comp-assoc* **by** *simp*
also have ... $= ((\tau_0\ (H.map_0\ a) \star_D F\ (H.unit\ a)) \cdot_D$
$\quad (\tau_0\ (H.map_0\ a) \star_D \tau.F.unit\ (H.map_0\ a))) \cdot_D$
$\quad \mathrm{r}_D{}^{-1}[\tau_0\ (H.map_0\ a)] \cdot_D \mathrm{l}_D[\tau_0\ (H.map_0\ a)]$
using *a* τ.*respects-unit D.comp-assoc* **by** *simp*
also have ... $= (\tau_0\ (H.map_0\ a) \star_D F\ (H.unit\ a) \cdot_D \tau.F.unit\ (H.map_0\ a)) \cdot_D$
$\quad \mathrm{r}_D{}^{-1}[\tau_0\ (H.map_0\ a)] \cdot_D \mathrm{l}_D[\tau_0\ (H.map_0\ a)]$
using *a D.whisker-left* **by** *simp*
also have ... $= (map_0\ a \star_D FoH.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
unfolding map_0-*def* map_1-*def*
using *a FoH.unit-char′* **by** *simp*
finally show $(map_0\ a \star_D FoH.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
$\quad = map_1\ a \cdot_D (GoH.unit\ a \star_D map_0\ a)$
by *simp*
qed
show $\bigwedge f\ g.\ [\![B.ide\ f;\ B.ide\ g;\ src_B\ g = trg_B\ f]\!] \Longrightarrow$
$\quad (map_0\ (trg_B\ g) \star_D FoH.cmp\ (g, f)) \cdot_D$
$\quad \mathrm{a}_D[map_0\ (trg_B\ g), (F \circ H)\ g, (F \circ H)\ f] \cdot_D$
$\quad (map_1\ g \star_D (F \circ H)\ f) \cdot_D$
$\quad D.inv\ \mathrm{a}_D[(G \circ H)\ g, map_0\ (src_B\ g), (F \circ H)\ f] \cdot_D$
$\quad ((G \circ H)\ g \star_D map_1\ f) \cdot_D$
$\quad \mathrm{a}_D[(G \circ H)\ g, (G \circ H)\ f, map_0\ (src_B\ f)]$
$\quad\ = map_1\ (g \star_B f) \cdot_D (GoH.cmp\ (g, f) \star_D map_0\ (src_B\ f))$
proof −
fix $f\ g$
assume f: $B.ide\ f$ **and** g: $B.ide\ g$ **and** fg: $src_B\ g = trg_B\ f$

have $map_1\ (g \star_B f) \cdot_D (GoH.cmp\ (g, f) \star_D map_0\ (src_B\ f))$
$= \tau_1\ (H\ (g \star_B f)) \cdot_D$
$(G\ (H\ (g \star_B f)) \cdot_D G\ (\Phi_H\ (g, f)) \cdot_D \Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
unfolding map_0*-def* map_1*-def*
using *f g fg GoH.cmp-def* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ **by** *simp*
also have ... $= (\tau_1\ (H\ (g \star_B f)) \cdot_D$
$(G\ (H\ (g \star_B f)) \star_D \tau_0\ (src_C\ (H\ f)))) \cdot_D$
$(G\ (\Phi_H\ (g, f)) \star_D \tau_0\ (src_C\ (H\ f))) \cdot_D$
$(\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *H.FF-def*
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *D.whisker-right*
D.comp-assoc
by *simp*
also have ... $= (\tau_0\ (trg_C\ (H\ g)) \star_D F\ (H\ (g \star_B f))) \cdot_D$
$(\tau_1\ (H\ (g \star_B f)) \cdot_D$
$(G\ (\Phi_H\ (g, f)) \star_D \tau_0\ (src_C\ (H\ f)))) \cdot_D$
$(\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
using *f g fg* τ*.naturality* [*of* $H\ (g \star_B f)$] *D.comp-assoc* **by** *auto*
also have ... $= (\tau_0\ (trg_C\ (H\ g)) \star_D F\ (H\ (g \star_B f))) \cdot_D$
$(\tau_0\ (trg_C\ (H\ g)) \star_D F\ (\Phi_H\ (g, f))) \cdot_D$
$\tau_1\ (H\ g \star_C H\ f) \cdot_D$
$(\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
using *f g fg* τ*.naturality* [*of* $\Phi_H\ (g, f)$] *D.comp-assoc* **by** *fastforce*
also have ... $= (\tau_0\ (trg_C\ (H\ g)) \star_D F\ (H\ (g \star_B f))) \cdot_D$
$(\tau_0\ (trg_C\ (H\ g)) \star_D F\ (\Phi_H\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ (H\ g)) \star_D \Phi_F\ (H\ g, H\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)), F\ (H\ g), F\ (H\ f)] \cdot_D$
$(\tau_1\ (H\ g) \star_D F\ (H\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[G\ (H\ g), \tau_0\ (src_C\ (H\ g)), F\ (H\ f)] \cdot_D$
$(G\ (H\ g) \star_D \tau_1\ (H\ f)) \cdot_D$
$\mathrm{a}_D[G\ (H\ g), G\ (H\ f), \tau_0\ (src_C\ (H\ f))] \cdot_D$
$(D.inv\ (\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f))) \cdot_D$
$(\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f))))$
proof −
have $\tau_1\ (H\ g \star_C H\ f) = (\tau_0\ (trg_C\ (H\ g)) \star_D \Phi_F\ (H\ g, H\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)), F\ (H\ g), F\ (H\ f)] \cdot_D$
$(\tau_1\ (H\ g) \star_D F\ (H\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[G\ (H\ g), \tau_0\ (src_C\ (H\ g)), F\ (H\ f)] \cdot_D$
$(G\ (H\ g) \star_D \tau_1\ (H\ f)) \cdot_D$
$\mathrm{a}_D[G\ (H\ g), G\ (H\ f), \tau_0\ (src_C\ (H\ f))] \cdot_D$
$D.inv\ (\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
proof −
have $\tau_1\ (H\ g \star_C H\ f) \cdot_D (\Phi_G\ (H\ g, H\ f) \star_D \tau_0\ (src_C\ (H\ f)))$
$= (\tau_0\ (trg_C\ (H\ g)) \star_D \Phi_F\ (H\ g, H\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)), F\ (H\ g), F\ (H\ f)] \cdot_D$
$(\tau_1\ (H\ g) \star_D F\ (H\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[G\ (H\ g), \tau_0\ (src_C\ (H\ g)), F\ (H\ f)] \cdot_D$
$(G\ (H\ g) \star_D \tau_1\ (H\ f)) \cdot_D$
$\mathrm{a}_D[G\ (H\ g), G\ (H\ f), \tau_0\ (src_C\ (H\ f))]$

using *f g fg* τ*.respects-hcomp* [*of H f H g*] **by** *simp*
moreover have *D.seq* $(\tau_0\ (trg_C\ (H\ g))\ \star_D\ \Phi_F\ (H\ g,\ H\ f))$
$(\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)),\ F\ (H\ g),\ F\ (H\ f)]\ \cdot_D$
$(\tau_1\ (H\ g)\ \star_D\ F\ (H\ f))\ \cdot_D$
D.inv $\mathrm{a}_D[G\ (H\ g),\ \tau_0\ (src_C\ (H\ g)),\ F\ (H\ f)]\ \cdot_D$
$(G\ (H\ g)\ \star_D\ \tau_1\ (H\ f))\ \cdot_D$
$\mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))])$
using *f g fg C.obj-simps C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ τ*.F.FF-def*
by *simp*
moreover have *D.iso* $(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f)))$
using *f g fg* τ*.G.cmp-components-are-iso* [*of H g H f*] **by** *simp*
ultimately
have $\tau_1\ (H\ g\ \star_C\ H\ f) = ((\tau_0\ (trg_C\ (H\ g))\ \star_D\ \Phi_F\ (H\ g,\ H\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)),\ F\ (H\ g),\ F\ (H\ f)]\ \cdot_D$
$(\tau_1\ (H\ g)\ \star_D\ F\ (H\ f))\ \cdot_D$
D.inv $\mathrm{a}_D[G\ (H\ g),\ \tau_0\ (src_C\ (H\ g)),\ F\ (H\ f)]\ \cdot_D$
$(G\ (H\ g)\ \star_D\ \tau_1\ (H\ f))\ \cdot_D$
$\mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))])\ \cdot_D$
D.inv $(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f)))$
using *D.invert-side-of-triangle*(*2*) **by** *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0\ (trg_C\ (H\ g))\ \star_D\ F\ (H\ (g\ \star_B\ f)))\ \cdot_D$
$(\tau_0\ (trg_C\ (H\ g))\ \star_D\ F\ (\Phi_H\ (g,\ f)))\ \cdot_D$
$(\tau_0\ (trg_C\ (H\ g))\ \star_D\ \Phi_F\ (H\ g,\ H\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)),\ F\ (H\ g),\ F\ (H\ f)]\ \cdot_D$
$(\tau_1\ (H\ g)\ \star_D\ F\ (H\ f))\ \cdot_D$
D.inv $\mathrm{a}_D[G\ (H\ g),\ \tau_0\ (src_C\ (H\ g)),\ F\ (H\ f)]\ \cdot_D$
$(G\ (H\ g)\ \star_D\ \tau_1\ (H\ f))\ \cdot_D$
$\mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))]$
proof −
have (*D.inv* $(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f)))\ \cdot_D$
$(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f))))$
$= (G\ (H\ g)\ \star_D\ G\ (H\ f))\ \star_D\ \tau_0\ (src_C\ (H\ f))$
proof −
have (*D.inv* $(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f)))\ \cdot_D$
$(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f))))$
$= (D.inv\ (\Phi_G\ (H\ g,\ H\ f))\ \star_D\ \tau_0\ (src_C\ (H\ f)))\ \cdot_D$
$(\Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f)))$
using *f g fg* **by** *simp*
also have ... $= D.inv\ (\Phi_G\ (H\ g,\ H\ f))\ \cdot_D\ \Phi_G\ (H\ g,\ H\ f)\ \star_D\ \tau_0\ (src_C\ (H\ f))$
using *f g fg D.whisker-right C.VV.arr-char*$_{SbC}$ **by** *simp*
also have ... $= (G\ (H\ g)\ \star_D\ G\ (H\ f))\ \star_D\ \tau_0\ (src_C\ (H\ f))$
using *f g fg D.comp-inv-arr'* τ*.G.cmp-components-are-iso* **by** *simp*

finally show *?thesis* **by** *blast*
qed
moreover have $\mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))]\ \cdot_D$ ...
$= \mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))]$
using *f g fg D.comp-arr-dom* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also have ... $= (\tau_0\ (trg_C\ (H\ g))\ \star_D\ F\ (H\ (g \star_B f))\ \cdot_D$
$F\ (\Phi_H\ (g, f))\ \cdot_D$
$\Phi_F\ (H\ g,\ H\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ (H\ g)),\ F\ (H\ g),\ F\ (H\ f)]\ \cdot_D$
$(\tau_1\ (H\ g)\ \star_D\ F\ (H\ f))\ \cdot_D$
$D.inv\ \mathrm{a}_D[G\ (H\ g),\ \tau_0\ (src_C\ (H\ g)),\ F\ (H\ f)]\ \cdot_D$
$(G\ (H\ g)\ \star_D\ \tau_1\ (H\ f))\ \cdot_D$
$\mathrm{a}_D[G\ (H\ g),\ G\ (H\ f),\ \tau_0\ (src_C\ (H\ f))]$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
$B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *H.FF-def*
D.whisker-left $C.VV.arr\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *D.comp-assoc*
by *auto*
also have ... $= (map_0\ (trg_B\ g)\ \star_D\ FoH.cmp\ (g, f))\ \cdot_D$
$\mathrm{a}_D[map_0\ (trg_B\ g),\ (F \circ H)\ g,\ (F \circ H)\ f]\ \cdot_D$
$(map_1\ g \star_D\ (F \circ H)\ f)\ \cdot_D$
$D.inv\ \mathrm{a}_D[(G \circ H)\ g,\ map_0\ (src_B\ g),\ (F \circ H)\ f]\ \cdot_D$
$((G \circ H)\ g \star_D\ map_1\ f)\ \cdot_D$
$\mathrm{a}_D[(G \circ H)\ g,\ (G \circ H)\ f,\ map_0\ (src_B\ f)]$
unfolding map_0*-def* map_1*-def*
using *f g fg FoH.cmp-def* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ **by** *simp*
finally show $(map_0\ (trg_B\ g)\ \star_D\ FoH.cmp\ (g, f))\ \cdot_D$
$\mathrm{a}_D[map_0\ (trg_B\ g),\ (F \circ H)\ g,\ (F \circ H)\ f]\ \cdot_D$
$(map_1\ g \star_D\ (F \circ H)\ f)\ \cdot_D$
$D.inv\ \mathrm{a}_D[(G \circ H)\ g,\ map_0\ (src_B\ g),\ (F \circ H)\ f]\ \cdot_D$
$((G \circ H)\ g \star_D\ map_1\ f)\ \cdot_D$
$\mathrm{a}_D[(G \circ H)\ g,\ (G \circ H)\ f,\ map_0\ (src_B\ f)]$
$= map_1\ (g \star_B f)\ \cdot_D\ (GoH.cmp\ (g, f) \star_D\ map_0\ (src_B\ f))$
by *simp*
qed
qed

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$(F\ o\ H)$ $FoH.cmp$ $(G\ o\ H)$ $GoH.cmp$ map_0 map_1
..

end

locale *pseudonatural-transformation-whisker-left* =
B: *bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ +
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +

$\tau.F$: *pseudofunctor* V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F +
$\tau.G$: *pseudofunctor* V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C G Φ_G +
H: *pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D H Φ_H +
τ: *pseudonatural-transformation*
V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F G Φ_G τ_0 τ_1
for V_B :: $'b$ *comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: $'b$ *comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and i_B :: $'b \Rightarrow 'b$ (‹i_B[-]›)
and src_B :: $'b \Rightarrow 'b$
and trg_B :: $'b \Rightarrow 'b$
and V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'b \Rightarrow 'c$
and Φ_F :: $'b * 'b \Rightarrow 'c$
and G :: $'b \Rightarrow 'c$
and Φ_G :: $'b * 'b \Rightarrow 'c$
and H :: $'c \Rightarrow 'd$
and Φ_H :: $'c * 'c \Rightarrow 'd$
and τ_0 :: $'b \Rightarrow 'c$
and τ_1 :: $'b \Rightarrow 'c$
begin

interpretation *HoF*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F H Φ_H
..
interpretation *HoG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G H Φ_H
..

definition map_0
where $map_0\ a = H\ (\tau_0\ a)$

definition map_1
where $map_1\ f = D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ f),\ F\ f)) \cdot_D H\ (\tau_1\ f) \cdot_D \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f))$

sublocale *pseudonatural-transformation* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
HoF.map HoF.cmp HoG.map HoG.cmp map_0 map_1
proof

show $\bigwedge a.\ B.obj\ a \Longrightarrow D.ide\ (map_0\ a)$
  using map_0-*def* **by** *simp*
show $\bigwedge f.\ B.ide\ f \Longrightarrow D.iso\ (map_1\ f)$
proof −
  fix f
  assume f: $B.ide\ f$
  have $D.seq\ (H\ (\tau_1\ f))\ (\Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))$
    using f **by** (*intro D.seqI*) *auto*
  moreover have $D.seq\ (D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ f),\ F\ f)))$
                $(H\ (\tau_1\ f) \cdot_D \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))$
    using f $\tau.map_1$-*in-hom* [*of f*] *calculation* **by** (*intro D.seqI*) *auto*
  ultimately show $D.iso\ (map_1\ f)$
    using f map_1-*def H.preserves-iso* τ.*iso-*map_1-*ide H.cmp-components-are-iso*
      $C.VV.arr\text{-}char_{SbC}$ *D.isos-compose*
    by *auto*
qed
show $\bigwedge a.\ B.obj\ a \Longrightarrow$ «$map_0\ a : src_D\ ((H \circ F)\ a) \rightarrow_D src_D\ ((H \circ G)\ a)$»
  using map_0-*def* **by** *fastforce*
show $\bigwedge f.\ B.ide\ f \Longrightarrow$
      «$map_1\ f : (H \circ G)\ f \star_D map_0\ (src_B\ f) \Rightarrow_D map_0\ (trg_B\ f) \star_D (H \circ F)\ f$»
  using map_0-*def* map_1-*def* **by** *fastforce*
show $\bigwedge \mu.\ B.arr\ \mu \Longrightarrow$
      $map_1\ (B.cod\ \mu) \cdot_D ((H \circ G)\ \mu \star_D map_0\ (src_B\ \mu))$
        $= (map_0\ (trg_B\ \mu) \star_D (H \circ F)\ \mu) \cdot_D map_1\ (B.dom\ \mu)$
proof −
  fix μ
  assume μ: $B.arr\ \mu$
  have $(map_0\ (trg_B\ \mu) \star_D (H \circ F)\ \mu) \cdot_D map_1\ (B.dom\ \mu)$
      $= (H\ (\tau_0\ (trg_B\ \mu)) \star_D (H \circ F)\ \mu) \cdot_D$
        $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ (B.dom\ \mu)),\ F\ (B.dom\ \mu))) \cdot_D$
        $H\ (\tau_1\ (B.dom\ \mu)) \cdot_D \Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ (B.dom\ \mu)))$
    unfolding map_0-*def* map_1-*def*
    using μ **by** *simp*
  also have ... $= ((H\ (\tau_0\ (trg_B\ \mu)) \star_D H\ (F\ \mu)) \cdot_D$
              $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.dom\ \mu)))) \cdot_D$
      $H\ (\tau_1\ (B.dom\ \mu)) \cdot_D \Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ \mu))$
    using μ *D.comp-assoc* **by** *simp*
  also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D$
              $(H\ (\tau_0\ (trg_B\ \mu) \star_C F\ \mu) \cdot_D$
              $H\ (\tau_1\ (B.dom\ \mu))) \cdot_D$
              $\Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ (B.dom\ \mu)))$
  proof −
    have $\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu)) \cdot_D (H\ (\tau_0\ (trg_B\ \mu)) \star_D H\ (F\ \mu))$
        $= H\ (\tau_0\ (trg_B\ \mu) \star_C F\ \mu) \cdot_D \Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.dom\ \mu))$
      using μ $H.\Phi.naturality$ [*of* $(\tau_0\ (trg_B\ \mu),\ F\ \mu)$]
        $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *H.FF-def*
      by *simp*
    moreover have $D.seq\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu)))\ (H\ (\tau_0\ (trg_B\ \mu)) \star_D H\ (F\ \mu))$
      using μ

by (*intro D.seqI D.hseqI′*) *auto*
moreover have $D.iso\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu)))$
using μ *H.cmp-components-are-iso* **by** *simp*
moreover have $D.iso\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.dom\ \mu)))$
using μ *H.cmp-components-are-iso* **by** *simp*
ultimately
have $(H\ (\tau_0\ (trg_B\ \mu)) \star_D H\ (F\ \mu)) \cdot_D D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.dom\ \mu)))$
$= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D H\ (\tau_0\ (trg_B\ \mu) \star_C F\ \mu)$
using μ *H.cmp-components-are-iso* $C.VV.arr\text{-}char_{SbC}$ *D.invert-opposite-sides-of-square*
by *blast*
thus *?thesis*
using μ *D.comp-assoc* **by** *simp*
qed
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D$
$H\ ((\tau_0\ (trg_B\ \mu) \star_C F\ \mu) \cdot_C \tau_1\ (B.dom\ \mu)) \cdot_D$
$\Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ (B.dom\ \mu)))$
using μ **by** *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D$
$H\ (\tau_1\ (B.cod\ \mu) \cdot_C (G\ \mu \star_C \tau_0\ (src_B\ \mu))) \cdot_D$
$\Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ (B.dom\ \mu)))$
using μ τ*.naturality* **by** *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D$
$H\ (\tau_1\ (B.cod\ \mu)) \cdot_D$
$H\ (G\ \mu \star_C \tau_0\ (src_B\ \mu)) \cdot_D$
$\Phi_H\ (G\ (B.dom\ \mu),\ \tau_0\ (src_B\ (B.dom\ \mu)))$
using μ *D.comp-assoc* **by** *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ \mu),\ F\ (B.cod\ \mu))) \cdot_D$
$H\ (\tau_1\ (B.cod\ \mu)) \cdot_D$
$\Phi_H\ (G\ (B.cod\ \mu),\ \tau_0\ (src_B\ (B.cod\ \mu))) \cdot_D$
$(H\ (G\ \mu) \star_D H\ (\tau_0\ (src_B\ \mu)))$
using μ $H.\Phi.naturality$ [*of* $(G\ \mu,\ \tau_0\ (src_B\ \mu))$]
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *H.FF-def*
by *force*
also have ... $= map_1\ (B.cod\ \mu) \cdot_D ((H \circ G)\ \mu \star_D map_0\ (src_B\ \mu))$
unfolding map_0*-def* map_1*-def*
using μ *D.comp-assoc* **by** *simp*
finally show $map_1\ (B.cod\ \mu) \cdot_D ((H \circ G)\ \mu \star_D map_0\ (src_B\ \mu))$
$= (map_0\ (trg_B\ \mu) \star_D (H \circ F)\ \mu) \cdot_D map_1\ (B.dom\ \mu)$
by *simp*
qed
show $\bigwedge a.\ B.obj\ a \Longrightarrow (map_0\ a \star_D HoF.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
$= map_1\ a \cdot_D (HoG.unit\ a \star_D map_0\ a)$
proof −
fix a
assume a: $B.obj\ a$
have $map_1\ a \cdot_D (HoG.unit\ a \star_D map_0\ a)$
$= D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$H\ (\tau_1\ a) \cdot_D$
$\Phi_H\ (G\ a,\ \tau_0\ a) \cdot_D$

$(H\ (\tau.G.unit\ a) \cdot_D H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
unfolding map_0-*def* map_1-*def*
using *a HoG.unit-char′ D.comp-assoc* **by** *auto*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$H\ (\tau_1\ a) \cdot_D$
$(\Phi_H\ (G\ a,\ \tau_0\ a) \cdot_D$
$(H\ (\tau.G.unit\ a) \star_D H\ (\tau_0\ a))) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
using *a D.whisker-right D.comp-assoc* **by** *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$(H\ (\tau_1\ a) \cdot_D$
$H\ (\tau.G.unit\ a \star_C \tau_0\ a)) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
using *a* $H.\Phi.naturality$ [*of* $(\tau.G.unit\ a,\ \tau_0\ a)$] *D.comp-assoc*
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *H.FF-def*
by *auto*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$H\ (\tau_1\ a \cdot_C (\tau.G.unit\ a \star_C \tau_0\ a)) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
proof −
have $C.arr\ (\tau_1\ a \cdot_C (\tau.G.unit\ a \star_C \tau_0\ a))$
using *a* **by** *force*
hence $H\ (\tau_1\ a) \cdot_D H\ (\tau.G.unit\ a \star_C \tau_0\ a) = H\ (\tau_1\ a \cdot_C (\tau.G.unit\ a \star_C \tau_0\ a))$
using *a* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$H\ ((\tau_0\ a \star_C \tau.F.unit\ a) \cdot_C \mathrm{r}_C{}^{-1}[\tau_0\ a] \cdot_C \mathrm{l}_C[\tau_0\ a]) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
using *a* $\tau.respects\text{-}unit$ **by** *simp*
also have ... $= (D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D$
$H\ (\tau_0\ a \star_C \tau.F.unit\ a)) \cdot_D$
$H\ (\mathrm{r}_C{}^{-1}[\tau_0\ a]) \cdot_D$
$H\ (\mathrm{l}_C[\tau_0\ a]) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
using *a D.comp-assoc B.obj-simps* **by** *simp*
also have ... $= (H\ (\tau_0\ a) \star_D H\ (\tau.F.unit\ a)) \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a)) \cdot_D$
$H\ (\mathrm{r}_C{}^{-1}[\tau_0\ a])) \cdot_D$
$H\ (\mathrm{l}_C[\tau_0\ a]) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \cdot_D H\ (\tau_0\ a \star_C \tau.F.unit\ a)$

$= (H\ (\tau_0\ a) \star_D H\ (\tau.F.unit\ a)) \cdot_D D.inv\ (\Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a))$
proof −
have $D.seq\ (H\ (\tau_0\ a \star_C \tau.F.unit\ a))\ (\Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a))$
using $a\ C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}char_{SbC}\ C.VV.cod\text{-}char_{SbC}\ C.obj\text{-}simps$ **by** *auto*
moreover have $D.iso\ (\Phi_H\ (\tau_0\ a,\ F\ a)) \wedge D.iso\ (\Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a))$
proof −
have $C.ide\ (F\ a) \wedge src_C\ (\tau_0\ a) = trg_C\ (F\ a) \wedge src_C\ (\tau_0\ a) = trg_C\ (\tau.F.map_0\ a)$
using a **by** *auto*
moreover have $C.ide\ (\tau.F.map_0\ a)$
proof −

have $C.obj\ (\tau.F.map_0\ a)$
using a **by** *simp*
thus *?thesis* **by** *auto*
qed
ultimately show *?thesis*
using $a\ H.cmp\text{-}components\text{-}are\text{-}iso\ B.obj\text{-}simps$ **by** *auto*
qed
ultimately show *?thesis*
using $a\ H.\Phi.naturality\ [of\ (\tau_0\ a,\ \tau.F.unit\ a)]$
$C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}char_{SbC}\ C.VV.cod\text{-}char_{SbC}\ H.FF\text{-}def$
$D.invert\text{-}opposite\text{-}sides\text{-}of\text{-}square$
$[of\ \Phi_H\ (\tau_0\ a,\ F\ a)\ H\ (\tau_0\ a) \star_D H\ (\tau.F.unit\ a)$
$H\ (\tau_0\ a \star_C \tau.F.unit\ a)\ \Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a)]$
by *auto*
qed
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= (H\ (\tau_0\ a) \star_D H\ (\tau.F.unit\ a)) \cdot_D$
$(H\ (\tau_0\ a) \star_D H.unit\ (\tau.F.map_0\ a)) \cdot_D$
$\mathrm{r}_D{}^{-1}[H\ (\tau_0\ a)] \cdot_D$
$(H\ (\mathrm{l}_C[\tau_0\ a]) \cdot_D$
$\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a)) \cdot_D$
$(H.unit\ (\tau.G.map_0\ a) \star_D H\ (\tau_0\ a))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ a,\ \tau.F.map_0\ a)) \cdot_D H\ (\mathrm{r}_C{}^{-1}[\tau_0\ a])$
$= (H\ (\tau_0\ a) \star_D H.unit\ (\tau.F.map_0\ a)) \cdot_D \mathrm{r}_D{}^{-1}[H\ (\tau_0\ a)]$
using $a\ H.preserves\text{-}runit(2)\ [of\ \tau_0\ a]\ D.comp\text{-}assoc$
by $(metis\ C.ideD(1)\ C.runit'\text{-}simps(1)\ C.src.preserves\text{-}ide\ C.trg\text{-}src$
$D.invert\text{-}side\text{-}of\text{-}triangle(1)\ \tau.F.map_0\text{-}def\ H.cmp\text{-}components\text{-}are\text{-}iso$
$H.preserves\text{-}reflects\text{-}arr\ \tau.ide\text{-}map_0\text{-}obj\ \tau.map_0\text{-}simps(2))$
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= (H\ (\tau_0\ a) \star_D H\ (\tau.F.unit\ a)) \cdot_D$
$(H\ (\tau_0\ a) \star_D H.unit\ (\tau.F.map_0\ a)) \cdot_D$
$\mathrm{r}_D{}^{-1}[H\ (\tau_0\ a)] \cdot_D$

$\mathrm{l}_D[H\ (\tau_0\ a)]\ \cdot_D$
$(D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))\ \cdot_D$
$(H.unit\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a))$

proof −
have $H\ (\mathrm{l}_C[\tau_0\ a])\ \cdot_D\ \Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a)$
$= \mathrm{l}_D[H\ (\tau_0\ a)]\ \cdot_D\ (D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))\ \cdot_D$
$D.inv\ (\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a))\ \cdot_D\ \Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a)$
using *a H.preserves-lunit(1) D.comp-assoc* **by** *auto*
also have ... $= \mathrm{l}_D[H\ (\tau_0\ a)]\ \cdot_D\ (D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))$
proof −
have $D.inv\ (\Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a))\ \cdot_D\ \Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a)$
$= H\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)$
using *a H.cmp-components-are-iso D.comp-inv-arr′*
by (*metis C.isomorphic-implies-hpar(1) τ.G.map₀-simps(2) τ.G.weakly-preserves-objects H.cmp-simps(4) τ.ide-map₀-obj τ.map₀-simps(3)*)
moreover have $(D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))\ \cdot_D$
$(H\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a))$
$= (D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))$
using *a H.unit-char(2) D.comp-arr-dom*
by (*metis D.arr-inv D.dom-inv D.whisker-right τ.G.map₀-simps(1) H.unit-simps(5) H.preserves-ide τ.ide-map₀-obj*)
ultimately show *?thesis* **by** *simp*
qed
finally have $H\ (\mathrm{l}_C[\tau_0\ a])\ \cdot_D\ \Phi_H\ (\tau.G.map_0\ a,\ \tau_0\ a)$
$= \mathrm{l}_D[H\ (\tau_0\ a)]\ \cdot_D\ (D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((H\ (\tau_0\ a)\ \star_D\ H\ (\tau.F.unit\ a))\ \cdot_D$
$(H\ (\tau_0\ a)\ \star_D\ H.unit\ (\tau.F.map_0\ a)))\ \cdot_D$
$\mathrm{r}_D{}^{-1}[H\ (\tau_0\ a)]\ \cdot_D$
$\mathrm{l}_D[H\ (\tau_0\ a)]$

proof −
have $(D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))\ \cdot_D$
$(H.unit\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a))$
$= D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \cdot_D\ H.unit\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)$
using *a D.whisker-right H.unit-char(2)* **by** *simp*
also have ... $= H.map_0\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)$
using *a H.unit-char(1−2) [of τ.G.map₀ a] D.comp-inv-arr′* **by** *simp*
finally have $(D.inv\ (H.unit\ (\tau.G.map_0\ a))\ \star_D\ H\ (\tau_0\ a))\ \cdot_D$
$(H.unit\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a))$
$= H.map_0\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)$
by *blast*
moreover have $\mathrm{l}_D[H\ (\tau_0\ a)]\ \cdot_D\ (H.map_0\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)) = \mathrm{l}_D[H\ (\tau_0\ a)]$
using *a D.comp-arr-dom [of* $\mathrm{l}_D[H\ (\tau_0\ a)]\ H.map_0\ (\tau.G.map_0\ a)\ \star_D\ H\ (\tau_0\ a)$*]*
by *auto*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= (H\ (\tau_0\ a) \star_D HoF.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[H\ (\tau_0\ a)] \cdot_D \mathrm{l}_D[H\ (\tau_0\ a)]$
using *a HoF.unit-char′ D.whisker-left* [*of* $H\ (\tau_0\ a)$] **by** *simp*
also have ... $= (map_0\ a \star_D HoF.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a]$
unfolding map_0*-def* **by** *simp*
finally show $(map_0\ a \star_D HoF.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[map_0\ a] \cdot_D \mathrm{l}_D[map_0\ a] =$
$map_1\ a \cdot_D (HoG.unit\ a \star_D map_0\ a)$
by *simp*
qed
show $\bigwedge f\ g.\ [\![B.ide\ f;\ B.ide\ g;\ src_B\ g = trg_B\ f]\!] \Longrightarrow$
$(map_0\ (trg_B\ g) \star_D HoF.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_B\ g), (H \circ F)\ g, (H \circ F)\ f] \cdot_D$
$(map_1\ g \star_D (H \circ F)\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[(H \circ G)\ g, map_0\ (src_B\ g), (H \circ F)\ f] \cdot_D$
$((H \circ G)\ g \star_D map_1\ f) \cdot_D \mathrm{a}_D[(H \circ G)\ g, (H \circ G)\ f, map_0\ (src_B\ f)]$
$= map_1\ (g \star_B f) \cdot_D (HoG.cmp\ (g, f) \star_D map_0\ (src_B\ f))$
proof −
fix *f g*
assume *f*: $B.ide\ f$ **and** *g*: $B.ide\ g$ **and** *fg*: $src_B\ g = trg_B\ f$
have $map_1\ (g \star_B f) \cdot_D (HoG.cmp\ (g, f) \star_D map_0\ (src_B\ f))$
$= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ (\tau_1\ (g \star_B f)) \cdot_D$
$\Phi_H\ (G\ (g \star_B f), \tau_0\ (src_B\ f)) \cdot_D$
$(H\ (G\ (g \star_B f)) \cdot_D$
$H\ (\Phi_G\ (g, f)) \cdot_D$
$\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
unfolding map_0*-def* map_1*-def HoG.cmp-def*
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *D.comp-assoc*
by *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ (\tau_1\ (g \star_B f)) \cdot_D$
$\Phi_H\ (G\ (g \star_B f), \tau_0\ (src_B\ f)) \cdot_D$
$((H\ (G\ (g \star_B f)) \star_D H\ (\tau_0\ (src_B\ f))) \cdot_D$
$(H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f)))) \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $\tau.G.FF\text{-}def$
D.comp-assoc D.whisker-right
by *auto*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ (\tau_1\ (g \star_B f)) \cdot_D$
$(\Phi_H\ (G\ (g \star_B f), \tau_0\ (src_B\ f)) \cdot_D$
$(H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f)))) \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $(H\ (G\ (g \star_B f)) \star_D H\ (\tau_0\ (src_B\ f))) \cdot_D (H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f)))$
$= H\ (G\ (g \star_B f)) \cdot_D H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f))$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *D.whisker-right*
by *simp*

also have ... $= H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f))$
using *f g fg D.comp-cod-arr* [*of* $H\ (\Phi_G\ (g, f))\ H\ (G\ (g \star_B f))$]
B.VV.arr-char$_{SbC}$ *B.VV.dom-char*$_{SbC}$ *B.VV.cod-char*$_{SbC}$
by *simp*
finally have $(H\ (G\ (g \star_B f)) \star_D H\ (\tau_0\ (src_B\ f))) \cdot_D$
$(H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f)))$
$= H\ (\Phi_G\ (g, f)) \star_D H\ (\tau_0\ (src_B\ f))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$(H\ (\tau_1\ (g \star_B f)) \cdot_D$
$H\ (\Phi_G\ (g, f) \star_C \tau_0\ (src_B\ f))) \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
using *f g fg H.*Φ*.naturality* [*of* $(\Phi_G\ (g, f), \tau_0\ (src_B\ f))$]
B.VV.arr-char$_{SbC}$ *B.VV.dom-char*$_{SbC}$ *B.VV.cod-char*$_{SbC}$ τ*.G.FF-def*
C.VV.arr-char$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *H.FF-def*
D.comp-assoc
by *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ (\tau_1\ (g \star_B f) \cdot_C (\Phi_G\ (g, f) \star_C \tau_0\ (src_B\ f))) \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
using *f g fg B.VV.arr-char*$_{SbC}$ *B.VV.dom-char*$_{SbC}$ *B.VV.cod-char*$_{SbC}$ *D.comp-assoc*
by *simp*
also have ... $= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ ((\tau_0\ (trg_B\ g) \star_C \Phi_F\ (g, f)) \cdot_C$
$\mathrm{a}_C[\tau_0\ (trg_B\ g), F\ g, F\ f] \cdot_C$
$(\tau_1\ g \star_C F\ f) \cdot_C$
$\mathrm{a}_C{}^{-1}[G\ g, \tau_0\ (src_B\ g), F\ f] \cdot_C$
$(G\ g \star_C \tau_1\ f) \cdot_C$
$\mathrm{a}_C[G\ g, G\ f, \tau_0\ (src_B\ f)]) \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
using *f g fg* τ*.respects-hcomp* **by** *simp*
also have ... $= (D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D$
$H\ (\tau_0\ (trg_B\ g) \star_C \Phi_F\ (g, f))) \cdot_D$
$H\ \mathrm{a}_C[\tau_0\ (trg_B\ g), F\ g, F\ f] \cdot_D$
$H\ (\tau_1\ g \star_C F\ f) \cdot_D$
$H\ (\mathrm{a}_C{}^{-1}[G\ g, \tau_0\ (src_B\ g), F\ f]) \cdot_D$
$H\ (G\ g \star_C \tau_1\ f) \cdot_D$
$H\ \mathrm{a}_C[G\ g, G\ f, \tau_0\ (src_B\ f)] \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
using *f g fg B.VV.arr-char*$_{SbC}$ *B.VV.dom-char*$_{SbC}$ *B.VV.cod-char*$_{SbC}$ τ*.F.FF-def*
D.comp-assoc
by *simp*

also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g \star_C F\ f)) \cdot_D$
$H\ \mathrm{a}_C[\tau_0\ (trg_B\ g), F\ g, F\ f]) \cdot_D$
$H\ (\tau_1\ g \star_C F\ f) \cdot_D$
$H\ (\mathrm{a}_C{}^{-1}[G\ g, \tau_0\ (src_B\ g), F\ f]) \cdot_D$
$H\ (G\ g \star_C \tau_1\ f) \cdot_D$
$H\ \mathrm{a}_C[G\ g, G\ f, \tau_0\ (src_B\ f)] \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \cdot_D H\ (\tau_0\ (trg_B\ g) \star_C \Phi_F\ (g, f))$
$= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g \star_C F\ f))$
proof −
have $\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f)) \cdot_D (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) =$
$H\ (\tau_0\ (trg_B\ g) \star_C \Phi_F\ (g, f)) \cdot_D \Phi_H\ (\tau_0\ (trg_B\ g), F\ g \star_C F\ f)$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $\tau.F.FF\text{-}def$ $H.FF\text{-}def$
$H.\Phi.naturality$ [*of* $(\tau_0\ (trg_B\ g), \Phi_F\ (g, f))$]
by *auto*
moreover have $D.seq\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f)))\ (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g,$
$f)))$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $H.FF\text{-}def$
by *auto*
moreover have $D.iso\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ (g \star_B f))) \wedge$
$D.iso\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g \star_C F\ f))$
using *f g fg* $H.cmp\text{-}components\text{-}are\text{-}iso$ **by** *simp*
ultimately show *?thesis*
using *f g fg* $D.invert\text{-}opposite\text{-}sides\text{-}of\text{-}square$ **by** *simp*
qed
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g, F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)), H\ (F\ g), H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g, F\ f)) \cdot_D$
$H\ (\tau_1\ g \star_C F\ f)) \cdot_D$
$H\ (\mathrm{a}_C{}^{-1}[G\ g, \tau_0\ (src_B\ g), F\ f]) \cdot_D$
$H\ (G\ g \star_C \tau_1\ f) \cdot_D$
$H\ \mathrm{a}_C[G\ g, G\ f, \tau_0\ (src_B\ f)] \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g \star_C F\ f)) \cdot_D H\ \mathrm{a}_C[\tau_0\ (trg_B\ g), F\ g, F\ f]$
$= (H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g, F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)), H\ (F\ g), H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g)) \star_D H\ (F\ f)) \cdot_D$

$D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g \star_C F\ f)) \cdot_D H\ \mathrm{a}_C[\tau_0\ (trg_B\ g),\ F\ g,\ F\ f]$
$= ((D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g \star_C F\ f)) \cdot_D$
$\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g \star_C F\ f)) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f))) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f))$
using *f g fg H.preserves-assoc(1) D.comp-assoc* **by** *simp*
also have ... $= ((H\ (\tau_0\ (trg_B\ g)) \star_D H\ (F\ g \star_C F\ f)) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f))) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f))$
using *f g fg B.VV.arr-char*$_{SbC}$ *B.VV.dom-char*$_{SbC}$ *B.VV.cod-char*$_{SbC}$
H.FF-def D.comp-assoc D.comp-inv-arr′ H.cmp-components-are-iso
by *simp*
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f))$
using *f g fg C.VV.arr-char*$_{SbC}$ *H.cmp-simps(5) D.comp-cod-arr D.comp-assoc* **by** *auto*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g,\ f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D$
$H\ (\mathrm{a}_C{}^{-1}[G\ g,\ \tau_0\ (src_B\ g),\ F\ f])) \cdot_D$
$H\ (G\ g \star_C \tau_1\ f) \cdot_D$
$H\ \mathrm{a}_C[G\ g,\ G\ f,\ \tau_0\ (src_B\ f)] \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f,\ \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f)) \cdot_D H\ (\tau_1\ g \star_C F\ f)$
$= (H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f))$
proof −
have $\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f) \cdot_D (H\ (\tau_1\ g) \star_D H\ (F\ f))$
$= H\ (\tau_1\ g \star_C F\ f) \cdot_D \Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)$
using *f g fg H.Φ.naturality* [*of* $(\tau_1\ g,\ F\ f)$]
C.VV.arr-char$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *H.FF-def*
by *simp*

moreover have $D.seq\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f))$
$(H\ (\tau_1\ g) \star_D H\ (F\ f))$
using *f g fg* **by** *fastforce*
moreover have $D.iso\ (\Phi_H\ (\tau_0\ (trg_B\ g) \star_C F\ g,\ F\ f)) \wedge$
$D.iso\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f))$
using *f g fg H.cmp-components-are-iso* **by** *simp*
ultimately show *?thesis*
using *f g fg D.invert-opposite-sides-of-square* **by** *metis*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g,\ f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D$
$(\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g) \star_C F\ f)) \cdot_D$
$H\ (G\ g \star_C \tau_1\ f)) \cdot_D$
$H\ \mathrm{a}_C[G\ g,\ G\ f,\ \tau_0\ (src_B\ f)] \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f,\ \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D H\ (\mathrm{a}_C{}^{-1}[G\ g,\ \tau_0\ (src_B\ g),\ F\ f])$
$= (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$D.inv\ (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g) \star_C F\ f))$
proof −
have $D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D H\ (\mathrm{a}_C{}^{-1}[G\ g,\ \tau_0\ (src_B\ g),\ F\ f])$
$= ((D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D$
$\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D$
$(\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f))) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$D.inv\ (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g) \star_C F\ f))$
using *f g fg H.preserves-assoc(2) D.comp-assoc* **by** *simp*
also have ... $= (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$D.inv\ (\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g) \star_C F\ f))$
proof −
have $D.inv\ (\Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)) \cdot_D \Phi_H\ (G\ g \star_C \tau_0\ (src_B\ g),\ F\ f)$
$= H\ (G\ g \star_C \tau_0\ (src_B\ g)) \star_D H\ (F\ f)$
using *f g fg H.cmp-components-are-iso D.comp-inv-arr′ H.FF-def*
$C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}char_{SbC}\ C.VV.cod\text{-}char_{SbC}$

by *simp*
moreover have $(H\ (G\ g \star_C \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D (\Phi_H\ (G\ g, \tau_0\ (src_B\ g)) \star_D H\ (F\ f))$
$= (\Phi_H\ (G\ g, \tau_0\ (src_B\ g)) \star_D H\ (F\ f))$
using *f g fg D.comp-cod-arr C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$*
by *simp*
ultimately show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g, F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)), H\ (F\ g), H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D$
$(\Phi_H\ (G\ g, \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g), H\ (\tau_0\ (src_B\ g)), H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g), F\ f))) \cdot_D$
$(H\ (G\ g) \star_D H\ (\tau_1\ f)) \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g, G\ f \star_C \tau_0\ (src_B\ f))) \cdot_D$
$H\ \mathrm{a}_C[G\ g, G\ f, \tau_0\ (src_B\ f)]) \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f, \tau_0\ (src_B\ f)) \cdot_D$
$(\Phi_H\ (G\ g, G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
proof −
have $D.inv\ (\Phi_H\ (G\ g, \tau_0\ (src_B\ g) \star_C F\ f)) \cdot_D H\ (G\ g \star_C \tau_1\ f)$
$= (H\ (G\ g) \star_D H\ (\tau_1\ f)) \cdot_D D.inv\ (\Phi_H\ (G\ g, G\ f \star_C \tau_0\ (src_B\ f)))$
proof −
have $\Phi_H\ (G\ g, \tau_0\ (trg_B\ f) \star_C F\ f) \cdot_D (H\ (G\ g) \star_D H\ (\tau_1\ f))$
$= H\ (G\ g \star_C \tau_1\ f) \cdot_D \Phi_H\ (G\ g, G\ f \star_C \tau_0\ (src_B\ f))$
using *f g fg H.Φ.naturality* [*of* $(G\ g, \tau_1\ f)$]
C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$ H.FF-def
by *simp*
thus *?thesis*
using *f g fg H.cmp-components-are-iso*
C.VV.arr-char$_{SbC}$ C.VV.dom-char$_{SbC}$ C.VV.cod-char$_{SbC}$
D.invert-opposite-sides-of-square
[*of* $\Phi_H\ (G\ g, \tau_0\ (trg_B\ f) \star_C F\ f)\ H\ (G\ g) \star_D H\ (\tau_1\ f)$
$H\ (G\ g \star_C \tau_1\ f)\ \Phi_H\ (G\ g, G\ f \star_C \tau_0\ (src_B\ f))$]
by *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g, F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)), H\ (F\ g), H\ (F\ f)] \cdot_D$

$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g))\ \star_D\ H\ (F\ f))\ \cdot_D$
$(H\ (\tau_1\ g)\ \star_D\ H\ (F\ f))\ \cdot_D$
$(\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g))\ \star_D\ H\ (F\ f))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)]\ \cdot_D$
$(H\ (G\ g)\ \star_D\ D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f)))\ \cdot_D$
$(H\ (G\ g)\ \star_D\ H\ (\tau_1\ f))\ \cdot_D$
$(H\ (G\ g)\ \star_D\ \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))\ \cdot_D$
$\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]\ \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g,\ G\ f))\ \star_D\ H\ (\tau_0\ (src_B\ f)))\ \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g\ \star_C\ G\ f,\ \tau_0\ (src_B\ f)))\ \cdot_D$
$\Phi_H\ (G\ g\ \star_C\ G\ f,\ \tau_0\ (src_B\ f)))\ \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f)\ \star_D\ H\ (\tau_0\ (src_B\ f)))$

proof −
have $D.inv\ (\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D\ H\ \mathrm{a}_C[G\ g,\ G\ f,\ \tau_0\ (src_B\ f)]$
$= (H\ (G\ g)\ \star_D\ \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))\ \cdot_D$
$\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]\ \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g,\ G\ f))\ \star_D\ H\ (\tau_0\ (src_B\ f)))\ \cdot_D$
$D.inv\ (\Phi_H\ (G\ g\ \star_C\ G\ f,\ \tau_0\ (src_B\ f)))$

proof −
have $D.inv\ (\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D\ H\ \mathrm{a}_C[G\ g,\ G\ f,\ \tau_0\ (src_B\ f)]$
$= ((D.inv\ (\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D$
$\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D$
$(H\ (G\ g)\ \star_D\ \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f))))\ \cdot_D$
$\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]\ \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g,\ G\ f))\ \star_D\ H\ (\tau_0\ (src_B\ f)))\ \cdot_D$
$D.inv\ (\Phi_H\ (G\ g\ \star_C\ G\ f,\ \tau_0\ (src_B\ f)))$
using *f g fg H.preserves-assoc(1)*
H.cmp-components-are-iso C.VV.arr-char$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$
D.comp-assoc
by *simp*
moreover have $D.inv\ (\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D$
$\Phi_H\ (G\ g,\ G\ f\ \star_C\ \tau_0\ (src_B\ f))$
$= H\ (G\ g)\ \star_D\ H\ (G\ f\ \star_C\ \tau_0\ (src_B\ f))$
using *f g fg H.cmp-components-are-iso C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$
D.comp-inv-arr′ H.FF-def
by *simp*
moreover have $(H\ (G\ g)\ \star_D\ H\ (G\ f\ \star_C\ \tau_0\ (src_B\ f)))\ \cdot_D$
$(H\ (G\ g)\ \star_D\ \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))$
$= H\ (G\ g)\ \star_D\ \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f))$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$
D.whisker-left D.comp-cod-arr
by *simp*
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((H\ (\tau_0\ (trg_B\ g))\ \star_D\ H\ (\Phi_F\ (g,\ f)))\ \cdot_D$

$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f))) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$((D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D$
$(\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f))) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$((H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$(H\ (G\ g) \star_D H\ (\tau_1\ f)) \cdot_D$
$(H\ (G\ g) \star_D \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))) \cdot_D$
$\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]$

proof −
have $\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))] \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g,\ G\ f)) \star_D H\ (\tau_0\ (src_B\ f))) \cdot_D$
$(D.inv\ (\Phi_H\ (G\ g \star_C G\ f,\ \tau_0\ (src_B\ f))) \cdot_D$
$\Phi_H\ (G\ g \star_C G\ f,\ \tau_0\ (src_B\ f))) \cdot_D$
$(\Phi_H\ (G\ g,\ G\ f) \star_D H\ (\tau_0\ (src_B\ f)))$
$= \mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.whisker-right [*of* $H\ (\tau_0\ (src_B\ f))$ $D.inv\ (\Phi_H\ (G\ g,\ G\ f))$ $\Phi_H\ (G\ g,\ G\ f)$]
D.comp-inv-arr′ C.VV.arr-char$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *H.FF-def*
H.cmp-components-are-iso
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= (H\ (\tau_0\ (trg_B\ g)) \star_D H\ (F\ (g \star_B f)) \cdot_D$
$H\ (\Phi_F\ (g, f)) \cdot_D$
$\Phi_H\ (F\ g,\ F\ f)) \cdot_D$
$\mathrm{a}_D[H\ (\tau_0\ (trg_B\ g)),\ H\ (F\ g),\ H\ (F\ f)] \cdot_D$
$(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g),\ F\ g)) \cdot_D$
$H\ (\tau_1\ g) \cdot_D$
$\Phi_H\ (G\ g,\ \tau_0\ (src_B\ g)) \star_D H\ (F\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[H\ (G\ g),\ H\ (\tau_0\ (src_B\ g)),\ H\ (F\ f)] \cdot_D$
$(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f)) \cdot_D$
$H\ (\tau_1\ f) \cdot_D$
$\Phi_H\ (G\ f,\ \tau_0\ (src_B\ f))) \cdot_D$
$\mathrm{a}_D[H\ (G\ g),\ H\ (G\ f),\ H\ (\tau_0\ (src_B\ f))]$

proof −
have $(H\ (G\ g) \star_D D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f))) \cdot_D$
$(H\ (G\ g) \star_D H\ (\tau_1\ f)) \cdot_D$
$(H\ (G\ g) \star_D \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f)))$
$= H\ (G\ g) \star_D$
$D.inv\ (\Phi_H\ (\tau_0\ (src_B\ g),\ F\ f)) \cdot_D H\ (\tau_1\ f) \cdot_D \Phi_H\ (G\ f,\ \tau_0\ (src_B\ f))$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-char*$_{SbC}$ *C.VV.cod-char*$_{SbC}$ *D.whisker-left*
H.cmp-components-are-iso
by *simp*
moreover have $(H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D$
$(H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g,\ F\ f))$

$= H\ (\tau_0\ (trg_B\ g)) \star_D$
$H\ (F\ (g \star_B f)) \cdot_D H\ (\Phi_F\ (g, f)) \cdot_D \Phi_H\ (F\ g, F\ f)$
proof −
have $(H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f))) \cdot_D (H\ (\tau_0\ (trg_B\ g)) \star_D \Phi_H\ (F\ g, F\ f))$
$= H\ (\tau_0\ (trg_B\ g)) \star_D H\ (\Phi_F\ (g, f)) \cdot_D \Phi_H\ (F\ g, F\ f)$
proof −
have $D.arr\ (H\ (\Phi_F\ (g, f)) \cdot_D \Phi_H\ (F\ g, F\ f))$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ $\tau.F.FF\text{-}def$
by (*intro* $D.seqI$) *auto*
thus *?thesis*
using *f g fg* $D.whisker\text{-}left$ $C.VV.arr\text{-}char_{SbC}$ **by** *simp*
qed
also have ... $= H\ (\tau_0\ (trg_B\ g)) \star_D$
$(H\ (F\ (g \star_B f) \cdot_C \Phi_F\ (g, f))) \cdot_D \Phi_H\ (F\ g, F\ f)$
using *f g fg* $C.comp\text{-}cod\text{-}arr$ $B.VV.arr\text{-}char_{SbC}$ $\tau.F.cmp\text{-}simps(5)$ **by** *auto*
also have ... $= H\ (\tau_0\ (trg_B\ g)) \star_D$
$(H\ (F\ (g \star_B f)) \cdot_D H\ (\Phi_F\ (g, f))) \cdot_D \Phi_H\ (F\ g, F\ f)$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $\tau.F.cmp\text{-}simps(5)$ **by** *auto*
also have ... $= H\ (\tau_0\ (trg_B\ g)) \star_D$
$H\ (F\ (g \star_B f)) \cdot_D H\ (\Phi_F\ (g, f)) \cdot_D \Phi_H\ (F\ g, F\ f)$
using $D.comp\text{-}assoc$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
moreover have $(D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g)) \star_D H\ (F\ f)) \cdot_D$
$(H\ (\tau_1\ g) \star_D H\ (F\ f)) \cdot_D$
$(\Phi_H\ (G\ g, \tau_0\ (src_B\ g)) \star_D H\ (F\ f))$
$= D.inv\ (\Phi_H\ (\tau_0\ (trg_B\ g), F\ g)) \cdot_D$
$H\ (\tau_1\ g) \cdot_D \Phi_H\ (G\ g, \tau_0\ (src_B\ g)) \star_D H\ (F\ f)$
using *f g fg* $D.whisker\text{-}right$ $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
ultimately show *?thesis*
by *simp*
qed
also have ... $= (map_0\ (trg_B\ g) \star_D HoF.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_B\ g), (H \circ F)\ g, (H \circ F)\ f] \cdot_D$
$(map_1\ g \star_D (H \circ F)\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[(H \circ G)\ g, map_0\ (src_B\ g), (H \circ F)\ f] \cdot_D$
$((H \circ G)\ g \star_D map_1\ f) \cdot_D \mathrm{a}_D[(H \circ G)\ g, (H \circ G)\ f, map_0\ (src_B\ f)]$
unfolding $map_0\text{-}def$ $map_1\text{-}def$ $HoF.cmp\text{-}def$
using *f g fg* $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ **by** *simp*
finally show $(map_0\ (trg_B\ g) \star_D HoF.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[map_0\ (trg_B\ g), (H \circ F)\ g, (H \circ F)\ f] \cdot_D$
$(map_1\ g \star_D (H \circ F)\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[(H \circ G)\ g, map_0\ (src_B\ g), (H \circ F)\ f] \cdot_D$
$((H \circ G)\ g \star_D map_1\ f) \cdot_D \mathrm{a}_D[(H \circ G)\ g, (H \circ G)\ f, map_0\ (src_B\ f)]$
$= map_1\ (g \star_B f) \cdot_D (HoG.cmp\ (g, f) \star_D map_0\ (src_B\ f))$
by *simp*
qed
qed

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
HoF.map HoF.cmp HoG.map HoG.cmp map_0 map_1
..

end

1.14.5 Pseudonatural Equivalences

A *pseudonatural equivalence* is a pseudonatural transformation whose components at objects are equivalence maps. Pseudonatural equivalences between pseudofunctors generalize natural isomorphisms between ordinary functors.

locale *pseudonatural-equivalence* =
pseudonatural-transformation +
assumes *components-are-equivalences*: *C.obj a* $\Longrightarrow$ *D.equivalence-map* (τ_0 *a*)

Identity Transformations are Pseudonatural Equivalences

sublocale *identity-pseudonatural-transformation* $\subseteq$
pseudonatural-equivalence V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ_F F Φ_F map_0 map_1
by *unfold-locales* (*simp add*: *D.obj-is-equivalence-map*)

Composition of Pseudonatural Equivalences

locale *composite-pseudonatural-equivalence* =
composite-pseudonatural-transformation +
σ: *pseudonatural-equivalence* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ_F G Φ_G σ_0 σ_1 +
ϱ: *pseudonatural-equivalence* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
G Φ_G H Φ_H ϱ_0 ϱ_1
begin

sublocale *pseudonatural-equivalence* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ_F H Φ_H map_0 map_1
apply *unfold-locales*
by (*metis D.equivalence-maps-compose D.hcomp-in-hhomE* ϱ*.components-are-equivalences*
σ*.components-are-equivalences* map_0*-def* map_0*-in-hom*(*1*))

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ_F H Φ_H map_0 map_1
..

end

locale *pseudonatural-equivalence-whisker-right* =
pseudonatural-transformation-whisker-right +

τ: *pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1
begin

interpretation *FoH*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D H Φ_H F Φ_F
..
interpretation *GoH*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D H Φ_H G Φ_G
..

sublocale *pseudonatural-equivalence* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
‹*F o H*› *FoH.cmp* ‹*G o H*› *GoH.cmp* map_0 map_1
using map_0*-def* τ*.components-are-equivalences*
by *unfold-locales simp*

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
(*F o H*) *FoH.cmp* (*G o H*) *GoH.cmp* map_0 map_1
..

end

locale *pseudonatural-equivalence-whisker-left* =
pseudonatural-transformation-whisker-left +
τ: *pseudonatural-equivalence*
V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F G Φ_G τ_0 τ_1
begin

interpretation *HoF*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F H Φ_H
..
interpretation *HoG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G H Φ_H
..

sublocale *pseudonatural-equivalence* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
‹*H o F*› *HoF.cmp* ‹*H o G*› *HoG.cmp* map_0 map_1
using map_0*-def* τ*.components-are-equivalences H.preserves-equivalence-maps*
by *unfold-locales simp*

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
(*H o F*) *HoF.cmp* (*H o G*) *HoG.cmp* map_0 map_1
..

end

Converse of a Pseudonatural Equivalence

It is easy to see that natural isomorphism between ordinary functors is a symmetric relation because a unique inverse to a natural isomorphism is obtained merely by inverting the components. However the situation is more difficult for pseudonatural equivalences because they do not have unique inverses. Instead, we have to choose a quasi-inverse for each of the components. In order to satisfy the required coherence conditions, it is necessary for these quasi-inverses to be part of chosen adjoint equivalences. Some long calculations to establish the coherence conditions seem unavoidable. The purpose of this section is to carry out the construction, given a pseudonatural equivalence, of a "converse" pseudonatural equivalence in the opposite direction.

locale *converse-pseudonatural-equivalence* =
  C: *bicategory* V_C H_C a_C i_C src_C trg_C +
  D: *bicategory* V_D H_D a_D i_D src_D trg_D +
  F: *pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F +
  G: *pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G +
  τ: *pseudonatural-equivalence*
begin

  abbreviation (*input*) F_0
  where $F_0 \equiv F.map_0$

  abbreviation (*input*) G_0
  where $G_0 \equiv G.map_0$

  definition map_0
  where $map_0\ a = (SOME\ g.\ \exists \eta\ \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (\tau_0\ a)\ g\ \eta\ \varepsilon)$

  abbreviation (*input*) $\tau_0{}'$
  where $\tau_0{}' \equiv map_0$

  definition *unit*
  where $unit\ a = (SOME\ \eta.\ \exists \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (\tau_0\ a)\ (\tau_0{}'\ a)\ \eta\ \varepsilon)$

  abbreviation (*input*) η
  where $\eta \equiv unit$

  definition *counit*
  where $counit\ a = (SOME\ \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (\tau_0\ a)\ (\tau_0{}'\ a)\ (\eta\ a)\ \varepsilon)$

  abbreviation (*input*) ε
  where $\varepsilon \equiv counit$

  lemma *chosen-adjoint-equivalence*:
  assumes $C.obj\ a$

shows *adjoint-equivalence-in-bicategory* V_D H_D a_D i_D src_D trg_D $(\tau_0\ a)$ $(\tau_0{}'\ a)$ $(\eta\ a)$ $(\varepsilon\ a)$
  using *assms* τ*.components-are-equivalences* map_0*-def unit-def counit-def*
    D.obj-is-equivalence-map D.equivalence-map-extends-to-adjoint-equivalence
    someI-ex [*of* $\lambda g.\ \exists\eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory*
      V_D H_D a_D i_D src_D trg_D $(\tau_0\ a)$ g η ε]
    someI-ex [*of* $\lambda\eta.\ \exists\varepsilon.$ *adjoint-equivalence-in-bicategory*
      V_D H_D a_D i_D src_D trg_D $(\tau_0\ a)$ $(\tau_0{}'\ a)$ η ε]
    someI-ex [*of* $\lambda\varepsilon.$ *adjoint-equivalence-in-bicategory*
      V_D H_D a_D i_D src_D trg_D $(\tau_0\ a)$ $(\tau_0{}'\ a)$ $(\eta\ a)$ ε]
  by *simp*

lemma map_0*-in-hhom* [*intro*]:
assumes *C.obj a*
shows «$\tau_0{}'\ a : G_0\ a \rightarrow_D F_0\ a$»
proof –
  interpret *adjoint-equivalence-in-bicategory*
    V_D H_D a_D i_D src_D trg_D ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *?thesis*
    using *assms* τ*.*map_0*-in-hhom* [*of a*] *antipar* **by** *auto*
qed

lemma map_0*-simps* [*simp*]:
assumes *C.obj a*
shows *D.ide* $(\tau_0{}'\ a)$ **and** src_D $(\tau_0{}'\ a) = G_0\ a$ **and** trg_D $(\tau_0{}'\ a) = F_0\ a$
proof –
  interpret *adjoint-equivalence-in-bicategory*
    V_D H_D a_D i_D src_D trg_D ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *D.ide* $(\tau_0{}'\ a)$
    by *simp*
  show src_D $(\tau_0{}'\ a) = G_0\ a$
    using *assms* map_0*-in-hhom* **by** *blast*
  show trg_D $(\tau_0{}'\ a) = F_0\ a$
    using *assms* map_0*-in-hhom* **by** *blast*
qed

lemma *equivalence-map-*map_0 [*simp*]:
assumes *C.obj a*
shows *D.equivalence-map* $(\tau_0{}'\ a)$
proof –
  interpret *adjoint-equivalence-in-bicategory*
    V_D H_D a_D i_D src_D trg_D ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *D.equivalence-map* $(\tau_0{}'\ a)$
    using *D.equivalence-map-def adjoint-equivalence-in-bicategory-axioms*
      dual-equivalence
    by *blast*
qed

lemma *unit-in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\eta\ a : F_0\ a \rightarrow_D F_0\ a$»
and «$\eta\ a : F_0\ a \Rightarrow_D \tau_0{}'\ a \star_D \tau_0\ a$»
proof −
  interpret *adjoint-equivalence-in-bicategory*
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show «$\eta\ a : F_0\ a \rightarrow_D F_0\ a$»
    using *assms unit-in-hom antipar*(*1*) **by** *auto*
  show «$\eta\ a : F_0\ a \Rightarrow_D \tau_0{}'\ a \star_D \tau_0\ a$»
    using *assms unit-in-hom antipar*(*1*) **by** *auto*
qed

lemma *unit-simps* [*simp*]:
assumes $C.obj\ a$
shows $D.iso\ (\eta\ a)$ **and** $D.arr\ (\eta\ a)$
and $src_D\ (\eta\ a) = F_0\ a$ **and** $trg_D\ (\eta\ a) = F_0\ a$
and $D.dom\ (\eta\ a) = F_0\ a$ **and** $D.cod\ (\eta\ a) = \tau_0{}'\ a \star_D \tau_0\ a$
proof −
  interpret *adjoint-equivalence-in-bicategory*
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show $D.iso\ (\eta\ a)$ **and** $D.arr\ (\eta\ a)$
  and $src_D\ (\eta\ a) = F_0\ a$ **and** $trg_D\ (\eta\ a) = F_0\ a$
  and $D.dom\ (\eta\ a) = F_0\ a$ **and** $D.cod\ (\eta\ a) = \tau_0{}'\ a \star_D \tau_0\ a$
    using *assms* **by** *auto*
qed

lemma *iso-unit*:
assumes $C.obj\ a$
shows $D.iso\ (\eta\ a)$
proof −
  interpret *adjoint-equivalence-in-bicategory*
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *?thesis* **by** *simp*
qed

lemma *counit-in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\varepsilon\ a : G_0\ a \rightarrow_D G_0\ a$»
and «$\varepsilon\ a : \tau_0\ a \star_D \tau_0{}'\ a \Rightarrow_D G_0\ a$»
proof −
  interpret *adjoint-equivalence-in-bicategory*
    $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show «$\varepsilon\ a : G_0\ a \rightarrow_D G_0\ a$»

using *assms counit-in-hom antipar*(*2*) **by** *auto*
show «$\varepsilon\ a : \tau_0\ a \star_D \tau_0{}'\ a \Rightarrow_D G_0\ a$»
using *assms counit-in-hom antipar*(*2*) **by** *simp*
qed

lemma *counit-simps* [*simp*]:
assumes $C.obj\ a$
shows $D.iso\ (\varepsilon\ a)$ **and** $D.arr\ (\varepsilon\ a)$
and $src_D\ (\varepsilon\ a) = G_0\ a$ **and** $trg_D\ (\varepsilon\ a) = G_0\ a$
and $D.dom\ (\varepsilon\ a) = \tau_0\ a \star_D \tau_0{}'\ a$ **and** $D.cod\ (\varepsilon\ a) = G_0\ a$
proof −
interpret *adjoint-equivalence-in-bicategory*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
using *assms chosen-adjoint-equivalence* **by** *simp*
show $D.iso\ (\varepsilon\ a)$ **and** $D.arr\ (\varepsilon\ a)$
and $src_D\ (\varepsilon\ a) = G_0\ a$ **and** $trg_D\ (\varepsilon\ a) = G_0\ a$
and $D.dom\ (\varepsilon\ a) = \tau_0\ a \star_D \tau_0{}'\ a$ **and** $D.cod\ (\varepsilon\ a) = G_0\ a$
using *assms* **by** *auto*
qed

lemma *iso-counit*:
assumes $C.obj\ a$
shows $D.iso\ (\varepsilon\ a)$
proof −
interpret *adjoint-equivalence-in-bicategory*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
using *assms chosen-adjoint-equivalence* **by** *simp*
show *?thesis* **by** *simp*
qed

lemma *quasi-inverts-components*:
assumes $C.obj\ a$
shows $D.isomorphic\ (\tau_0{}'\ a \star_D \tau_0\ a)\ (F_0\ a)$
and $D.isomorphic\ (\tau_0\ a \star_D \tau_0{}'\ a)\ (G_0\ a)$
and $D.quasi\text{-}inverses\ (\tau_0\ a)\ (\tau_0{}'\ a)$
proof −
interpret *adjoint-equivalence-in-bicategory*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ a$› ‹$\tau_0{}'\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
using *assms chosen-adjoint-equivalence* **by** *simp*
show $D.isomorphic\ (\tau_0{}'\ a \star_D \tau_0\ a)\ (F_0\ a)$
using *assms D.isomorphic-def D.isomorphic-symmetric unit-is-iso* **by** *blast*
show $D.isomorphic\ (\tau_0\ a \star_D \tau_0{}'\ a)\ (G_0\ a)$
using *assms D.isomorphic-def counit-is-iso* **by** *blast*
show $D.quasi\text{-}inverses\ (\tau_0\ a)\ (\tau_0{}'\ a)$
using *D.quasi-inverses-def equivalence-in-bicategory-axioms* **by** *auto*
qed

definition map_1
where $map_1\ f = (\tau_0{}'\ (trg_C\ f) \star_D \mathrm{r}_D[G\ f]) \cdot_D$

$(\tau_0{}'\ (trg_C\ f) \star_D\ G\ f \star_D\ \varepsilon\ (src_C\ f)) \cdot_D$
$(\tau_0{}'\ (trg_C\ f) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ (src_C\ f),\ \tau_0{}'\ (src_C\ f)]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ (trg_C\ f),\ G\ f \star_D\ \tau_0\ (src_C\ f),\ \tau_0{}'\ (src_C\ f)] \cdot_D$
$((\tau_0{}'\ (trg_C\ f) \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ (src_C\ f)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ (trg_C\ f),\ \tau_0\ (trg_C\ f),\ F\ f] \star_D\ \tau_0{}'\ (src_C\ f)) \cdot_D$
$((\eta\ (trg_C\ f) \star_D\ F\ f) \star_D\ \tau_0{}'\ (src_C\ f)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ (src_C\ f))$

abbreviation (*input*) $\tau_1{}'$
where $\tau_1{}' \equiv map_1$

lemma map_1*-in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\tau_1{}'\ f : G_0\ (src_C\ f) \rightarrow_D\ F_0\ (trg_C\ f)$»
and «$\tau_1{}'\ f : F\ f \star_D\ \tau_0{}'\ (src_C\ f) \Rightarrow_D\ \tau_0{}'\ (trg_C\ f) \star_D\ G\ f$»
proof −
  show «$\tau_1{}'\ f : F\ f \star_D\ \tau_0{}'\ (src_C\ f) \Rightarrow_D\ \tau_0{}'\ (trg_C\ f) \star_D\ G\ f$»
    using *assms* τ*.iso-map*$_1$*-ide* τ*.map*$_1$*-in-vhom*
    by (*unfold* map_1*-def*, *intro D.comp-in-homI*) *auto*
  thus «$\tau_1{}'\ f : G_0\ (src_C\ f) \rightarrow_D\ F_0\ (trg_C\ f)$»
    using *assms D.vconn-implies-hpar*(*1*−*2*) **by** *auto*
qed

lemma map_1*-simps* [*simp*]:
assumes $C.ide\ f$
shows $D.arr\ (\tau_1{}'\ f)$
and $src_D\ (\tau_1{}'\ f) = G_0\ (src_C\ f)$ **and** $trg_D\ (\tau_1{}'\ f) = F_0\ (trg_C\ f)$
and $D.dom\ (\tau_1{}'\ f) = F\ f \star_D\ \tau_0{}'\ (src_C\ f)$ **and** $D.cod\ (\tau_1{}'\ f) = \tau_0{}'\ (trg_C\ f) \star_D\ G\ f$
  using *assms* map_1*-in-hom* **by** *auto*

lemma *iso-*map_1*-ide*:
assumes $C.ide\ f$
shows $D.iso\ (\tau_1{}'\ f)$
proof −
  interpret *src*: *adjoint-equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
    ‹$\tau_0\ (src_C\ f)$› ‹$\tau_0{}'\ (src_C\ f)$› ‹$\eta\ (src_C\ f)$› ‹$\varepsilon\ (src_C\ f)$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  interpret *trg*: *adjoint-equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
    ‹$\tau_0\ (trg_C\ f)$› ‹$\tau_0{}'\ (trg_C\ f)$› ‹$\eta\ (trg_C\ f)$› ‹$\varepsilon\ (trg_C\ f)$›

    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *?thesis*
    unfolding map_1*-def*
    using *assms* τ*.iso-map*$_1$*-ide*
    by (*intro D.isos-compose*) *auto*
qed

interpretation *EV*: *self-evaluation-map* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ **..**
notation *EV.eval* (‹⦃-⦄›)

sublocale *pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ F\ \Phi_F\ \tau_0{}'\ \tau_1{}'$
proof
show $\bigwedge a.\ C.obj\ a \Longrightarrow D.ide\ (\tau_0{}'\ a)$
using map_0*-simps(1)* **by** *simp*
show $\bigwedge f.\ C.ide\ f \Longrightarrow D.iso\ (\tau_1{}'\ f)$
using *iso-map*$_1$*-ide* **by** *simp*
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$\tau_0{}'\ a : src_D\ (G\ a) \rightarrow_D src_D\ (F\ a)$»
by *fastforce*
show $\bigwedge f.\ C.ide\ f \Longrightarrow$ «$\tau_1{}'\ f : F\ f \star_D \tau_0{}'\ (src_C\ f) \Rightarrow_D \tau_0{}'\ (trg_C\ f) \star_D G\ f$»
by *auto*
show $\bigwedge a.\ C.obj\ a \Longrightarrow D.equivalence\text{-}map\ (\tau_0{}'\ a)$
by *simp*
show $\bigwedge \mu.\ C.arr\ \mu \Longrightarrow \tau_1{}'\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0{}'\ (src_C\ \mu))$
$= (\tau_0{}'\ (trg_C\ \mu) \star_D G\ \mu) \cdot_D \tau_1{}'\ (C.dom\ \mu)$
proof −
fix μ
assume μ: $C.arr\ \mu$
let $?a = src_C\ \mu$
let $?b = trg_C\ \mu$
let $?f = C.dom\ \mu$
let $?g = C.cod\ \mu$
have $\tau_1{}'\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b, G\ ?g \star_D \tau_0\ ?a, \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ ?g) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ \mu \star_D \tau_0{}'\ ?a)$
unfolding map_1*-def*
using μ *D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b, G\ ?g \star_D \tau_0\ ?a, \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\eta\ ?b \star_D F\ ?g) \star_D \tau_0{}'\ ?a) \cdot_D$
$((trg_D\ (F\ \mu) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D \tau_0{}'\ ?a)$
using μ *D.whisker-right D.lunit′-naturality* [*of* $F\ \mu$] *D.comp-assoc*
D.whisker-right [*of* $\tau_0{}'\ ?a\ trg_D\ (F\ \mu) \star_D F\ \mu\ \mathrm{l}_D{}^{-1}[F\ ?f]$]
by *simp*
also have ... $= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$

$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ ?g \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D \tau_0{}'\ ?a)$
proof −
have $((\eta\ ?b \star_D F\ ?g) \star_D \tau_0{}'\ ?a) \cdot_D ((trg_D\ (F\ \mu) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a) \cdot_D ((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a)$
proof −
have $((\eta\ ?b \star_D F\ ?g) \star_D \tau_0{}'\ ?a) \cdot_D ((trg_D\ (F\ \mu) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$
$= (\eta\ ?b \star_D F\ ?g) \cdot_D (trg_D\ (F\ \mu) \star_D F\ \mu) \star_D \tau_0{}'\ ?a$
using μ *D.whisker-right D.obj-simps* **by** *simp*
also have ... $= (\eta\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a$
using μ *D.interchange* [*of* $\eta\ ?b\ trg_D\ (F\ \mu)\ F\ ?g\ F\ \mu$]
D.comp-arr-dom D.comp-cod-arr
by *simp*
also have ... $= ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \cdot_D (\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a$
using μ *D.interchange* [*of* $\tau_0{}'\ ?b \star_D \tau_0\ ?b\ \eta\ ?b\ F\ \mu\ F\ ?f$]
D.comp-arr-dom D.comp-cod-arr
by *simp*
also have ... $= (((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a) \cdot_D ((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a)$
using μ *D.whisker-right* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ ?g \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D \tau_0{}'\ ?a)$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D (((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?g] \cdot_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a$
using μ *D.whisker-right* **by** *simp*
also have ... $= (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a$
using μ *D.assoc-naturality* [*of* $\tau_0{}'\ ?b\ \tau_0\ ?b\ F\ \mu$] **by** *simp*
also have ... $= ((\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a)$
using μ *D.whisker-right* **by** *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?g] \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$

$= ((\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ ?g \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D \tau_0{}'\ ?a)$
proof −
have $((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \cdot_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a$
using μ τ*.iso-map*$_1$*-ide D.whisker-right* **by** *simp*
also have ... $= (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g) \cdot_D (\tau_0\ ?b \star_D F\ \mu)) \star_D \tau_0{}'\ ?a$
using μ τ*.iso-map*$_1$*-ide D.whisker-left* **by** *simp*
also have ... $= (\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a) \cdot_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a$
using μ τ*.iso-map*$_1$*-ide* τ*.naturality*
D.invert-opposite-sides-of-square
[*of* τ_1 *?g* G μ $\star_D$ τ_0 *?a* τ_0 *?b* $\star_D$ F μ τ_1 *?f*]
by *simp*
also have ... $= (\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a)) \cdot_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a$
using μ τ*.iso-map*$_1$*-ide D.whisker-left* **by** *simp*
also have ... $= ((\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a)$
using μ τ*.iso-map*$_1$*-ide D.whisker-right* **by** *simp*
finally have $((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?g)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ \mu) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ ?g]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ ?g \star_D \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ ?g,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?b \star_D (G\ \mu \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ ?f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ ?f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ ?f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D \tau_0{}'\ ?a)$

using μ *D.assoc-naturality* [*of* $\tau_0{}'$ *?b* G μ $\star_D$ τ_0 *?a* $\tau_0{}'$ *?a*] *D.comp-assoc*
by *simp*
also have ... = $(\tau_0{}'$ *?b* $\star_D$ $\mathrm{r}_D[G$ *?g*$])$ $\cdot_D$
$((\tau_0{}'$ *?b* $\star_D$ G *?g* $\star_D$ ε *?a*$)$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$))$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?f*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$ $\cdot_D$
$\mathrm{a}_D[\tau_0{}'$ *?b*, G *?f* $\star_D$ τ_0 *?a*, $\tau_0{}'$ *?a*$]$ $\cdot_D$
$((\tau_0{}'$ *?b* $\star_D$ *D.inv* $(\tau_1$ *?f*$))$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\mathrm{a}_D[\tau_0{}'$ *?b*, τ_0 *?b*, F *?f*$]$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$((\eta$ *?b* $\star_D$ F *?f*$)$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\mathrm{l}_D{}^{-1}[F$ *?f*$]$ $\star_D$ $\tau_0{}'$ *?a*$)$
proof −
have $(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?g*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $(G$ μ $\star_D$ τ_0 *?a*$)$ $\star_D$ $\tau_0{}'$ *?a*$)$
= $\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?g*, τ_0 *?a*, $\tau_0{}'$ *?a*$]$ $\cdot_D$ $((G$ μ $\star_D$ τ_0 *?a*$)$ $\star_D$ $\tau_0{}'$ *?a*$)$
using μ *D.whisker-left* **by** *simp*
also have ... = $\tau_0{}'$ *?b* $\star_D$ $(G$ μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$ $\mathrm{a}_D[G$ *?f*, τ_0 *?a*, $\tau_0{}'$ *?a*$]$
using μ *D.assoc-naturality* [*of* G μ τ_0 *?a* $\tau_0{}'$ *?a*] **by** *simp*
also have ... = $(\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?f*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$
using μ *D.whisker-left* **by** *simp*
finally have $(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?g*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $(G$ μ $\star_D$ τ_0 *?a*$)$ $\star_D$ $\tau_0{}'$ *?a*$)$
= $(\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?f*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $((\tau_0{}'$ *?b* $\star_D$ $\mathrm{r}_D[G$ *?g*$])$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ G_0 *?a*$))$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ G *?f* $\star_D$ ε *?a*$)$ $\cdot_D$
$(\tau_0{}'$ *?b* $\star_D$ $\mathrm{a}_D[G$ *?f*, τ_0 *?a*, $\tau_0{}'$ *?a*$])$ $\cdot_D$
$\mathrm{a}_D[\tau_0{}'$ *?b*, G *?f* $\star_D$ τ_0 *?a*, $\tau_0{}'$ *?a*$]$ $\cdot_D$
$((\tau_0{}'$ *?b* $\star_D$ *D.inv* $(\tau_1$ *?f*$))$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\mathrm{a}_D[\tau_0{}'$ *?b*, τ_0 *?b*, F *?f*$]$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$((\eta$ *?b* $\star_D$ F *?f*$)$ $\star_D$ $\tau_0{}'$ *?a*$)$ $\cdot_D$
$(\mathrm{l}_D{}^{-1}[F$ *?f*$]$ $\star_D$ $\tau_0{}'$ *?a*$)$
proof −
have $(\tau_0{}'$ *?b* $\star_D$ G *?g* $\star_D$ ε *?a*$)$ $\cdot_D$ $(\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$)$
= $\tau_0{}'$ *?b* $\star_D$ $(G$ *?g* $\star_D$ ε *?a*$)$ $\cdot_D$ $(G$ μ $\star_D$ τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*$)$
using μ *D.whisker-left* **by** *simp*
also have ... = $\tau_0{}'$ *?b* $\star_D$ G μ $\star_D$ ε *?a*
using μ *D.interchange* [*of* G *?g* G μ ε *?a* τ_0 *?a* $\star_D$ $\tau_0{}'$ *?a*]
D.comp-arr-dom D.comp-cod-arr
by *simp*
also have ... = $\tau_0{}'$ *?b* $\star_D$ $(G$ μ $\star_D$ G_0 *?a*$)$ $\cdot_D$ $(G$ *?f* $\star_D$ ε *?a*$)$
using μ *D.interchange* [*of* G μ G *?f* G_0 *?a* ε *?a*]
D.comp-arr-dom D.comp-cod-arr

by *simp*
also have ... $= (\tau_0{}'\ ?b \star_D\ G\ \mu \star_D\ G_0\ ?a) \cdot_D (\tau_0{}'\ ?b \star_D\ G\ ?f \star_D\ \varepsilon\ ?a)$
using μ *D.whisker-left D.obj-simps* **by** *auto*
finally have $(\tau_0{}'\ ?b \star_D\ G\ ?g \star_D\ \varepsilon\ ?a) \cdot_D (\tau_0{}'\ ?b \star_D\ G\ \mu \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?b \star_D\ G\ \mu \star_D\ G_0\ ?a) \cdot_D (\tau_0{}'\ ?b \star_D\ G\ ?f \star_D\ \varepsilon\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?b \star_D\ G\ \mu) \cdot_D$
$(\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?f]) \cdot_D$
$(\tau_0{}'\ ?b \star_D\ G\ ?f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D\ \mathrm{a}_D[G\ ?f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ ?f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ ?f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ ?f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D\ F\ ?f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ ?f] \star_D\ \tau_0{}'\ ?a)$
proof −
have $(\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?g]) \cdot_D (\tau_0{}'\ ?b \star_D\ G\ \mu \star_D\ G_0\ ?a)$
$= \tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?g] \cdot_D (G\ \mu \star_D\ G_0\ ?a)$
using μ *D.whisker-left D.obj-simps* **by** *simp*
also have ... $= \tau_0{}'\ ?b \star_D\ G\ \mu \cdot_D \mathrm{r}_D[G\ ?f]$
using μ *D.runit-naturality* [*of* $G\ \mu$] **by** *simp*
also have ... $= (\tau_0{}'\ ?b \star_D\ G\ \mu) \cdot_D (\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?f])$
using μ *D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?g]) \cdot_D (\tau_0{}'\ ?b \star_D\ G\ \mu \star_D\ G_0\ ?a)$
$= (\tau_0{}'\ ?b \star_D\ G\ \mu) \cdot_D (\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ ?f])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?b \star_D\ G\ \mu) \cdot_D \tau_1{}'\ ?f$
unfolding map_1*-def*
using μ *D.comp-assoc* **by** *simp*
finally show $\tau_1{}'\ ?g \cdot_D (F\ \mu \star_D\ \tau_0{}'\ ?a) = (\tau_0{}'\ ?b \star_D\ G\ \mu) \cdot_D \tau_1{}'\ ?f$
by *blast*
qed
show $\bigwedge a.\ C.obj\ a \Longrightarrow$
$(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a] \cdot_D \mathrm{l}_D[\tau_0{}'\ a] = \tau_1{}'\ a \cdot_D (F.unit\ a \star_D\ \tau_0{}'\ a)$
proof −
fix *a*
assume *C.obj a*
hence *a*: $C.obj\ a \wedge C.arr\ a \wedge C.ide\ a \wedge src_C\ a = a \wedge trg_C\ a = a$
by *auto*
have *1*: $D.ide\ (F_0\ a)$
using *a* $F.map_0$*-simps*(*1*) **by** *blast*
have *2*: $D.ide\ (G_0\ a)$
using *a* $G.map_0$*-simps*(*1*) **by** *blast*

have $\tau_1{}'\ a \cdot_D (F.unit\ a \star_D \tau_0{}'\ a)$
$= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a, G\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ a] \star_D \tau_0{}'\ a) \cdot_D$
$(F.unit\ a \star_D \tau_0{}'\ a)$
unfolding *map*$_1$*-def*
using *a D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a, G\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \star_D \tau_0{}'\ a) \cdot_D$
$(((\eta\ a \star_D F\ a) \star_D \tau_0{}'\ a) \cdot_D$
$((F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ a] \star_D \tau_0{}'\ a) \cdot_D (F.unit\ a \star_D \tau_0{}'\ a) = \mathrm{l}_D{}^{-1}[F\ a] \cdot_D F.unit\ a \star_D \tau_0{}'\ a$
using *a D.whisker-right* **by** *simp*
also have ... $= (F_0\ a \star_D F.unit\ a) \cdot_D \mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a$
using *a D.lunit′-naturality* [*of F.unit a*] **by** *simp*
also have ... $= ((F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D (\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
using *a 1 D.whisker-right* **by** *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ a] \star_D \tau_0{}'\ a) \cdot_D (F.unit\ a \star_D \tau_0{}'\ a) =$
$((F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D (\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a, G\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \star_D \tau_0{}'\ a) \cdot_D$
$(((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a)) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $((\eta\ a \star_D F\ a) \star_D \tau_0{}'\ a) \cdot_D ((F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= (\eta\ a \star_D F\ a) \cdot_D (F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a$
using *a 1 D.whisker-right* **by** *simp*
also have ... $= (\eta\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a$
using *a D.interchange* [*of* $\eta\ a\ F_0\ a\ F\ a\ F.unit\ a$]

$D.comp\text{-}cod\text{-}arr\ D.comp\text{-}arr\text{-}dom$
by *simp*
also have ... $= ((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \cdot_D (\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a$
using $a\ D.interchange$ [*of* $\tau_0{}'\ a \star_D \tau_0\ a\ \eta\ a\ F.unit\ a\ F_0\ a$]
$D.comp\text{-}cod\text{-}arr\ D.comp\text{-}arr\text{-}dom$
by *simp*
also have ... $= (((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D ((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)$
using $a\ 1\ D.whisker\text{-}right$ **by** *simp*
finally have $((\eta\ a \star_D F\ a) \star_D \tau_0{}'\ a) \cdot_D ((F_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= (((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D ((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a, G\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$(((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \star_D \tau_0{}'\ a) \cdot_D (((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= \mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \cdot_D ((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a$
using $a\ D.whisker\text{-}right$ **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \cdot_D \mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F_0\ a] \star_D \tau_0{}'\ a$
using $a\ D.assoc\text{-}naturality$ [*of* $\tau_0{}'\ a\ \tau_0\ a\ F.unit\ a$] **by** *simp*
also have ... $= ((\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F_0\ a] \star_D \tau_0{}'\ a)$
using $a\ 1\ D.whisker\text{-}right$ **by** *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F\ a] \star_D \tau_0{}'\ a) \cdot_D$
$(((\tau_0{}'\ a \star_D \tau_0\ a) \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= ((\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F_0\ a] \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, G\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a)) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$

$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D ((\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= (\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \cdot_D (\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a$
using *a τ.iso-map₁-ide D.whisker-right* **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a) \cdot_D (\tau_0\ a \star_D F.unit\ a)) \star_D \tau_0{}'\ a$
using *a τ.iso-map₁-ide D.whisker-left* **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D (G.unit\ a \star_D \tau_0\ a) \cdot_D \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a$
proof −
have $(\tau_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{l}_D[\tau_0\ a] = \tau_1\ a \cdot_D (G.unit\ a \star_D \tau_0\ a)$
using *a τ.respects-unit* **by** *simp*
hence $D.inv\ (\tau_1\ a) \cdot_D (\tau_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{l}_D[\tau_0\ a]$
$= G.unit\ a \star_D \tau_0\ a$
using *a τ.iso-map₁-ide D.invert-side-of-triangle(1)* **by** *simp*
hence $D.inv\ (\tau_1\ a) \cdot_D (\tau_0\ a \star_D F.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a]$
$= (G.unit\ a \star_D \tau_0\ a) \cdot_D \mathrm{l}_D{}^{-1}[\tau_0\ a]$
using *a D.iso-lunit D.invert-side-of-triangle(2) D.comp-assoc* **by** *simp*
hence $D.inv\ (\tau_1\ a) \cdot_D (\tau_0\ a \star_D F.unit\ a)$
$= (G.unit\ a \star_D \tau_0\ a) \cdot_D \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{r}_D[\tau_0\ a]$
using *a D.iso-runit D.comp-assoc*
D.invert-side-of-triangle(2)
[*of* $(G.unit\ a \star_D \tau_0\ a) \cdot_D \mathrm{l}_D{}^{-1}[\tau_0\ a]$ $D.inv\ (\tau_1\ a) \cdot_D (\tau_0\ a \star_D F.unit\ a)$
$\mathrm{r}_D{}^{-1}[\tau_0\ a]$]
by *simp*
thus *?thesis* **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D G.unit\ a \star_D \tau_0\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a$
using *a D.whisker-left* **by** *simp*
also have ... $= ((\tau_0{}'\ a \star_D G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a)$
using *a D.whisker-right* **by** *simp*
finally have $((\tau_0{}'\ a \star_D D.inv\ (\tau_1\ a)) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \tau_0\ a \star_D F.unit\ a) \star_D \tau_0{}'\ a)$
$= ((\tau_0{}'\ a \star_D G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a, \tau_0\ a, \tau_0{}'\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D (G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a, G_0\ a \star_D \tau_0\ a, \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$

$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
using *a D.assoc-naturality* [*of* $\tau_0{}'\ a$ *G.unit* $a \star_D \tau_0\ a\ \tau_0{}'\ a$] *D.comp-assoc*
by *simp*
also have ... $= (\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$((\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D$ *G.unit* $a \star_D \tau_0\ a \star_D \tau_0{}'\ a)) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ G_0\ a \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D (\tau_0{}'\ a \star_D (G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a)$
$= \tau_0{}'\ a \star_D \mathrm{a}_D[G\ a,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D ((G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a)$
using *a D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ a \star_D (G.unit\ a \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]$
using *a D.assoc-naturality* [*of G.unit* $a\ \tau_0\ a\ \tau_0{}'\ a$] **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D$ *G.unit* $a \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a])$
using *a 2 D.whisker-left* [*of* $\tau_0{}'\ a$] **by** *simp*
finally have $(\tau_0{}'\ a \star_D \mathrm{a}_D[G\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D (G.unit\ a \star_D \tau_0\ a) \star_D \tau_0{}'\ a)$
$= (\tau_0{}'\ a \star_D$ *G.unit* $a \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\tau_0{}'\ a \star_D \mathrm{r}_D[G\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D$ *G.unit* $a \star_D G_0\ a)) \cdot_D$
$(\tau_0{}'\ a \star_D G_0\ a \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ G_0\ a \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[\tau_0\ a]) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\tau_0{}'\ a \star_D G\ a \star_D \varepsilon\ a) \cdot_D (\tau_0{}'\ a \star_D$ *G.unit* $a \star_D \tau_0\ a \star_D \tau_0{}'\ a)$
$= \tau_0{}'\ a \star_D$ *G.unit* $a \star_D \varepsilon\ a$
using *a D.whisker-left D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $G\ a$ *G.unit* $a\ \varepsilon\ a\ \tau_0\ a \star_D \tau_0{}'\ a$]
by *simp*

also have ... = $(\tau_0{}'\ a \star_D\ G.unit\ a \star_D\ G_0\ a) \cdot_D (\tau_0{}'\ a \star_D\ G_0\ a \star_D\ \varepsilon\ a)$
using *a 2 D.whisker-left D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of G.unit a* G_0 *a* G_0 *a* ε *a*]
by *simp*
finally have $(\tau_0{}'\ a \star_D\ G\ a \star_D\ \varepsilon\ a) \cdot_D (\tau_0{}'\ a \star_D\ G.unit\ a \star_D\ \tau_0\ a \star_D\ \tau_0{}'\ a)$
$= (\tau_0{}'\ a \star_D\ G.unit\ a \star_D\ G_0\ a) \cdot_D (\tau_0{}'\ a \star_D\ G_0\ a \star_D\ \varepsilon\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D$
$(\tau_0{}'\ a \star_D\ \mathrm{r}_D[G_0\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D\ G_0\ a \star_D\ \varepsilon\ a) \cdot_D$
$((\tau_0{}'\ a \star_D\ \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ G_0\ a \star_D\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$((\tau_0{}'\ a \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D\ \tau_0{}'\ a) \cdot_D$
$(((\tau_0{}'\ a \star_D\ \mathrm{r}_D[\tau_0\ a]) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ F_0\ a] \star_D\ \tau_0{}'\ a)) \cdot_D$
$((\eta\ a \star_D\ F_0\ a) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D\ \tau_0{}'\ a)$
proof −
have $(\tau_0{}'\ a \star_D\ \mathrm{r}_D[G\ a]) \cdot_D (\tau_0{}'\ a \star_D\ G.unit\ a \star_D\ G_0\ a)$
$= \tau_0{}'\ a \star_D\ \mathrm{r}_D[G\ a] \cdot_D (G.unit\ a \star_D\ G_0\ a)$
using *a 2 D.whisker-left* **by** *simp*
also have ... = $\tau_0{}'\ a \star_D\ G.unit\ a \cdot_D \mathrm{r}_D[G_0\ a]$
using *a D.runit-naturality* [*of G.unit a*] **by** *simp*
also have ... = $(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D (\tau_0{}'\ a \star_D\ \mathrm{r}_D[G_0\ a])$
using *a 2 D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ a \star_D\ \mathrm{r}_D[G\ a]) \cdot_D (\tau_0{}'\ a \star_D\ G.unit\ a \star_D\ G_0\ a)$
$= (\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D (\tau_0{}'\ a \star_D\ \mathrm{r}_D[G_0\ a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D$
$(\tau_0{}'\ a \star_D\ \mathrm{r}_D[G_0\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D\ G_0\ a \star_D\ \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ a \star_D\ \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ a,\ G_0\ a \star_D\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ a \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a]) \star_D\ \tau_0{}'\ a)) \cdot_D$
$(\mathrm{r}_D[\tau_0{}'\ a \star_D\ \tau_0\ a] \star_D\ \tau_0{}'\ a) \cdot_D$
$((\eta\ a \star_D\ F_0\ a) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D\ \tau_0{}'\ a)$
using *a 1 D.whisker-right D.runit-hcomp*(*1*) *D.comp-assoc* **by** *simp*
also have ... = $(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D$
$(\tau_0{}'\ a \star_D\ \mathrm{r}_D[G_0\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D\ G_0\ a \star_D\ \varepsilon\ a) \cdot_D$
$((\tau_0{}'\ a \star_D\ \mathrm{a}_D[G_0\ a,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a] \star_D\ \tau_0{}'\ a)) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$(\mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a] \star_D \tau_0{}'\ a)\ \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
using *a D.assoc-naturality* [*of* $\tau_0{}'\ a\ \mathrm{l}_D{}^{-1}[\tau_0\ a]\ \tau_0{}'\ a$] *D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D G.unit\ a)\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a])\ \cdot_D$
$((\tau_0{}'\ a \star_D G_0\ a \star_D \varepsilon\ a)\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a \star_D \tau_0{}'\ a]))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$(\mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a] \star_D \tau_0{}'\ a)\ \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
using *a 2 D.whisker-left D.lunit-hcomp(4) D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D G.unit\ a)\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a])\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a])\ \cdot_D$
$(\tau_0{}'\ a \star_D \varepsilon\ a)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$((\mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a] \star_D \tau_0{}'\ a)\ \cdot_D$
$((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a))\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\tau_0{}'\ a \star_D G_0\ a \star_D \varepsilon\ a)\ \cdot_D\ (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a \star_D \tau_0{}'\ a])$
$= \tau_0{}'\ a \star_D (G_0\ a \star_D \varepsilon\ a)\ \cdot_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a \star_D \tau_0{}'\ a]$
using *a 2 D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a]\ \cdot_D\ \varepsilon\ a$
using *a D.lunit′-naturality* [*of* $\varepsilon\ a$] **by** *simp*
also have ... $= (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a])\ \cdot_D\ (\tau_0{}'\ a \star_D \varepsilon\ a)$
using *a 2 D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ a \star_D G_0\ a \star_D \varepsilon\ a)\ \cdot_D\ (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a \star_D \tau_0{}'\ a])$
$= (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a])\ \cdot_D\ (\tau_0{}'\ a \star_D \varepsilon\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ a \star_D G.unit\ a)\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a])\ \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a])\ \cdot_D$
$((\tau_0{}'\ a \star_D \varepsilon\ a)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ a,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$(\eta\ a \star_D \tau_0{}'\ a))\ \cdot_D$
$(\mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
have $(\mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a] \star_D \tau_0{}'\ a)\ \cdot_D\ ((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)$
$= \mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a]\ \cdot_D\ (\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a$
using *a 1 D.whisker-right* **by** *simp*
also have ... $= \eta\ a\ \cdot_D\ \mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a$

using *a D.runit-naturality* [*of* η *a*] **by** *simp*
also have ... = $(\eta\ a \star_D \tau_0{}'\ a) \cdot_D (\mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a)$
using *a 1 D.whisker-right* **by** *simp*
finally have $(\mathrm{r}_D[\tau_0{}'\ a \star_D \tau_0\ a] \star_D \tau_0{}'\ a) \cdot_D ((\eta\ a \star_D F_0\ a) \star_D \tau_0{}'\ a)$
$= (\eta\ a \star_D \tau_0{}'\ a) \cdot_D (\mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(\tau_0{}'\ a \star_D G.unit\ a) \cdot_D$
$((\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a]) \cdot_D$
$(\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a]) \cdot_D$
$\mathrm{r}_D{}^{-1}[\tau_0{}'\ a]) \cdot_D$
$\mathrm{l}_D[\tau_0{}'\ a] \cdot_D$
$(\mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a)$
proof −
interpret *adjoint-equivalence-in-bicategory* V_D H_D a_D i_D src_D trg_D
‹τ_0 *a*› ‹$\tau_0{}'$ *a*› ‹η *a*› ‹ε *a*›
using *a chosen-adjoint-equivalence* **by** *simp*
have $(\tau_0{}'\ a \star_D \varepsilon\ a) \cdot_D \mathrm{a}_D[\tau_0{}'\ a, \tau_0\ a, \tau_0{}'\ a] \cdot_D (\eta\ a \star_D \tau_0{}'\ a)$
$= \mathrm{r}_D{}^{-1}[\tau_0{}'\ a] \cdot_D \mathrm{l}_D[\tau_0{}'\ a]$
using *triangle-right* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(\tau_0{}'\ a \star_D G.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a] \cdot_D \mathrm{l}_D[\tau_0{}'\ a]$
proof −
have $\mathrm{l}_D[\tau_0{}'\ a] \cdot_D (\mathrm{r}_D[F_0\ a] \star_D \tau_0{}'\ a) \cdot_D (\mathrm{l}_D{}^{-1}[F_0\ a] \star_D \tau_0{}'\ a) = \mathrm{l}_D[\tau_0{}'\ a]$
proof −
have *D.seq* $\mathrm{r}_D[F_0\ a]$ $\mathrm{l}_D{}^{-1}[F_0\ a]$
using *a 1* **by** *auto*
thus *?thesis*
using *a D.unitor-coincidence* [*of* F_0 *a*] *D.comp-arr-dom D.whisker-right*
by (*metis D.comp-arr-inv′ D.iso-runit D.lunit-simps*(*1*,*4*)
D.objE D.runit-simps(*5*) *F.map*$_0$*-simps*(*1*) *map*$_0$*-simps*(*1*,*3*))
qed
moreover have $(\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a]) \cdot_D (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a]) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
$= \mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
proof −
have $(\tau_0{}'\ a \star_D \mathrm{r}_D[G_0\ a]) \cdot_D (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a]) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a] =$
$((\tau_0{}'\ a \star_D \mathrm{l}_D[G_0\ a]) \cdot_D (\tau_0{}'\ a \star_D \mathrm{l}_D{}^{-1}[G_0\ a])) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
using *a D.unitor-coincidence D.comp-assoc* **by** *simp*
also have ... = $(\tau_0{}'\ a \star_D \mathrm{l}_D[G_0\ a] \cdot_D \mathrm{l}_D{}^{-1}[G_0\ a]) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
using *a 2 D.whisker-left* **by** *simp*
also have ... = $(\tau_0{}'\ a \star_D G_0\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
using *a 2 D.comp-arr-inv′* **by** *simp*
also have ... = $\mathrm{r}_D{}^{-1}[\tau_0{}'\ a]$
using *a 2 D.comp-cod-arr* **by** *simp*

finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show $(\tau_0{}'\ a \star_D\ G.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ a] \cdot_D \mathrm{l}_D[\tau_0{}'\ a]$
$= \tau_1{}'\ a \cdot_D (F.unit\ a \star_D\ \tau_0{}'\ a)$
by *simp*
qed
show $\bigwedge f\ g.\ [\![C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f]\!] \Longrightarrow$
$(\tau_0{}'\ (trg_C\ g) \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ (trg_C\ g),\ G\ g,\ G\ f] \cdot_D$
$(\tau_1{}'\ g \star_D\ G\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0{}'\ (src_C\ g),\ G\ f] \cdot_D$
$(F\ g \star_D\ \tau_1{}'\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ (src_C\ f)]$
$= \tau_1{}'\ (g \star_C\ f) \cdot_D (\Phi_F\ (g, f) \star_D\ \tau_0{}'\ (src_C\ f))$
proof −
fix $f\ g$
assume f: $C.ide\ f$ **and** g: $C.ide\ g$ **and** fg: $src_C\ g = trg_C\ f$
let $?a = src_C\ f$
let $?b = trg_C\ f$
let $?c = trg_C\ g$

have $*$: $D.ide\ (G_0\ ?a) \wedge D.ide\ (G_0\ ?b) \wedge D.ide\ (G_0\ ?b) \wedge D.ide\ (G_0\ ?c)$
using $f\ g$ $C.ideD(1)$ $G.map_0\text{-}simps(1)$ **by** *blast*
have $**$: $D.ide\ (F_0\ ?a) \wedge D.ide\ (F_0\ ?b) \wedge D.ide\ (F_0\ ?b) \wedge D.ide\ (F_0\ ?c)$
using $f\ g$ $C.ideD(1)$ $F.map_0\text{-}simps(1)$ **by** *blast*
have $(\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$(\tau_1{}'\ g \star_D\ G\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0{}'\ (src_C\ g),\ G\ f] \cdot_D$
$(F\ g \star_D\ \tau_1{}'\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
$= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b)$
$\star_D\ G\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(F\ g \star_D\ (\tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ ?b \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
unfolding *map*$_1$*-def*
using *f g fg D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ \tau_0{}'\ ?b,\ G\ f]) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b)$
$\star_D G\ f$
$= ((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f)$
proof −
have *1*: *D.arr* $((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \cdot_D$

$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b))$
using *f g fg* τ*.iso-map*$_1$*-ide* **by** *auto*
thus *?thesis*
using *f g fg D.whisker-right* [*of G f*]
by (*metis D.seqE G.preserves-ide*)
qed
moreover have $F\ g \star_D (\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)$
$= (F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have *1*: $D.arr\ ((\tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a))$
using *f g fg* τ*.iso-map*$_1$*-ide map*$_1$*-def map*$_1$*-simps*(*1*) **by** *presburger*
thus *?thesis*
using *f g fg D.whisker-left* [*of F g*]
by (*metis D.seqE F.preserves-ide*)
qed
ultimately show *?thesis*
using *f g fg D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\eta\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b \star_D\ G\ f) \cdot_D$
$(F\ g \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ f])) \cdot_D$
$(F\ g \star_D\ \tau_0{}'\ ?b \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(F\ g \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D\ (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D\ \mathrm{a}_D{}^{-1}[F\ g,\ \tau_0{}'\ ?b,\ G\ f]$
$= \mathrm{a}_D{}^{-1}[F_0\ ?c \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D\ (\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b \star_D\ G\ f)$
using *f g fg D.assoc′-naturality* [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ \tau_0{}'\ ?b\ G\ f$] **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\eta\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(F\ g \star_D\ \tau_0{}'\ ?b \star_D\ G\ f \star_D\ \varepsilon\ ?a)) \cdot_D$
$(F\ g \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D\ (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b \star_D\ G\ f) \cdot_D\ (F\ g \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ f])$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D\ \tau_0{}'\ ?b \star_D\ \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*

$D.interchange$ [*of* $\mathrm{l}_D{}^{-1}[F\ g]$ $F\ g\ \tau_0{}'\ ?b \star_D G\ f\ \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= ((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
$D.interchange$ [*of* $F_0\ ?c \star_D F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]$
$\tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f) \cdot_D (F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f])$
$= ((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a) \cdot_D (F\ g \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
$D.interchange$ [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ \tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a$
$\tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a$]
by *simp*
also have ... $= ((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
$D.interchange$ [*of* $F_0\ ?c \star_D F\ g\ \mathrm{l}_D{}^{-1}[F\ g]$
$\tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a$
$\tau_0{}'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$]

by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D\ G_0\ \ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \varepsilon\ ?a)$
$= ((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \tau_0\ \ ?a \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ \ ?b, \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ \ ?b, \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ \ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\eta\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ \ ?c \star_D\ F\ g, \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ \ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ \ ?a, \tau_0{}'\ ?a])) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ \ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ \ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \tau_0\ \ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a])$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ \tau_0{}'\ ?b \star_D\ G\ f \star_D \tau_0\ \ ?a \star_D \tau_0{}'\ ?a$
$\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ \ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F_0\ \ ?c \star_D\ F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]$
$\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ \ ?a) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \tau_0\ \ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a])$
$= ((F_0\ \ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ \ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ \ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\eta\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((F_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F_0\ ?c \star_D\ F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]$
$(\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= ((F_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$

$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\eta\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$
using *f g fg τ.iso-map₁-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a$
$(\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((F_0\ ?c \star_D\ F\ g) \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg τ.iso-map₁-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F_0\ ?c \star_D\ F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$
$(\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= ((F_0\ ?c \star_D\ F\ g) \star_D (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$

$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F_0\ ?c \star_D F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= ((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$

$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a$
$(\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F_0\ ?c \star_D F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$(F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(F\ g \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −

have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D (F\ g \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ g]\ F\ g\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a\ \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f)) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$
proof −
have $(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D \mathrm{a}_D{}^{-1}[F_0\ ?c \star_D F\ g, \tau_0{}'\ ?b,\ G\ f]$
$= \mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f)$
using *f g fg D.assoc′-naturality* [*of* $\eta\ ?c \star_D F\ g\ \tau_0{}'\ ?b\ G\ f$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D G_0\ ?a) \cdot_D$

$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a)) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0'\ ?a]$

proof −
have $((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f) \cdot_D ((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f])$
$= (\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g$
$\tau_0'\ ?b \star_D G\ f\ \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= (((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]\ \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a$]
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f])$
$= (((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a])) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0'\ ?a]$

proof –
have $((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a)$
$= (\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g\ \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a$
$\tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a$]
by *simp*
also have ... $= (((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a\ \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a$]
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a)$
$= (((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0'\ ?a])) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0'\ ?a]$
proof –
have $((\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a])$
$= (\eta\ ?c \star_D F\ g) \star_D \tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g\ \tau_0'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0'\ ?a$
$\tau_0'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]$]

by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
$\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (\eta\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g\ \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]$

$(\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= (\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g$
$(\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a$
$(\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$(\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a$
$(\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f)) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (G\ f \star_D \tau_0\ ?a)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$

$((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$
proof −
have $((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a)$
$= (\eta\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g$
$(\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\eta\ ?c \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\eta\ ?c \star_D\ F\ g) \star_D\ ((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
$= (\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D\ F\ g\ F_0\ ?c \star_D\ F\ g$
$((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$
$(\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g,\ \tau_0{}'\ ?b,\ G\ f \star_D\ G_0\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D\ G\ f \star_D\ \varepsilon\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ (\tau_0{}'\ ?b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$

$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{r}_D[G\ f])$
$= ((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f \star_D G_0\ ?a]$
using *f g fg D.assoc′-naturality* [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \tau_0{}'\ ?b\ \mathrm{r}_D[G\ f]$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f \star_D G_0\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D G\ f \star_D \varepsilon\ ?a)$
$= ((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a]$
using *f g fg D.assoc′-naturality* [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \tau_0{}'\ ?b\ G\ f \star_D \varepsilon\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$

$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, G\ f \star_D \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$

proof −
have $\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a])$
$= ((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
using *f g fg*
D.assoc′-naturality [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \tau_0{}'\ ?b\ \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, G\ g, G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$

proof −
have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, G\ f \star_D \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, G\ f \star_D \tau_0\ ?a, \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-left* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide*

$D.assoc\text{-}naturality\ [of\ \tau_0{}'\ ?b\ D.inv\ (\tau_1\ f)\ \tau_0{}'\ ?a]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
using *f g fg* $\tau.iso\text{-}map_1\text{-}ide$ *D.whisker-left* **by** *simp*
finally have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D (\tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= ((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
using *f g fg* $\tau.iso\text{-}map_1\text{-}ide$ $D.assoc'\text{-}naturality$
$[of\ (\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \tau_0{}'\ ?b\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$

$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a)) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b$
$G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$\mathrm{r}_D[G\ f]\ G\ f \star_D\ G_0\ ?a$]
by *simp*
finally have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$

$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$

proof −
have $((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b$
$G\ f \star_D G_0\ ?a\ G\ f \star_D \varepsilon\ ?a$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b$
$G\ f \star_D \varepsilon\ ?a\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, G\ g, G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$

$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b$
$G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]))\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$

$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
$(\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b$

$G\ f\ \mathrm{r}_D[G\ f]]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b$
$\mathrm{r}_D[G\ f]\ G\ f \star_D\ G_0\ ?a]$
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f])$
$= (((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ \tau_0{}'\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c) \star_D\ F\ g) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= ((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \tau_0{}'\ ?b$
$G\ f \star_D\ G_0\ ?a\ G\ f \star_D\ \varepsilon\ ?a]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b$

$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$G\ f \star_D \varepsilon\ ?a\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)$
$= (((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b$
$G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a\ \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$

$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D\ F\ f, \tau_0{}'\ ?a])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$
proof −
have $((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.assoc′-naturality [*of* $\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c,\ F\ g]\ \tau_0{}'\ ?b$
$(\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D\ F\ f, \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c,\ F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f, \tau_0{}'\ ?a]$

proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$ $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g$
$\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g$ $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]$
$(\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f)) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$ $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g$

$(\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a]$
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g\ \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide*
D.assoc′-naturality
[*of* $\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)\ \tau_0{}'\ ?b\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)\ \tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g$
$\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f)) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b\ \tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]$
$(\tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f)) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \tau_0{}'\ ?b \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$

$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg τ.iso-map₁-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)\ \tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g$
$(\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a]$
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg τ.iso-map₁-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b\ \tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)$
$\mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\tau_0{}'\ ?b \star_D \tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D\ F\ g) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ ((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ ((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \cdot_D\ (\eta\ ?c \star_D\ F\ g) \star_D$
$(((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D\ ((\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
using *f g fg*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$ $\eta\ ?c \star_D\ F\ g$
$((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$
$(\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]$
by *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \cdot_D\ (\eta\ ?c \star_D\ F\ g) \star_D\ ((\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.comp-cod-arr* **by** *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \cdot_D\ (\eta\ ?c \star_D\ F\ g) \star_D$
$((\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D\ ((F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom* **by** *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
using *f g fg* ** *D.interchange* **by** *auto*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ ((\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$

$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D ((\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg τ.iso-map₁-ide D.comp-cod-arr*
D.interchange
[*of* $\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)\ \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$
$((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a\ (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg τ.iso-map₁-ide D.comp-arr-dom*
D.interchange
[*of* $\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)\ \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$
$(\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D ((\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D G\ f)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$

$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f)$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f$
using *f g fg D.pentagon*
D.invert-side-of-triangle(*2*)
[*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b]$
D.comp-assoc
by *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f)$
using *f g fg D.whisker-right* **by** *simp*
finally have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f$
using *f g fg D.whisker-right* [*of* $G\ f$] **by** *simp*

also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \cdot_D ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f$
using *f g fg D.assoc-naturality* [*of* $\tau_0{}'\ ?c\ G\ g\ \varepsilon\ ?b$] **by** *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f)$
using *f g fg* $*$ *D.whisker-right* [*of* $G\ f$] **by** *simp*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \star_D\ G\ f)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \star_D\ G\ f)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D\ \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \cdot_D\ \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
using *f g fg D.pentagon* **by** *simp*
hence $\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \cdot_D (\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b)$
using *f g fg D.comp-assoc D.invert-side-of-triangle*(*1*) **by** *simp*
hence $\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b)$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \cdot_D (\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
using *f g fg D.comp-assoc*
D.invert-side-of-triangle(*2*)
[*of* $\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b] \cdot_D (\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$

$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b]\ \star_D\ \tau_0{}'\ ?b]$
by *simp*
hence $\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]\ \cdot_D\ (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b]\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ G\ f$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b]\ \cdot_D\ (\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]\ \star_D\ G\ f$
using *f g fg D.whisker-right* **by** *simp*
hence $(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]\ \star_D\ G\ f)\ \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b]\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ G\ f)$
$= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b]\ \star_D\ G\ f)\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b])\ \star_D\ G\ f)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]\ \star_D\ G\ f)$
using *f g fg D.whisker-right* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g])\ \star_D\ G\ f)\ \cdot_D$
$(((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \star_D\ G\ f)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \star_D\ G\ f))\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f))\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b])\ \star_D\ G\ f)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]\ \star_D\ G\ f)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ \mathrm{r}_D[G\ f])\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ G\ f \star_D\ \varepsilon\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \tau_0{}'\ ?b)\ \star_D\ D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f)\ \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b)\ \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f]\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g))\ \star_D\ (\eta\ ?b \star_D\ F\ f)\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]\ \star_D\ (F_0\ ?b \star_D\ F\ f)\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D\ F\ g)\ \star_D\ (F_0\ ?b \star_D\ F\ f)\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g]\ \star_D\ \mathrm{l}_D{}^{-1}[F\ f]\ \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D\ G\ g)\ \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b]\ \star_D\ G\ f)$
$= ((\tau_0{}'\ ?c \star_D\ G\ g)\ \star_D\ \varepsilon\ ?b)\ \cdot_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b]\ \star_D\ G\ f$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \cdot_D\ (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f$
using *f g fg D.assoc'-naturality* [*of* $\tau_0{}'\ ?c\ G\ g\ \varepsilon\ ?b$] **by** *simp*
also have ... $= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \star_D\ G\ f)\ \cdot_D\ ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f)$
using *f g fg* ∗ *D.whisker-right* **by** *simp*
finally have $(((\tau_0{}'\ ?c \star_D\ G\ g)\ \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b]\ \star_D\ G\ f)$
$= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \star_D\ G\ f)\ \cdot_D\ ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b)\ \star_D\ G\ f)$
by *blast*

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f$
using *f g fg* ∗ *D.comp-arr-inv′ D.whisker-right*
by (*metis C.ideD*(*1*) *C.obj-trg D.comp-assoc-assoc′*(*1*) *D.hcomp-simps*(*2*) *D.hseqI′ D.ideD*(*1*) *G.map*$_0$*-simps*(*3*) *G.preserves-ide G.preserves-src G.preserves-trg map*$_0$*-simps*(*1*−*2*))
moreover have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f$
using *f g fg D.comp-cod-arr* **by** *simp*
ultimately have $((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ (G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$

$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D\ F\ f, \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$

proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= \mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b]$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b]$
$\mathrm{r}_D[G\ f]\ G\ f \star_D\ G_0\ ?a$]
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= ((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ G_0\ ?a)$
by *blast*
thus *?thesis*
using *f g fg D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, G\ g, G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g, \tau_0\ ?b, \tau_0{}'\ ?b]) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b, \tau_0{}'\ ?b, (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b \star_D\ F\ f, \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b, \tau_0\ ?b, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$

$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b$
$\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\mathrm{r}_D[G\ f]\ G\ f \star_D G_0\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= ((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −

have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f])$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b\ \tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b$
$G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b\ \tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b$
$\mathrm{r}_D[G\ f]\ G\ f \star_D\ G_0\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{r}_D[G\ f])$
$= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ (G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b$
$G\ f \star_D\ G_0\ ?a\ G\ f \star_D\ \varepsilon\ ?a$]
by *simp*

also have ... $= ((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$G\ f \star_D \varepsilon\ ?a\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)$
$= ((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \varepsilon\ ?b) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a)$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\tau_0{}'\ ?c \star_D (G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$G\ f \star_D G_0\ ?a\ G\ f \star_D \varepsilon\ ?a$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b$
$\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$G\ f \star_D \varepsilon\ ?a\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$]

by *simp*
finally have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ (G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b$
$\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b$
$G\ f \star_D\ G_0\ ?a\ G\ f \star_D\ \varepsilon\ ?a$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b$
$\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b$
$G\ f \star_D\ \varepsilon\ ?a\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ G_0\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= ((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a)$
by *blast*

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
proof −
have $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
using *f g fg D.pentagon* **by** *simp*
hence $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b)$
using *f g fg D.comp-assoc*
D.invert-side-of-triangle(*2*)
[*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b]$
by *simp*
hence $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b)$
$\star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$
by *simp*

thus *?thesis*
using *f g fg D.whisker-right* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b]$
$\star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \cdot_D ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b)$
$\star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a$
using *f g fg D.assoc-naturality* [*of* $\tau_0{}'\ ?c\ G\ g\ \varepsilon\ ?b$] **by** *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg* ∗ *D.whisker-right* **by** *simp*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$

$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= ((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]]$
by *simp*
also have ... $= ((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= ((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$

$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]))\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= \mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f)\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f])\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \varepsilon\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a]\ \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b\ (\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b$
$G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b\ (\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a$]
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \tau_0{}'\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b$
$(\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
also have ... $= ((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= ((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*

$D.interchange$ [*of* $\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
$D.interchange$ [*of* $(\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b]$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
$D.interchange$ [*of* $(\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b\ (\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b$
$(G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a]$
by *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$

using $f\ g\ fg\ \tau$.*iso-map*$_1$-*ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b\ (\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
$= (((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$= ((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using $f\ g\ fg\ \tau$.*iso-map*$_1$-*ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b\ \tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)$
$(\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\varepsilon\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\varepsilon\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a$
$= (((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom* **by** *simp*
also have ... $= ((((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \varepsilon\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]$

proof −
have $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= \{\!\!\{\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star (\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}[\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star (\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle) \star \mathbf{a}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle]\}\!\!\}$
using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char*$_{SbC}$ *D.VVV.arr-char*$_{SbC}$
D.VV.ide-char$_{SbC}$ *D.VV.arr-char*$_{SbC}$
by *simp*
also have ... =
$\{\!\!\{((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle) \star (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle\}\!\!\}$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)
also have ... = $((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char D.VVV.arr-char*$_{SbC}$
D.VV.ide-char$_{SbC}$ *D.VV.arr-char*$_{SbC}$
by *simp*
finally have $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
proof −
have $(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg*
D.assoc′-naturality [*of* $\tau_0{}'\ ?c \star_D G\ g\ \varepsilon\ ?b\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *auto*

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
using *f g fg* ∗ *D.comp-arr-dom D.comp-cod-arr D.whisker-left*
D.interchange [*of* $\varepsilon\ ?b\ (\tau_0\ ?b \star_D \tau_0{}'\ ?b)\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
$\mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg* ∗ *D.comp-arr-dom D.comp-cod-arr D.whisker-left*
D.interchange [*of* $G_0\ ?b\ \varepsilon\ ?b\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$
$\tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\varepsilon\ ?b \star_D \tau_0\ ?b) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D G\ g) \star_D$
$(\varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]$
using *f g fg D.whisker-left* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D G\ g) \star_D$
$\mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\varepsilon\ ?b \star_D \tau_0\ ?b) \star_D F\ f \star_D \tau_0{}'\ ?a)$

using *f g fg D.assoc-naturality* [*of* ε *?b* τ_0 *?b* $F\ f \star_D \tau_0'$ *?a*]
by *simp*
also have ... $= ((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[G_0\ ?b, \tau_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D (\varepsilon\ ?b \star_D \tau_0\ ?b) \star_D F\ f \star_D \tau_0'\ ?a)$
using *f g fg* $*$ *D.whisker-left* **by** *simp*
finally have $((\tau_0'\ ?c \star_D G\ g) \star_D \varepsilon\ ?b \star_D \tau_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b \star_D \tau_0'\ ?b, \tau_0\ ?b, F\ f \star_D \tau_0'\ ?a])$
$= ((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[G_0\ ?b, \tau_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D (\varepsilon\ ?b \star_D \tau_0\ ?b) \star_D F\ f \star_D \tau_0'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
moreover
have $(\tau_0'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a$
$= (\mathrm{a}_D[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[\tau_0'\ ?b \star_D \tau_0\ ?b, F\ f, \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, (\tau_0'\ ?b \star_D \tau_0\ ?b) \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b, \tau_0'\ ?b \star_D \tau_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D \eta\ ?b) \star_D F\ f \star_D \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b, F_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, F_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b, F\ f, \tau_0'\ ?a])$
proof −
have $(\tau_0'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a$
$= ((\tau_0'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0'\ ?a)$
using *f g fg* $**$ *D.comp-arr-dom* **by** *simp*
also have ... $= (((\tau_0'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[F_0\ ?b, F\ f, \tau_0'\ ?a])) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, F_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b, F_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b, F_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, F_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b, F\ f, \tau_0'\ ?a])$

proof −
have $(\mathrm{a}_D[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[F_0\ ?b, F\ f, \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, F_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b, F_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b, F_0\ ?b, F\ f \star_D \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c \star_D G\ g, \tau_0\ ?b, F_0\ ?b \star_D F\ f \star_D \tau_0'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c, G\ g, \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b, F\ f, \tau_0'\ ?a])$
$= \{\!|(\mathbf{a}[\langle\tau_0'\ ?c\rangle, \langle G\ g\rangle, \langle\tau_0\ ?b\rangle] \star \mathbf{a}^{-1}[\langle F_0\ ?b\rangle_0, \langle F\ f\rangle, \langle\tau_0'\ ?a\rangle]) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0'\ ?c\rangle \star \langle G\ g\rangle, \langle\tau_0\ ?b\rangle, \langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle \star \langle\tau_0'\ ?a\rangle] \cdot$
$((\langle\tau_0'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}[\langle\tau_0\ ?b\rangle, \langle F_0\ ?b\rangle_0, \langle F\ f\rangle \star \langle\tau_0'\ ?a\rangle]) \cdot$
$((\langle\tau_0'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle, \langle F_0\ ?b\rangle_0, \langle F\ f\rangle \star \langle\tau_0'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0'\ ?c\rangle \star \langle G\ g\rangle, \langle\tau_0\ ?b\rangle, \langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle \star \langle\tau_0'\ ?a\rangle] \cdot$

$(\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star \mathbf{a}[\langle F_0\ ?b\rangle_0,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle])\}$
using *f g fg* ** *D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*
also have ... $= \{\!|(\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle \star \langle\tau_0\ ?b\rangle) \star (\langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle|\!\}$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= (\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* ** *D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \eta\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
proof −
have $((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D (\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D$
$((\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* ** *D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b\ \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b]$
$(\eta\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a\ \mathrm{a}_D{}^{-1}[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$(\eta\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg* ** *D.assoc′-naturality* [*of* $\eta\ ?b\ F\ f\ \tau_0{}'\ ?a$]
by *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \eta\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg* ** *D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b]\ (\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$
$\eta\ ?b \star_D F\ f \star_D \tau_0{}'\ ?a$]
by *simp*

finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ (\eta\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
proof −
have $(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.assoc′-naturality* [*of* $\tau_0{}'\ ?c \star_D\ G\ g\ \tau_0\ ?b$
$\eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a$]
by *force*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D$
$(\tau_0\ ?b \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D\ \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]$
using *f g fg ∗∗ D.whisker-left* **by** *auto*
also have ... $= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D$
$\mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
using *f g fg ∗∗ D.assoc-naturality* [*of* $\tau_0\ ?b\ \eta\ ?b\ F\ f \star_D\ \tau_0{}'\ ?a$]
by *auto*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
using *f g fg ∗∗ D.whisker-left* **by** *auto*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \eta\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a])$

$= ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D (\tau_0\ ?b \star_D \eta\ ?b) \star_D\ F\ f \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D (\varepsilon\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D (\tau_0\ ?b \star_D \eta\ ?b) \star_D\ F\ f \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D (F_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D (\tau_0\ ?b \star_D \tau_0{}'\ ?b) \star_D \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \tau_0{}'\ ?b) \star_D (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \tau_0\ ?b) \star_D \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D \tau_0\ ?b) \star_D\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D \tau_0{}'\ ?a$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b \star_D \tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \tau_0{}'\ ?b) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b \star_D\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ \tau_0{}'\ ?b] \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \tau_0{}'\ ?b) \star_D\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b,\ \tau_0{}'\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{a}_D[\tau_0{}'\ ?b,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (\tau_0{}'\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[\tau_0\ ?b,\ \tau_0{}'\ ?b \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a])$
$= \{\!|((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star (\langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle) \star \mathbf{a}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle \star \langle\tau_0{}'\ ?b\rangle,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle\tau_0{}'\ ?b\rangle] \star (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$((\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star \langle\tau_0{}'\ ?b\rangle) \star (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle \star \langle\tau_0\ ?b\rangle,\ \langle\tau_0{}'\ ?b\rangle,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle \star \langle\tau_0\ ?b\rangle) \star \mathbf{a}[\langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle \star \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle \star \langle\tau_0\ ?b\rangle) \star \mathbf{a}[\langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star \mathbf{a}^{-1}[\langle\tau_0{}'\ ?b\rangle \star \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle,\ (\langle\tau_0{}'\ ?b\rangle \star \langle\tau_0\ ?b\rangle) \star \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}[\langle\tau_0\ ?b\rangle,\ \langle\tau_0{}'\ ?b\rangle \star \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle])\|\}$

using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*

also have ... =
$\{\!|(\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle\tau_0{}'\ ?b\rangle,\ \langle\tau_0\ ?b\rangle] \star \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle|\}$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)

also have ... = $(\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a$
using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*

finally show *?thesis*
using *D.comp-assoc* **by** *simp*

qed

thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed

also have ... = $(\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$

$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ ?b] \cdot_D\ \mathrm{r}_D[\tau_0\ ?b] \star_D\ (F\ f \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\varepsilon\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D$
$((\varepsilon\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
using *f g fg D.whisker-left* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D$
$(\varepsilon\ ?b \star_D\ \tau_0\ ?b) \cdot_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \cdot_D\ (\tau_0\ ?b \star_D\ \eta\ ?b)$
$\star_D\ F\ f \star_D\ \tau_0{}'\ ?a$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ ?b] \cdot_D\ \mathrm{r}_D[\tau_0\ ?b] \star_D\ (F\ f \star_D\ \tau_0{}'\ ?a)$
proof −
interpret *adjoint-equivalence-in-bicategory*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹τ_0 ?b› ‹$\tau_0{}'$?b› ‹η ?b› ‹ε ?b›
using *f chosen-adjoint-equivalence* **by** *simp*
show *?thesis*
using *fg triangle-left* **by** *simp*
qed
finally have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\varepsilon\ ?b \star_D\ \tau_0\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ \tau_0{}'\ ?b,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ (\tau_0\ ?b \star_D\ \eta\ ?b) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ ?b] \cdot_D\ \mathrm{r}_D[\tau_0\ ?b] \star_D\ (F\ f \star_D\ \tau_0{}'\ ?a)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ G\ f) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ \mathrm{r}_D[G\ f])) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −

have $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ ?b] \cdot_D\ \mathrm{r}_D[\tau_0\ ?b] \star_D\ (F\ f \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= (\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$

proof −

have $\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g,\ G_0\ ?b,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G_0\ ?b,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ ?b] \cdot_D\ \mathrm{r}_D[\tau_0\ ?b] \star_D\ (F\ f \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F_0\ ?b \star_D\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{a}_D[F_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= \{\!|\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle G_0\ ?b\rangle_0,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \langle G_0\ ?b\rangle_0 \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}[\langle G_0\ ?b\rangle_0,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{l}^{-1}[\langle\tau_0\ ?b\rangle] \cdot \mathbf{r}[\langle\tau_0\ ?b\rangle] \star (\langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle)) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle F_0\ ?b\rangle_0,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star \mathbf{a}[\langle F_0\ ?b\rangle_0,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle])|\!\}$

using *f g fg* ∗ ∗∗ *D.α-def D.α′.map-ide-simp D.VVV.ide-char*$_{SbC}$ *D.VVV.arr-char*$_{SbC}$ *D.VV.ide-char*$_{SbC}$ *D.VV.arr-char*$_{SbC}$ *D.𝔩-ide-simp D.𝔯-ide-simp*

by *simp*

also have ... $= \{\!|(\mathbf{r}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle] \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \langle\tau_0\ ?b\rangle \star \mathbf{l}[\langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle,\ (\langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star (\langle F_0\ ?b\rangle_0 \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle)|\!\}$

using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)

also have ... $= (\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)$

using *f g fg* ∗ ∗∗ *D.α-def D.α′.map-ide-simp D.VVV.ide-char*$_{SbC}$ *D.VVV.arr-char*$_{SbC}$ *D.VV.ide-char*$_{SbC}$ *D.VV.arr-char*$_{SbC}$ *D.𝔩-ide-simp D.𝔯-ide-simp*

by *simp*

finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D G\ f) \cdot_D ((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D \mathrm{r}_D[G\ f])$
$= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]\ \tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b\ G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
moreover have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg* ∗ *D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ (\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b$
$G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D \tau_0\ ?b) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$

$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0'\ ?a]$

proof −
have $((\tau_0'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a]$
$= \mathrm{a}_D[\tau_0'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0'\ ?a] \cdot_D$
$(((\tau_0'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a)$
using *f g fg D.assoc-naturality* [*of* $\tau_0'\ ?c \star_D\ G\ g\ \tau_0\ ?b\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G_0\ ?b] \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$(((\tau_0'\ ?c \star_D\ G\ g) \star_D\ G_0\ ?b) \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0'\ ?c \star_D\ G\ g] \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0'\ ?a) \cdot_D$
$(((\tau_0'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D\ D.inv\ (\tau_1\ g)) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0'\ ?a]$

proof −
have $(((\tau_0'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a)$
$= \mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b\ \mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b]$
$\mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a$]
by *simp*
also have ... $= (\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b]\ \tau_0'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b$
$F\ f \star_D\ \tau_0'\ ?a\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a$]
by *simp*
finally have $(((\tau_0'\ ?c \star_D\ G\ g) \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0'\ ?a)$
$= (\mathrm{a}_D{}^{-1}[\tau_0'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0'\ ?a) \cdot_D$
$((\tau_0'\ ?c \star_D\ G\ g \star_D\ \tau_0\ ?b) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b\ \tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)$
$\mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)\ \tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g$
$F\ f \star_D \tau_0{}'\ ?a\ \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D G\ g \star_D \tau_0\ ?b) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$

$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= \mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b\ \mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g]$
$D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= (\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g]\ \tau_0{}'\ ?c \star_D G\ g\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$
$(D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]$]
D.whisker-left
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D G\ g) \star_D G_0\ ?b) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= (\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D G\ g,\ \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$

proof −
have $(\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom* **by** *simp*
also have ... $= (((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= ((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-left D.comp-assoc* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ g \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-inv'* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-cod-arr* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-inv-arr'* **by** *simp*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D G\ g) \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ g \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide*
D.assoc′-naturality [*of* $\tau_0{}'$ *?c G g D.inv* $(\tau_1\ f) \star_D \tau_0{}'$ *?a*]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D G\ g \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
$= \tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.assoc-naturality* [*of G g D.inv* $(\tau_1\ f)\ \tau_0{}'$ *?a*]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D G\ g \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b \star_D F\ f,\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
moreover have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.comp-arr-dom* **by** *auto*
also have ... $= (((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
$= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
using *f g fg D.whisker-left D.comp-assoc* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D (\tau_0\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
using *f g fg D.comp-arr-inv'* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
using *f g fg D.comp-cod-arr* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-inv-arr'* **by** *simp*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a])) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g)) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g) \star_D F\ f \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide*

$D.assoc'$-*naturality* [*of* $\tau_0{}'$ *?c* $D.inv$ $(\tau_1\ g)$ $F\ f \star_D \tau_0{}'$ *?a*]
by *simp*
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D \tau_0\ ?b,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a])$
$= \tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* $\tau.iso\text{-}map_1\text{-}ide$ $D.whisker\text{-}left$ **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$((D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* $\tau.iso\text{-}map_1\text{-}ide$ $D.assoc\text{-}naturality$ [*of* $D.inv$ $(\tau_1\ g)$ $F\ f$ $\tau_0{}'$ *?a*]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* $\tau.iso\text{-}map_1\text{-}ide$ $D.whisker\text{-}left$ **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
finally show *?thesis*
by *simp*
qed
ultimately show *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g) \star_D\ \mathrm{l}_D[F\ f] \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ (F_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= \tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a$

proof −
have $(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b \star_D\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ (\tau_0\ ?b \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ G\ g,\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ \tau_0\ ?b] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g \star_D\ \tau_0\ ?b,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g \star_D\ \tau_0\ ?b,\ F\ f,\ \tau_0{}'\ ?a])$
$= \{\!|(\langle\tau_0{}'\ ?c\rangle \star \mathbf{a}^{-1}[\langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle \star \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ (\langle\tau_0\ ?b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle G\ g\rangle, \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle] \star \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle,\ \langle G\ g\rangle \star \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\langle\tau_0{}'\ ?c\rangle \star \mathbf{a}[\langle G\ g\rangle \star \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle])|\!\}$
using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*
also have ... $= \{\!|\langle\tau_0{}'\ ?c\rangle \star \mathbf{a}[\langle G\ g\rangle,\ \langle\tau_0\ ?b\rangle,\ \langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle|\!\}$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= \tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D\ \tau_0{}'\ ?a$
using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ \mathrm{r}_D[G\ f])\ \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a)\ \cdot_D$

$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\tau_0{}'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= \tau_0{}'\ ?c \star_D ((G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg τ.iso-map₁-ide D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D$
$(G\ g \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (D.inv\ (\tau_1\ g) \star_D F\ f)$
$\star_D \tau_0{}'\ ?a$
using *f g fg τ.iso-map₁-ide D.whisker-right* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D$
$\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D$
$(D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f)) \cdot_D$
$((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
$\star_D \tau_0{}'\ ?a$
proof −
have *1*: $(\tau_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)$
$= D.inv\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]) \cdot_D$
$\tau_1\ (g \star_C f) \cdot_D$
$(\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]$
proof −
have $((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \cdot_D$
$(\tau_1\ g \star_D F\ f) \cdot_D D.inv\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)) \cdot_D$
$\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a]$
$= \tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a)$
using *f g fg τ.respects-hcomp D.comp-assoc* **by** *simp*
hence $((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]) \cdot_D$
$(\tau_1\ g \star_D F\ f) \cdot_D D.inv\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)$
$= \tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]$
using *f g fg D.comp-assoc*

$D.invert\text{-}side\text{-}of\text{-}triangle(2)$
$[of\ \tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a)$
$(\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \cdot_D$
$(\tau_1\ g \star_D F\ f) \cdot_D D.inv\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)$
$\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a]]$
by *fastforce*
moreover have $D.iso\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
using $f\ g\ fg\ C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}simp\ C.VV.cod\text{-}simp\ F.cmp\text{-}components\text{-}are\text{-}iso$
$F.FF\text{-}def$
by $(intro\ D.isos\text{-}compose\ D.seqI\ D.hseqI')\ auto$
moreover have $D.seq\ (\tau_1\ (g \star_C f))$
$((\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
using $f\ g\ fg\ C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}simp\ C.VV.cod\text{-}simp\ G.FF\text{-}def$
by $(intro\ D.seqI)\ auto$
ultimately show *?thesis*
using $f\ g\ fg\ \tau.iso\text{-}map_1\text{-}ide$
$D.invert\text{-}side\text{-}of\text{-}triangle(1)$
$[of\ \tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]$
$(\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]$
$(\tau_1\ g \star_D F\ f) \cdot_D D.inv\ \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)]$
by *simp*
qed
have $(G\ g \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (D.inv\ (\tau_1\ g) \star_D F\ f)$
$= D.inv\ ((\tau_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f))$
proof −
have $D.inv\ ((\tau_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f))$
$= D.inv\ (\mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f)) \cdot_D D.inv\ (\tau_1\ g \star_D F\ f)$
proof −
have $D.iso\ ((\tau_1\ g \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f))$
using $f\ g\ fg\ \tau.iso\text{-}map_1\text{-}ide$
by $(intro\ D.isos\text{-}compose)\ auto$
moreover have $D.iso\ (\tau_1\ g \star_D F\ f)$
using $f\ g\ fg\ \tau.iso\text{-}map_1\text{-}ide$ **by** *auto*
ultimately show *?thesis*
using $\tau.iso\text{-}map_1\text{-}ide\ D.inv\text{-}comp\text{-}left$ **by** *simp*
qed
also have $... = ((G\ g \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f]) \cdot_D$
$(D.inv\ (\tau_1\ g) \star_D F\ f)$
proof −
have $D.iso\ (\mathrm{a}_D{}^{-1}[G\ g,\ \tau_0\ (src_C\ g),\ F\ f] \cdot_D (G\ g \star_D \tau_1\ f))$
using $f\ g\ fg\ \tau.iso\text{-}map_1\text{-}ide$
by $(intro\ D.isos\text{-}compose)\ auto$
thus *?thesis*
using $f\ g\ fg\ \tau.iso\text{-}map_1\text{-}ide\ D.inv\text{-}comp\text{-}left$ **by** *simp*
qed
finally show *?thesis*
using $D.comp\text{-}assoc$ **by** *simp*
qed
also have $... = D.inv\ (D.inv\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]) \cdot_D$

$\tau_1\ (g \star_C f) \cdot_D$
$(\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
using *1* **by** *simp*
also have ... $= D.inv\ (\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]) \cdot_D$
$((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
proof −
have *2*: $D.iso\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
using *f g fg* τ*.iso-map*$_1$*-ide C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp F.cmp-components-are-iso F.FF-def*
by (*intro D.isos-compose*) *auto*
moreover have $D.iso\ (D.inv\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]) \cdot_D$
$\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
using *2 f g fg* τ*.iso-map*$_1$*-ide C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp F.FF-def G.FF-def D.inv-comp-left*
by (*intro D.isos-compose*) *auto*
ultimately show *?thesis*
using *D.inv-comp-left*
[*of* $D.inv\ ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
$\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]]$
D.inv-inv D.iso-inv-iso
by *metis*
qed
also have ... $= \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f)) \cdot_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f])$
proof −
have $D.inv\ (\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
$= \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f))$
proof −
have *2*: $D.iso\ (\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
using *f g fg* τ*.iso-map*$_1$*-ide C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp G.FF-def*
by (*intro D.isos-compose D.seqI D.hseqI′*) *auto*
have $D.inv\ (\tau_1\ (g \star_C f) \cdot_D (\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a])$
$= D.inv\ ((\Phi_G\ (g, f) \star_D \tau_0\ ?a) \cdot_D \mathrm{a}_D{}^{-1}[G\ g,\ G\ f,\ \tau_0\ ?a]) \cdot_D D.inv\ (\tau_1\ (g$
$\star_C f))$
using *2 f g fg* $*$ τ*.iso-map*$_1$*-ide G.cmp-components-are-iso C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp G.FF-def D.inv-comp-left*
by *simp*
also have ... $= (\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a)) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f))$
using *2 f g fg* $*$ τ*.iso-map*$_1$*-ide G.cmp-components-are-iso C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp G.FF-def D.inv-comp-left*
by *simp*
also have ... $= \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f))$
using *D.comp-assoc* **by** *simp*
finally show *?thesis*

using *D.comp-assoc* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally have $(G\ g \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \cdot_D (D.inv\ (\tau_1\ g) \star_D F\ f)$
$= \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \cdot_D$
$D.inv\ (\tau_1\ (g \star_C f)) \cdot_D (\tau_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f]$
by *blast*
thus *?thesis* **by** *simp*
qed
also have ... $= \tau_0'\ ?c \star_D$
$(\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0'\ ?a) \cdot_D$
$((D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0'\ ?a) \cdot_D$
$(D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0'\ ?a) \cdot_D$
$(((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0'\ ?a)) \cdot_D$
$(\mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-right*
C.VV.arr-char$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp G.cmp-components-are-iso F.FF-def G.FF-def*
by *simp*
also have ... $= (\tau_0'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0'\ ?a)) \cdot_D$
$(\tau_0'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide D.whisker-left*
C.VV.arr-char$_{SbC}$ *C.VV.dom-simp C.VV.cod-simp G.cmp-components-are-iso F.FF-def G.FF-def*
by *simp*
finally have $(\tau_0'\ ?c \star_D (G\ g \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D \mathrm{a}_D[G\ g,\ \tau_0\ ?b,\ F\ f] \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D (D.inv\ (\tau_1\ g) \star_D F\ f) \star_D \tau_0'\ ?a)$
$= (\tau_0'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0'\ ?a) \cdot_D$
$(\tau_0'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0'\ ?a)) \cdot_D$
$(\tau_0'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0'\ ?a) \cdot_D$

$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g,\ f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g,\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g\ \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]$
$\mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
D.whisker-right
by *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g]\ (\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g$
$F\ f \star_D \tau_0{}'\ ?a\ \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$]
D.whisker-right
by *simp*
finally have $((\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g,\ f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$

$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(((F_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= (\eta\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g\ \eta\ ?c \star_D F\ g$
$\mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a\ (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= ((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ ?c \star_D F\ g\ F_0\ ?c \star_D F\ g\ F\ f \star_D \tau_0{}'\ ?a\ \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D (F_0\ ?b \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((F_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$

$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((F_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a) =$
$\mathrm{l}_D{}^{-1}[F\ g] \star_D (\mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)$
using *f g fg D.comp-cod-arr*
D.interchange [*of* $F_0\ ?c \star_D F\ g\ \mathrm{l}_D{}^{-1}[F\ g]\ \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a\ \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= \mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D[F\ f] \cdot_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.whisker-right* **by** *simp*
also have ... $= \mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a$
using *f D.comp-arr-inv'* **by** *simp*
finally have $((F_0\ ?c \star_D F\ g) \star_D \mathrm{l}_D[F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ ?a)$
$= \mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g]) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g \star_D G_0\ ?b) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D{}^{-1}[G\ g]) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g,\ f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g,\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \cdot_D \mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg* ∗ *D.comp-arr-dom*
D.interchange [*of* $\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b]\ \mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D G\ g]$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D{}^{-1}[G\ g]) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.runit-hcomp*(*4*) [*of* $\tau_0{}'\ ?c\ G\ g$] **by** *simp*
finally have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G_0\ ?b] \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$

$(\mathrm{r}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ G\ g] \star_D\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ G\ f \star_D\ \varepsilon\ ?a)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ (G\ f \star_D\ \varepsilon\ ?a) \cdot_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.comp-cod-arr*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b\ \tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]$
$G\ f \star_D\ \varepsilon\ ?a\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
using *f g fg D.comp-arr-dom*
D.interchange [*of* $\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]\ \tau_0{}'\ ?c \star_D\ G\ g\ G\ f \star_D\ \varepsilon\ ?a$
$\mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
finally have $((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G_0\ ?b) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
$= ((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$

$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D\ ((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \cdot_D\ (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ \mathrm{r}_D[G\ f] \cdot_D\ (G\ f \star_D\ \varepsilon\ ?a)$
using *f g fg D.interchange* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g] \cdot_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ \mathrm{r}_D[G\ f] \cdot_D\ (G\ f \star_D\ \varepsilon\ ?a)$
using *f g fg D.whisker-left* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f] \cdot_D\ (G\ f \star_D\ \varepsilon\ ?a)$
using *g D.comp-arr-inv'* **by** *simp*
also have ... $= ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D\ ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
using *f g fg D.whisker-left* **by** *simp*
finally have $((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ g]) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \mathrm{r}_D{}^{-1}[G\ g]) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
$= ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D\ ((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f \star_D\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$

$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ (G\ f \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a] \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ g \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
using *f g fg*
D.assoc′-naturality [*of* $\tau_0{}'\ ?c\ G\ g\ \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D$
$((\tau_0{}'\ ?c \star_D G\ g) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(((\tau_0{}'\ ?c \star_D G\ g) \star_D G\ f \star_D \varepsilon\ ?a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g,\ f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g,\ f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$

proof −
have $(\tau_0{}'\ ?c \star_D G\ g \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a)$
$= \tau_0{}'\ ?c \star_D (G\ g \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a)$
using *f g fg D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D[G\ g \star_D G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]$
using *f g fg D.pentagon* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
using *f g fg D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D G\ g \star_D \mathrm{a}_D[G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a] \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D \tau_0{}'\ ?a]) \cdot_D$

$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f]\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ \mathrm{r}_D[G\ f])\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D\ G_0\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a))\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D\ G\ f \star_D\ \varepsilon\ ?a)\ \cdot_D\ \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D\ G_0\ ?a]\ \cdot_D\ (\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a)$
using *f g fg D.assoc′-naturality* [*of* $\tau_0{}'\ ?c\ G\ g\ G\ f \star_D\ \varepsilon\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f))\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f]\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ \mathrm{r}_D[G\ f]))\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a))\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −

have $((\tau_0{}'\ ?c \star_D\ G\ g) \star_D \mathrm{r}_D[G\ f]) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f \star_D\ G_0\ ?a]$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D (\tau_0{}'\ ?c \star_D\ G\ g \star_D \mathrm{r}_D[G\ f])$
using *f g fg D.assoc'-naturality* [*of* $\tau_0{}'$ *?c G g* $\mathrm{r}_D[G\ f]$] **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ g \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a])) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c,\ G\ g,\ G\ f]) \cdot_D (\tau_0{}'\ ?c \star_D\ G\ g \star_D \mathrm{r}_D[G\ f])$
$= (\tau_0{}'\ ?c \star_D\ G\ g \star_D \mathrm{r}_D[G\ f])$
using *f g fg D.comp-arr-inv' D.comp-cod-arr* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \Phi_G\ (g,\ f)) \cdot_D$
$((\tau_0{}'\ ?c \star_D\ G\ g \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ G_0\ ?a])) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (G\ g \star_D\ G\ f) \star_D\ \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ (D.inv\ (\Phi_G\ (g,\ f)) \star_D\ \tau_0\ ?a) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g,\ f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a])$
$= \tau_0{}'\ ?c \star_D\ (G\ g \star_D\ G\ f \star_D\ \varepsilon\ ?a) \cdot_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D\ \tau_0{}'\ ?a]$
using *f g fg D.whisker-left* **by** *simp*

also have ... $= \tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ G_0\ ?a] \cdot_D ((G\ g \star_D\ G\ f) \star_D \varepsilon\ ?a)$
using *f g fg D.assoc-naturality* [*of G g G f* ε *?a*] **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ G_0\ ?a]) \cdot_D (\tau_0{}'\ ?c \star_D (G\ g \star_D\ G\ f) \star_D \varepsilon\ ?a)$
using *f g fg* $*$ *D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D\ G\ g \star_D\ G\ f \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ \tau_0\ ?a \star_D \tau_0{}'\ ?a])$
$= (\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g,\ G\ f,\ G_0\ ?a]) \cdot_D (\tau_0{}'\ ?c \star_D (G\ g \star_D\ G\ f) \star_D \varepsilon\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D\ G\ f]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (G\ g \star_D\ G\ f) \star_D \varepsilon\ ?a) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
using *f g fg* $*$ *D.runit-hcomp D.whisker-left D.comp-assoc* **by** *simp*

also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D\ G\ f]) \cdot_D$
$((\tau_0{}'\ ?c \star_D (G\ g \star_D\ G\ f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f),\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D\ F\ g) \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D\ F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
$= \tau_0{}'\ ?c \star_D$
$\mathrm{a}_D[G\ g \star_D\ G\ f,\ \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
using *f g fg G.cmp-components-are-iso D.whisker-left* **by** *simp*

also have ... $= \tau_0{}'\ ?c \star_D$
$(D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]$
using *f g fg G.cmp-components-are-iso*
D.assoc-naturality [*of D.inv* $(\Phi_G\ (g, f))\ \tau_0\ ?a\ \tau_0{}'\ ?a$]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])$
using *f g fg G.cmp-components-are-iso D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ g \star_D G\ f, \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a) \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D G\ f]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D (G\ g \star_D G\ f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
$= \tau_0{}'\ ?c \star_D ((G\ g \star_D G\ f) \star_D \varepsilon\ ?a) \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$
using *f g fg G.cmp-components-are-iso D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D \varepsilon\ ?a)$
using *f g fg* $*$ *D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $G\ g \star_D G\ f\ D.inv\ (\Phi_G\ (g, f))\ \varepsilon\ ?a\ \tau_0\ ?a \star_D \tau_0{}'\ ?a$]
by *simp*
also have ... $= \tau_0{}'\ ?c \star_D (D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a) \cdot_D (G\ (g \star_C f) \star_D \varepsilon\ ?a)$
using *f g fg* $*$ *D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $D.inv\ (\Phi_G\ (g, f))\ G\ (g \star_C f)\ G_0\ ?a\ \varepsilon\ ?a$]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a)$
using *f g fg* $*$ *G.cmp-components-are-iso D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D (G\ g \star_D G\ f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D \tau_0\ ?a \star_D \tau_0{}'\ ?a)$

$= (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (((\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)))) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$
proof −
have $(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D G\ f]) \cdot_D (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a)$
$= \tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D G\ f] \cdot_D (D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a)$
using *f g fg* ∗ *G.cmp-components-are-iso D.whisker-left* **by** *simp*
also have ... $= \tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \cdot_D \mathrm{r}_D[G\ (g \star_C f)]$
using *f g fg* ∗ *G.cmp-components-are-iso D.runit-naturality* [*of D.inv* $(\Phi_G\ (g, f))$]
by *simp*
also have ... $= (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])$
using *f g fg* ∗ *G.cmp-components-are-iso D.whisker-left* **by** *simp*
finally have $(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ g \star_D G\ f]) \cdot_D (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)) \star_D G_0\ ?a) =$
$(\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f))) \cdot_D (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F\ g, F\ f, \tau_0{}'\ ?a]$
proof −
have $((\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)))) \cdot_D$

$(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])$
$= \tau_0{}'\ ?c \star_D (\Phi_G\ (g, f)) \cdot_D D.inv\ (\Phi_G\ (g, f))) \cdot_D \mathrm{r}_D[G\ (g \star_C f)]$
using *f g fg D.whisker-left G.cmp-components-are-iso*
$C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
by *simp*
also have ... $= \tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]$
using *f g fg G.cmp-components-are-iso D.comp-arr-inv′ D.comp-cod-arr* **by** *simp*
finally have $((\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D (\tau_0{}'\ ?c \star_D D.inv\ (\Phi_G\ (g, f)))) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])$
$= \tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[F_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.assoc-naturality* [*of* $\mathrm{l}_D{}^{-1}[F\ g]$ $F\ f\ \tau_0{}'\ ?a$] *D.comp-assoc* **by** *simp*
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a] \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, F\ f, \tau_0{}'\ ?a]) \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)$
proof −
have $((\eta\ ?c \star_D F\ g) \star_D F\ f \star_D \tau_0{}'\ ?a) \cdot_D \mathrm{a}_D[F_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]$
$= \mathrm{a}_D[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, F\ f, \tau_0{}'\ ?a] \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* **
D.assoc-naturality [*of* η $?c \star_D F\ g\ F\ f\ \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a]\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)$
proof −
have $(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f \star_D \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[(\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D F\ g, F\ f, \tau_0{}'\ ?a]$
$= \mathrm{a}_D[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg D.assoc-naturality* [*of* $\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g]\ F\ f\ \tau_0{}'\ ?a$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a]\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a))\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a$
$= \mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f]\ \cdot_D\ \mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg D.lunit-hcomp*(*2*) [*of F g F f*] **by** *simp*
also have ... $= (\mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D\ (\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
using *f g fg* ** *D.whisker-right* [*of* $\tau_0{}'\ ?a$] **by** *simp*
finally have $(\mathrm{l}_D{}^{-1}[F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a$
$= (\mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D\ (\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$((\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a]\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a))\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have $(((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D (\mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a)$
$= ((\eta\ ?c \star_D F\ g) \star_D F\ f)\ \cdot_D \mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg* ** *D.whisker-right* **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c, F\ g, F\ f]\ \cdot_D (\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a$
using *f g fg* ** *D.assoc′-naturality* [*of* η *?c F g F f*] **by** *simp*
also have ... $= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)$
using *f g fg* ** *D.whisker-right* **by** *simp*
finally have $(((\eta\ ?c \star_D F\ g) \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[trg_D\ (F\ g), F\ g, F\ f] \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\tau_0{}'\ ?c \star_D ((\tau_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g \star_D F\ f, \tau_0{}'\ ?a])\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have $(\tau_0{}'\ ?c \star_D \mathrm{a}_D[\tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g, F\ f \star_D \tau_0{}'\ ?a]\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D F\ g, F\ f, \tau_0{}'\ ?a]\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ g] \star_D F\ f) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D \tau_0\ ?c, F\ g, F\ f] \star_D \tau_0{}'\ ?a)$
$= \mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ g \star_D F\ f, \tau_0{}'\ ?a]\ \cdot_D$

$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D\ F\ f] \star_D\ \tau_0{}'\ ?a)$

proof −

have $(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[\tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g,\ F\ f \star_D\ \tau_0{}'\ ?a] \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ F\ g,\ F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g] \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ ?c \star_D\ \tau_0\ ?c,\ F\ g,\ F\ f] \star_D\ \tau_0{}'\ ?a)$
$= \{\!|(\langle\tau_0{}'\ ?c\rangle \star \mathbf{a}[\langle\tau_0\ ?c\rangle,\ \langle F\ g\rangle,\ \langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$(\langle\tau_0{}'\ ?c\rangle \star \mathbf{a}^{-1}[\langle\tau_0\ ?c\rangle \star \langle F\ g\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle\tau_0\ ?c\rangle \star \langle F\ g\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ ?a\rangle] \cdot$
$\mathbf{a}[\langle\tau_0{}'\ ?c\rangle \star \langle\tau_0\ ?c\rangle \star \langle F\ g\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle] \cdot$
$((\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle\tau_0\ ?c\rangle,\ \langle F\ g\rangle] \star \langle F\ f\rangle) \star \langle\tau_0{}'\ ?a\rangle) \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ ?c\rangle \star \langle\tau_0\ ?c\rangle,\ \langle F\ g\rangle,\ \langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle)|\!\}$

using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*

by *simp*

also have ... $= \{\!|\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle\tau_0\ ?c\rangle \star \langle F\ g\rangle \star \langle F\ f\rangle,\ \langle\tau_0{}'\ ?a\rangle] \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ ?c\rangle,\ \langle\tau_0\ ?c\rangle,\ \langle F\ g\rangle \star \langle F\ f\rangle] \star \langle\tau_0{}'\ ?a\rangle)|\!\}$

using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)

also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g \star_D\ F\ f,\ \tau_0{}'\ ?a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D\ F\ f] \star_D\ \tau_0{}'\ ?a)$

using *f g fg D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*

by *simp*

finally show *?thesis* **by** *simp*

qed

thus *?thesis*

using *D.comp-assoc* **by** *simp*

qed

also have ... $= (\tau_0{}'\ ?c \star_D\ \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ G\ (g \star_C f) \star_D\ \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ \mathrm{a}_D[G\ (g \star_C f),\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$(\tau_0{}'\ ?c \star_D\ D.inv\ (\tau_1\ (g \star_C f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ (g \star_C f),\ \tau_0{}'\ ?a] \cdot_D$
$(((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ \Phi_F\ (g, f)) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D\ F\ f] \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$((\eta\ ?c \star_D\ F\ g \star_D\ F\ f) \star_D\ \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D\ F\ f] \star_D\ \tau_0{}'\ ?a)$

proof −

have $(\tau_0{}'\ ?c \star_D\ ((\tau_0\ ?c \star_D\ \Phi_F\ (g, f)) \star_D\ \tau_0{}'\ ?a)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ g \star_D\ F\ f,\ \tau_0{}'\ ?a]$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D\ F\ (g \star_C f),\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D\ \tau_0\ ?c \star_D\ \Phi_F\ (g, f)) \star_D\ \tau_0{}'\ ?a)$

using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def D.assoc-naturality* [*of* $\tau_0{}'\ ?c\ \tau_0\ ?c \star_D\ \Phi_F\ (g, f)\ \tau_0{}'\ ?a$]

by *simp*

thus *?thesis*

using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f),\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ (g \star_C f),\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a))\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have $((\tau_0{}'\ ?c \star_D (\tau_0\ ?c \star_D \Phi_F\ (g, f))) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
$= (\tau_0{}'\ ?c \star_D \tau_0\ ?c \star_D \Phi_F\ (g, f))\ \cdot_D\ \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-char*$_{SbC}$ *F.FF-def D.whisker-right*
by *simp*
also have ... $= \mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)]\ \cdot_D$
$((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-char*$_{SbC}$ *F.FF-def D.assoc-naturality* [*of* $\tau_0{}'\ ?c\ \tau_0\ ?c\ \Phi_F\ (g, f)$]
by *simp*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)$
using *f g fg C.VV.arr-char*$_{SbC}$ *C.VV.dom-simp C.VV.cod-char*$_{SbC}$ *F.FF-def D.whisker-right*
by *simp*
finally have $((\tau_0{}'\ ?c \star_D (\tau_0\ ?c \star_D \Phi_F\ (g, f))) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
$= (\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a)\ \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f),\ \tau_0\ ?a,\ \tau_0{}'\ ?a])\ \cdot_D$
$(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ (g \star_C f),\ \tau_0{}'\ ?a]\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)] \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ (g \star_C f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
proof −
have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)\ \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f))\ \cdot_D\ (\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a$

using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def D.whisker-right*
by *simp*
also have ... $= (\eta\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def
D.interchange [of $\tau_0{}'\ ?c \star_D \tau_0\ ?c\ \eta\ ?c\ \Phi_F\ (g, f)\ F\ g \star_D F\ f$*]*
by *simp*
also have ... $= (\eta\ ?c \star_D F\ (g \star_C f)) \cdot_D (F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a$
using *f g fg D.comp-arr-dom D.comp-cod-arr*
C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def
D.interchange [of $\eta\ ?c\ F_0\ ?c\ F\ (g \star_C f)\ \Phi_F\ (g, f)$*]*
by *simp*
also have ... $= ((\eta\ ?c \star_D F\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D ((F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)$
using *f g fg ∗∗ C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def D.whisker-right*
by *simp*
finally have $(((\tau_0{}'\ ?c \star_D \tau_0\ ?c) \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ g \star_D F\ f) \star_D \tau_0{}'\ ?a)$
$= ((\eta\ ?c \star_D F\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D ((F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f), \tau_0\ ?a, \tau_0{}'\ ?a]) \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c \star_D F\ (g \star_C f), \tau_0{}'\ ?a]) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c, \tau_0\ ?c, F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$
proof −
have $((F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
$= (F_0\ ?c \star_D \Phi_F\ (g, f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a$
using *f g fg ∗∗ C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def D.whisker-right*
by *simp*
also have ... $= \mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \cdot_D \Phi_F\ (g, f) \star_D \tau_0{}'\ ?a$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def*
D.lunit′-naturality [of $\Phi_F\ (g, f)$*]*
by *simp*
also have ... $= (\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D (\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$
using *f g fg C.VV.arr-char$_{SbC}$ C.VV.dom-simp C.VV.cod-char$_{SbC}$ F.FF-def D.whisker-right*
by *simp*
finally have $((F_0\ ?c \star_D \Phi_F\ (g, f)) \star_D \tau_0{}'\ ?a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ g \star_D F\ f] \star_D \tau_0{}'\ ?a)$
$= (\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D (\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$

by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= (\tau_0{}'\ ?c \star_D \mathrm{r}_D[G\ (g \star_C f)]) \cdot_D$
$(\tau_0{}'\ ?c \star_D G\ (g \star_C f) \star_D \varepsilon\ ?a) \cdot_D$
$(\tau_0{}'\ ?c \star_D \mathrm{a}_D[G\ (g \star_C f),\ \tau_0\ ?a,\ \tau_0{}'\ ?a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ G\ (g \star_C f) \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f))) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c,\ F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D$
$((\eta\ ?c \star_D F\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (g \star_C f)] \star_D \tau_0{}'\ ?a) \cdot_D$
$(\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$
proof −
have $(\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f)) \star_D \tau_0{}'\ ?a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ ?c,\ \tau_0\ ?c \star_D F\ (g \star_C f),\ \tau_0{}'\ ?a]$
$= \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ (g \star_C f) \star_D \tau_0\ ?a,\ \tau_0{}'\ ?a] \cdot_D$
$((\tau_0{}'\ ?c \star_D D.inv\ (\tau_1\ (g \star_C f))) \star_D \tau_0{}'\ ?a)$
using *f g fg* τ*.iso-map*$_1$*-ide*
D.assoc-naturality [*of* $\tau_0{}'$ *?c D.inv* $(\tau_1\ (g \star_C f))$ $\tau_0{}'$ *?a*]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \tau_1{}'\ (g \star_C f) \cdot_D (\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$
unfolding *map*$_1$*-def*
using *f g fg D.comp-assoc* **by** *simp*
finally show $(\tau_0{}'\ ?c \star_D \Phi_G\ (g, f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ ?c,\ G\ g,\ G\ f] \cdot_D (\tau_1{}'\ g \star_D G\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0{}'\ (src_C\ g),\ G\ f] \cdot_D (F\ g \star_D \tau_1{}'\ f) \cdot_D \mathrm{a}_D[F\ g,\ F\ f,\ \tau_0{}'\ ?a]$
$= \tau_1{}'\ (g \star_C f) \cdot_D (\Phi_F\ (g, f) \star_D \tau_0{}'\ ?a)$
by *blast*
qed
qed

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ F\ \Phi_F\ \tau_0{}'\ \tau_1{}'$
..

end

1.14.6 Pseudonaturally Equivalent Pseudofunctors

Pseudofunctors F and G are *pseudonaturally equivalent* if there is a pseudonatural equivalence between them.

definition *pseudonaturally-equivalent*
where *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G \equiv$

$\exists\, \tau_0\ \tau_1.$ *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$

lemma *pseudonaturally-equivalent-reflexive*:
assumes *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
shows *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ F\ \Phi_F$
proof −
interpret F: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
using *assms* **by** *simp*
interpret *identity-pseudonatural-transformation*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
..
show *?thesis*
using *pseudonatural-equivalence-axioms pseudonaturally-equivalent-def* **by** *blast*
qed

lemma *pseudonaturally-equivalent-symmetric*:
assumes *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G$
shows *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$
proof −
obtain $\tau_0\ \tau_1$ **where** τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
using *assms pseudonaturally-equivalent-def* **by** *blast*
interpret τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
using τ **by** *simp*
interpret σ: *converse-pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
..
show *?thesis*
using σ*.is-pseudonatural-equivalence pseudonaturally-equivalent-def* **by** *blast*
qed

lemma *pseudonaturally-equivalent-transitive*:
assumes *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G$
and *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ H\ \Phi_H$
shows *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ H\ \Phi_H$
proof −
obtain $\tau_0\ \tau_1$ **where** τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
using *assms pseudonaturally-equivalent-def* **by** *blast*
interpret τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$

using τ **by** *simp*
obtain σ_0 σ_1 **where** σ: *pseudonatural-equivalence*
V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C G Φ_G H Φ_H σ_0 σ_1
using *assms pseudonaturally-equivalent-def* **by** *blast*
interpret σ: *pseudonatural-equivalence*
V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C G Φ_G H Φ_H σ_0 σ_1
using σ **by** *simp*
interpret $\sigma\tau$: *composite-pseudonatural-equivalence* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C F Φ_F G Φ_G H Φ_H τ_0 τ_1 σ_0 σ_1
..
show *?thesis*
using $\sigma\tau$.*pseudonatural-equivalence-axioms pseudonaturally-equivalent-def* **by** *blast*
qed

If τ is a pseudonatural equivalence from pseudofunctor F to pseudofunctor G and σ is the converse equivalence, then F is locally naturally isomorphic to the functor that takes a 2-cell μ of C to σ_0 $(trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)$ and symmetrically for G. Here we just establish the naturality property and and that each 1-cell «$g : a \to_C a'$» is isomorphic to $\tau_0\ a' \star_D (\sigma_0\ a' \star_D g \star_D \tau_0\ a) \star_D \sigma_0\ a$. We ultimately need these results to prove that a pseudofunctor extends to an equivalence of bicategories if and only if it is an equivalence pseudofunctor.

context *pseudonatural-equivalence*
begin

interpretation σ: *converse-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1
..

abbreviation (*input*) σ_0
where $\sigma_0 \equiv \sigma.map_0$

definition φ
where $\varphi\ f \equiv \mathrm{a}_D[\tau_0\ (trg_C\ f),\ F\ f,\ \sigma_0\ (src_C\ f)] \cdot_D$
$(\tau_1\ f \star_D \sigma_0\ (src_C\ f)) \cdot_D$
${\mathrm{a}_D}^{-1}[G\ f,\ \tau_0\ (src_C\ f),\ \sigma_0\ (src_C\ f)] \cdot_D$
$(G\ f \star_D D.inv\ (\sigma.\varepsilon\ (src_C\ f))) \cdot_D$
${\mathrm{r}_D}^{-1}[G\ f]$

lemma φ-*in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\varphi\ f : G\ f \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ f \star_D \sigma_0\ (src_C\ f)$»
unfolding φ-*def*
using *assms* **by** (*intro D.comp-in-homI′ D.hseqI′*) *auto*

lemma *iso-*φ:
assumes $C.ide\ f$
shows $D.iso\ (\varphi\ f)$
unfolding φ-*def*
using *assms iso-*map_1*-ide*

by (*intro D.isos-compose*) *auto*

definition ψ
where $\psi\ f \equiv (\sigma_0\ (trg_C\ f) \star_D D.inv\ (\tau_1\ f)) \cdot_D$
$\mathrm{a}_D[\sigma_0\ (trg_C\ f),\ \tau_0\ (trg_C\ f),\ F\ f] \cdot_D$
$(\sigma.\eta\ (trg_C\ f) \star_D F\ f) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ f]$

lemma *ψ-in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\psi\ f : F\ f \Rightarrow_D \sigma_0\ (trg_C\ f) \star_D G\ f \star_D \tau_0\ (src_C\ f)$»
unfolding *ψ-def*
using *assms iso-map*$_1$*-ide*
by (*intro D.comp-in-homI′ D.hseqI′*) *auto*

lemma *iso-ψ*:
assumes $C.ide\ f$
shows $D.iso\ (\psi\ f)$
unfolding *ψ-def*
using *assms iso-map*$_1$*-ide*
by (*intro D.isos-compose*) *auto*

lemma *ψ-naturality*:
assumes $C.arr\ \mu$
shows $(\sigma_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)) \cdot_D \psi\ (C.dom\ \mu) = \psi\ (C.cod\ \mu) \cdot_D F\ \mu$
and $D.inv\ (\psi\ (C.cod\ \mu)) \cdot_D (\sigma_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)) \cdot_D \psi\ (C.dom\ \mu) = F\ \mu$
proof −
let $?a = src_C\ \mu$ **and** $?a' = trg_C\ \mu$
let $?f = C.dom\ \mu$ **and** $?f' = C.cod\ \mu$
have $(\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D \psi\ (C.dom\ \mu)$
$= ((\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D$
$(\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f))) \cdot_D$
$\mathrm{a}_D[\sigma_0\ ?a',\ \tau_0\ ?a',\ F\ ?f] \cdot_D$
$(\sigma.\eta\ ?a' \star_D F\ ?f) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ ?f]$
unfolding *ψ-def*
using *assms D.comp-assoc* **by** *simp*
also have ... $= (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f')) \cdot_D$
$((\sigma_0\ ?a' \star_D \tau_0\ ?a' \star_D F\ \mu) \cdot_D$
$\mathrm{a}_D[\sigma_0\ ?a',\ \tau_0\ ?a',\ F\ ?f]) \cdot_D$
$(\sigma.\eta\ ?a' \star_D F\ ?f) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ ?f]$
proof −
have $(\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f))$
$= \sigma_0\ ?a' \star_D (G\ \mu \star_D \tau_0\ ?a) \cdot_D D.inv\ (\tau_1\ ?f)$
using *assms D.whisker-left iso-map*$_1$*-ide* **by** *simp*
also have ... $= \sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f') \cdot_D (\tau_0\ ?a' \star_D F\ \mu)$
using *assms naturality iso-map*$_1$*-ide*
D.invert-opposite-sides-of-square

$[of\ \tau_1\ ?f'\ G\ \mu \star_D \tau_0\ ?a\ \tau_0\ ?a' \star_D F\ \mu\ \tau_1\ ?f]$
by *simp*
also have ... $= (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f')) \cdot_D (\sigma_0\ ?a' \star_D \tau_0\ ?a' \star_D F\ \mu)$
using *assms D.whisker-left iso-map*$_1$*-ide* **by** *simp*
finally have $(\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f))$
$= (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f')) \cdot_D (\sigma_0\ ?a' \star_D \tau_0\ ?a' \star_D F\ \mu)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f')) \cdot_D$
$\mathrm{a}_D[\sigma_0\ ?a', \tau_0\ ?a', F\ ?f'] \cdot_D$
$(((\sigma_0\ ?a' \star_D \tau_0\ ?a') \star_D F\ \mu) \cdot_D$
$(\sigma.\eta\ ?a' \star_D F\ ?f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ ?f]$
proof −
have $(\sigma_0\ ?a' \star_D \tau_0\ ?a' \star_D F\ \mu) \cdot_D \mathrm{a}_D[\sigma_0\ ?a', \tau_0\ ?a', F\ ?f]$
$= \mathrm{a}_D[\sigma_0\ ?a', \tau_0\ ?a', F\ ?f'] \cdot_D ((\sigma_0\ ?a' \star_D \tau_0\ ?a') \star_D F\ \mu)$
using *assms D.assoc-naturality* $[of\ \sigma_0\ ?a'\ \tau_0\ ?a'\ F\ \mu]$ **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\sigma_0\ ?a' \star_D D.inv\ (\tau_1\ ?f')) \cdot_D$
$\mathrm{a}_D[\sigma_0\ ?a', \tau_0\ ?a', F\ ?f'] \cdot_D$
$(\sigma.\eta\ ?a' \star_D F\ ?f') \cdot_D$
$(F.map_0\ ?a' \star_D F\ \mu) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ ?f]$
proof −
have $((\sigma_0\ ?a' \star_D \tau_0\ ?a') \star_D F\ \mu) \cdot_D (\sigma.\eta\ ?a' \star_D F\ ?f)$
$= (\sigma_0\ ?a' \star_D \tau_0\ ?a') \cdot_D \sigma.\eta\ ?a' \star_D F\ \mu \cdot_D F\ ?f$
using *assms D.interchange* $[of\ \sigma_0\ ?a' \star_D \tau_0\ ?a'\ \sigma.\eta\ ?a'\ F\ \mu\ F\ ?f]$
by *simp*
also have ... $= \sigma.\eta\ ?a' \cdot_D F.map_0\ ?a' \star_D F\ ?f' \cdot_D F\ \mu$
using *assms D.comp-arr-dom D.comp-cod-arr* **by** *simp*
also have ... $= (\sigma.\eta\ ?a' \star_D F\ ?f') \cdot_D (F.map_0\ ?a' \star_D F\ \mu)$
using *assms D.interchange* $[of\ \sigma.\eta\ ?a'\ F.map_0\ ?a'\ F\ ?f'\ F\ \mu]$
σ*.unit-in-hom* $[of\ ?a']$
by *fastforce*
finally have $((\sigma_0\ ?a' \star_D \tau_0\ ?a') \star_D F\ \mu) \cdot_D (\sigma.\eta\ ?a' \star_D F\ ?f)$
$= (\sigma.\eta\ ?a' \star_D F\ ?f') \cdot_D (F.map_0\ ?a' \star_D F\ \mu)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \psi\ ?f' \cdot_D F\ \mu$
proof −
have $(F.map_0\ ?a' \star_D F\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[F\ ?f] = \mathrm{l}_D{}^{-1}[F\ ?f'] \cdot_D F\ \mu$
using *assms D.lunit′-naturality* $[of\ F\ \mu]$ **by** *simp*
thus *?thesis*

unfolding ψ-*def*
using *assms D.comp-assoc* **by** *simp*
qed
finally show $(\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D \psi\ (C.dom\ \mu) = \psi\ ?f' \cdot_D F\ \mu$
by *blast*
thus $D.inv\ (\psi\ ?f') \cdot_D (\sigma_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a) \cdot_D \psi\ (C.dom\ \mu) = F\ \mu$
using *assms* ψ-*in-hom iso*-ψ
by (*metis C.ide-cod D.in-homE D.invert-side-of-triangle*(*1*) *D.seqI*
F.preserves-arr F.preserves-cod)
qed

lemma *isomorphic-expansion*:
assumes $C.obj\ a$ **and** $C.obj\ a'$ **and** «$g : G.map_0\ a \rightarrow_D G.map_0\ a'$» **and** $D.ide\ g$
shows $D.isomorphic\ (\tau_0\ a' \star_D (\sigma_0\ a' \star_D g \star_D \tau_0\ a) \star_D \sigma_0\ a)\ g$
proof −
let $?g' = \sigma_0\ a' \star_D g \star_D \tau_0\ a$
have g': «$?g' : F.map_0\ a \rightarrow_D F.map_0\ a'$»
using *assms ide-map*$_0$-*obj* σ.*map*$_0$-*simps*(*3*) *C.obj-simps*
by (*intro D.in-hhomI D.hseqI'*) *auto*
have *ide-g'*: $D.ide\ ?g'$
using *assms* g' σ.*ide-map*$_0$-*obj ide-map*$_0$-*obj* **by** *blast*
let $?\psi = (\sigma_0\ a' \star_D \mathrm{r}_D[g]) \cdot_D (\sigma_0\ a' \star_D g \star_D \sigma.\varepsilon\ a) \cdot_D$
$(\sigma_0\ a' \star_D \mathrm{a}_D[g, \tau_0\ a, \sigma_0\ a]) \cdot_D \mathrm{a}_D[\sigma_0\ a', g \star_D \tau_0\ a, \sigma_0\ a]$
have ψ: «$?\psi : ?g' \star_D \sigma_0\ a \Rightarrow_D \sigma_0\ a' \star_D g$»
proof (*intro D.comp-in-homI*)
show «$\mathrm{a}_D[\sigma_0\ a', g \star_D \tau_0\ a, \sigma_0\ a] :$
$(\sigma_0\ a' \star_D g \star_D \tau_0\ a) \star_D \sigma_0\ a \Rightarrow_D \sigma_0\ a' \star_D (g \star_D \tau_0\ a) \star_D \sigma_0\ a$»
using *assms* g' **by** *fastforce*
show «$\sigma_0\ a' \star_D \mathrm{a}_D[g, \tau_0\ a, \sigma_0\ a] :$
$\sigma_0\ a' \star_D (g \star_D \tau_0\ a) \star_D \sigma_0\ a \Rightarrow_D \sigma_0\ a' \star_D g \star_D \tau_0\ a \star_D \sigma_0\ a$»
using *assms* g' **by** *fastforce*
show «$\sigma_0\ a' \star_D g \star_D \sigma.\varepsilon\ a :$
$\sigma_0\ a' \star_D g \star_D \tau_0\ a \star_D \sigma_0\ a \Rightarrow_D \sigma_0\ a' \star_D g \star_D G.map_0\ a$»
using *assms C.obj-simps*
by (*intro D.hcomp-in-vhom*) *auto*
show «$\sigma_0\ a' \star_D \mathrm{r}_D[g] : \sigma_0\ a' \star_D g \star_D G.map_0\ a \Rightarrow_D \sigma_0\ a' \star_D g$»
using *assms*
apply (*intro D.hcomp-in-vhom*)
apply *auto*
by *fastforce*
qed
have *iso*-ψ: $D.iso\ ?\psi$
using *assms* g' ψ *ide-g' ide-map*$_0$-*obj C.obj-simps*
by (*intro D.isos-compose*) *auto*
let $?\varphi = \mathrm{l}_D[g] \cdot_D (\sigma.\varepsilon\ a' \star_D g) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ a', \sigma_0\ a', g] \cdot_D (\tau_0\ a' \star_D ?\psi)$
have φ: «$?\varphi : \tau_0\ a' \star_D ?g' \star_D \sigma_0\ a \Rightarrow_D g$»
proof (*intro D.comp-in-homI*)
show «$\tau_0\ a' \star_D ?\psi : \tau_0\ a' \star_D ?g' \star_D \sigma_0\ a \Rightarrow_D \tau_0\ a' \star_D \sigma_0\ a' \star_D g$»
using *assms* g' ψ *C.obj-simps*

by (*intro D.hcomp-in-vhom*) *auto*
show «$\mathrm{a}_D{}^{-1}[\tau_0\ a',\ \sigma_0\ a',\ g] : \tau_0\ a' \star_D \sigma_0\ a' \star_D g \Rightarrow_D (\tau_0\ a' \star_D \sigma_0\ a') \star_D g$»
proof −
have $src_D\ (\tau_0\ a') = trg_D\ (\sigma_0\ a')$
using *assms* **by** *auto*
moreover have $src_D\ (\sigma_0\ a') = trg_D\ g$
using *assms* **by** *auto*
ultimately show *?thesis*
using *assms ide-map*$_0$*-obj* σ*.ide-map*$_0$*-obj* **by** *auto*
qed
show «$\sigma.\varepsilon\ a' \star_D g : (\tau_0\ a' \star_D \sigma_0\ a') \star_D g \Rightarrow_D G.map_0\ a' \star_D g$»
using *assms* **by** *fastforce*
show «$\mathrm{l}_D[g] : G.map_0\ a' \star_D g \Rightarrow_D g$»
using *assms* **by** *auto*
qed
have *iso-*φ: *D.iso ?*φ
using *assms* g' φ ψ *iso-*ψ *ide-map*$_0$*-obj iso-map*$_1$*-ide* σ*.ide-map*$_0$*-obj*
apply (*intro D.isos-compose*)
apply (*meson D.arrI D.hseqE D.ide-is-iso D.iso-hcomp D.seqE*)
apply (*metis D.hseqE D.in-hhomE D.iso-assoc′ D.trg-hcomp F.map*$_0$*-def map*$_0$*-simps*(*2*))
by *auto*
show *?thesis*
using φ *iso-*φ *D.isomorphic-def* **by** *auto*
qed

end

Here we show that pseudonatural equivalence respects equivalence pseudofunctors, in the sense that a pseudofunctor G, pseudonaturally equivalent to an equivalence pseudofunctor F, is itself an equivalence pseudofunctor.

locale *pseudofunctor-pseudonaturally-equivalent-to-equivalence-pseudofunctor* =
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F: *equivalence-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$ +
pseudofunctor $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$ +
τ: *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
for V_C :: *′c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *′c comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: *′d comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: *′d comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$

and $F :: 'c \Rightarrow 'd$
and $\Phi_F :: 'c * 'c \Rightarrow 'd$
and $G :: 'c \Rightarrow 'd$
and $\Phi_G :: 'c * 'c \Rightarrow 'd$
and $\tau_0 :: 'c \Rightarrow 'd$
and $\tau_1 :: 'c \Rightarrow 'd$
begin

interpretation σ': *converse-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
..

sublocale G: *equivalence-pseudofunctor*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
proof
show $\bigwedge \mu\ \mu'.\ [\![C.par\ \mu\ \mu';\ G\ \mu = G\ \mu']\!] \Longrightarrow \mu = \mu'$
proof −
fix $\mu\ \mu'$
assume *par*: $C.par\ \mu\ \mu'$ **and** *eq*: $G\ \mu = G\ \mu'$
let $?a = src_C\ \mu$ **and** $?a' = trg_C\ \mu$
let $?f = C.dom\ \mu$ **and** $?f' = C.cod\ \mu$
have $\tau_0\ ?a' \star_D F\ \mu = (\tau_0\ ?a' \star_D F\ \mu) \cdot_D \tau_1\ ?f \cdot_D D.inv\ (\tau_1\ ?f)$
using *par* τ.*ide-map*$_0$*-obj* τ.*iso-map*$_1$*-ide D.comp-arr-dom D.comp-arr-inv'* τ.*map*$_1$*-in-hom* [*of* μ]
by (*metis C.dom-trg C.ide-dom C.obj-trg C.trg.preserves-dom D.whisker-left F.as-nat-trans.naturality2 F.as-nat-trans.naturality F.preserves-arr* τ.*map*$_1$*-simps*(*5*))
also have ... $= ((\tau_0\ ?a' \star_D F\ \mu) \cdot_D \tau_1\ ?f) \cdot_D D.inv\ (\tau_1\ ?f)$
using *D.comp-assoc* **by** *simp*
also have ... $= ((\tau_0\ ?a' \star_D F\ \mu') \cdot_D \tau_1\ ?f) \cdot_D D.inv\ (\tau_1\ ?f)$
using *eq par* τ.*naturality* [*of* μ] τ.*naturality* [*of* μ'] *C.src-cod C.trg-cod* **by** *metis*
also have *1*: ... $= (\tau_0\ ?a' \star_D F\ \mu') \cdot_D \tau_1\ ?f \cdot_D D.inv\ (\tau_1\ ?f)$
using *D.comp-assoc* **by** *blast*
also have ... $= \tau_0\ ?a' \star_D F\ \mu'$
using *par* τ.*iso-map*$_1$*-ide D.comp-arr-dom D.comp-arr-inv'* τ.*map*$_1$*-in-hom* [*of* μ']
by (*metis 1 C.ide-dom C.trg-cod D.hseqE D.hseqI' D.hseq-char' D.seqE F.preserves-arr F.preserves-trg calculation*)
finally have $\tau_0\ ?a' \star_D F\ \mu = \tau_0\ ?a' \star_D F\ \mu'$
by *blast*
hence $F\ \mu = F\ \mu'$
using *par* τ.*components-are-equivalences D.equivalence-cancel-left* [*of* $\tau_0\ ?a'\ F\ \mu\ F\ \mu'$]
by *simp*
thus $\mu = \mu'$
using *par F.is-faithful* **by** *blast*
qed
show $\bigwedge b.\ D.obj\ b \Longrightarrow \exists\, a.\ C.obj\ a \wedge D.equivalent\text{-}objects\ (map_0\ a)\ b$
proof −
fix b

assume b: $D.obj\ b$
obtain a **where** a: $C.obj\ a \wedge D.equivalent\text{-}objects\ (F.map_0\ a)\ b$
using b $F.biessentially\text{-}surjective\text{-}on\text{-}objects$ **by** $blast$
have $D.equivalent\text{-}objects\ (F.map_0\ a)\ (map_0\ a)$
using a $\tau.components\text{-}are\text{-}equivalences$ $D.equivalent\text{-}objects\text{-}def$
by ($metis$ $F.map_0\text{-}def$ $map_0\text{-}def$ $\tau.map_0\text{-}in\text{-}hhom$)
hence $D.equivalent\text{-}objects\ (map_0\ a)\ b$
using a $D.equivalent\text{-}objects\text{-}symmetric$ $D.equivalent\text{-}objects\text{-}transitive$ **by** $blast$
thus $\exists a.\ C.obj\ a \wedge D.equivalent\text{-}objects\ (map_0\ a)\ b$
using a **by** $auto$
qed
show $\bigwedge a\ a'\ g.\ [\![C.obj\ a;\ C.obj\ a';\ «g : map_0\ a \rightarrow_D map_0\ a'»;\ D.ide\ g]\!]$
$\Longrightarrow \exists f.\ «f : a \rightarrow_C a'» \wedge C.ide\ f \wedge D.isomorphic\ (G\ f)\ g$
proof −
fix $a\ a'\ g$
assume a: $C.obj\ a$ **and** a': $C.obj\ a'$
assume g: $«g : map_0\ a \rightarrow_D map_0\ a'»$ **and** $ide\text{-}g$: $D.ide\ g$
interpret σ: $converse\text{-}pseudonatural\text{-}equivalence$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
..
interpret σ: $pseudonatural\text{-}equivalence$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$G\ \Phi_G\ F\ \Phi_F\ \sigma'.map_0\ \sigma'.map_1$
using $\sigma'.is\text{-}pseudonatural\text{-}equivalence$ **by** $simp$
let $?g' = \sigma'.map_0\ a' \star_D g \star_D \tau_0\ a$
have g': $«?g' : F.map_0\ a \rightarrow_D F.map_0\ a'»$
using $a\ a'\ g\ \tau.ide\text{-}map_0\text{-}obj\ \sigma'.map_0\text{-}simps(3)\ C.obj\text{-}simps$
by ($intro\ D.in\text{-}hhomI\ D.hseqI'$) $auto$
have $ide\text{-}g'$: $D.ide\ ?g'$
using $a\ a'\ g\ g'\ ide\text{-}g$
by ($meson\ D.hcomp\text{-}in\text{-}hhomE\ D.hseqE\ D.ide\text{-}hcomp\ \sigma'.ide\text{-}map_0\text{-}obj\ \tau.ide\text{-}map_0\text{-}obj$)
obtain f **where** f: $«f : a \rightarrow_C a'» \wedge C.ide\ f \wedge D.isomorphic\ (F\ f)\ ?g'$
using $a\ a'\ g\ g'\ ide\text{-}g'\ F.locally\text{-}essentially\text{-}surjective\ [of\ a\ a'\ ?g']$ **by** $auto$
obtain ξ **where** ξ: $«\xi : F\ f \Rightarrow_D ?g'» \wedge D.iso\ \xi$
using f $D.isomorphic\text{-}def$ **by** $blast$
have $D.isomorphic\ (G\ f)\ g$
proof −
have $D.isomorphic\ (G\ f)\ (\tau_0\ a' \star_D F\ f \star_D \sigma'.map_0\ a)$
using $a\ a'\ f\ \tau.iso\text{-}\varphi\ D.isomorphic\text{-}def$ **by** $blast$
also have $D.isomorphic\ ...\ (\tau_0\ a' \star_D ?g' \star_D \sigma'.map_0\ a)$
using $a\ a'\ g'\ \xi\ D.isomorphic\text{-}def$
by ($metis\ D.hcomp\text{-}ide\text{-}isomorphic\ D.hcomp\text{-}isomorphic\text{-}ide\ D.hseqE\ D.ideD(1)$
$D.isomorphic\text{-}implies\text{-}ide(2)\ \sigma'.ide\text{-}map_0\text{-}obj\ calculation\ \tau.ide\text{-}map_0\text{-}obj$)
also have $D.isomorphic\ ...\ g$
using $a\ a'\ g\ ide\text{-}g\ \tau.isomorphic\text{-}expansion$ **by** $simp$
finally show $?thesis$ **by** $blast$
qed
thus $\exists f.\ «f : a \rightarrow_C a'» \wedge C.ide\ f \wedge D.isomorphic\ (G\ f)\ g$
using f **by** $auto$

qed
show $\bigwedge f\, f'\, \nu.\ [\![C.ide\ f;\ C.ide\ f';\ src_C\ f = src_C\ f';\ trg_C\ f = trg_C\ f';\ «\nu : G\ f \Rightarrow_D G\ f'»]\!]$
$\Longrightarrow \exists \mu.\ «\mu : f \Rightarrow_C f'» \wedge G\ \mu = \nu$
proof −
fix $f\, f'\, \nu$
assume f: $C.ide\ f$ **and** f': $C.ide\ f'$
and *eq-src*: $src_C\ f = src_C\ f'$ **and** *eq-trg*: $trg_C\ f = trg_C\ f'$
and ν: $«\nu : G\ f \Rightarrow_D G\ f'»$
let $?a = src_C\ f$ **and** $?a' = trg_C\ f$
let $?\nu' = D.inv\ (\tau.\psi\ f') \cdot_D (\sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a) \cdot_D \tau.\psi\ f$
have ν': $«?\nu' : F\ f \Rightarrow_D F\ f'»$
proof (*intro D.comp-in-homI*)
show $«\tau.\psi\ f : F\ f \Rightarrow_D \sigma'.map_0\ ?a' \star_D G\ f \star_D \tau_0\ ?a»$
using f $\tau.\psi$*-in-hom* [*of* f] **by** *blast*
show $«\sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a :$
$\sigma'.map_0\ ?a' \star_D G\ f \star_D \tau_0\ ?a \Rightarrow_D \sigma'.map_0\ ?a' \star_D G\ f' \star_D \tau_0\ ?a»$
using f ν **by** (*intro D.hcomp-in-vhom*) *auto*
show $«D.inv\ (\tau.\psi\ f') : \sigma'.map_0\ ?a' \star_D G\ f' \star_D \tau_0\ ?a \Rightarrow_D F\ f'»$
using f' τ*.iso-*ψ *eq-src eq-trg* **by** *auto*
qed
obtain μ **where** μ: $«\mu : f \Rightarrow_C f'» \wedge F\ \mu = ?\nu'$
using $f\, f'$ *eq-src eq-trg* ν' *F.locally-full* [*of* $f\, f'$ $?\nu'$] **by** *blast*
have $G\ \mu = \nu$
proof −
have $D.inv\ (\tau.\psi\ f') \cdot_D (\sigma'.map_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)) \cdot_D \tau.\psi\ f =$
$D.inv\ (\tau.\psi\ f') \cdot_D (\sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a) \cdot_D \tau.\psi\ f$
using μ $\tau.\psi$*-naturality* [*of* μ] **by** *auto*
hence $\sigma'.map_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu) = \sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a$
using $f\, f'\, \nu'$ τ*.iso-*ψ
D.iso-cancel-left
[*of* $D.inv\ (\tau.\psi\ f')\ (\sigma'.map_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)) \cdot_D \tau.\psi\ f$
$(\sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a) \cdot_D \tau.\psi\ f$]
D.iso-cancel-right
[*of* $\tau.\psi\ f\ \sigma'.map_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)\ \sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a$]
by *auto*
hence $\sigma'.map_0\ ?a' \star_D G\ \mu \star_D \tau_0\ ?a = \sigma'.map_0\ ?a' \star_D \nu \star_D \tau_0\ ?a$
using μ **by** *auto*
moreover have $D.par\ (G\ \mu \star_D \tau_0\ (src_C\ f))\ (\nu \star_D \tau_0\ (src_C\ f))$
using $f\ \mu\ \nu\ \nu'$
apply (*intro conjI D.hseqI'*)
apply (*auto simp add*: *D.vconn-implies-hpar*(*1*))
apply *fastforce*
by (*metis C.ideD*(*1*) *C.vconn-implies-hpar*(*1*) *D.hcomp-simps*(*4*) *D.hseqE D.hseqI'*
D.seqE D.vconn-implies-hpar(*1*) *preserves-arr preserves-cod preserves-src*
weak-arrow-of-homs-axioms category.in-homE horizontal-homs-def
weak-arrow-of-homs-def)
ultimately have $G\ \mu \star_D \tau_0\ ?a = \nu \star_D \tau_0\ ?a$
using $f\, f'\, \mu\ \nu$ $\tau.ide\text{-}map_0\text{-}obj$ $\sigma'.ide\text{-}map_0\text{-}obj$
D.equivalence-cancel-left [*of* $\sigma'.map_0\ ?a'\ G\ \mu \star_D \tau_0\ ?a\ \nu \star_D \tau_0\ ?a$]

σ'.*components-are-equivalences*
by *auto*
moreover have $D.par\ (G\ \mu)\ \nu$
using $f\ f'\ \mu\ \nu$ **by** *fastforce*
ultimately show *?thesis*
using $f\ f'\ \mu\ \nu\ \tau$.*ide-map*$_0$*-obj*
D.*equivalence-cancel-right* [*of* τ_0 *?a* $G\ \mu\ \nu$] τ.*components-are-equivalences*
by *auto*
qed
thus $\exists \mu.$ «$\mu : f \Rightarrow_C f'$» $\wedge\ G\ \mu = \nu$
using μ **by** *auto*
qed
qed

lemma *is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
..

end

lemma *pseudonaturally-equivalent-respects-equivalence-pseudofunctor*:
assumes *pseudonaturally-equivalent*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G$
and *equivalence-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
shows *equivalence-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G$
proof −
obtain $\tau_0\ \tau_1$ **where** τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
using *assms pseudonaturally-equivalent-def* **by** *blast*
interpret F: *equivalence-pseudofunctor*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
using *assms* **by** *simp*
interpret G: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G$
using *assms* τ
by (*simp add*: *pseudonatural-equivalence-def pseudonatural-transformation-def*)
interpret τ: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
using τ **by** *simp*
interpret G: *pseudofunctor-pseudonaturally-equivalent-to-equivalence-pseudofunctor*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
..
show *equivalence-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G$
using G.*is-equivalence-pseudofunctor* **by** *simp*
qed

end

1.15 Modifications

Modifications are morphisms of pseudonatural transformations. In this section, we give a definition of "modification", and we prove that the mappings η and ε, which were chosen in the course of showing that a pseudonatural equivalence τ has a converse pseudonatural equivalence τ', are invertible modifications that relate the composites $\tau'\tau$ and $\tau\tau'$ to the identity pseudonatural transformations on F and G. This means that τ and τ' are quasi-inverses in a suitable bicategory of pseudofunctors, pseudonatural transformations, and modifications, though we do not go so far as to give a formal construction of such a bicategory.

theory *Modification*
imports *PseudonaturalTransformation*
begin

locale *modification* =
  C: *bicategory* V_C H_C a_C i_C src_C trg_C +
  D: *bicategory* V_D H_D a_D i_D src_D trg_D +
  τ: *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1 +
  τ': *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G $\tau_0{}'$ $\tau_1{}'$
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and τ_0 :: $'c \Rightarrow 'd$
and τ_1 :: $'c \Rightarrow 'd$
and $\tau_0{}'$ :: $'c \Rightarrow 'd$
and $\tau_1{}'$ :: $'c \Rightarrow 'd$
and Γ :: $'c \Rightarrow 'd$ +
assumes Γ-*in-hom*: $C.obj\ a \Longrightarrow$ «$\Gamma\ a : \tau_0\ a \Rightarrow_D \tau_0{}'\ a$»
and *naturality*: $[\![$ «$f : a \rightarrow_C b$»; $C.ide\ f$ $]\!] \Longrightarrow \tau_1{}'\ f \cdot_D (G\ f \star_D \Gamma\ a) = (\Gamma\ b \star_D F\ f) \cdot_D \tau_1\ f$

locale *invertible-modification* =
  modification +
  assumes *components-are-iso*: $C.obj\ a \Longrightarrow D.iso\ (\Gamma\ a)$

locale *identity-modification* =
  C: *bicategory* V_C H_C a_C i_C src_C trg_C +
  D: *bicategory* V_D H_D a_D i_D src_D trg_D +
  τ: *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and τ_0 :: $'c \Rightarrow 'd$
and τ_1 :: $'c \Rightarrow 'd$
begin

  abbreviation *map*
  where *map* $\equiv \tau_0$

  sublocale *invertible-modification*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1 τ_0 τ_1 *map*
    using *D.comp-arr-dom D.comp-cod-arr*
    by *unfold-locales auto*

end

locale *composite-modification* =
  C: *bicategory* V_C H_C a_C i_C src_C trg_C +
  D: *bicategory* V_D H_D a_D i_D src_D trg_D +
  ϱ: *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G ϱ_0 ϱ_1 +
  σ: *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G σ_0 σ_1 +
  τ: *pseudonatural-transformation*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G τ_0 τ_1 +
  Γ: *modification*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G ϱ_0 ϱ_1 σ_0 σ_1 Γ +
  Δ: *modification*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G σ_0 σ_1 τ_0 τ_1 Δ

for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
and G :: $'c \Rightarrow 'd$
and Φ_G :: $'c * 'c \Rightarrow 'd$
and ϱ_0 :: $'c \Rightarrow 'd$
and ϱ_1 :: $'c \Rightarrow 'd$
and σ_0 :: $'c \Rightarrow 'd$
and σ_1 :: $'c \Rightarrow 'd$
and τ_0 :: $'c \Rightarrow 'd$
and τ_1 :: $'c \Rightarrow 'd$
and Γ :: $'c \Rightarrow 'd$
and Δ :: $'c \Rightarrow 'd$
begin

abbreviation *map*
where $map\ a \equiv \Delta\ a \cdot_D \Gamma\ a$

sublocale *modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \varrho_0\ \varrho_1\ \tau_0\ \tau_1\ map$
using $\Delta.\Gamma$-*in-hom* $\Gamma.\Gamma$-*in-hom* Δ.*naturality* Γ.*naturality*
apply *unfold-locales*
apply *auto*[*1*]
proof −
fix $f\ a\ b$
assume f: «$f : a \rightarrow_C b$» **and** *ide-f*: $C.ide\ f$
have $\tau_1\ f \cdot_D (G\ f \star_D map\ a) = (\tau_1\ f \cdot_D (G\ f \star_D \Delta\ a)) \cdot_D (G\ f \star_D \Gamma\ a)$
proof −
have $D.seq\ (\Delta\ a)\ (\Gamma\ a)$
using f $\Delta.\Gamma$-*in-hom* $\Gamma.\Gamma$-*in-hom* **by** *blast*
thus *?thesis*
using f *ide-f* *D.whisker-left* [*of* $G\ f\ \Delta\ a\ \Gamma\ a$] *D.comp-assoc*
by *simp*
qed
also have ... $= (\Delta\ b \star_D F\ f) \cdot_D \sigma_1\ f \cdot_D (G\ f \star_D \Gamma\ a)$
using f *ide-f* Δ.*naturality* [*of* $f\ a\ b$] *D.comp-assoc* **by** *simp*
also have ... $= ((\Delta\ b \star_D F\ f) \cdot_D (\Gamma\ b \star_D F\ f)) \cdot_D \varrho_1\ f$
using f *ide-f* Γ.*naturality* [*of* $f\ a\ b$] *D.comp-assoc* **by** *simp*

also have ... $= (map\ b \star_D F\ f) \cdot_D \varrho_1\ f$
proof −
have $D.seq\ (\Delta\ b)\ (\Gamma\ b)$
using $f\ \Delta.\Gamma$-*in-hom* $\Gamma.\Gamma$-*in-hom* **by** *blast*
thus *?thesis*
using f *ide-f* $D.whisker\text{-}right$ [*of* $F\ f\ \Delta\ b\ \Gamma\ b$] **by** *simp*
qed
finally show $\tau_1\ f \cdot_D (G\ f \star_D map\ a) = (map\ b \star_D F\ f) \cdot_D \varrho_1\ f$ **by** *simp*
qed

end

context *converse-pseudonatural-equivalence*
begin

interpretation EV: *self-evaluation-map* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ **..**
notation $EV.eval$ (‹⦃-⦄›)

interpretation ι_F: *identity-pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$
..
interpretation ι_G: *identity-pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
..

interpretation $\tau'\tau$: *composite-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ F\ \Phi_F$
$\tau_0\ \tau_1\ map_0\ map_1$
..
interpretation $\tau\tau'$: *composite-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ F\ \Phi_F\ G\ \Phi_G$
$map_0\ map_1\ \tau_0\ \tau_1$
..

interpretation η: *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F$
$\iota_F.map_0\ \iota_F.map_1\ \tau'\tau.map_0\ \tau'\tau.map_1\ \eta$
proof
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$\eta\ a : F.map_0\ a \Rightarrow_D \tau'\tau.map_0\ a$»
using *unit-in-hom* $\tau'\tau.map_0$-*def* **by** *simp*
show $\bigwedge a.\ C.obj\ a \Longrightarrow D.iso\ (\eta\ a)$
by *simp*
show $\bigwedge f\ a\ b.\ [\![$ «$f : a \to_C b$»; $C.ide\ f]\!]$
$\Longrightarrow \tau'\tau.map_1\ f \cdot_D (F\ f \star_D \eta\ a) = (\eta\ b \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]$
proof −
fix $f\ a\ b$
assume f: «$f : a \to_C b$» **and** *ide-f*: $C.ide\ f$
have a: $C.obj\ a$ **and** b: $C.obj\ b$
using f **by** *auto*

have $\tau'\tau.map_1\ f\ \cdot_D\ (F\ f\ \star_D\ \eta\ a) =$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ \tau_1\ f)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a]\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ \mathrm{r}_D[G\ f])\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ G\ f\ \star_D\ \varepsilon\ a)\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f\ \star_D\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ D.inv\ (\tau_1\ f))\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$((\eta\ b\ \star_D\ F\ f)\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f]\ \star_D\ \tau_0{}'\ a)$
$\star_D\ \tau_0\ a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ \tau_0{}'\ a,\ \tau_0\ a])\ \cdot_D$
$(F\ f\ \star_D\ \eta\ a)$
unfolding $\tau'\tau.map_1$-*def* map_1-*def*
using *a b f D.comp-assoc* **by** *auto*
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ \tau_1\ f)\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a]\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ \mathrm{r}_D[G\ f])\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ G\ f\ \star_D\ \varepsilon\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a])\ \star_D\ \tau_0\ a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f\ \star_D\ \tau_0\ a,\ \tau_0{}'\ a]\ \star_D\ \tau_0\ a)\ \cdot_D$
$(((\tau_0{}'\ b\ \star_D\ D.inv\ (\tau_1\ f))\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$(((\eta\ b\ \star_D\ F\ f)\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$(((\mathrm{l}_D{}^{-1}[F\ f]\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ \tau_0{}'\ a,\ \tau_0\ a])\ \cdot_D$
$(F\ f\ \star_D\ \eta\ a)$
proof −
have $(\tau_0{}'\ b\ \star_D\ \mathrm{r}_D[G\ f])\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ G\ f\ \star_D\ \varepsilon\ a)\ \cdot_D$
$(\tau_0{}'\ b\ \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f\ \star_D\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ D.inv\ (\tau_1\ f))\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$((\eta\ b\ \star_D\ F\ f)\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f]\ \star_D\ \tau_0{}'\ a)$
$\star_D\ \tau_0\ a$
$= ((\tau_0{}'\ b\ \star_D\ \mathrm{r}_D[G\ f])\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ G\ f\ \star_D\ \varepsilon\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\tau_0{}'\ b\ \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a])\ \star_D\ \tau_0\ a)\ \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f\ \star_D\ \tau_0\ a,\ \tau_0{}'\ a]\ \star_D\ \tau_0\ a)\ \cdot_D$
$(((\tau_0{}'\ b\ \star_D\ D.inv\ (\tau_1\ f))\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$(((\eta\ b\ \star_D\ F\ f)\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)\ \cdot_D$
$((\mathrm{l}_D{}^{-1}[F\ f]\ \star_D\ \tau_0{}'\ a)\ \star_D\ \tau_0\ a)$
proof −

have $D.arr$ $((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ b \star_D\ G\ f \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ b \star_D\ F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a))$
using $a\ b\ f$ *ide-f* τ*.iso-map*$_1$*-ide*
by (*metis* (*no-types*, *lifting*) *C.in-hhomE map*$_1$*-def map*$_1$*-simps*(*1*))
thus *?thesis*
using $a\ b\ f$ *D.whisker-right* [*of* $\tau_0\ a$] **by** *fastforce*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$(((\eta\ b \star_D\ F\ f) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ b \star_D\ F\ f,\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$(F\ f \star_D \eta\ a)$
proof −
have $((\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ \tau_0{}'\ a,\ \tau_0\ a] =$
$\mathrm{a}_D{}^{-1}[F_0\ b \star_D\ F\ f,\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D (\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a)$
using $a\ b\ f$ *ide-f D.assoc′-naturality* [*of* $\mathrm{l}_D{}^{-1}[F\ f]\ \tau_0{}'\ a\ \tau_0\ a$] **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((((\eta\ b \star_D\ F\ f) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[F_0\ b \star_D\ F\ f,\ \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$((F_0\ b \star_D\ F\ f) \star_D \eta\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D\ F_0\ a)$

proof −
  have $(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D (F\ f \star_D \eta\ a) = \mathrm{l}_D{}^{-1}[F\ f] \star_D \eta\ a$
    using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
      D.interchange [*of* $\mathrm{l}_D{}^{-1}[F\ f]\ F\ f\ \tau_0{}'\ a \star_D \tau_0\ a\ \eta\ a$]
    by *simp*
  also have ... $= ((F_0\ b \star_D F\ f) \star_D \eta\ a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
    using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
      D.interchange [*of* $F_0\ b \star_D F\ f\ \mathrm{l}_D{}^{-1}[F\ f]\ \eta\ a\ F_0\ a$]
    by *auto*
  finally have $(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D (F\ f \star_D \eta\ a) =$
      $((F_0\ b \star_D F\ f) \star_D \eta\ a) \cdot_D (\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
    by *blast*
  thus *?thesis*
    using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b, \tau_0\ b, F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b, G\ f, \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f, \tau_0\ a, \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b, G\ f \star_D \tau_0\ a, \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f, \tau_0{}'\ a, \tau_0\ a] \cdot_D$
$(((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$((F_0\ b \star_D F\ f) \star_D \eta\ a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
  have $(((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D \mathrm{a}_D{}^{-1}[F_0\ b \star_D F\ f, \tau_0{}'\ a, \tau_0\ a] =$
      $\mathrm{a}_D{}^{-1}[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f, \tau_0{}'\ a, \tau_0\ a] \cdot_D ((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a \star_D \tau_0\ a)$
    using *a b f ide-f D.assoc′-naturality* [*of* $\eta\ b \star_D F\ f\ \tau_0{}'\ a\ \tau_0\ a$] **by** *auto*
  thus *?thesis*
    using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b, \tau_0\ b, F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b, G\ f, \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f, \tau_0\ a, \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b, G\ f \star_D \tau_0\ a, \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$((((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f, \tau_0{}'\ a, \tau_0\ a] \cdot_D$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −

have $((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D ((F_0\ b \star_D F\ f) \star_D \eta\ a)$
$= (\eta\ b \star_D F\ f) \star_D \eta\ a$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\eta\ b \star_D F\ f\ F_0\ b \star_D F\ f\ \tau_0{}'\ a \star_D \tau_0\ a\ \eta\ a$]
by *auto*
also have ... $= (((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) \cdot_D ((\eta\ b \star_D F\ f) \star_D F_0\ a)$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f\ \eta\ b \star_D F\ f\ \eta\ a\ F_0\ a$]
by *auto*
finally have $((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D ((F_0\ b \star_D F\ f) \star_D \eta\ a) =$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) \cdot_D ((\eta\ b \star_D F\ f) \star_D F_0\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b, \tau_0\ b, F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b, G\ f, \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f, \tau_0\ a, \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b, G\ f \star_D \tau_0\ a, \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$((((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f, \tau_0{}'\ a, \tau_0\ a]) \cdot_D$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $(((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$
$((\mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) =$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a$
using *a b f ide-f D.whisker-right* τ*.iso-map*$_1$*-ide* **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b, \tau_0\ b, F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b, G\ f, \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f, \tau_0\ a, \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b, G\ f \star_D \tau_0\ a, \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a), \tau_0{}'\ a, \tau_0\ a] \cdot_D$
$(((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a)) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $(((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\ f] \star_D \tau_0{}'\ a) \star_D \tau_0\ a) \cdot_D$

$\mathrm{a}_D{}^{-1}[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f,\ \tau_0{}'\ a,\ \tau_0\ a] =$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a),\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a)$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
D.assoc′-naturality
[*of* $(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]\ \tau_0{}'\ a\ \tau_0\ a$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a),\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a) \star_D \eta\ a) \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) =$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]) \cdot_D ((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D$
$(\tau_0{}'\ a \star_D \tau_0\ a) \cdot_D \eta\ a$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
D.interchange [*of* $(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]$
$(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f\ \tau_0{}'\ a \star_D \tau_0\ a\ \eta\ a$]
by *auto*
also have ... $= (\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \eta\ a$
using *a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr D.comp-assoc*
by *auto*
also have ... $= (\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a) \cdot_D$
$(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]$
$\star_D \eta\ a \cdot_D F_0\ a$
using *a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr* **by** *auto*
also have ... $= ((\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a) \star_D \eta\ a) \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a)$
proof −
have *D.seq* $(\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a)\ ((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f])$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
by (*intro D.seqI D.hseqI′*) *auto*
thus *?thesis*
using *a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $(\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a)$
$(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]$

$\eta\ a\ F_0\ a]$
by *simp*
qed
finally have $((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f) \star_D \eta\ a) =$
$((\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a) \star_D \eta\ a) \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D G\ f) \star_D \varepsilon\ a \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a),\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a,\ \tau_0{}'\ a \star_D \tau_0\ a]) \cdot_D$
$((\tau_0{}'\ b \star_D G\ f) \star_D \tau_0\ a \star_D \eta\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D F_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $(\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a$
$= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D G\ f) \star_D \varepsilon\ a \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a] \star_D \tau_0\ a)$
proof −
have $(\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \star_D \tau_0\ a$
$= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D G\ f) \star_D \varepsilon\ a) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a] \star_D \tau_0\ a)$
using *a b f ide-f D.hcomp-reassoc D.whisker-right* [*of* $\tau_0\ a$] **by** *auto*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D G\ f) \star_D \varepsilon\ a \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a] \star_D \tau_0\ a)$

using *a b f ide-f D.hcomp-reassoc(1)* [*of* $\tau_0{}'\ b \star_D\ G\ f\ \varepsilon\ a\ \tau_0\ a$]
by *auto*
finally show *?thesis* **by** *blast*
qed
moreover have $(\tau_0{}'\ b \star_D\ G\ f \star_D\ \tau_0\ a) \star_D\ \eta\ a$
$= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ \tau_0{}'\ a \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ \tau_0{}'\ a \star_D\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a \star_D\ \eta\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a)$
proof −
have $(\tau_0{}'\ b \star_D\ G\ f \star_D\ \tau_0\ a) \star_D\ \eta\ a =$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D ((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D$
$(\tau_0{}'\ a \star_D\ \tau_0\ a) \cdot_D\ \eta\ a \cdot_D\ F_0\ a$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.hcomp-reassoc(2) [*of* $\tau_0{}'\ b\ G\ f\ \tau_0\ a$]
by *auto*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ \tau_0{}'\ a \star_D\ \tau_0\ a) \cdot_D$
$(((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a) \star_D\ \eta\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a)$
using *a b f ide-f D.inv-inv D.iso-inv-iso D.interchange* **by** *auto*
also have ... $= (\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ \tau_0{}'\ a \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ \tau_0{}'\ a \star_D\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a \star_D\ \eta\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a)$
using *a b f ide-f D.hcomp-reassoc(1)* [*of* $\tau_0{}'\ b \star_D\ G\ f\ \tau_0\ a\ \eta\ a$] **by** *auto*
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D\ \tau_1\ f) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ \mathrm{r}_D[G\ f]) \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D\ G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$(((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \varepsilon\ a \star_D\ \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a \star_D\ \eta\ a)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \cdot_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D\ F_0\ a) \cdot_D$
$((\eta\ b \star_D\ F\ f) \star_D\ F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D\ F_0\ a)$
proof −
have $\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a \star_D\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D\ \tau_0{}'\ a] \star_D\ \tau_0\ a) \cdot_D$

$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a),\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a,\ \tau_0{}'\ a \star_D \tau_0\ a] =$
$(\tau_0{}'\ b \star_D G\ f) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a]$
proof −
have $\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a \star_D \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D (G\ f \star_D \tau_0\ a),\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D \tau_0{}'\ a \star_D \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a,\ \tau_0{}'\ a \star_D \tau_0\ a]$
$= \{\!|\mathbf{a}[\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle,\ \langle\tau_0\ a\rangle \star \langle\tau_0{}'\ a\rangle,\ \langle\tau_0\ a\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle,\ \langle\tau_0\ a\rangle \star \langle\tau_0{}'\ a\rangle] \star \langle\tau_0\ a\rangle) \cdot$
$((\langle\tau_0{}'\ b\rangle \star \mathbf{a}[\ \langle G\ f\rangle,\ \langle\tau_0\ a\rangle,\ \langle\tau_0{}'\ a\rangle]) \star \langle\tau_0\ a\rangle) \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle \star \langle\tau_0\ a\rangle,\ \langle\tau_0{}'\ a\rangle] \star \langle\tau_0\ a\rangle) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle \star (\langle G\ f\rangle \star \langle\tau_0\ a\rangle),\ \langle\tau_0{}'\ a\rangle,\ \langle\tau_0\ a\rangle] \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle,\ \langle\tau_0\ a\rangle] \star \langle\tau_0{}'\ a\rangle \star \langle\tau_0\ a\rangle) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle,\ \langle\tau_0\ a\rangle,\ \langle\tau_0{}'\ a\rangle \star \langle\tau_0\ a\rangle]|\!\}$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char*$_{SbC}$ *D.VVV.arr-char*$_{SbC}$ *D.VV.ide-char*$_{SbC}$ *D.VV.arr-char*$_{SbC}$
by *auto*
also have ... $= \{\!|(\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle) \star \mathbf{a}^{-1}[\langle\tau_0\ a\rangle,\ \langle\tau_0{}'\ a\rangle,\ \langle\tau_0\ a\rangle]|\!\}$
using *a b f ide-f* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= (\tau_0{}'\ b \star_D G\ f) \star_D \mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a]$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char*$_{SbC}$ *D.VVV.arr-char*$_{SbC}$ *D.VV.ide-char*$_{SbC}$ *D.VV.arr-char*$_{SbC}$
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D G\ f) \star_D \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{r}_D[\tau_0\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D F_0\ a)) \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
interpret *adjoint-equivalence-in-bicategory*

$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ \langle\tau_0\ a\rangle\ \langle\tau_0{}'\ a\rangle\ \langle\eta\ a\rangle\ \langle\varepsilon\ a\rangle$
using *a chosen-adjoint-equivalence* **by** *simp*
have $((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \varepsilon\ a \star_D\ \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a \star_D\ \eta\ a)$
$= (\tau_0{}'\ b \star_D\ G\ f) \star_D\ (\varepsilon\ a \star_D\ \tau_0\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a] \cdot_D$
$(\tau_0\ a \star_D\ \eta\ a)$
using *a b f ide-f D.whisker-left* **by** *auto*
also have ... $= (\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D\ \mathrm{r}_D[\tau_0\ a]$
using *triangle-left* **by** *simp*
finally have $((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \varepsilon\ a \star_D\ \tau_0\ a) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{a}_D{}^{-1}[\tau_0\ a,\ \tau_0{}'\ a,\ \tau_0\ a]) \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \tau_0\ a \star_D\ \eta\ a)$
$= (\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D\ \mathrm{r}_D[\tau_0\ a]$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$(\tau_0{}'\ b \star_D\ \tau_1\ f) \cdot_D$
$(\mathrm{r}_D[\tau_0{}'\ b \star_D\ G\ f \star_D\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \cdot_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D\ F_0\ a)) \cdot_D$
$((\eta\ b \star_D\ F\ f) \star_D\ F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D\ F_0\ a)$
proof −
have $\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ \mathrm{r}_D[G\ f]) \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D\ G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D\ \mathrm{r}_D[\tau_0\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a)$
$= \mathrm{r}_D[\tau_0{}'\ b \star_D\ G\ f \star_D\ \tau_0\ a]$
proof −
have $\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ \mathrm{r}_D[G\ f]) \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ G\ f,\ G_0\ a] \star_D\ \tau_0\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D\ G\ f,\ G_0\ a,\ \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ G\ f) \star_D\ \mathrm{l}_D{}^{-1}[\tau_0\ a] \cdot_D\ \mathrm{r}_D[\tau_0\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b \star_D\ G\ f,\ \tau_0\ a,\ F_0\ a]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ G\ f,\ \tau_0\ a] \star_D\ F_0\ a)$
$= \{\!\!\{\mathbf{a}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle,\ \langle\tau_0\ a\rangle] \cdot$
$((\langle\tau_0{}'\ b\rangle \star \mathbf{r}[\langle G\ f\rangle]) \star \langle\tau_0\ a\rangle) \cdot$
$(\mathbf{a}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle,\ \langle G_0\ a\rangle_0] \star \langle\tau_0\ a\rangle) \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle,\ \langle G_0\ a\rangle_0,\ \langle\tau_0\ a\rangle] \cdot$
$((\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle) \star \mathbf{l}^{-1}[\langle\tau_0\ a\rangle] \cdot \mathbf{r}[\langle\tau_0\ a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0{}'\ b\rangle \star \langle G\ f\rangle,\ \langle\tau_0\ a\rangle,\ \langle F_0\ a\rangle_0]) \cdot$
$(\mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle,\ \langle G\ f\rangle,\ \langle\tau_0\ a\rangle] \star \langle F_0\ a\rangle_0)\}\!\!\}$

using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$ D.𝔩-ide-simp D.𝔯-ide-simp*
by *auto*
also have ... $= \{\!|\mathbf{r}[\langle \tau_0{}'\ b\rangle \star \langle G\ f\rangle \star \langle \tau_0\ a\rangle]|\!\}$
using *a b f ide-f* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= \mathrm{r}_D[\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a]$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$ D.𝔩-ide-simp D.𝔯-ide-simp*
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$((\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D$
$(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)))) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D \mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f] \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $\mathrm{r}_D[\tau_0{}'\ b \star_D G\ f \star_D \tau_0\ a] \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D F_0\ a)$
$= (\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$\mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f]$
using *a b f ide-f D.comp-assoc τ.iso-map$_1$-ide D.runit-naturality* [*of* $(\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]) \cdot_D$
$\mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f]) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D ((\tau_0{}'\ b \star_D \tau_1\ f) \cdot_D (\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)))$
$= \mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]$
using *a b f ide-f D.comp-arr-inv′ D.comp-arr-dom τ.iso-map$_1$-ide D.whisker-left* [*of* $\tau_0{}'\ b\ \tau_1\ f\ D.inv\ (\tau_1\ f)$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f] \cdot_D$
$((\eta\ b \star_D F\ f) \star_D F_0\ a)) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$

proof −
have $(\mathrm{a}_D{}^{-1}[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \cdot_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f]) \cdot_D \mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f]$
$= \mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f]$
using *a b f ide-f D.comp-inv-arr′ D.comp-cod-arr* **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\eta\ b \star_D F\ f) \cdot_D \mathrm{r}_D[F_0\ b \star_D F\ f] \cdot_D (\mathrm{l}_D{}^{-1}[F\ f] \star_D F_0\ a)$
proof −
have $\mathrm{r}_D[(\tau_0{}'\ b \star_D \tau_0\ b) \star_D F\ f] \cdot_D ((\eta\ b \star_D F\ f) \star_D F_0\ a)$
$= (\eta\ b \star_D F\ f) \cdot_D \mathrm{r}_D[F_0\ b \star_D F\ f]$
using *a b f ide-f D.runit-naturality* [*of* $\eta\ b \star_D F\ f$] **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\eta\ b \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]$
using *a b f ide-f D.runit-naturality* [*of* $\mathrm{l}_D{}^{-1}[F\ f]$] **by** *auto*
finally show $\tau'\tau.map_1\ f \cdot_D (F\ f \star_D \eta\ a) = (\eta\ b \star_D F\ f) \cdot_D \mathrm{l}_D{}^{-1}[F\ f] \cdot_D \mathrm{r}_D[F\ f]$
by *blast*
qed
qed

lemma *unit-is-invertible-modification*:
shows *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F$
$\iota_F.map_0\ \iota_F.map_1\ \tau'\tau.map_0\ \tau'\tau.map_1\ \eta$
..

interpretation ε: *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G\ \Phi_G$
$\tau\tau'.map_0\ \tau\tau'.map_1\ \iota_G.map_0\ \iota_G.map_1\ \varepsilon$
proof
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$\varepsilon\ a : \tau\tau'.map_0\ a \Rightarrow_D G_0\ a$»
using *counit-in-hom* $\tau\tau'.map_0$*-def* **by** *simp*
show $\bigwedge a.\ C.obj\ a \Longrightarrow D.iso\ (\varepsilon\ a)$
by *simp*
show $\bigwedge f\ a\ b.\ [\![$«$f : a \to_C b$»; $C.ide\ f]\!]$
$\Longrightarrow (\mathrm{l}_D{}^{-1}[G\ f] \cdot_D \mathrm{r}_D[G\ f]) \cdot_D (G\ f \star_D \varepsilon\ a) = (\varepsilon\ b \star_D G\ f) \cdot_D \tau\tau'.map_1\ f$
proof −
fix $f\ a\ b$
assume f: «$f : a \to_C b$» **and** *ide-f*: $C.ide\ f$
have a: $C.obj\ a$ **and** b: $C.obj\ b$
using f **by** *auto*
have $(\varepsilon\ b \star_D G\ f) \cdot_D \tau\tau'.map_1\ f$
$= (\varepsilon\ b \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f] \cdot_D$
$(\tau_0\ b \star_D$
$(\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$

$(\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
unfolding $\tau\tau'.map_1$-*def* map_1-*def*
using *a b f D.comp-assoc* **by** *auto*
also have ... $= (\varepsilon\ b \star_D G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f] \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$((\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D (\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a)) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $\tau_0\ b \star_D$
$(\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$((\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a)$
$= (\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D (\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a)$
proof −
have $D.arr\ ((\tau_0{}'\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(\tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ b \star_D D.inv\ (\tau_1\ f)) \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$

$((\eta\ b \star_D\ F\ f) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ a))$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
by (*intro D.seqI*) *auto*
thus *?thesis*
using *a b f D.whisker-left* [*of* τ_0 *b*] **by** *fastforce*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\varepsilon\ b \star_D\ G\ f) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f] \cdot_D$
$(\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ \mathrm{r}_D[G\ f]) \cdot_D$
$(\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ G\ f \star_D\ \varepsilon\ a) \cdot_D$
$((\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ a)) \cdot_D$
$(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D\ \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D\ (\eta\ b \star_D\ F\ f) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D\ \mathrm{l}_D{}^{-1}[F\ f] \star_D\ \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D\ \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D\ (\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ a)$
$= \tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D$
$((\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ a)$
using *a b f ide-f D.whisker-left* [*of* τ_0 *b*] τ*.iso-map*$_1$*-ide* **by** *auto*
also have ... $= \tau_0\ b \star_D\ (\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a]$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
D.assoc-naturality [*of* $\tau_0{}'\ b\ D.inv\ (\tau_1\ f)\ \tau_0{}'\ a$]
by *auto*
also have ... $= (\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a])$
using *a b f ide-f D.whisker-left* [*of* τ_0 *b*] τ*.iso-map*$_1$*-ide* **by** *auto*
finally have $(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ G\ f \star_D\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D\ (\tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f)) \star_D\ \tau_0{}'\ a)$
$= (\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ D.inv\ (\tau_1\ f) \star_D\ \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D\ \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a])$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\varepsilon\ b \star_D\ G\ f) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f] \cdot_D$
$(\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ \mathrm{r}_D[G\ f])) \cdot_D$
$(\tau_0\ b \star_D\ \tau_0{}'\ b \star_D\ G\ f \star_D\ \varepsilon\ a) \cdot_D$

$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$

proof −
have $(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$
$= \tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$
using *a b f ide-f D.whisker-left* τ*.iso-map*$_1$*-ide* **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\varepsilon\ b \star_D G\ f) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D G_0\ a] \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D G\ f \star_D \varepsilon\ a)) \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$

proof −
have $\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f] \cdot_D (\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{r}_D[G\ f])$
$= ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\ f]) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D G_0\ a]$
using *a b f ide-f D.assoc′-naturality* [*of* $\tau_0\ b\ \tau_0{}'\ b\ \mathrm{r}_D[G\ f]$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\varepsilon\ b \star_D G\ f) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$

proof −

have $\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D\ G_0\ a] \cdot_D (\tau_0\ b \star_D \tau_0{}'\ b \star_D\ G\ f \star_D \varepsilon\ a)$
$= ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D\ G\ f \star_D \varepsilon\ a) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a]$
using *a b f ide-f D.assoc′-naturality* [*of* $\tau_0\ b\ \tau_0{}'\ b\ G\ f \star_D \varepsilon\ a$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\varepsilon\ b \star_D\ G\ f) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\ f])) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D\ G\ f \star_D \varepsilon\ a) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D\ F\ f) \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D\ F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \tau_0{}'\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))$
$= ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D\ F\ f) \star_D \tau_0{}'\ a]$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
D.assoc′-naturality
[*of* $\tau_0\ b\ \tau_0{}'\ b\ \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$((\varepsilon\ b \star_D\ G\ f \star_D\ G_0\ a) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D\ G\ f \star_D \varepsilon\ a)) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D\ F\ f) \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D\ F\ f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $(\varepsilon\ b \star_D\ G\ f) \cdot_D ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\ f]) = \varepsilon\ b \star_D \mathrm{r}_D[G\ f]$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $\varepsilon\ b\ \tau_0\ b \star_D \tau_0{}'\ b\ G\ f\ \mathrm{r}_D[G\ f]$]
by *simp*
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D (\varepsilon\ b \star_D\ G\ f \star_D\ G_0\ a)$

using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* G_0 b ε b $\mathrm{r}_D[G\,f]$ $G\,f \star_D G_0\ a$]
by *auto*
finally have $(\varepsilon\ b \star_D G\,f) \cdot_D ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{r}_D[G\,f]) =$
$(G_0\ b \star_D \mathrm{r}_D[G\,f]) \cdot_D (\varepsilon\ b \star_D G\,f \star_D G_0\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\,f]) \cdot_D$
$(G_0\ b \star_D G\,f \star_D \varepsilon\ a) \cdot_D$
$((\varepsilon\ b \star_D G\,f \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\,f, \tau_0\ a, \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, (\tau_0\ b \star_D F\,f) \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b \star_D F\,f, \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\,f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\,f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\,f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b, F\,f, \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\,f, \tau_0\ a, \tau_0{}'\ a]$
proof −
have $(\varepsilon\ b \star_D G\,f \star_D G_0\ a) \cdot_D ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D G\,f \star_D \varepsilon\ a)$
$= \varepsilon\ b \star_D G\,f \star_D \varepsilon\ a$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* ε b τ_0 $b \star_D \tau_0{}'\ b$ $G\,f \star_D G_0\ a$ $G\,f \star_D \varepsilon\ a$]
by *auto*
also have ... $= (G_0\ b \star_D G\,f \star_D \varepsilon\ a) \cdot_D (\varepsilon\ b \star_D G\,f \star_D \tau_0\ a \star_D \tau_0{}'\ a)$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* G_0 b ε b $G\,f \star_D \varepsilon\ a$ $G\,f \star_D \tau_0\ a \star_D \tau_0{}'\ a$]
by *auto*
finally have $(\varepsilon\ b \star_D G\,f \star_D G_0\ a) \cdot_D ((\tau_0\ b \star_D \tau_0{}'\ b) \star_D G\,f \star_D \varepsilon\ a) =$
$(G_0\ b \star_D G\,f \star_D \varepsilon\ a) \cdot_D (\varepsilon\ b \star_D G\,f \star_D \tau_0\ a \star_D \tau_0{}'\ a)$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\,f]) \cdot_D$
$(G_0\ b \star_D G\,f \star_D \varepsilon\ a) \cdot_D$
$(G_0\ b \star_D \mathrm{a}_D[G\,f, \tau_0\ a, \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$(\varepsilon\ b \star_D (\tau_0\ b \star_D F\,f) \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, (\tau_0\ b \star_D F\,f) \star_D \tau_0{}'\ a] \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b \star_D F\,f, \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b, \tau_0\ b, F\,f] \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D (\eta\ b \star_D F\,f) \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\,f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b, F\,f, \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\,f, \tau_0\ a, \tau_0{}'\ a]$

proof −
  have *1*: *D.seq* $(G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a)$
    $(\mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))$
    using *a b f ide-f* τ*.iso-map*$_1$*-ide*
    by (*intro D.seqI D.hseqI′*) *auto*
  have *2*: *D.seq* $(\mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))$
    $((\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a)$
    using *a b f ide-f* τ*.iso-map*$_1$*-ide*
    by (*intro D.seqI D.hseqI′*) *auto*
  have $(\varepsilon\ b \star_D G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D$
    $((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) =$
    $\varepsilon\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$
    using *1 a b f ide-f D.comp-arr-dom D.comp-cod-arr* τ*.iso-map*$_1$*-ide*
      D.interchange [*of* $\varepsilon\ b\ \tau_0\ b \star_D \tau_0{}'\ b\ G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a$
        $\mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$]
    by *auto*
  also have ... $= G_0\ b \cdot_D \varepsilon\ b \star_D$
      $(\mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
      $((\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a)$
    using *a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-arr-dom D.comp-cod-arr* **by** *auto*
  also have ... $= (G_0\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
      $(\varepsilon\ b \star_D (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a)$
    using *2 a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-assoc*
      D.interchange [*of* $G_0\ b\ \varepsilon\ b$
        $\mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)$
        $(\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a$]
    by *simp*
  finally have $(\varepsilon\ b \star_D G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a) \cdot_D$
      $((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))$
      $= (G_0\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
        $(\varepsilon\ b \star_D (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a)$
    by *blast*
  thus *?thesis*
    using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
  $(G_0\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
  $(G_0\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
  $(G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
  $(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
  $((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
  $(\mathrm{a}_D{}^{-1}[\tau_0\ b \star_D \tau_0{}'\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
  $((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
  $\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a] \cdot_D$
  $(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a]) \cdot_D$
  $(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a) \cdot_D$
  $(\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
  $\mathrm{a}_D[\tau_0\ b,\ \tau_0{}'\ b \star_D \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
  $((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$

$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$

proof −
have $\varepsilon\ b \star_D (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a$
$= (G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
$((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b \star_D \tau_0{}'\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])$
proof −
have $\varepsilon\ b \star_D (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a$
$= (G_0\ b \cdot_D \varepsilon\ b \cdot_D (\tau_0\ b \star_D \tau_0{}'\ b)) \star_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a] \cdot_D (\tau_0\ b \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr*
D.hcomp-reassoc(*1*) [*of* $\tau_0\ b\ F\ f\ \tau_0{}'\ a$]
by *auto*
also have ... $= (G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\varepsilon\ b \star_D \tau_0\ b \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])$
using *a b f ide-f D.comp-arr-dom D.comp-cod-arr D.assoc′-naturality1*
D.interchange [*of* $G_0\ b\ \varepsilon\ b \cdot_D (\tau_0\ b \star_D \tau_0{}'\ b)\ \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$
$(\tau_0\ b \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$]
D.interchange [*of* $\varepsilon\ b\ \tau_0\ b \star_D \tau_0{}'\ b\ \tau_0\ b \star_D F\ f \star_D \tau_0{}'\ a$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$]
by *fastforce*
also have ... $= (G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
$((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b \star_D \tau_0{}'\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])$
using *a b f ide-f D.hcomp-reassoc*(*2*) [*of* $\varepsilon\ b\ \tau_0\ b\ F\ f \star_D \tau_0{}'\ a$]
by *auto*
finally show *?thesis* **by** *blast*
qed
moreover have $\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a$
$= (\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[\tau_0\ b,\ \tau_0{}'\ b \star_D \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
$((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a])$
proof −
have $\tau_0\ b \star_D (\eta\ b \star_D F\ f) \star_D \tau_0{}'\ a =$
$(\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \eta\ b \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a])$

using *a b f ide-f D.hcomp-reassoc(1)* [*of* η *b F f* $\tau_0{}'$ *a*]
D.whisker-left [*of* τ_0 *b*]
by *auto*
also have ... = $(\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$(\mathrm{a}_D[\tau_0\ b,\ \tau_0{}'\ b \star_D \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]\ \cdot_D$
$((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a])\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a])$
using *a b f ide-f D.hcomp-reassoc(2)* [*of* τ_0 *b* η *b F f* $\star_D$ $\tau_0{}'$ *a*]
by *auto*
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(G_0\ b \star_D \mathrm{r}_D[G\ f])\ \cdot_D$
$(G_0\ b \star_D G\ f \star_D \varepsilon\ a)\ \cdot_D$
$(G_0\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D\ (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a))\ \cdot_D$
$(G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]\ \cdot_D$
$(((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a)\ \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ \tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a)\ \cdot_D$
$((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a])\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a)\ \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]\ \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $\mathrm{a}_D{}^{-1}[\tau_0\ b \star_D \tau_0{}'\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]\ \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a]\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a)\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ \tau_0{}'\ b \star_D \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]$
$= \mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ \tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a$
proof −
have $\mathrm{a}_D{}^{-1}[\tau_0\ b \star_D \tau_0{}'\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]\ \cdot_D$
$((\tau_0\ b \star_D \tau_0{}'\ b) \star_D \mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ \tau_0{}'\ b,\ (\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a]\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b \star_D F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[\tau_0{}'\ b,\ \tau_0\ b,\ F\ f] \star_D \tau_0{}'\ a)\ \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0{}'\ b \star_D \tau_0\ b,\ F\ f,\ \tau_0{}'\ a])\ \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ \tau_0{}'\ b \star_D \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a]$
$= \{\!\!\{\mathbf{a}^{-1}[\langle\tau_0\ b\rangle \star \langle\tau_0{}'\ b\rangle,\ \langle\tau_0\ b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle]\ \cdot$
$((\langle\tau_0\ b\rangle \star \langle\tau_0{}'\ b\rangle) \star \mathbf{a}[\langle\tau_0\ b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ a\rangle])\ \cdot$
$\mathbf{a}^{-1}[\langle\tau_0\ b\rangle,\ \langle\tau_0{}'\ b\rangle,\ (\langle\tau_0\ b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ a\rangle]\ \cdot$

$(\langle\tau_0\ b\rangle \star \mathbf{a}[\langle\tau_0{}'\ b\rangle, \langle\tau_0\ b\rangle \star \langle F\ f\rangle, \langle\tau_0{}'\ a\rangle]) \cdot$
$(\langle\tau_0\ b\rangle \star \mathbf{a}[\langle\tau_0{}'\ b\rangle, \langle\tau_0\ b\rangle, \langle F\ f\rangle] \star \langle\tau_0{}'\ a\rangle) \cdot$
$(\langle\tau_0\ b\rangle \star \mathbf{a}^{-1}[\langle\tau_0{}'\ b\rangle \star \langle\tau_0\ b\rangle, \langle F\ f\rangle, \langle\tau_0{}'\ a\rangle]) \cdot$
$\mathbf{a}[\langle\tau_0\ b\rangle, \langle\tau_0{}'\ b\rangle \star \langle\tau_0\ b\rangle, \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle]\rangle\}$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *auto*
also have ... $= \{\!|\mathbf{a}^{-1}[\langle\tau_0\ b\rangle, \langle\tau_0{}'\ b\rangle, \langle\tau_0\ b\rangle] \star \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle|\!\}$
using *a b f ide-f* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= \mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, \tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$*
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *auto*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(G_0\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$(G_0\ b \star_D \mathrm{a}_D[G\ f, \tau_0\ a, \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$((G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b, F\ f, \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[G_0\ b, \tau_0\ b, F\ f \star_D \tau_0{}'\ a] \cdot_D$
$(\mathrm{l}_D{}^{-1}[\tau_0\ b] \cdot_D \mathrm{r}_D[\tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b, F_0\ b, F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b, F\ f, \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b, F\ f, \tau_0{}'\ a]) \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f, \tau_0\ a, \tau_0{}'\ a]$
proof −
have $((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, \tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a)$
$= \mathrm{l}_D{}^{-1}[\tau_0\ b] \cdot_D \mathrm{r}_D[\tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a$
proof −
interpret *adjoint-equivalence-in-bicategory*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ‹$\tau_0\ b$› ‹$\tau_0{}'\ b$› ‹$\eta\ b$› ‹$\varepsilon\ b$›
using *b chosen-adjoint-equivalence* **by** *simp*
have $((\varepsilon\ b \star_D \tau_0\ b) \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$(\mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, \tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$((\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a)$
$= (\varepsilon\ b \star_D \tau_0\ b) \cdot_D \mathrm{a}_D{}^{-1}[\tau_0\ b, \tau_0{}'\ b, \tau_0\ b] \cdot_D (\tau_0\ b \star_D \eta\ b) \star_D F\ f \star_D \tau_0{}'\ a$
using *a b f ide-f D.whisker-right* **by** *auto*
also have ... $= \mathrm{l}_D{}^{-1}[\tau_0\ b] \cdot_D \mathrm{r}_D[\tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a$
using *triangle-left* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*

using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(G_0\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$((G_0\ b \star_D \mathrm{a}_D[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a] \cdot_D (D.inv\ (\tau_1\ f) \star_D \tau_0{}'\ a)) \cdot_D$
$\mathrm{l}_D{}^{-1}[(\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_1\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[G\ f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $(G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
$(\mathrm{l}_D{}^{-1}[\tau_0\ b] \cdot_D \mathrm{r}_D[\tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$
$= \mathrm{l}_D{}^{-1}[(\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a]$
proof −
have $(G_0\ b \star_D \mathrm{a}_D{}^{-1}[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\mathrm{a}_D[G_0\ b,\ \tau_0\ b,\ F\ f \star_D \tau_0{}'\ a] \cdot_D$
$(\mathrm{l}_D{}^{-1}[\tau_0\ b] \cdot_D \mathrm{r}_D[\tau_0\ b] \star_D F\ f \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D{}^{-1}[\tau_0\ b,\ F_0\ b,\ F\ f \star_D \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{a}_D[F_0\ b,\ F\ f,\ \tau_0{}'\ a]) \cdot_D$
$(\tau_0\ b \star_D \mathrm{l}_D{}^{-1}[F\ f] \star_D \tau_0{}'\ a) \cdot_D$
$\mathrm{a}_D[\tau_0\ b,\ F\ f,\ \tau_0{}'\ a]$
$= \{\!|(\langle G_0\ b\rangle_0 \star \mathbf{a}^{-1}[\langle\tau_0\ b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ a\rangle]) \cdot$
$(\mathbf{a}[\langle G_0\ b\rangle_0,\ \langle\tau_0\ b\rangle,\ \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle] \cdot$
$(\mathbf{l}^{-1}[\langle\tau_0\ b\rangle] \cdot \mathbf{r}[\langle\tau_0\ b\rangle] \star \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle) \cdot$
$\mathbf{a}^{-1}[\langle\tau_0\ b\rangle,\ \langle F_0\ b\rangle_0,\ \langle F\ f\rangle \star \langle\tau_0{}'\ a\rangle]) \cdot$
$(\langle\tau_0\ b\rangle \star \mathbf{a}[\langle F_0\ b\rangle_0,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ a\rangle]) \cdot$
$(\langle\tau_0\ b\rangle \star \mathbf{l}^{-1}[\langle F\ f\rangle] \star \langle\tau_0{}'\ a\rangle) \cdot$
$\mathbf{a}[\langle\tau_0\ b\rangle,\ \langle F\ f\rangle,\ \langle\tau_0{}'\ a\rangle]|\!\}$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$ D.𝔩-ide-simp D.𝔯-ide-simp*
by *auto*
also have ... $= \{\!|\mathbf{l}^{-1}[(\langle\tau_0\ b\rangle \star \langle F\ f\rangle) \star \langle\tau_0{}'\ a\rangle]|\!\}$
using *a b f ide-f* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= \mathrm{l}_D{}^{-1}[(\tau_0\ b \star_D F\ f) \star_D \tau_0{}'\ a]$
using *a b f ide-f D.α-def D.α′.map-ide-simp D.VVV.ide-char$_{SbC}$ D.VVV.arr-char$_{SbC}$ D.VV.ide-char$_{SbC}$ D.VV.arr-char$_{SbC}$ D.𝔩-ide-simp*
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b \star_D \mathrm{r}_D[G\ f]) \cdot_D$
$(G_0\ b \star_D G\ f \star_D \varepsilon\ a) \cdot_D$
$\mathrm{l}_D{}^{-1}[G\ f \star_D \tau_0\ a \star_D \tau_0{}'\ a] \cdot_D$

$\mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$((D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ a)\ \cdot_D$
$(\tau_1\ f\ \star_D\ \tau_0{}'\ a))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $(G_0\ b\ \star_D\ \mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D\ (D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ a))\ \cdot_D$
$\mathrm{l}_D{}^{-1}[(\tau_0\ b\ \star_D\ F\,f)\ \star_D\ \tau_0{}'\ a]$
$= \mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a]\ \cdot_D$
$\mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$(D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ a)$
using *a b f ide-f* τ*.iso-map*$_1$*-ide*
D.lunit′-naturality [*of* $\mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D\ (D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ a)$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b\ \star_D\ \mathrm{r}_D[G\,f])\ \cdot_D$
$(G_0\ b\ \star_D\ G\,f\ \star_D\ \varepsilon\ a)\ \cdot_D$
$\mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a]\ \cdot_D$
$\mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D$
$\mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]$
proof −
have $((D.inv\ (\tau_1\ f)\ \star_D\ \tau_0{}'\ a)\ \cdot_D\ (\tau_1\ f\ \star_D\ \tau_0{}'\ a))\ \cdot_D\ \mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a] =$
$((D.inv\ (\tau_1\ f)\ \cdot_D\ \tau_1\ f)\ \star_D\ \tau_0{}'\ a)\ \cdot_D\ \mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]$
using *a b f ide-f* τ*.iso-map*$_1$*-ide D.whisker-right* [*of* $\tau_0{}'\ a$] **by** *simp*
also have ... $= \mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]$
using *a b f ide-f* τ*.iso-map*$_1$*-ide D.comp-inv-arr′ D.comp-cod-arr* **by** *auto*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (G_0\ b\ \star_D\ \mathrm{r}_D[G\,f])\ \cdot_D$
$(G_0\ b\ \star_D\ G\,f\ \star_D\ \varepsilon\ a)\ \cdot_D$
$\mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a]$
proof −
have $\mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a]\ \cdot_D\ \mathrm{a}_D[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a]\ \cdot_D\ \mathrm{a}_D{}^{-1}[G\,f,\ \tau_0\ a,\ \tau_0{}'\ a] =$
$\mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a]$
using *a b f ide-f D.comp-arr-inv′ D.comp-arr-dom* **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((G_0\ b\ \star_D\ \mathrm{r}_D[G\,f])\ \cdot_D\ \mathrm{l}_D{}^{-1}[G\,f\ \star_D\ G_0\ a])\ \cdot_D\ (G\,f\ \star_D\ \varepsilon\ a)$
proof −
have $(G_0\ b\ \star_D\ \mathrm{r}_D[G\,f])\ \cdot_D\ (G_0\ b\ \star_D\ G\,f\ \star_D\ \varepsilon\ a)\ \cdot_D\ \mathrm{l}_D{}^{-1}[G\,f\ \star_D\ \tau_0\ a\ \star_D\ \tau_0{}'\ a] =$
$(G_0\ b\ \star_D\ \mathrm{r}_D[G\,f])\ \cdot_D\ \mathrm{l}_D{}^{-1}[G\,f\ \star_D\ G_0\ a]\ \cdot_D\ (G\,f\ \star_D\ \varepsilon\ a)$
using *a b f ide-f D.lunit′-naturality* [*of* $G\,f\ \star_D\ \varepsilon\ a$] **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{l}_D{}^{-1}[G\,f]\ \cdot_D\ \mathrm{r}_D[G\,f])\ \cdot_D\ (G\,f\ \star_D\ \varepsilon\ a)$

using *a b f ide-f D.lunit′-naturality* [*of* $\mathrm{r}_D[G\,f]$] **by** *auto*
finally show $(\mathrm{l}_D{}^{-1}[G\,f] \cdot_D \mathrm{r}_D[G\,f]) \cdot_D (G\,f \star_D \varepsilon\ a) = (\varepsilon\ b \star_D G\,f) \cdot_D \tau\tau'.map_1\ f$
by *simp*
qed
qed

lemma *counit-is-invertible-modification*:
shows *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G\ \Phi_G$
$\tau\tau'.map_0\ \tau\tau'.map_1\ \iota_G.map_0\ \iota_G.map_1\ \varepsilon$
..

end

end

1.16 Equivalence of Bicategories

In this section, we define the notion "equivalence of bicategories", which generalizes the notion of equivalence of categories, and we establish various of its properties. In particular, we show that "equivalent bicategories" is an equivalence relation, and that a pseudofunctor is part of an equivalence of bicategories if and only if it is an equivalence pseudofunctor (*i.e.* it is faithful, locally full, locally essentially surjective, and biessentially surjective on objects).

theory *EquivalenceOfBicategories*
imports *InternalAdjunction PseudonaturalTransformation*
begin

1.16.1 Definition of Equivalence of Bicategories

An equivalence of bicategories between bicategories C and D consists of pseudofunctors $F : D \to C$ and $G : C \to D$ and pseudonatural equivalences η: $I_D \approx GF$ and ε: $FG \approx I_C$.

locale *equivalence-of-bicategories* =
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F: *pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$ +
G: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$ +
FG: *composite-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$ +
GF: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G$ +
I_C: *identity-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
I_D: *identity-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
η: *pseudonatural-equivalence* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$I_D.map\ I_D.cmp\ GF.map\ GF.cmp\ \eta_0\ \eta_1$ +
ε: *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$

FG.map FG.cmp I_C*.map* I_C*.cmp* ε_0 ε_1
for V_C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *'c comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: *'c* ⇒ *'c* ⇒ *'c* ⇒ *'c* (‹a_C[-, -, -]›)
and i_C :: *'c* ⇒ *'c* (‹i_C[-]›)
and src_C :: *'c* ⇒ *'c*
and trg_C :: *'c* ⇒ *'c*
and V_D :: *'d comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: *'d comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: *'d* ⇒ *'d* ⇒ *'d* ⇒ *'d* (‹a_D[-, -, -]›)
and i_D :: *'d* ⇒ *'d* (‹i_D[-]›)
and src_D :: *'d* ⇒ *'d*
and trg_D :: *'d* ⇒ *'d*
and F :: *'d* ⇒ *'c*
and Φ_F :: *'d* ∗ *'d* ⇒ *'c*
and G :: *'c* ⇒ *'d*
and Φ_G :: *'c* ∗ *'c* ⇒ *'d*
and η_0 :: *'d* ⇒ *'d*
and η_1 :: *'d* ⇒ *'d*
and ε_0 :: *'c* ⇒ *'c*
and ε_1 :: *'c* ⇒ *'c*
begin

notation *C.isomorphic* (**infix** ‹$\cong_C$› *50*)
notation *D.isomorphic* (**infix** ‹$\cong_D$› *50*)

notation *C.iso-in-hom* (‹«- : - $\cong_C$ -»›)
notation *D.iso-in-hom* (‹«- : - $\cong_D$ -»›)

notation *C.some-quasi-inverse* (‹-$^{\sim C}$› [*1000*] *1000*)
notation *D.some-quasi-inverse* (‹-$^{\sim D}$› [*1000*] *1000*)

interpretation η': *converse-pseudonatural-equivalence*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D
I_D*.map* I_D*.cmp GF.map GF.cmp* η_0 η_1
..
interpretation ε': *converse-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
FG.map FG.cmp I_C*.map* I_C*.cmp* ε_0 ε_1
..

Although it will be some trouble yet to prove that F is an equivalence pseudofunctor, it is not as difficult to prove that the composites GF and FG are equivalence pseudofunctors, by virtue of their being pseudonaturally equivalent to identity pseudofunctors.

interpretation *GF*: *equivalence-pseudofunctor*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D *GF.map GF.cmp*
proof −
interpret *GF*: *pseudofunctor-pseudonaturally-equivalent-to-equivalence-pseudofunctor*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D

I_D.*map* I_D.*cmp* *GF.map* *GF.cmp* η_0 η_1
..
show *equivalence-pseudofunctor*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D *GF.map* *GF.cmp*
using *GF.is-equivalence-pseudofunctor* **by** *simp*
qed

interpretation *FG*: *equivalence-pseudofunctor*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C *FG.map* *FG.cmp*
proof −
interpret *FG*: *pseudofunctor-pseudonaturally-equivalent-to-equivalence-pseudofunctor*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
I_C.*map* I_C.*cmp* *FG.map* *FG.cmp* ε'.map_0 ε'.map_1
..
show *equivalence-pseudofunctor*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C *FG.map* *FG.cmp*
using *FG.is-equivalence-pseudofunctor* **by** *simp*
qed

It is also easy to establish that F and G are faithful. We will use the facts, that GF is locally full and G is faithful, to prove that F is locally full.

interpretation *F*: *faithful-functor* V_D V_C F
proof
fix μ μ'
assume *par*: *D.par* μ μ' **and** *eq*: F $\mu = F$ μ'
have src_D $\mu = src_D$ $\mu' \wedge trg_D$ $\mu = trg_D$ μ'
using *par* *D.src-dom* *D.trg-cod* **by** (*metis* *D.src-cod* *D.trg-cod*)
hence η_0 (trg_D μ) $\star_D$ $\mu = \eta_0$ (trg_D μ) $\star_D$ μ'
using *par* *eq* η.*iso-map*$_1$*-ide* *D.comp-arr-inv'* *D.trg.preserves-dom*
D.comp-arr-dom *D.comp-assoc*
by (*metis* *GF.is-faithful* *o-apply*)
thus $\mu = \mu'$
using *par* η.*ide-map*$_0$*-obj* η.*components-are-equivalences*
D.equivalence-cancel-left [*of* η_0 (trg_D μ) μ μ']
by *simp*
qed

interpretation *G*: *faithful-functor* V_C V_D G
proof
show $\bigwedge\mu$ μ'. $[\![C.par\ \mu\ \mu';\ G\ \mu = G\ \mu']\!] \Longrightarrow \mu = \mu'$
proof −
fix μ μ'
assume *par*: *C.par* μ μ' **and** *eq*: G $\mu = G$ μ'
have src_C $\mu = src_C$ $\mu' \wedge trg_C$ $\mu = trg_C$ μ'
using *par* **by** (*metis* *C.src-cod* *C.trg-cod*)
hence $\mu \star_C \varepsilon_0$ (src_C μ) $= \mu' \star_C \varepsilon_0$ (src_C μ)
using *par* *eq* ε.*iso-map*$_1$*-ide* *C.comp-inv-arr'* *C.src.preserves-dom*
C.comp-arr-dom *C.comp-assoc*
by (*metis* *FG.is-faithful* *o-apply*)

```
    thus μ = μ'
      using par ε.ide-map₀-obj ε.components-are-equivalences
            C.equivalence-cancel-right [of ε₀ (src_C μ) μ μ']
      by simp
  qed
qed
```

It is perhaps not so easy to discover a proof that F is locally essentially surjective. Here we follow the proof of [7], Lemma 6.2.13, except we have expressed it in a way that explicitly shows its constructive nature, given that we have already chosen an extension of each component of η and ε to an adjoint equivalence.

```
abbreviation Φ
where Φ ψ f f' ≡ r_C[f'] ·_C
                 (f' ⋆_C ε'.counit (src_C f)) ·_C
                 a_C[f', ε₀ (src_C f), ε'.map₀ (src_C f)] ·_C
                 (C.inv (ε₁ f') ⋆_C ε'.map₀ (src_C f)) ·_C
                 a_C⁻¹[ε₀ (trg_C f), F (G f'), ε'.map₀ (src_C f)] ·_C
                 (ε₀ (trg_C f) ⋆_C F ψ ⋆_C ε'.map₀ (src_C f)) ·_C
                 (ε₀ (trg_C f) ⋆_C C.inv (ε'.map₁ f)) ·_C
                 a_C[ε₀ (trg_C f), ε'.map₀ (trg_C f), f] ·_C
                 (C.inv (ε'.counit (trg_C f)) ⋆_C f) ·_C
                 l_C⁻¹[f]

lemma G-reflects-isomorphic:
assumes C.ide f and C.ide f' and src_C f = src_C f' and trg_C f = trg_C f'
and «ψ : G f ≅_D G f'»
shows «Φ ψ f f' : f ≅_C f'»
proof −
  let ?a = src_C f and ?a' = trg_C f
  have f: «f : ?a →_C ?a'» ∧ C.ide f
    using assms by simp
  have f': «f' : ?a →_C ?a'» ∧ C.ide f'
    using assms by simp
  have ψ: «ψ : G.map₀ ?a →_D G.map₀ ?a'»
    using assms D.vconn-implies-hpar(1−2)
    by (elim D.iso-in-homE) auto
  hence Fψ: «F ψ : F.map₀ (G.map₀ ?a) →_C F.map₀ (G.map₀ ?a')»
    by auto
  show «Φ ψ f f' : f ≅_C f'»
  proof (intro C.comp-iso-in-hom)
    show «l_C⁻¹[f] : f ≅_C ?a' ⋆_C f»
      using f by auto
    show «C.inv (ε'.counit ?a') ⋆_C f : ?a' ⋆_C f ≅_C (ε₀ ?a' ⋆_C ε'.map₀ ?a') ⋆_C f»
      using f ε'.counit-in-hom [of ?a'] ε'.iso-counit [of ?a']
      by (intro C.hcomp-iso-in-hom) auto
    show «a_C[ε₀ ?a', ε'.map₀ ?a', f] :
            (ε₀ ?a' ⋆_C ε'.map₀ ?a') ⋆_C f ≅_C ε₀ ?a' ⋆_C ε'.map₀ ?a' ⋆_C f»
      using f ε'.counit-in-hom [of ?a] ε'.ide-map₀-obj
      by (intro C.iso-in-homI) auto
```

show «ε_0 *?a'* $\star_C$ *C.inv* (ε'.*map*$_1$ *f*) :
ε_0 *?a'* $\star_C$ ε'.*map*$_0$ *?a'* $\star_C$ *f* $\cong_C$ ε_0 *?a'* $\star_C$ *F* (*G f*) $\star_C$ ε'.*map*$_0$ *?a*»
using *f* ε'.*map*$_1$*-in-hom* [*of f*] ε'.*iso-map*$_1$*-ide* [*of f*] *C.ide-iso-in-hom*
by (*intro C.hcomp-iso-in-hom*) *auto*
show «ε_0 *?a'* $\star_C$ *F* ψ $\star_C$ ε'.*map*$_0$ *?a* :
ε_0 *?a'* $\star_C$ *F* (*G f*) $\star_C$ ε'.*map*$_0$ *?a* $\cong_C$ ε_0 *?a'* $\star_C$ *F* (*G f'*) $\star_C$ ε'.*map*$_0$ *?a*»
using *assms f f' F*ψ *F.preserves-iso C.ide-iso-in-hom*
by (*intro C.hcomp-iso-in-hom*) *auto*
show «$\mathrm{a}_C{}^{-1}$[ε_0 *?a'*, *F* (*G f'*), ε'.*map*$_0$ *?a*] :
ε_0 *?a'* $\star_C$ *F* (*G f'*) $\star_C$ ε'.*map*$_0$ *?a* $\cong_C$ (ε_0 *?a'* $\star_C$ *F* (*G f'*)) $\star_C$ ε'.*map*$_0$ *?a*»
using *assms f'* ε.*map*$_0$*-in-hom*(*1*) [*of ?a'*] ε.*ide-map*$_0$*-obj* [*of ?a'*]
ε'.*map*$_0$*-in-hom*(*1*) [*of ?a*] ε'.*ide-map*$_0$*-obj* [*of ?a*]
C.assoc'-in-hom C.iso-assoc'
by *auto*
show «*C.inv* (ε_1 *f'*) $\star_C$ ε'.*map*$_0$ *?a* :
(ε_0 *?a'* $\star_C$ *F* (*G f'*)) $\star_C$ ε'.*map*$_0$ *?a* $\cong_C$ (*f'* $\star_C$ ε_0 *?a*) $\star_C$ ε'.*map*$_0$ *?a*»
using *assms f'* ε.*map*$_1$*-in-hom* ε.*iso-map*$_1$*-ide* ε'.*map*$_0$*-in-hom*(*1*) [*of ?a*]
C.ide-iso-in-hom
by (*intro C.hcomp-iso-in-hom*) *auto*
show «a_C[*f'*, ε_0 *?a*, ε'.*map*$_0$ *?a*] :
(*f'* $\star_C$ ε_0 *?a*) $\star_C$ ε'.*map*$_0$ *?a* $\cong_C$ *f'* $\star_C$ ε_0 *?a* $\star_C$ ε'.*map*$_0$ *?a*»
using *assms f'* ε.*map*$_0$*-in-hom*(*1*) [*of ?a*] ε'.*map*$_0$*-in-hom*(*1*) [*of ?a*]
ε.*ide-map*$_0$*-obj* ε'.*ide-map*$_0$*-obj C.assoc-in-hom*
by *auto*
show «*f'* $\star_C$ ε'.*counit ?a* : *f'* $\star_C$ ε_0 *?a* $\star_C$ ε'.*map*$_0$ *?a* $\cong_C$ *f'* $\star_C$ *?a*»
using *f f'* ε'.*counit-in-hom*
by (*intro C.hcomp-iso-in-hom*) *auto*
show «r_C[*f'*] : *f'* $\star_C$ *?a* $\cong_C$ *f'*»
using *assms f'* **by** *auto*
qed
qed

abbreviation Ψ
where Ψ *f b b'* $\equiv$ r_D[*G* (*F* (η'.*map*$_0$ *b'* $\star_D$ *G f* $\star_D$ η_0 *b*))] $\cdot_D$
(G (F (η'.*map*$_0$ b' $\star_D$ G f $\star_D$ η_0 b)) $\star_D$ $\eta'.\varepsilon$ b) $\cdot_D$
a_D[G (F (η'.*map*$_0$ b' $\star_D$ G f $\star_D$ η_0 b)), η_0 b, η'.*map*$_0$ b] $\cdot_D$
(D.*inv* (η_1 (η'.*map*$_0$ b' $\star_D$ G f $\star_D$ η_0 b)) $\star_D$ η'.*map*$_0$ b) $\cdot_D$
(a_D[η_0 b', η'.*map*$_0$ b', G f $\star_D$ η_0 b] $\star_D$ η'.*map*$_0$ b) $\cdot_D$
$\mathrm{a}_D{}^{-1}$[η_0 b' $\star_D$ η'.*map*$_0$ b', G f $\star_D$ η_0 b, η'.*map*$_0$ b] $\cdot_D$
((η_0 b' $\star_D$ η'.*map*$_0$ b') $\star_D$ $\mathrm{a}_D{}^{-1}$[G f, η_0 b, η'.*map*$_0$ b]) $\cdot_D$
(D.*inv* (η'.*counit* b') $\star_D$ G f $\star_D$ D.*inv* (η'.*counit* b)) $\cdot_D$
$\mathrm{l}_D{}^{-1}$[G f $\star_D$ G.*map*$_0$ (F.*map*$_0$ b)] $\cdot_D$
$\mathrm{r}_D{}^{-1}$[G f]

lemma *F-is-locally-essentially-surjective*:
assumes *D.obj b* **and** *D.obj b'* **and** «*f* : *F.map*$_0$ *b* $\rightarrow_C$ *F.map*$_0$ *b'*» **and** *C.ide f*
shows «Φ (Ψ *f b b'*) *f* (*F* (η'.*map*$_0$ *b'* $\star_D$ *G f* $\star_D$ η_0 *b*)) :
f $\cong_C$ *F* (η'.*map*$_0$ *b'* $\star_D$ *G f* $\star_D$ η_0 *b*)»
proof −

let $?g = \eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b$
have *g-in-hhom*: «$?g : b \rightarrow_D b'$»
  using *assms* η.*ide-map*$_0$*-obj* η.*map*$_0$*-in-hhom* **by** *auto*
have *ide-g*: *D.ide ?g*
  using *assms g-in-hhom* η.*ide-map*$_0$*-obj* η'.*ide-map*$_0$*-obj G.preserves-ide* **by** *blast*
let $?\psi = \Psi\ f\ b\ b'$
have «$?\psi : G\ f \cong_D G\ (F\ ?g)$»
proof (*intro D.comp-iso-in-hom*)
  show «$\mathrm{r}_D{}^{-1}[G\ f] : G\ f \cong_D G\ f \star_D G.map_0\ (F.map_0\ b)$»
    using *assms*
    by (*intro D.iso-in-homI*) *auto*
  show «$\mathrm{l}_D{}^{-1}[G\ f \star_D G.map_0\ (F.map_0\ b)] :$
        $G\ f \star_D G.map_0\ (F.map_0\ b)$
          $\cong_D G.map_0\ (F.map_0\ b') \star_D G\ f \star_D G.map_0\ (F.map_0\ b)$»
  proof −
    have $D.ide\ (G\ f \star_D G.map_0\ (F.map_0\ b))$
      using *assms D.ide-hcomp* [*of* $G\ f\ G.map_0\ (F.map_0\ b)$]
      by (*metis D.hcomp-in-hhomE D.hseqE D.objE F.map*$_0$*-simps*(*1*) *G.map*$_0$*-simps*(*1*)
        G.preserves-ide GF.map$_0$*-simp* η.*map*$_0$*-simps*(*3*) *g-in-hhom*)
    thus *?thesis*
      using *assms* η.*ide-map*$_0$*-obj*
      by (*intro D.iso-in-homI*) *auto*
  qed
  show «$D.inv\ (\eta'.counit\ b') \star_D G\ f \star_D D.inv\ (\eta'.counit\ b) :$
        $G.map_0\ (F.map_0\ b') \star_D G\ f \star_D G.map_0\ (F.map_0\ b)$
          $\cong_D (\eta_0\ b' \star_D \eta'.map_0\ b') \star_D G\ f \star_D (\eta_0\ b \star_D \eta'.map_0\ b)$»
  using *assms* η'.*iso-counit* η'.*counit-in-hom*(*2*) *D.vconn-implies-hpar*(*4*) *D.ide-iso-in-hom*
    by (*intro D.hcomp-iso-in-hom*) *auto*
  show «$(\eta_0\ b' \star_D \eta'.map_0\ b') \star_D \mathrm{a}_D{}^{-1}[G\ f, \eta_0\ b, \eta'.map_0\ b] :$
        $(\eta_0\ b' \star_D \eta'.map_0\ b') \star_D G\ f \star_D (\eta_0\ b \star_D \eta'.map_0\ b)$
          $\cong_D (\eta_0\ b' \star_D \eta'.map_0\ b') \star_D (G\ f \star_D \eta_0\ b) \star_D \eta'.map_0\ b$»
  proof −
    have «$\mathrm{a}_D{}^{-1}[G\ f, \eta_0\ b, \eta'.map_0\ b] :$
          $G\ f \star_D (\eta_0\ b \star_D \eta'.map_0\ b) \cong_D (G\ f \star_D \eta_0\ b) \star_D \eta'.map_0\ b$»
      using *assms* η.*ide-map*$_0$*-obj* η'.*ide-map*$_0$*-obj* η.*map*$_0$*-in-hom* η'.*map*$_0$*-in-hom*
        D.assoc'-in-hom D.iso-assoc'
      by (*intro D.iso-in-homI*) *auto*
    thus *?thesis*
      using *assms*
      by (*intro D.hcomp-iso-in-hom*) *auto*
  qed
  show «$\mathrm{a}_D{}^{-1}[\eta_0\ b' \star_D \eta'.map_0\ b', G\ f \star_D \eta_0\ b, \eta'.map_0\ b] :$
        $(\eta_0\ b' \star_D \eta'.map_0\ b') \star_D (G\ f \star_D \eta_0\ b) \star_D \eta'.map_0\ b$
          $\cong_D ((\eta_0\ b' \star_D \eta'.map_0\ b') \star_D (G\ f \star_D \eta_0\ b)) \star_D \eta'.map_0\ b$»
    using *assms* η.*ide-map*$_0$*-obj* η'.*ide-map*$_0$*-obj* η.*map*$_0$*-in-hom* η'.*map*$_0$*-in-hom*
      D.assoc'-in-hom D.iso-assoc'
    by (*intro D.iso-in-homI*) *auto*
  show «$\mathrm{a}_D[\eta_0\ b', \eta'.map_0\ b', G\ f \star_D \eta_0\ b] \star_D \eta'.map_0\ b :$
        $((\eta_0\ b' \star_D \eta'.map_0\ b') \star_D (G\ f \star_D \eta_0\ b)) \star_D \eta'.map_0\ b$

$\cong_D$ (η_0 b' $\star_D$?g) $\star_D$ η'.map_0 b»
using *assms* η.*ide-map*$_0$*-obj* η'.*ide-map*$_0$*-obj* η.*map*$_0$*-in-hom* η'.*map*$_0$*-in-hom* *D.assoc-in-hom D.iso-assoc*
by (*intro D.hcomp-iso-in-hom D.iso-in-homI*) *auto*
show «*D.inv* (η_1 ?g) $\star_D$ η'.map_0 b :
(η_0 b' $\star_D$?g) $\star_D$ η'.map_0 b $\cong_D$ (G (F ?g) $\star_D$ η_0 b) $\star_D$ η'.map_0 b»
using *assms* η.*map*$_1$*-in-hom* [*of ?g*] η.*iso-map*$_1$*-ide* η'.*map*$_0$*-in-hom g-in-hhom ide-g*
by (*intro D.hcomp-iso-in-hom D.iso-in-homI*) *auto*
show «a_D[G (F ?g), η_0 b, η'.map_0 b] :
(G (F ?g) $\star_D$ η_0 b) $\star_D$ η'.map_0 b $\cong_D$ G (F ?g) $\star_D$ η_0 b $\star_D$ η'.map_0 b»
using *assms* η.*ide-map*$_0$*-obj* η'.*ide-map*$_0$*-obj* η.*map*$_0$*-in-hom* η'.*map*$_0$*-in-hom* *D.assoc-in-hom D.iso-assoc*
by (*intro D.iso-in-homI*) *auto*
show «G (F ?g) $\star_D$ η'.*counit* b :
G (F ?g) $\star_D$ η_0 b $\star_D$ η'.map_0 b $\cong_D$ G (F ?g) $\star_D$ G.map_0 (F.map_0 b)»
using *assms* η'.*counit-in-hom D.ide-in-hom*(*2*) *ide-g*
by (*intro D.hcomp-iso-in-hom*) *auto*
show «r_D[G (F ?g)] : G (F ?g) $\star_D$ G.map_0 (F.map_0 b) $\cong_D$ G (F ?g)»
using *assms ide-g*
by (*intro D.iso-in-homI*) *auto*
qed
thus «Φ ?ψ f (F ?g) : f $\cong_C$ F ?g»
using *assms ide-g G-reflects-isomorphic* [*of f F ?g ?ψ*]
by (*metis C.horizontal-homs-axioms D.in-hhomE F.preserves-ide F.preserves-src F.preserves-trg g-in-hhom horizontal-homs.in-hhomE*)
qed

interpretation *F*: *equivalence-pseudofunctor*
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C F Φ_F
proof
show $\bigwedge$b. C.obj b $\Longrightarrow$ $\exists$ a. D.obj a $\wedge$ C.*equivalent-objects* (F.map_0 a) b
by (*metis FG.biessentially-surjective-on-objects FG.map*$_0$*-simp G.map*$_0$*-simps*(*1*))
show $\bigwedge$g g' μ. ⟦D.ide g; D.ide g'; src_D g = src_D g'; trg_D g = trg_D g';
«μ : F g $\Rightarrow_C$ F g'»⟧
$\Longrightarrow$ $\exists$$\nu$. «$\nu$: g $\Rightarrow_D$ g'» $\wedge$ F ν = μ
proof −
fix g g' μ
assume *g*: D.ide g **and** g': D.ide g'
assume *eq-src*: src_D g = src_D g' **and** *eq-trg*: trg_D g = trg_D g'
assume μ: «μ : F g $\Rightarrow_C$ F g'»
obtain ν **where** ν: «ν : g $\Rightarrow_D$ g'» $\wedge$ G (F ν) = G μ
using g g' *eq-src eq-trg* μ *GF.locally-full* [*of* g g' G μ] **by** *auto*
have F ν = μ
using μ ν *G.is-faithful*
by (*metis* (*mono-tags*, *lifting*) *C.in-homE F.preserves-hom*)
thus $\exists$$\nu$. «$\nu$: g $\Rightarrow_D$ g'» $\wedge$ F ν = μ
using ν **by** *auto*
qed
show $\bigwedge$b b' f. ⟦D.obj b; D.obj b'; «f : F.map_0 b $\rightarrow_C$ F.map_0 b'»; C.ide f⟧

$\Longrightarrow \exists\, g.$ «$g : b \to_D b'$» $\wedge$ $D.ide\ g \wedge C.isomorphic\ (F\ g)\ f$

proof −

fix $b\ b'\ f$

assume b: $D.obj\ b$ **and** b': $D.obj\ b'$ **and** f: «$f : F.map_0\ b \to_C F.map_0\ b'$»

assume *ide-f*: $C.ide\ f$

let $?g = \eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b$

have *g-in-hhom*: «$?g : b \to_D b'$»

using $b\ b'\ f\ \eta.ide\text{-}map_0\text{-}obj\ \eta.map_0\text{-}in\text{-}hhom$ **by** *auto*

have *ide-g*: $D.ide\ ?g$

using $b\ b'\ f$ *ide-f g-in-hhom* $\eta.ide\text{-}map_0\text{-}obj\ \eta'.ide\text{-}map_0\text{-}obj\ G.preserves\text{-}ide$ **by** *blast*

have $f \cong_C F\ ?g$

using $b\ b'\ f$ *ide-f F-is-locally-essentially-surjective C.isomorphic-symmetric*

by (*meson C.isomorphicI'*)

thus $\exists\, g.$ «$g : b \to_D b'$» $\wedge$ $D.ide\ g \wedge F\ g \cong_C f$

using *g-in-hhom ide-g C.isomorphic-symmetric C.isomorphic-def* **by** *blast*

qed

qed

lemma *equivalence-pseudofunctor-left*:

shows $equivalence\text{-}pseudofunctor\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$

..

end

1.16.2 Equivalences Respect Pseudonatural Equivalence

In this section we prove that, in an equivalence of bicategories, either pseudofunctor may be replaced by a pseudonaturally equivalent one and still obtain an equivalence of bicategories.

locale *equivalence-of-bicategories-and-pseudonatural-equivalence-left* =

C: $bicategory\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +

D: $bicategory\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +

E: *equivalence-of-bicategories* +

τ: *pseudonatural-equivalence*

$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ F'\ \Phi_F{}'\ \tau_0\ \tau_1$

for F'

and $\Phi_F{}'$

and τ_0

and τ_1

begin

interpretation GF': $composite\text{-}pseudofunctor\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$

$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F'\ \Phi_F{}'\ G\ \Phi_G$

..

interpretation $F'G$: $composite\text{-}pseudofunctor\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$

$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F'\ \Phi_F{}'$

..

interpretation τ': *converse-pseudonatural-equivalence*

V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C F Φ_F F' $\Phi_F{}'$ τ_0 τ_1
..
interpretation $\tau' o G$: *pseudonatural-equivalence-whisker-right* V_C H_C a_C i_C src_C trg_C
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F' $\Phi_F{}'$ F Φ_F G Φ_G $\tau'.map_0$ $\tau'.map_1$
..
interpretation $Go\tau$: *pseudonatural-equivalence-whisker-left* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ_F F' $\Phi_F{}'$ G Φ_G τ_0 τ_1
..
sublocale *unit*: *composite-pseudonatural-equivalence*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D
$E.I_D.map$ $E.I_D.cmp$ $E.GF.map$ $E.GF.cmp$ $GF'.map$ $GF'.cmp$
η_0 η_1 $Go\tau.map_0$ $Go\tau.map_1$
..
sublocale *counit*: *composite-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
$F'G.map$ $F'G.cmp$ $E.FG.map$ $E.FG.cmp$ $E.I_C.map$ $E.I_C.cmp$
$\tau' oG.map_0$ $\tau' oG.map_1$ ε_0 ε_1
..

definition $unit_0$
where $unit_0 \equiv unit.map_0$

definition $unit_1$
where $unit_1 \equiv unit.map_1$

definition $counit_0$
where $counit_0 \equiv counit.map_0$

definition $counit_1$
where $counit_1 \equiv counit.map_1$

sublocale *equivalence-of-bicategories*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F' $\Phi_F{}'$ G Φ_G
$unit_0$ $unit_1$ $counit_0$ $counit_1$
unfolding $unit_0$-*def* $unit_1$-*def* $counit_0$-*def* $counit_1$-*def*
..

lemma *induces-equivalence-of-bicategories*:
shows *equivalence-of-bicategories*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F' $\Phi_F{}'$ G Φ_G
$unit_0$ $unit_1$ $counit_0$ $counit_1$
..

end

locale *equivalence-of-bicategories-and-pseudonatural-equivalence-right* =
C: *bicategory* V_C H_C a_C i_C src_C trg_C +

D: *bicategory* V_D H_D a_D i_D src_D trg_D +
E: *equivalence-of-bicategories* +
τ: *pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G G' $\Phi_G{}'$ τ_0 τ_1
for G'
and $\Phi_G{}'$
and τ_0
and τ_1
begin

interpretation $G'F$: *composite-pseudofunctor* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G' $\Phi_G{}'$
..
interpretation FG': *composite-pseudofunctor* V_C H_C a_C i_C src_C trg_C
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C G' $\Phi_G{}'$ F Φ_F
..
interpretation τ': *converse-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G G' $\Phi_G{}'$ τ_0 τ_1
..
interpretation τoF: *pseudonatural-equivalence-whisker-right* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
G Φ_G G' $\Phi_G{}'$ F Φ_F τ_0 τ_1
..
interpretation $Fo\tau'$: *pseudonatural-equivalence-whisker-left*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
G' $\Phi_G{}'$ G Φ_G F Φ_F $\tau'.map_0$ $\tau'.map_1$
..
sublocale *unit*: *composite-pseudonatural-equivalence*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D
$E.I_D.map$ $E.I_D.cmp$ $E.GF.map$ $E.GF.cmp$ $G'F.map$ $G'F.cmp$
η_0 η_1 $\tau oF.map_0$ $\tau oF.map_1$
..
sublocale *counit*: *composite-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
$FG'.map$ $FG'.cmp$ $E.FG.map$ $E.FG.cmp$ $E.I_C.map$ $E.I_C.cmp$
$Fo\tau'.map_0$ $Fo\tau'.map_1$ ε_0 ε_1
..

definition $unit_0$
where $unit_0 \equiv unit.map_0$

definition $unit_1$
where $unit_1 \equiv unit.map_1$

definition $counit_0$
where $counit_0 \equiv counit.map_0$

definition $counit_1$
where $counit_1 \equiv counit.map_1$

sublocale *equivalence-of-bicategories*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G' $\Phi_G{}'$
$unit_0$ $unit_1$ $counit_0$ $counit_1$
unfolding $unit_0$*-def* $unit_1$*-def* $counit_0$*-def* $counit_1$*-def*
..

lemma *induces-equivalence-of-bicategories*:
shows *equivalence-of-bicategories*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G' $\Phi_G{}'$
$unit_0$ $unit_1$ $counit_0$ $counit_1$
..

end

1.16.3 Converse of an Equivalence

Equivalence of bicategories is a symmetric notion, in the sense that from an equivalence of bicategories from C to D we may obtain an equivalence of bicategories from D to C. The converse equivalence is obtained by interchanging the pseudofunctors F and G and replacing the pseudonatural equivalences η and ε by converse equivalences. Essentially all the work goes into proving that pseudonatural equivalences have pseudonatural converses, which we have already done.

locale *converse-equivalence-of-bicategories* =
C: *bicategory* V_C H_C a_C i_C src_C trg_C +
D: *bicategory* V_D H_D a_D i_D src_D trg_D +
I_C: *identity-pseudofunctor* V_C H_C a_C i_C src_C trg_C +
I_D: *identity-pseudofunctor* V_D H_D a_D i_D src_D trg_D +
E: *equivalence-of-bicategories*
begin

sublocale *counit*: *converse-pseudonatural-equivalence*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D
$I_D.map$ $I_D.cmp$ $E.GF.map$ $E.GF.cmp$ η_0 η_1
..
sublocale *unit*: *converse-pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
$E.FG.map$ $E.FG.cmp$ $I_C.map$ $I_C.cmp$ ε_0 ε_1
..

definition $unit_0$
where $unit_0 \equiv unit.map_0$

definition $unit_1$
where $unit_1 \equiv unit.map_1$

definition $counit_0$
where $counit_0 \equiv counit.map_0$

definition $counit_1$
where $counit_1 \equiv counit.map_1$

sublocale *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$
$unit_0\ unit_1\ counit_0\ counit_1$
unfolding $unit_0$-*def* $unit_1$-*def* $counit_0$-*def* $counit_1$-*def*
..

lemma *is-equivalence-of-bicategories*:
shows *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$
$unit_0\ unit_1\ counit_0\ counit_1$
..

end

1.16.4 Composition of Equivalences

An equivalence of bicategories from B to C and an equivalence of bicategories from C to D may be composed to obtain an equivalence of bicategories from B to D.

locale *composite-equivalence-of-bicategories* =
B: *bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ +
C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
I_B: *identity-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ +
I_C: *identity-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
I_D: *identity-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
F-: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ F\ \Phi_F$ +
G-: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G$ +
H: *pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ H\ \Phi_H$ +
K: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ K\ \Phi_K$ +
FG: *equivalence-of-bicategories*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F\ G\ \Phi_G\ \varrho_0\ \varrho_1\ \sigma_0\ \sigma_1$ +
HK: *equivalence-of-bicategories*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ H\ \Phi_H\ K\ \Phi_K\ \zeta_0\ \zeta_1\ \xi_0\ \xi_1$
for V_B :: *'b comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: *'b comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: *'b* ⇒ *'b* ⇒ *'b* ⇒ *'b* (‹a_B[-, -, -]›)
and i_B :: *'b* ⇒ *'b* (‹i_B[-]›)
and src_B :: *'b* ⇒ *'b*
and trg_B :: *'b* ⇒ *'b*
and V_C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: *'c comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: *'c* ⇒ *'c* ⇒ *'c* ⇒ *'c* (‹a_C[-, -, -]›)
and i_C :: *'c* ⇒ *'c* (‹i_C[-]›)
and src_C :: *'c* ⇒ *'c*
and trg_C :: *'c* ⇒ *'c*

and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'b$
and Φ_F :: $'c * 'c \Rightarrow 'b$
and G :: $'b \Rightarrow 'c$
and Φ_G :: $'b * 'b \Rightarrow 'c$
and H :: $'d \Rightarrow 'c$
and Φ_H :: $'d * 'd \Rightarrow 'c$
and K :: $'c \Rightarrow 'd$
and Φ_K :: $'c * 'c \Rightarrow 'd$
and ϱ_0 :: $'c \Rightarrow 'c$
and ϱ_1 :: $'c \Rightarrow 'c$
and σ_0 :: $'b \Rightarrow 'b$
and σ_1 :: $'b \Rightarrow 'b$
and ζ_0 :: $'d \Rightarrow 'd$
and ζ_1 :: $'d \Rightarrow 'd$
and ξ_0 :: $'c \Rightarrow 'c$
and ξ_1 :: $'c \Rightarrow 'c$
begin

notation $B.\mathrm{a}'$ (‹$\mathrm{a}_B{}^{-1}$[-, -, -]›)

At this point we could make the explicit definitions:

- $\eta_0 = K\ (\varrho_0\ (H.map_0\ a)) \star_D \zeta_0\ a$
- $\eta_1 = \mathrm{a}_D{}^{-1}[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))), \zeta_0\ (trg_D\ f), f] \cdot_D (K\ (\varrho_0\ (H.map_0\ (trg_D\ f))) \star_D \zeta_1\ f) \cdot_D \mathrm{a}_D[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))), K\ (H\ f), \zeta_0\ (src_D\ f)] \cdot_D (D.inv\ (\Phi_K\ (\varrho_0\ (H.map_0\ (trg_D\ f)), H\ f)) \cdot_D K\ (\varrho_1\ (H\ f)) \cdot_D \Phi_K\ (G\ (F\ (H\ f)), \varrho_0\ (H.map_0\ (src_D\ f))) \star_D \zeta_0\ (src_D\ f)) \cdot_D \mathrm{a}_D{}^{-1}[K\ (G\ (F\ (H\ f))), K\ (\varrho_0\ (H.map_0\ (src_D\ f))), \zeta_0\ (src_D\ f)]$
- $\varepsilon_0 = \sigma_0\ a \star_B F\ (\xi_0\ (G\text{-}.map_0\ a))$
- $\varepsilon_1 = \mathrm{a}_B{}^{-1}[\sigma_0\ (trg_B\ f), F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f))), F\ (H\ (K\ (G\ f)))] \cdot_B (\sigma_0\ (trg_B\ f) \star_B B.inv\ (\Phi_F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f)), H\ (K\ (G\ f)))) \cdot_B F\ (\xi_1\ (G\ f)) \cdot_B \Phi_F\ (G\ f, \xi_0\ (G\text{-}.map_0\ (src_B\ f)))) \cdot_B \mathrm{a}_B[\sigma_0\ (trg_B\ f), F\ (G\ f), F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))] \cdot_B (\sigma_1\ f \star_B F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))) \cdot_B \mathrm{a}_B{}^{-1}[f, \sigma_0\ (src_B\ f), F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))]$

but then it is a daunting task to establish the necessary coherence conditions. It is easier (and more useful) to use general results about composite pseudonatural equivalences, which are somewhat easier to prove, though long calculations are still required for those.

sublocale *FH*: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ H\ \Phi_H\ F\ \Phi_F$

..
sublocale *KG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G K Φ_K
..
interpretation *IG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
G Φ_G $I_C.map$ $I_C.cmp$
..
interpretation *IH*: *composite-pseudofunctor* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
H Φ_H $I_C.map$ $I_C.cmp$
..
interpretation *HKG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
G Φ_G $HK.FG.map$ $HK.FG.cmp$
..
interpretation *GFH*: *composite-pseudofunctor* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
H Φ_H $FG.GF.map$ $FG.GF.cmp$
..
interpretation *KGFH*: *composite-pseudofunctor* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
$GFH.map$ $GFH.cmp$ K Φ_K
..
interpretation *FHKG*: *composite-pseudofunctor* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_B H_B a_B i_B src_B trg_B
$HKG.map$ $HKG.cmp$ F Φ_F
..
interpretation ϱoH: *pseudonatural-equivalence-whisker-right* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
$I_C.map$ $I_C.cmp$ $FG.GF.map$ $FG.GF.cmp$ H Φ_H ϱ_0 ϱ_1
..
interpretation $Ko\varrho oH$: *pseudonatural-equivalence-whisker-left* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
H Φ_H $GFH.map$ $GFH.cmp$ K Φ_K $\varrho oH.map_0$ $\varrho oH.map_1$
proof −
interpret $Ko\varrho oH$: *pseudonatural-equivalence-whisker-left* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
$IH.map$ $IH.cmp$ $GFH.map$ $GFH.cmp$ K Φ_K $\varrho oH.map_0$ $\varrho oH.map_1$
..
have $IH.map = H$
using $H.extensionality$ $IH.extensionality$ $H.functor\text{-}axioms$ **by** *force*
moreover have $IH.cmp = \Phi_H$
proof
fix $\mu\nu$
show $IH.cmp\ \mu\nu = \Phi_H\ \mu\nu$
using $IH.cmp\text{-}def$ $D.VV.arr\text{-}char_{SbC}$ $D.VV.dom\text{-}simp$ $D.VV.cod\text{-}simp$ $H.FF\text{-}def$
$C.comp\text{-}arr\text{-}dom$ $C.comp\text{-}cod\text{-}arr$ $H.\Phi.naturality1$ $H.\Phi.extensionality$
by (*cases* $D.VV.arr\ \mu\nu$) *auto*

qed
ultimately show *pseudonatural-equivalence-whisker-left* V_D H_D a_D i_D src_D trg_D
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
H Φ_H $GFH.map$ $GFH.cmp$ K Φ_K $\varrho oH.map_0$ $\varrho oH.map_1$
using $Ko\varrho oH.pseudonatural\text{-}equivalence\text{-}whisker\text{-}left\text{-}axioms$ **by** *simp*
qed
interpretation ξoG: *pseudonatural-equivalence-whisker-right* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
$HK.FG.map$ $HK.FG.cmp$ $I_C.map$ $I_C.cmp$ G Φ_G ξ_0 ξ_1
..
interpretation $Fo\xi oG$: *pseudonatural-equivalence-whisker-left* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_B H_B a_B i_B src_B trg_B
$HKG.map$ $HKG.cmp$ G Φ_G F Φ_F $\xi oG.map_0$ $\xi oG.map_1$
proof −
interpret $Fo\xi oG$: *pseudonatural-equivalence-whisker-left* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_B H_B a_B i_B src_B trg_B
$HKG.map$ $HKG.cmp$ $IG.map$ $IG.cmp$ F Φ_F $\xi oG.map_0$ $\xi oG.map_1$
..
have $IG.map = G$
using $G\text{-}.extensionality$ $IG.extensionality$
by (*meson* $G\text{-}.functor\text{-}axioms$ $comp\text{-}identity\text{-}functor$)
moreover have $IG.cmp = \Phi_G$
proof
fix $\mu\nu$
show $IG.cmp$ $\mu\nu = \Phi_G$ $\mu\nu$
using $IG.cmp\text{-}def$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}simp$ $B.VV.cod\text{-}simp$ $G\text{-}.FF\text{-}def$
$C.comp\text{-}arr\text{-}dom$ $C.comp\text{-}cod\text{-}arr$ $G\text{-}.\Phi.naturality1$ $G\text{-}.\Phi.extensionality$
by (*cases* $B.VV.arr$ $\mu\nu$) *auto*
qed
ultimately show *pseudonatural-equivalence-whisker-left* V_B H_B a_B i_B src_B trg_B
V_C H_C a_C i_C src_C trg_C V_B H_B a_B i_B src_B trg_B
$HKG.map$ $HKG.cmp$ G Φ_G F Φ_F $\xi oG.map_0$ $\xi oG.map_1$
using $Fo\xi oG.pseudonatural\text{-}equivalence\text{-}whisker\text{-}left\text{-}axioms$ **by** *simp*
qed

sublocale *unit*: *composite-pseudonatural-equivalence*
V_D H_D a_D i_D src_D trg_D V_D H_D a_D i_D src_D trg_D
$I_D.map$ $I_D.cmp$ $HK.GF.map$ $HK.GF.cmp$ $KGFH.map$ $KGFH.cmp$
ζ_0 ζ_1 $Ko\varrho oH.map_0$ $Ko\varrho oH.map_1$
..
sublocale *counit*: *composite-pseudonatural-equivalence*
V_B H_B a_B i_B src_B trg_B V_B H_B a_B i_B src_B trg_B
$FHKG.map$ $FHKG.cmp$ $FG.FG.map$ $FG.FG.cmp$ $I_B.map$ $I_B.cmp$
$Fo\xi oG.map_0$ $Fo\xi oG.map_1$ σ_0 σ_1
..

abbreviation *left-map*
where *left-map* $\equiv FH.map$

abbreviation *right-map*
where *right-map* $\equiv$ *KG.map*

abbreviation *left-cmp*
where *left-cmp* $\equiv$ *FH.cmp*

abbreviation *right-cmp*
where *right-cmp* $\equiv$ *KG.cmp*

definition $unit_0$
where $unit_0 \equiv unit.map_0$

definition $unit_1$
where $unit_1 \equiv unit.map_1$

definition $counit_0$
where $counit_0 \equiv counit.map_0$

definition $counit_1$
where $counit_1 == counit.map_1$

lemma $unit_0$*-simp*:
assumes *D.obj a*
shows $unit_0\ a = K\ (\varrho_0\ (H.map_0\ a)) \star_D \zeta_0\ a$
using *assms* $unit_0$*-def* $unit.map_0$*-def* $Ko\varrho oH.map_0$*-def* $\varrho oH.map_0$*-def* **by** *simp*

lemma $unit_1$*-simp*:
assumes *D.ide f*
shows $unit_1\ f = \mathrm{a}_D{}^{-1}[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))),\ \zeta_0\ (trg_D\ f),\ f] \cdot_D$
$(K\ (\varrho_0\ (H.map_0\ (trg_D\ f))) \star_D \zeta_1\ f) \cdot_D$
$\mathrm{a}_D[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))),\ K\ (H\ f),\ \zeta_0\ (src_D\ f)] \cdot_D$
$(D.inv\ (\Phi_K\ (\varrho_0\ (H.map_0\ (trg_D\ f)),\ H\ f)) \cdot_D$
$K\ (\varrho_1\ (H\ f)) \cdot_D$
$\Phi_K\ (G\ (F\ (H\ f)),\ \varrho_0\ (H.map_0\ (src_D\ f))) \star_D \zeta_0\ (src_D\ f)) \cdot_D$
$\mathrm{a}_D{}^{-1}[K\ (G\ (F\ (H\ f))),\ K\ (\varrho_0\ (H.map_0\ (src_D\ f))),\ \zeta_0\ (src_D\ f)]$
using *assms* $unit_1$*-def* $unit.map_1$*-def* $Ko\varrho oH.map_0$*-def* $\varrho oH.map_0$*-def* $Ko\varrho oH.map_1$*-def* $\varrho oH.map_1$*-def*
by *simp*

lemma $counit_0$*-simp*:
assumes *B.obj a*
shows $counit_0\ a = \sigma_0\ a \star_B F\ (\xi_0\ (G\text{-}.map_0\ a))$
using *assms* $counit_0$*-def* $counit.map_0$*-def* $Fo\xi oG.map_0$*-def* $\xi oG.map_0$*-def* **by** *simp*

lemma $counit_1$*-simp*:
assumes *B.ide f*
shows $counit_1\ f = \mathrm{a}_B{}^{-1}[\sigma_0\ (trg_B\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f))),\ F\ (H\ (K\ (G\ f)))] \cdot_B$
$(\sigma_0\ (trg_B\ f) \star_B$
$B.inv\ (\Phi_F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f)),\ H\ (K\ (G\ f)))) \cdot_B$

$F\ (\xi_1\ (G\ f))\ \cdot_B$
$\Phi_F\ (G\ f,\ \xi_0\ (G\text{-}.map_0\ (src_B\ f))))\ \cdot_B$
$\mathrm{a}_B[\sigma_0\ (trg_B\ f),\ F\ (G\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))]\ \cdot_B$
$(\sigma_1\ f\ \star_B\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f))))\ \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ \sigma_0\ (src_B\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))]$
using *assms* $counit_1$*-def counit.*map_1*-def Fo*ξ*oG.*map_0*-def Fo*ξ*oG.*map_1*-def*
ξ*oG.*map_0*-def* ξ*oG.*map_1*-def*
by *simp*

sublocale *equivalence-of-bicategories* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
FH.map FH.cmp KG.map KG.cmp $unit_0\ unit_1\ counit_0\ counit_1$
proof −
interpret *FH-KG*: *composite-pseudofunctor*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
KG.map KG.cmp FH.map FH.cmp
..
interpret *KG-FH*: *composite-pseudofunctor*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
FH.map FH.cmp KG.map KG.cmp
..
have *FH-KG.map* = *FHKG.map* **by** *auto*
moreover have *FH-KG.cmp* = *FHKG.cmp*
proof
fix $\mu\nu$
show *FH-KG.cmp* $\mu\nu$ = *FHKG.cmp* $\mu\nu$
proof (*cases B.VV.arr* $\mu\nu$)
case *False*
thus *?thesis*
using *FH-KG.*Φ*.extensionality FHKG.*Φ*.extensionality* **by** *simp*
next
case *True*
have *FH-KG.cmp* $\mu\nu$ =
$F\ (H\ (K\ (G\ (I_B.cmp\ \mu\nu))))\ \cdot_B$
$F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)\ \star_B\ B.dom\ (snd\ \mu\nu)))))\ \cdot_B$
$F\ (H\ (K\ (\Phi_G\ (B.dom\ (fst\ \mu\nu),\ B.dom\ (snd\ \mu\nu)))))\ \cdot_B$
$F\ (H\ (\Phi_K\ (G\ (B.dom\ (fst\ \mu\nu)),\ G\ (B.dom\ (snd\ \mu\nu)))))\ \cdot_B$
$(F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)))\ \star_D\ K\ (G\ (B.dom\ (snd\ \mu\nu)))))\ \cdot_B$
$F\ (\Phi_H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))),\ K\ (G\ (B.dom\ (snd\ \mu\nu))))))\ \cdot_B$
$\Phi_F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)))),\ H\ (K\ (G\ (B.dom\ (snd\ \mu\nu)))))$
using *True FH-KG.cmp-def FHKG.cmp-def HKG.cmp-def*
FH.cmp-def HK.FG.cmp-def KG.cmp-def
$B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
$C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
$D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
G-.FF-def H.FF-def K.FF-def B.comp-assoc
by *auto*
also have ... = $F\ (H\ (K\ (G\ (I_B.cmp\ \mu\nu))))\ \cdot_B$
$F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)\ \star_B\ B.dom\ (snd\ \mu\nu)))))\ \cdot_B$
$F\ (H\ (K\ (\Phi_G\ (B.dom\ (fst\ \mu\nu),\ B.dom\ (snd\ \mu\nu)))))\ \cdot_B$

$F\ (H\ (\Phi_K\ (G\ (B.dom\ (fst\ \mu\nu)),\ G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$F\ (\Phi_H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))),\ K\ (G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$\Phi_F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)))),\ H\ (K\ (G\ (B.dom\ (snd\ \mu\nu)))))$

proof −
have $F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))) \star_D K\ (G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$F\ (\Phi_H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))),\ K\ (G\ (B.dom\ (snd\ \mu\nu))))) =$
$F\ (\Phi_H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))),\ K\ (G\ (B.dom\ (snd\ \mu\nu)))))$
using *True* $B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
$C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
$D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
B.comp-cod-arr
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= F\ (H\ (K\ (G\ (I_B.cmp\ \mu\nu)))) \cdot_B$
$F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu) \star_B B.dom\ (snd\ \mu\nu))))) \cdot_B$
$F\ (H\ (K\ (\Phi_G\ (B.dom\ (fst\ \mu\nu),\ B.dom\ (snd\ \mu\nu))))) \cdot_B$
$(F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)) \star_C G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$F\ (H\ (\Phi_K\ (G\ (B.dom\ (fst\ \mu\nu)),\ G\ (B.dom\ (snd\ \mu\nu)))))) \cdot_B$
$F\ (\Phi_H\ (K\ (G\ (B.dom\ (fst\ \mu\nu))),\ K\ (G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$\Phi_F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)))),\ H\ (K\ (G\ (B.dom\ (snd\ \mu\nu)))))$

proof −
have $F\ (H\ (K\ (G\ (B.dom\ (fst\ \mu\nu)) \star_C G\ (B.dom\ (snd\ \mu\nu))))) \cdot_B$
$F\ (H\ (\Phi_K\ (G\ (B.dom\ (fst\ \mu\nu)),\ G\ (B.dom\ (snd\ \mu\nu))))) =$
$F\ (H\ (\Phi_K\ (G\ (B.dom\ (fst\ \mu\nu)),\ G\ (B.dom\ (snd\ \mu\nu)))))$
using *True* $B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
$C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
$D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
G-.FF-def H.FF-def K.FF-def B.comp-assoc
B.comp-cod-arr
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= FHKG.cmp\ \mu\nu$
using *True FH-KG.cmp-def FHKG.cmp-def HKG.cmp-def*
FH.cmp-def HK.FG.cmp-def KG.cmp-def
$B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
$C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
$D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
G-.FF-def H.FF-def K.FF-def B.comp-assoc
by *simp*
finally show *?thesis* **by** *blast*
qed
qed
moreover have *KG-FH.map* = *KGFH.map* **by** *auto*
moreover have *KG-FH.cmp* = *KGFH.cmp*
proof

fix $\mu\nu$
show *KG-FH.cmp* $\mu\nu$ = *KGFH.cmp* $\mu\nu$
proof (*cases D.VV.arr* $\mu\nu$)
  case *False*
  thus *?thesis*
    using *KG-FH*.Φ.*extensionality KGFH*.Φ.*extensionality* **by** *simp*
  next
  case *True*
  have *KG-FH.cmp* $\mu\nu$ =
      $K\ (G\ (F\ (H\ (I_D.cmp\ \mu\nu))))\ \cdot_D$
      $K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)\ \star_D\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (F\ (\Phi_H\ (D.dom\ (fst\ \mu\nu),\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (\Phi_F\ (H\ (D.dom\ (fst\ \mu\nu)),\ H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $(K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)))\ \star_B\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (\Phi_G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))),\ F\ (H\ (D.dom\ (snd\ \mu\nu))))))\ \cdot_D$
      $\Phi_K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)))),\ G\ (F\ (H\ (D.dom\ (snd\ \mu\nu)))))$
    using *True KG-FH.cmp-def FH.cmp-def KG.cmp-def*
      $B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
      $C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
      $D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
      G-.FF-def H.FF-def K.FF-def
      D.comp-assoc
    by *auto*
  also have ... = $K\ (G\ (F\ (H\ (I_D.cmp\ \mu\nu))))\ \cdot_D$
      $K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)\ \star_D\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (F\ (\Phi_H\ (D.dom\ (fst\ \mu\nu),\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (\Phi_F\ (H\ (D.dom\ (fst\ \mu\nu)),\ H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (\Phi_G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))),\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $\Phi_K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)))),\ G\ (F\ (H\ (D.dom\ (snd\ \mu\nu)))))$
  proof −
    have $K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)))\ \star_B\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (\Phi_G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))),\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))$ =
        $K\ (\Phi_G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))),\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))$
      using *True KG-FH.cmp-def FH.cmp-def KG.cmp-def*
        $B.VV.arr\text{-}char_{SbC}$ *B.VV.dom-simp B.VV.cod-simp*
        $C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
        $D.VV.arr\text{-}char_{SbC}$ *D.VV.dom-simp D.VV.cod-simp*
        D.comp-cod-arr
      by *simp*
    thus *?thesis*
      using *D.comp-assoc* **by** *simp*
  qed
  also have ... = $K\ (G\ (F\ (H\ (I_D.cmp\ \mu\nu))))\ \cdot_D$
      $K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)\ \star_D\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (F\ (\Phi_H\ (D.dom\ (fst\ \mu\nu),\ D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $(K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))\ \star_C\ H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $K\ (G\ (\Phi_F\ (H\ (D.dom\ (fst\ \mu\nu)),\ H\ (D.dom\ (snd\ \mu\nu))))))\ \cdot_D$
      $K\ (\Phi_G\ (F\ (H\ (D.dom\ (fst\ \mu\nu))),\ F\ (H\ (D.dom\ (snd\ \mu\nu)))))\ \cdot_D$
      $\Phi_K\ (G\ (F\ (H\ (D.dom\ (fst\ \mu\nu)))),\ G\ (F\ (H\ (D.dom\ (snd\ \mu\nu)))))$

```
      proof −
        have K (G (F (H (D.dom (fst μν)) ⋆C H (D.dom (snd μν))))) ·D
              K (G (ΦF (H (D.dom (fst μν)), H (D.dom (snd μν))))) =
                K (G (ΦF (H (D.dom (fst μν)), H (D.dom (snd μν)))))
          using True KGFH.cmp-def GFH.cmp-def FG.GF.cmp-def
                B.VV.arr-charSbC B.VV.dom-simp B.VV.cod-simp
                C.VV.arr-charSbC C.VV.dom-simp C.VV.cod-simp
                D.VV.arr-charSbC D.VV.dom-simp D.VV.cod-simp
                D.comp-cod-arr
          by simp
        thus ?thesis
          using D.comp-assoc by simp
      qed
      also have ... = KGFH.cmp μν
        using True KGFH.cmp-def GFH.cmp-def FG.GF.cmp-def
              B.VV.arr-charSbC B.VV.dom-simp B.VV.cod-simp
              C.VV.arr-charSbC C.VV.dom-simp C.VV.cod-simp
              D.VV.arr-charSbC D.VV.dom-simp D.VV.cod-simp
              F-.FF-def G-.FF-def H.FF-def K.FF-def
              D.comp-assoc
        by auto
      finally show ?thesis by blast
    qed
  qed
  ultimately show equivalence-of-bicategories
                  VB HB aB iB srcB trgB VD HD aD iD srcD trgD
                  FH.map FH.cmp KG.map KG.cmp unit0 unit1 counit0 counit1
    unfolding unit0-def unit1-def counit0-def counit1-def
    using B.bicategory-axioms D.bicategory-axioms FH.pseudofunctor-axioms
          KG.pseudofunctor-axioms unit.pseudonatural-equivalence-axioms
          counit.pseudonatural-equivalence-axioms IB.identity-pseudofunctor-axioms
          ID.identity-pseudofunctor-axioms FH-KG.composite-pseudofunctor-axioms
          KG-FH.composite-pseudofunctor-axioms
    unfolding equivalence-of-bicategories-def by simp
qed

lemma is-equivalence-of-bicategories:
shows equivalence-of-bicategories VB HB aB iB srcB trgB VD HD aD iD srcD trgD
        left-map left-cmp right-map right-cmp unit0 unit1 counit0 counit1
  ..

end
```

1.16.5 Equivalence with a Dense Sub-bicategory

The purpose of this section is to show that, given a bicategory B and a sub-bicategory defined by a "dense" set of objects of B, the embedding of the sub-bicategory in B extends to an equivalence of bicategories. Here by "dense" we mean that every object of B is equivalent to some object of the subbicategory.

locale *dense-subbicategory* =
  B: *bicategory* V_B H_B a_B i_B src_B trg_B +
  subbicategory V_B H_B a_B i_B src_B trg_B ‹$\lambda\mu$. $B.arr\ \mu \wedge src_B\ \mu \in Obj \wedge trg_B\ \mu \in Obj$›
for V_B :: $'b$ *comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: $'b$ *comp* (**infixr** ‹$\star_B$› *53*)
and a_B :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and i_B :: $'b \Rightarrow 'b$ (‹i_B[-]›)
and src_B :: $'b \Rightarrow 'b$
and trg_B :: $'b \Rightarrow 'b$
and *Obj* :: $'b$ *set* +
assumes *dense*: $\bigwedge a.\ B.obj\ a \Longrightarrow \exists\, a'.\ a' \in Obj \wedge B.equivalent\text{-}objects\ a'\ a$
begin

  notation $B.\mathrm{a}'$ (‹${\mathrm{a}_B}^{-1}$[-, -, -]›)
  notation $B.lunit$ (‹l_B[-]›)
  notation $B.lunit'$ (‹${\mathrm{l}_B}^{-1}$[-]›)
  notation $B.runit$ (‹r_B[-]›)
  notation $B.runit'$ (‹${\mathrm{r}_B}^{-1}$[-]›)

  notation *comp* (**infixr** ‹$\cdot$› *55*)
  notation *hcomp* (**infixr** ‹$\star$› *53*)
  notation *in-hom* (‹«- : - $\Rightarrow$ -»›)
  notation *in-hhom* (‹«- : - $\rightarrow$ -»›)
  notation a (‹a[-, -, -]›)
  notation a' (‹a^{-1}[-, -, -]›)
  notation *lunit* (‹l[-]›)
  notation *lunit'* (‹l^{-1}[-]›)
  notation *runit* (‹r[-]›)
  notation *runit'* (‹r^{-1}[-]›)

  abbreviation (*input*) *Arr*
  where $Arr \equiv \lambda\mu.\ B.arr\ \mu \wedge src_B\ \mu \in Obj \wedge trg_B\ \mu \in Obj$

  sublocale *emb*: *embedding-pseudofunctor* V_B H_B a_B i_B src_B trg_B *Arr*
    ..

  abbreviation E
  where $E \equiv emb.map$

  abbreviation Φ_E
  where $\Phi_E \equiv emb.cmp$

We define a projection P by transporting arrows of B across chosen equivalences between objects of B and objects of the sub-bicategory.

  definition P_0
  where $P_0\ a \equiv SOME\ a'.\ obj\ a' \wedge B.equivalent\text{-}objects\ a'\ a$

  lemma P_0*-props*:
  assumes $B.obj\ a$

shows *obj* $(P_0\ a)$
and *B.equivalent-objects* $(P_0\ a)\ a$
and *B.equivalent-objects* $a\ a' \Longrightarrow P_0\ a = P_0\ a'$
and $P_0\ (P_0\ a) = P_0\ a$
and *B.obj* $(P_0\ a)$
and $P_0\ a \in Obj$
proof −
  have $\exists\, a'.$ *obj* $a' \wedge$ *B.equivalent-objects* $a'\ a$
    using *assms dense* [*of a*] *obj-char arr-char*$_{SbC}$
   by (*metis* (*no-types*, *lifting*) *B.equivalent-objects-def B.in-hhomE B.obj-def B.obj-simps*(*3*))
  hence *1*: *obj* $(P_0\ a) \wedge$ *B.equivalent-objects* $(P_0\ a)\ a$
    unfolding P_0*-def*
    using *someI-ex* [*of* $\lambda a'.$ *obj* $a' \wedge$ *B.equivalent-objects* $a'\ a$] **by** *blast*
  show *obj* $(P_0\ a)$
    using *1* **by** *simp*
  show *2*: *B.equivalent-objects* $(P_0\ a)\ a$
    using *1* **by** *simp*
  show *3*: $\bigwedge a'.$ *B.equivalent-objects* $a\ a' \Longrightarrow P_0\ a = P_0\ a'$
    unfolding P_0*-def*
    using *B.equivalent-objects-symmetric B.equivalent-objects-transitive* **by** *meson*
  show $P_0\ (P_0\ a) = P_0\ a$
    using *2 3* [*of* $P_0\ a$] *B.equivalent-objects-symmetric* **by** *auto*
  show *B.obj* $(P_0\ a)$
    using *1 B.equivalent-objects-def* **by** *auto*
  thus $P_0\ a \in Obj$
    using *1 3 obj-char arr-char*$_{SbC}$ **by** *auto*
qed

For each object a of B, we choose an adjoint equivalence from a to $P_0\ a$. The use of adjoint equivalences is necessary in order to establish the required coherence conditions.

definition *e*
where $e\ a = (SOME\ e.\ «e : a \rightarrow_B P_0\ a» \wedge$
$(\exists\, d\ \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ e\ d\ \eta\ \varepsilon))$

definition *d*
where $d\ a = (SOME\ d.\ \exists\, \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ d\ \eta\ \varepsilon)$

definition η
where $\eta\ a = (SOME\ \eta.\ \exists\, \varepsilon.$ *adjoint-equivalence-in-bicategory*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ (d\ a)\ \eta\ \varepsilon)$

definition ε
where $\varepsilon\ a = (SOME\ \varepsilon.$ *adjoint-equivalence-in-bicategory*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ (d\ a)\ (\eta\ a)\ \varepsilon)$

lemma *chosen-adjoint-equivalence*:
assumes *B.obj a*

shows *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B $(e\ a)$ $(d\ a)$ $(\eta\ a)$ $(\varepsilon\ a)$
and «$e\ a : a \to_B P_0\ a$» **and** $B.ide\ (d\ a)$ **and** $B.ide\ (e\ a)$ **and** $B.iso\ (\eta\ a)$ **and** $B.iso\ (\varepsilon\ a)$
proof −
have $\exists\, e.$ «$e : a \to_B P_0\ a$» $\wedge$
($\exists\, d\ \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B $e\ d\ \eta\ \varepsilon$)
proof −
obtain e **where** e: «$e : a \to_B P_0\ a$» $\wedge$ *B.equivalence-map e*
using *assms* P_0*-props*(*2*) *B.equivalent-objects-symmetric B.equivalent-objects-def*
by *meson*
obtain $d\ \eta\ \varepsilon$ **where** d: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B $e\ d\ \eta\ \varepsilon$
using *e B.equivalence-map-extends-to-adjoint-equivalence* **by** *blast*
thus *?thesis*
using *e d* **by** *auto*
qed
hence *1*: «$e\ a : a \to_B P_0\ a$» $\wedge$
adjoint-equivalence-in-bicategory V_B H_B a_B i_B src_B trg_B $(e\ a)$ $(d\ a)$ $(\eta\ a)$ $(\varepsilon\ a)$
using *d-def e-def* η*-def* ε*-def arr-char*$_{SbC}$
someI-ex [*of* $\lambda e.$ «$e : a \to_B P_0\ a$» $\wedge$
($\exists\, d\ \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B $e\ d\ \eta\ \varepsilon$)]
someI-ex [*of* $\lambda d.$ ($\exists\, \eta\ \varepsilon.$ *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B $(e\ a)$ $d\ \eta\ \varepsilon$)]
someI-ex [*of* $\lambda\eta.$ $\exists\, \varepsilon.$ *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B $(e\ a)$ $(d\ a)$ $\eta\ \varepsilon$]
someI-ex [*of* $\lambda\varepsilon.$ *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B $(e\ a)$ $(d\ a)$ $(\eta\ a)$ ε]
by *simp*
interpret *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B ‹$e\ a$› ‹$d\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
using *1* **by** *simp*
show *adjoint-equivalence-in-bicategory*
V_B H_B a_B i_B src_B trg_B $(e\ a)$ $(d\ a)$ $(\eta\ a)$ $(\varepsilon\ a)$ **..**
show «$e\ a : a \to_B P_0\ a$» **using** *1* **by** *simp*
show $B.ide\ (d\ a)$ **by** *simp*
show $B.ide\ (e\ a)$ **by** *simp*
show $B.iso\ (\eta\ a)$ **by** *simp*
show $B.iso\ (\varepsilon\ a)$ **by** *simp*
qed

lemma *equivalence-data-in-hom*$_B$ [*intro*]:
assumes $B.obj\ a$
shows «$e\ a : a \to_B P_0\ a$» **and** «$d\ a : P_0\ a \to_B a$»
and «$e\ a : e\ a \Rightarrow_B e\ a$» **and** «$d\ a : d\ a \Rightarrow_B d\ a$»
and «$\eta\ a : a \to_B a$» **and** «$\varepsilon\ a : P_0\ a \to_B P_0\ a$»
and «$\eta\ a : a \Rightarrow_B d\ a \star_B e\ a$» **and** «$\varepsilon\ a : e\ a \star_B d\ a \Rightarrow_B P_0\ a$»
proof −
interpret *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B ‹$e\ a$› ‹$d\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
using *assms chosen-adjoint-equivalence* **by** *simp*
show e: «$e\ a : a \to_B P_0\ a$»

using *assms chosen-adjoint-equivalence* **by** *simp*
show «$d\ a : P_0\ a \rightarrow_B a$» **using** *e antipar* **by** *auto*
show «$e\ a : e\ a \Rightarrow_B e\ a$» **by** *auto*
show «$d\ a : d\ a \Rightarrow_B d\ a$» **by** *auto*
show «$\eta\ a : a \rightarrow_B a$» **using** *unit-in-hom e* **by** *auto*
show «$\varepsilon\ a : P_0\ a \rightarrow_B P_0\ a$» **using** *counit-in-hom e* **by** *auto*
show «$\eta\ a : a \Rightarrow_B d\ a \star_B e\ a$» **using** *unit-in-hom e* **by** *auto*
show «$\varepsilon\ a : e\ a \star_B d\ a \Rightarrow_B P_0\ a$» **using** *counit-in-hom e* **by** *auto*
qed

lemma *equivalence-data-simps*$_B$ [*simp*]:
assumes *B.obj a*
shows *B.ide* $(d\ a)$ **and** *B.ide* $(e\ a)$ **and** *B.iso* $(\eta\ a)$ **and** *B.iso* $(\varepsilon\ a)$
and $src_B\ (e\ a) = a$ **and** $trg_B\ (e\ a) = P_0\ a$ **and** $src_B\ (d\ a) = P_0\ a$ **and** $trg_B\ (d\ a) = a$
and *B.dom* $(e\ a) = e\ a$ **and** *B.cod* $(e\ a) = e\ a$
and *B.dom* $(d\ a) = d\ a$ **and** *B.cod* $(d\ a) = d\ a$
and $src_B\ (\eta\ a) = a$ **and** $trg_B\ (\eta\ a) = a$ **and** $src_B\ (\varepsilon\ a) = P_0\ a$ **and** $trg_B\ (\varepsilon\ a) = P_0\ a$
and *B.dom* $(\eta\ a) = a$ **and** *B.cod* $(\eta\ a) = d\ a \star_B e\ a$
and *B.dom* $(\varepsilon\ a) = e\ a \star_B d\ a$ **and** *B.cod* $(\varepsilon\ a) = P_0\ a$
using *assms chosen-adjoint-equivalence equivalence-data-in-hom*$_B$ *B.in-hhom-def*
apply *auto*
by (*meson B.in-homE*)+

lemma *equivalence-data-in-hom* [*intro*]:
assumes *obj a*
shows «$e\ a : a \rightarrow P_0\ a$» **and** «$d\ a : P_0\ a \rightarrow a$»
and «$e\ a : e\ a \Rightarrow e\ a$» **and** «$d\ a : d\ a \Rightarrow d\ a$»
and «$\eta\ a : a \rightarrow a$» **and** «$\varepsilon\ a : P_0\ a \rightarrow P_0\ a$»
and «$\eta\ a : a \Rightarrow d\ a \star e\ a$» **and** «$\varepsilon\ a : e\ a \star d\ a \Rightarrow P_0\ a$»
proof −
have *a*: $B.obj\ a \wedge a \in Obj$
using *assms obj-char arr-char*$_{SbC}$ **by** *auto*
have P_0a: $obj\ (P_0\ a) \wedge P_0\ a \in Obj$
using *assms a* P_0*-props* [*of a*] *arr-char*$_{SbC}$ *obj-char* **by** *auto*
show *ea*: «$e\ a : a \rightarrow P_0\ a$»
using *a* P_0a *arr-char*$_{SbC}$ *chosen-adjoint-equivalence*(*2*) *src-def trg-def* **by** *auto*
show «$e\ a : e\ a \Rightarrow e\ a$»
using *a ea chosen-adjoint-equivalence*(*4*) *ide-char*$_{SbC}$ *in-hom-char*$_{SbC}$ **by** *auto*
show *da*: «$d\ a : P_0\ a \rightarrow a$»
using *a* P_0a *arr-char*$_{SbC}$ *src-def trg-def* P_0*-props*(*1*) *equivalence-data-in-hom*$_B$(*2*)
by *auto*
show «$d\ a : d\ a \Rightarrow d\ a$»
using *a da chosen-adjoint-equivalence*(*3*) *ide-char*$_{SbC}$ *in-hom-char*$_{SbC}$ **by** *auto*
show ηa: «$\eta\ a : a \Rightarrow d\ a \star e\ a$»
proof
show *1*: $arr\ (\eta\ a)$
using *assms a* P_0a *da ea arr-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*7*) *hcomp-closed*
by (*metis* (*no-types*, *lifting*) *equivalence-data-in-hom*$_B$(*5*) *B.in-hhom-def*)
show $dom\ (\eta\ a) = a$

using *a 1 dom-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*7*) **by** *auto*
show *cod* (η *a*) = *d a* $\star$ *e a*
using *a 1 da ea cod-char*$_{SbC}$ *in-hom-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*7*) *hcomp-def*
by (*metis* (*no-types*, *lifting*) *hcomp-char in-hhomE B.in-homE*)
qed
show *εa*: «ε *a* : *e a* $\star$ *d a* $\Rightarrow$ P_0 *a*»
proof
show *1*: *arr* (ε *a*)
using *assms a* P_0*a da ea arr-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*8*) *hcomp-closed*
by (*metis* (*no-types*, *lifting*) *equivalence-data-in-hom*$_B$(*6*) *B.in-hhom-def*)
show *dom* (ε *a*) = *e a* $\star$ *d a*
using *a 1 da ea dom-char*$_{SbC}$ *in-hom-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*8*) *hcomp-def*
by (*metis* (*no-types*, *lifting*) *hcomp-char in-hhomE B.in-homE*)
show *cod* (ε *a*) = P_0 *a*
using *a 1 cod-char*$_{SbC}$ *equivalence-data-in-hom*$_B$(*8*) **by** *auto*
qed
show «η *a* : *a* $\rightarrow$ *a*»
using *assms ηa src-def trg-def src-dom trg-dom*
by (*metis* (*no-types*, *lifting*) *in-hhom-def in-hom-char*$_{SbC}$ *obj-simps*(*2*−*3*)
vconn-implies-hpar(*1*−*2*))
show «ε *a* : P_0 *a* $\rightarrow$ P_0 *a*»
using P_0*a εa src-def trg-def src-cod trg-cod*
by (*metis* (*no-types*, *lifting*) *in-hhom-def in-hom-char*$_{SbC}$ *obj-simps*(*2*−*3*)
vconn-implies-hpar(*1*−*4*))
qed

lemma *equivalence-data-simps* [*simp*]:
assumes *obj a*
shows *ide* (*d a*) **and** *ide* (*e a*) **and** *iso* (η *a*) **and** *iso* (ε *a*)
and *src* (*e a*) = *a* **and** *trg* (*e a*) = P_0 *a* **and** *src* (*d a*) = P_0 *a* **and** *trg* (*d a*) = *a*
and *dom* (*e a*) = *e a* **and** *cod* (*e a*) = *e a*
and *dom* (*d a*) = *d a* **and** *cod* (*d a*) = *d a*
and *src* (η *a*) = *a* **and** *trg* (η *a*) = *a* **and** *src* (ε *a*) = P_0 *a* **and** *trg* (ε *a*) = P_0 *a*
and *dom* (η *a*) = *a* **and** *cod* (η *a*) = *d a* $\star_B$ *e a*
and *dom* (ε *a*) = *e a* $\star_B$ *d a* **and** *cod* (ε *a*) = P_0 *a*

using *assms equivalence-data-in-hom*(*2*) *ide-char*$_{SbC}$ *obj-char* **apply** *auto*[*1*]
using *assms equivalence-data-in-hom*(*1*) *ide-char*$_{SbC}$ *obj-char* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *iso-char*$_{SbC}$ *B.iso-is-arr* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *iso-char*$_{SbC}$ P_0*-props B.iso-is-arr* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ P_0*-props src-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ P_0*-props trg-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ P_0*-props src-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ P_0*-props trg-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *dom-char*$_{SbC}$ P_0*-props* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *cod-char*$_{SbC}$ P_0*-props* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *dom-char*$_{SbC}$ P_0*-props* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *cod-char*$_{SbC}$ P_0*-props* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *B.iso-is-arr src-def* **apply** *auto*[*1*]

using *assms obj-char arr-char*$_{SbC}$ *B.iso-is-arr trg-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *dom-char*$_{SbC}$ P_0*-props B.iso-is-arr src-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *cod-char*$_{SbC}$ P_0*-props B.iso-is-arr trg-def* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *dom-char*$_{SbC}$ *B.iso-is-arr* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *cod-char*$_{SbC}$ *B.iso-is-arr* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *dom-char*$_{SbC}$ P_0*-props B.iso-is-arr* **apply** *auto*[*1*]
using *assms obj-char arr-char*$_{SbC}$ *cod-char*$_{SbC}$ P_0*-props B.iso-is-arr* **by** *auto*[*1*]

definition *P*
where $P\ \mu = e\ (trg_B\ \mu) \star_B \mu \star_B d\ (src_B\ \mu)$

lemma *P-in-hom*$_B$ [*intro*]:
assumes *B.arr* μ
shows «$P\ \mu : P_0\ (src_B\ \mu) \rightarrow_B P_0\ (trg_B\ \mu)$»
and «$P\ \mu : P\ (B.dom\ \mu) \Rightarrow_B P\ (B.cod\ \mu)$»
unfolding *P-def* **using** *assms* **by** *auto*

lemma *P-simps*$_B$ [*simp*]:
assumes *B.arr* μ
shows *B.arr* $(P\ \mu)$
and $src_B\ (P\ \mu) = P_0\ (src_B\ \mu)$ **and** $trg_B\ (P\ \mu) = P_0\ (trg_B\ \mu)$
and $B.dom\ (P\ \mu) = P\ (B.dom\ \mu)$ **and** $B.cod\ (P\ \mu) = P\ (B.cod\ \mu)$
using *assms P-in-hom*$_B$ **by** *blast+*

lemma *P-in-hom* [*intro*]:
assumes *B.arr* μ
shows «$P\ \mu : P_0\ (src_B\ \mu) \rightarrow P_0\ (trg_B\ \mu)$»
and «$P\ \mu : P\ (B.dom\ \mu) \Rightarrow P\ (B.cod\ \mu)$»
proof −
show *1*: «$P\ \mu : P_0\ (src_B\ \mu) \rightarrow P_0\ (trg_B\ \mu)$»
using *assms P-in-hom*$_B$(*1*) *arr-char*$_{SbC}$ *src-def trg-def* P_0*-props*(*1*)
by (*metis* (*no-types*, *lifting*) *in-hhom-def obj-char B.in-hhomE B.obj-simps*(*2*)
B.obj-src B.obj-trg)
show «$P\ \mu : P\ (B.dom\ \mu) \Rightarrow P\ (B.cod\ \mu)$»
using *assms 1 dom-char*$_{SbC}$ *cod-char*$_{SbC}$
by (*intro in-homI*) *auto*
qed

lemma *P-simps* [*simp*]:
assumes *B.arr* μ
shows *arr* $(P\ \mu)$
and $src\ (P\ \mu) = P_0\ (src_B\ \mu)$ **and** $trg\ (P\ \mu) = P_0\ (trg_B\ \mu)$
and $dom\ (P\ \mu) = P\ (B.dom\ \mu)$ **and** $cod\ (P\ \mu) = P\ (B.cod\ \mu)$
using *assms P-in-hom* **by** *blast+*

interpretation *P*: *functor* V_B *comp P*
proof
show $\bigwedge\mu.\ \neg\ B.arr\ \mu \Longrightarrow P\ \mu = null$
using *P-def null-char B.hseq-char B.hseq-char′* **by** *auto*

```
  have 0: ⋀μ. B.arr μ ⟹ B.arr (P μ)
    by simp
  show 1: ⋀μ. B.arr μ ⟹ arr (P μ)
    using P-simps_B(1) by simp
  show 2: ⋀μ. B.arr μ ⟹ dom (P μ) = P (B.dom μ)
    using 1 dom-simp by auto
  show 3: ⋀μ. B.arr μ ⟹ cod (P μ) = P (B.cod μ)
    using 1 cod-simp by auto
  show ⋀μ ν. B.seq μ ν ⟹ P (μ ·_B ν) = P μ · P ν
  proof −
    fix μ ν
    assume seq: B.seq μ ν
    show P (μ ·_B ν) = P μ · P ν
    proof −
      have 4: P (μ ·_B ν) = e (trg_B μ) ⋆_B μ ·_B ν ⋆_B d (src_B μ)
        unfolding P-def using seq B.src-vcomp B.trg-vcomp by simp
      also have ... = P μ ·_B P ν
        unfolding P-def
        using seq 0 4 B.whisker-left B.whisker-right
        by (metis equivalence-data-simps_B(1−2) B.hseqE B.obj-trg B.seqE B.src-vcomp
            B.vseq-implies-hpar(2))
      also have ... = P μ · P ν
        using seq 1 seq-char_SbC comp-char by auto
      finally show ?thesis by blast
    qed
  qed
qed

interpretation faithful-functor V_B comp P
proof
  fix μ μ'
  assume par: B.par μ μ'
  have 1: src_B μ = src_B μ' ∧ trg_B μ = trg_B μ'
    using par by (metis B.src-dom B.trg-dom)
  assume eq: P μ = P μ'
  interpret src: adjoint-equivalence-in-bicategory V_B H_B a_B i_B src_B trg_B
                 ‹e (src_B μ)› ‹d (src_B μ)› ‹η (src_B μ)› ‹ε (src_B μ)›
    using par chosen-adjoint-equivalence by simp
  interpret trg: adjoint-equivalence-in-bicategory V_B H_B a_B i_B src_B trg_B
                 ‹e (trg_B μ)› ‹d (trg_B μ)› ‹η (trg_B μ)› ‹ε (trg_B μ)›
    using par chosen-adjoint-equivalence by simp
  show μ = μ'
    using eq par 1
    unfolding P-def
    using B.equivalence-cancel-left [of e (trg_B μ) μ ⋆_B d (src_B μ) μ' ⋆_B d (src_B μ')]
          B.equivalence-cancel-right [of d (src_B μ) μ μ']
          B.equivalence-map-def src.dual-equivalence trg.equivalence-in-bicategory-axioms
    by auto
qed
```

interpretation P: *weak-arrow-of-homs* V_B src_B trg_B *comp src trg* P
proof
  fix μ
  assume μ: $B.arr\ \mu$
  interpret *src*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
    ‹$e\ (src_B\ \mu)$› ‹$d\ (src_B\ \mu)$› ‹$\eta\ (src_B\ \mu)$› ‹$\varepsilon\ (src_B\ \mu)$›
    using μ *chosen-adjoint-equivalence* **by** *simp*
  show *isomorphic* $(P\ (src_B\ \mu))\ (src\ (P\ \mu))$
  proof (*unfold isomorphic-def*)
    show $\exists f.$ «$f : P\ (src_B\ \mu) \Rightarrow src\ (P\ \mu)$» $\wedge$ *iso* f
    proof −
      let $?\varphi = \varepsilon\ (src_B\ \mu) \cdot_B (e\ (src_B\ \mu) \star_B \mathrm{l}_B[d\ (src_B\ \mu)])$
      have «$?\varphi : P\ (src_B\ \mu) \Rightarrow src\ (P\ \mu)$» $\wedge$ *iso* $?\varphi$
      proof −
        have *1*: «$?\varphi : P\ (src_B\ \mu) \Rightarrow_B P_0\ (src_B\ \mu)$» $\wedge$ $B.iso\ ?\varphi$
          unfolding *P-def* **using** μ **by** *auto*
        moreover have $arr\ ?\varphi \wedge arr\ (B.inv\ ?\varphi)$
          using μ *1* $arr\text{-}char_{SbC}$ P_0*-props*(*1*)
        by (*metis* (*no-types*, *lifting*) P_0*-props*(*6*) *B.arrI B.arr-inv* $equivalence\text{-}data\text{-}simps_B(16)$
            B.obj-src src.counit-simps(*4*−*5*) *B.src-inv B.src-vcomp*
            B.trg-inv B.trg-vcomp)
        moreover have $P_0\ (src_B\ \mu) = src\ (P\ \mu)$
          using μ **by** *simp*
        ultimately show *?thesis*
          using μ $in\text{-}hom\text{-}char_{SbC}$ $arr\text{-}char_{SbC}$ $iso\text{-}char_{SbC}$
          by (*metis* (*no-types*, *lifting*) *src-closed* $equivalence\text{-}data\text{-}simps_B(6)$
            B.obj-src P.preserves-arr src.counit-simps(*4*) *B.src.preserves-arr B.src-vcomp*)
      qed
      thus *?thesis* **by** *blast*
    qed
  qed
  interpret *trg*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
    ‹$e\ (trg_B\ \mu)$› ‹$d\ (trg_B\ \mu)$› ‹$\eta\ (trg_B\ \mu)$› ‹$\varepsilon\ (trg_B\ \mu)$›
    using μ *chosen-adjoint-equivalence* **by** *simp*
  show *isomorphic* $(P\ (trg_B\ \mu))\ (trg\ (P\ \mu))$
  proof (*unfold isomorphic-def*)
    show $\exists f.$ «$f : P\ (trg_B\ \mu) \Rightarrow trg\ (P\ \mu)$» $\wedge$ *iso* f
    proof −
      let $?\psi = \varepsilon\ (trg_B\ \mu) \cdot_B (e\ (trg_B\ \mu) \star_B \mathrm{l}_B[d\ (trg_B\ \mu)])$
      have «$?\psi : P\ (trg_B\ \mu) \Rightarrow trg\ (P\ \mu)$» $\wedge$ *iso* $?\psi$
      proof −
        have *1*: «$?\psi : P\ (trg_B\ \mu) \Rightarrow_B P_0\ (trg_B\ \mu)$» $\wedge$ $B.iso\ ?\psi$
          unfolding *P-def* **using** μ **by** *auto*
        moreover have $arr\ ?\psi \wedge arr\ (B.inv\ ?\psi)$
          using μ *1* $arr\text{-}char_{SbC}$ P_0*-props*(*1*)
          by (*metis* (*no-types*, *lifting*) *obj-char B.arrI B.arr-inv B.obj-trg B.src-inv B.trg-inv*
            B.vconn-implies-hpar(*1*−*4*))
        moreover have $arr\ (P\ (trg_B\ \mu)) \wedge arr\ (P_0\ (trg_B\ \mu))$

using μ *arr-char*$_{SbC}$ P_0*-props*(*1*) *P-simps*(*1*) *obj-char* **by** *auto*
moreover have P_0 (trg_B μ) = *trg* (P μ)
using μ **by** *simp*
ultimately show *?thesis*
using *in-hom-char*$_{SbC}$ *iso-char*$_{SbC}$ **by** *force*
qed
thus *?thesis* **by** *blast*
qed
qed
qed

The following seems to be needed to avoid non-confluent simplifications, *e.g.* of $S.src$ (P μ) to $P.map_0$ a and to P_0 a.

lemma *P-map*$_0$*-simp* [*simp*]:
assumes *B.obj a*
shows $P.map_0$ $a = P_0$ a
using *assms P.map*$_0$*-def B.obj-simps*(*1*−*2*) **by** *simp*

interpretation *HoPP*: *composite-functor B.VV.comp VV.comp comp*
P.FF ‹$\lambda\mu\nu$. *hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
..
interpretation *PoH*: *composite-functor B.VV.comp* V_B *comp* ‹($\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$)› P
..

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

definition *CMP*
where *CMP f g* ≡ (e (trg_B f) $\star_B$ $\mathrm{a}_B{}^{-1}$[f, g, d (src_B g)]) $\cdot_B$
(e (trg_B f) $\star_B$ f $\star_B$
l_B[g $\star_B$ d (src_B g)] $\cdot_B$ (*B.inv* (η (trg_B g)) $\star_B$ g $\star_B$ d (src_B g))) $\cdot_B$
(e (trg_B f) $\star_B$ f $\star_B$ $\mathrm{a}_B{}^{-1}$[d (src_B f), e (trg_B g), g $\star_B$ d (src_B g)]) $\cdot_B$
a_B[e (trg_B f), f, d (src_B f) $\star_B$ P g] $\cdot_B$
a_B[e (trg_B f) $\star_B$ f, d (src_B f), P g] $\cdot_B$
($\mathrm{a}_B{}^{-1}$[e (trg_B f), f, d (src_B f)] $\star_B$ P g)

The 2-cell *CMP f g* has the right type to be a compositor for a pseudofunctor whose underlying mapping is P.

lemma *CMP-in-hom* [*intro*]:
assumes *B.ide f* **and** *B.ide g* **and** src_B $f = trg_B$ g
shows «*CMP f g* : P_0 (src_B g) → P_0 (trg_B f)»
and «*CMP f g* : P f $\star$ P g ⇒ P (f $\star_B$ g)»
and «*CMP f g* : P_0 (src_B g) $\rightarrow_B$ P_0 (trg_B f)»
and «*CMP f g* : P f $\star_B$ P g $\Rightarrow_B$ P (f $\star_B$ g)»
proof −
show *1*: «*CMP f g* : P f $\star_B$ P g $\Rightarrow_B$ P (f $\star_B$ g)»
using *assms*
by (*unfold P-def CMP-def*, *intro B.comp-in-homI′ B.seqI B.hseqI′*) *auto*
show *2*: «*CMP f g* : P_0 (src_B g) $\rightarrow_B$ P_0 (trg_B f)»
using *assms 1 B.src-cod B.trg-cod B.vconn-implies-hpar*(*1*−*2*) **by** *auto*

show *3*: «$CMP\ f\ g : P\ f \star P\ g \Rightarrow P\ (f \star_B g)$»
using *assms 1 B.arrI B.vconn-implies-hpar(1−2) P-simps(1) arr-char$_{SbC}$ hcomp-eqI hseqI′ in-hom-char$_{SbC}$*
by *force*
show «$CMP\ f\ g : P_0\ (src_B\ g) \rightarrow P_0\ (trg_B\ f)$»
using *2 3 arr-char$_{SbC}$ src-def trg-def* **by** *fastforce*
qed

lemma *CMP-simps* [*simp*]:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $arr\ (CMP\ f\ g)$
and $src\ (CMP\ f\ g) = P_0\ (src_B\ g)$ **and** $trg\ (CMP\ f\ g) = P_0\ (trg_B\ f)$
and $dom\ (CMP\ f\ g) = P\ f \star P\ g$ **and** $cod\ (CMP\ f\ g) = P\ (f \star_B g)$
and $B.arr\ (CMP\ f\ g)$
and $src_B\ (CMP\ f\ g) = P_0\ (src_B\ g)$ **and** $trg_B\ (CMP\ f\ g) = P_0\ (trg_B\ f)$
and $B.dom\ (CMP\ f\ g) = P\ f \star_B P\ g$ **and** $B.cod\ (CMP\ f\ g) = P\ (f \star_B g)$
using *assms CMP-in-hom* [*of f g*] **by** *auto*

lemma *iso-CMP*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $iso\ (CMP\ f\ g)$
and $B.iso\ (CMP\ f\ g)$
proof −
show $B.iso\ (CMP\ f\ g)$
unfolding *CMP-def P-def*
using *assms B.VV.arr-char$_{SbC}$ P.as-nat-iso.components-are-iso*
by (*intro B.isos-compose B.iso-hcomp*) *auto*
thus $iso\ (CMP\ f\ g)$
using *assms iso-char$_{SbC}$ arr-char$_{SbC}$ CMP-simps(1)* **by** *auto*
qed

abbreviation (*input*) *SRC*
where $SRC\ \mu \equiv d\ (src_B\ \mu) \star_B e\ (src_B\ \mu)$

abbreviation (*input*) *TRG*
where $TRG\ \mu \equiv d\ (trg_B\ \mu) \star_B e\ (trg_B\ \mu)$

definition *LUNIT*
where $LUNIT\ f \equiv \mathrm{l}_B[f] \cdot_B (B.inv\ (\eta\ (trg_B\ f)) \star_B f)$

definition *RUNIT*
where $RUNIT\ f \equiv \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ (src_B\ f)))$

Here we prove a series of results that would be automatic if we had some notion of "bicategory with *SRC* and *TRG* as alternative source and target". Perhaps this idea can be developed in future work and used to simplify the overall development.

lemma *LUNIT-in-hom* [*intro*]:
assumes $B.ide\ f$
shows «$LUNIT\ f : src_B\ f \rightarrow_B trg_B\ f$»

and «$LUNIT\ f$: $TRG\ f \star_B f \Rightarrow_B f$»
proof −
  interpret *e-trg-f*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
    ‹$e\ (trg_B\ f)$› ‹$d\ (trg_B\ f)$› ‹$\eta\ (trg_B\ f)$› ‹$\varepsilon\ (trg_B\ f)$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show «$LUNIT\ f$: $src_B\ f \rightarrow_B trg_B\ f$»
    unfolding *LUNIT-def*
    using *assms e-trg-f.unit-is-iso* **by** *auto*
  show «$LUNIT\ f$: $TRG\ f \star_B f \Rightarrow_B f$»
    unfolding *LUNIT-def*
    using *assms e-trg-f.unit-is-iso* **by** *auto*
qed

lemma *LUNIT-simps* [*simp*]:
assumes $B.ide\ f$
shows $B.arr\ (LUNIT\ f)$
and $src_B\ (LUNIT\ f) = src_B\ f$ **and** $trg_B\ (LUNIT\ f) = trg_B\ f$
and $B.dom\ (LUNIT\ f) = TRG\ f \star_B f$
and $B.cod\ (LUNIT\ f) = f$
  using *assms LUNIT-in-hom* **by** *auto*

lemma *RUNIT-in-hom* [*intro*]:
assumes $B.ide\ f$
shows «$RUNIT\ f$: $src_B\ f \rightarrow_B trg_B\ f$»
and «$RUNIT\ f$: $f \star_B SRC\ f \Rightarrow_B f$»
proof −
  interpret *e-src-f*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
    ‹$e\ (src_B\ f)$› ‹$d\ (src_B\ f)$› ‹$\eta\ (src_B\ f)$› ‹$\varepsilon\ (src_B\ f)$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show «$RUNIT\ f$: $src_B\ f \rightarrow_B trg_B\ f$»
    unfolding *RUNIT-def*
    using *assms e-src-f.unit-is-iso* **by** *auto*
  show «$RUNIT\ f$: $f \star_B SRC\ f \Rightarrow_B f$»
    unfolding *RUNIT-def*
    using *assms e-src-f.unit-is-iso* **by** *auto*
qed

lemma *RUNIT-simps* [*simp*]:
assumes $B.ide\ f$
shows $B.arr\ (RUNIT\ f)$
and $src_B\ (RUNIT\ f) = src_B\ f$ **and** $trg_B\ (RUNIT\ f) = trg_B\ f$
and $B.dom\ (RUNIT\ f) = f \star_B SRC\ f$
and $B.cod\ (RUNIT\ f) = f$
  using *assms RUNIT-in-hom* **by** *auto*

lemma *iso-LUNIT*:
assumes $B.ide\ f$
shows $B.iso\ (LUNIT\ f)$
  by (*simp add*: *assms B.isos-compose LUNIT-def*)

lemma *iso-RUNIT*:
assumes $B.ide\ f$
shows $B.iso\ (RUNIT\ f)$
  by (*simp add*: *assms B.isos-compose RUNIT-def*)

lemma *LUNIT-naturality*:
assumes $B.arr\ \mu$
shows $\mu \cdot_B LUNIT\ (B.dom\ \mu) = LUNIT\ (B.cod\ \mu) \cdot_B (TRG\ \mu \star_B \mu)$
proof −
  interpret *e-trg-μ*: *adjoint-equivalence-in-bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
                ‹$e\ (trg_B\ \mu)$› ‹$d\ (trg_B\ \mu)$› ‹$\eta\ (trg_B\ \mu)$› ‹$\varepsilon\ (trg_B\ \mu)$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *?thesis*
  proof −
    have $\mu \cdot_B LUNIT\ (B.dom\ \mu) = (\mu \cdot_B \mathrm{l}_B[B.dom\ \mu]) \cdot_B (B.inv\ (\eta\ (trg_B\ \mu)) \star_B B.dom\ \mu)$
      unfolding *LUNIT-def*
      using *assms B.comp-assoc* **by** *simp*
    also have ... $= \mathrm{l}_B[B.cod\ \mu] \cdot_B (trg_B\ \mu \star_B \mu) \cdot_B (B.inv\ (\eta\ (trg_B\ \mu)) \star_B B.dom\ \mu)$
      using *assms B.lunit-naturality B.comp-assoc* **by** *simp*
    also have ... $= \mathrm{l}_B[B.cod\ \mu] \cdot_B (B.inv\ (\eta\ (trg_B\ \mu)) \star_B \mu)$
      using *assms B.interchange* [*of* $trg_B\ \mu\ B.inv\ (\eta\ (trg_B\ \mu))\ \mu\ B.dom\ \mu$]
          e-trg-μ.unit-is-iso B.comp-arr-dom B.comp-cod-arr **by** *simp*
    also have ... $= \mathrm{l}_B[B.cod\ \mu] \cdot_B (B.inv\ (\eta\ (trg_B\ \mu)) \star_B B.cod\ \mu) \cdot_B (TRG\ \mu \star_B \mu)$
      using *assms B.interchange* [*of* $B.inv\ (\eta\ (trg_B\ \mu))\ TRG\ \mu\ B.cod\ \mu\ \mu$]
          e-trg-μ.unit-is-iso B.comp-arr-dom B.comp-cod-arr **by** *simp*
    also have ... $= LUNIT\ (B.cod\ \mu) \cdot_B (TRG\ \mu \star_B \mu)$
      unfolding *LUNIT-def*
      using *assms B.comp-assoc* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
qed

lemma *RUNIT-naturality*:
assumes $B.arr\ \mu$
shows $\mu \cdot_B RUNIT\ (B.dom\ \mu) = RUNIT\ (B.cod\ \mu) \cdot_B (\mu \star_B SRC\ \mu)$
proof −
  interpret *e-src-μ*: *adjoint-equivalence-in-bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
                ‹$e\ (src_B\ \mu)$› ‹$d\ (src_B\ \mu)$› ‹$\eta\ (src_B\ \mu)$› ‹$\varepsilon\ (src_B\ \mu)$›
    using *assms chosen-adjoint-equivalence* **by** *simp*
  show *?thesis*
  proof −
    have $\mu \cdot_B RUNIT\ (B.dom\ \mu) =$
        $(\mu \cdot_B \mathrm{r}_B[B.dom\ \mu]) \cdot_B (B.dom\ \mu \star_B B.inv\ (\eta\ (src_B\ \mu)))$
      unfolding *RUNIT-def*
      using *assms B.comp-assoc* **by** *simp*
    also have ... $= \mathrm{r}_B[B.cod\ \mu] \cdot_B (\mu \star_B src_B\ \mu) \cdot_B (B.dom\ \mu \star_B B.inv\ (\eta\ (src_B\ \mu)))$
      using *assms B.runit-naturality B.comp-assoc* **by** *simp*
    also have ... $= \mathrm{r}_B[B.cod\ \mu] \cdot_B (\mu \star_B B.inv\ (\eta\ (src_B\ \mu)))$

using *assms B.interchange* [*of* μ *B.dom* μ src_B μ *B.inv* (η (src_B μ))]
e-src-μ.unit-is-iso B.comp-arr-dom B.comp-cod-arr **by** *simp*
also have ... = r_B[*B.cod* μ] $\cdot_B$ (*B.cod* μ $\star_B$ *B.inv* (η (src_B μ))) $\cdot_B$ (μ $\star_B$ *SRC* μ)
using *assms B.interchange* [*of B.cod* μ μ *B.inv* (η (src_B μ)) *SRC* μ]
e-src-μ.unit-is-iso B.comp-arr-dom B.comp-cod-arr
by *simp*
also have ... = *RUNIT* (*B.cod* μ) $\cdot_B$ (μ $\star_B$ *SRC* μ)
unfolding *RUNIT-def*
using *assms B.comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *LUNIT-hcomp*:
assumes *B.ide f* **and** *B.ide g* **and** src_B f = trg_B g
shows *LUNIT* (f $\star_B$ g) $\cdot_B$ a_B[d (trg_B f) $\star_B$ e (trg_B f), f, g] = *LUNIT* f $\star_B$ g
proof −
interpret *e-trg-f*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
‹e (trg_B f)› ‹d (trg_B f)› ‹η (trg_B f)› ‹ε (trg_B f)›
using *assms chosen-adjoint-equivalence* **by** *simp*
have *LUNIT* (f $\star_B$ g) $\cdot_B$ a_B[*TRG* f, f, g] =
l_B[f $\star_B$ g] $\cdot_B$ (*B.inv* (η (trg_B f)) $\star_B$ f $\star_B$ g) $\cdot_B$ a_B[*TRG* f, f, g]
unfolding *LUNIT-def*
using *assms B.comp-assoc* **by** *simp*
also have ... = (l_B[f $\star_B$ g] $\cdot_B$ a_B[trg_B f, f, g]) $\cdot_B$ ((*B.inv* (η (trg_B f)) $\star_B$ f) $\star_B$ g)
using *assms B.assoc-naturality* [*of B.inv* (η (trg_B f)) f g] *e-trg-f.unit-is-iso*
B.comp-assoc
by *simp*
also have ... = (l_B[f] $\star_B$ g) $\cdot_B$ ((*B.inv* (η (trg_B f)) $\star_B$ f) $\star_B$ g)
using *assms B.lunit-hcomp* **by** *simp*
also have ... = *LUNIT* f $\star_B$ g
using *assms LUNIT-def LUNIT-simps*(*1*) *B.whisker-right* **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *RUNIT-hcomp*:
assumes *B.ide f* **and** *B.ide g* **and** src_B f = trg_B g
shows *RUNIT* (f $\star_B$ g) = (f $\star_B$ *RUNIT* g) $\cdot_B$ a_B[f, g, *SRC* g]
proof −
interpret *e-src-g*: *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B
‹e (src_B g)› ‹d (src_B g)› ‹η (src_B g)› ‹ε (src_B g)›
using *assms chosen-adjoint-equivalence* **by** *simp*
have (f $\star_B$ *RUNIT* g) $\cdot_B$ a_B[f, g, *SRC* g] =
(f $\star_B$ r_B[g]) $\cdot_B$ (f $\star_B$ g $\star_B$ *B.inv* (η (src_B g))) $\cdot_B$ a_B[f, g, *SRC* g]
unfolding *RUNIT-def*
using *assms B.whisker-left e-src-g.unit-is-iso B.comp-assoc* **by** *simp*
also have ... = ((f $\star_B$ r_B[g]) $\cdot_B$ a_B[f, g, src_B g]) $\cdot_B$ ((f $\star_B$ g) $\star_B$ *B.inv* (η (src_B g)))
using *assms B.assoc-naturality* [*of f g B.inv* (η (src_B g))] *e-src-g.unit-is-iso*
B.comp-assoc

by *simp*
also have ... = $\mathrm{r}_B[f \star_B g] \cdot_B ((f \star_B g) \star_B B.inv\ (\eta\ (src_B\ g)))$
using *assms B.runit-hcomp* **by** *simp*
also have ... = $RUNIT\ (f \star_B g)$
using *assms RUNIT-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *TRIANGLE*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $(f \star_B LUNIT\ g) \cdot_B \mathrm{a}_B[f,\ SRC\ f,\ g] = RUNIT\ f \star_B g$
proof −
interpret *e-trg-g*: *adjoint-equivalence-in-bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
‹$e\ (trg_B\ g)$› ‹$d\ (trg_B\ g)$› ‹$\eta\ (trg_B\ g)$› ‹$\varepsilon\ (trg_B\ g)$›
using *assms chosen-adjoint-equivalence* **by** *simp*
show *?thesis*
proof −
have $(f \star_B LUNIT\ g) \cdot_B \mathrm{a}_B[f,\ SRC\ f,\ g] =$
$(f \star_B \mathrm{l}_B[g]) \cdot_B (f \star_B B.inv\ (\eta\ (trg_B\ g)) \star_B g) \cdot_B \mathrm{a}_B[f,\ SRC\ f,\ g]$
using *assms B.whisker-left e-trg-g.unit-is-iso LUNIT-def LUNIT-simps*(*1*) *B.comp-assoc*
by *auto*
also have ... = $((f \star_B \mathrm{l}_B[g]) \cdot_B \mathrm{a}_B[f,\ src_B\ f,\ g]) \cdot_B ((f \star_B B.inv\ (\eta\ (trg_B\ g))) \star_B g)$
using *assms B.assoc-naturality* [*of f B.inv* $(\eta\ (trg_B\ g))$ *g*] *e-trg-g.unit-is-iso*
B.comp-assoc
by *auto*
also have ... = $(\mathrm{r}_B[f] \star_B g) \cdot_B ((f \star_B B.inv\ (\eta\ (trg_B\ g))) \star_B g)$
using *assms B.triangle* **by** *simp*
also have ... = $RUNIT\ f \star_B g$
using *assms B.whisker-right e-trg-g.unit-is-iso RUNIT-def RUNIT-simps*
by *metis*
finally show *?thesis* **by** *simp*
qed
qed

The *CMP f g* also satisfy the naturality conditions required of compositors.

lemma *CMP-naturality*:
assumes $B.arr\ \mu$ **and** $B.arr\ \nu$ **and** $src_B\ \mu = trg_B\ \nu$
shows $CMP\ (B.cod\ \mu)\ (B.cod\ \nu) \cdot_B (P\ \mu \star_B P\ \nu)$
$= P\ (\mu \star_B \nu) \cdot_B CMP\ (B.dom\ \mu)\ (B.dom\ \nu)$
proof −
let $?a = src_B\ \nu$
let $?b = src_B\ \mu$
let $?c = trg_B\ \mu$
let $?f = B.dom\ \mu$
let $?g = B.cod\ \mu$
let $?h = B.dom\ \nu$
let $?k = B.cod\ \nu$
have $CMP\ ?g\ ?k \cdot_B (P\ \mu \star_B P\ \nu)$
$= (e\ ?c \star_B \mathrm{a}_B^{-1}[?g,\ ?k,\ d\ ?a]) \cdot_B$

$(e\ ?c \star_B\ ?g \star_B\ LUNIT\ (?k \star_B\ d\ ?a)) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ?k \star_B\ d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ ?g,\ d\ ?b \star_B\ P\ ?k] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B\ ?g,\ d\ ?b,\ P\ ?k] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?g,\ d\ ?b] \star_B\ P\ ?k) \cdot_B$
$(P\ \mu \star_B\ P\ \nu)$
unfolding *CMP-def LUNIT-def* **using** *assms B.comp-assoc* **by** *simp*
also have ... $= (e\ ?c \star_B\ \mathrm{a}_B{}^{-1}[?g,\ ?k,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ LUNIT\ (?k \star_B\ d\ ?a)) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ?k \star_B\ d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ ?g,\ d\ ?b \star_B\ P\ ?k] \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B\ ?g,\ d\ ?b,\ P\ ?k] \cdot_B$
$(((e\ ?c \star_B\ \mu) \star_B\ d\ ?b) \star_B\ P\ \nu)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?h)$
proof −
have $(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?g,\ d\ ?b] \star_B\ P\ ?k) \cdot_B\ (P\ \mu \star_B\ P\ \nu)$
$= \mathrm{a}_B{}^{-1}[e\ ?c,\ ?g,\ d\ ?b] \cdot_B\ P\ \mu \star_B\ P\ ?k \cdot_B\ P\ \nu$
using *assms P-def B.interchange* **by** *fastforce*
also have
... $= ((e\ ?c \star_B\ \mu) \star_B\ d\ ?b) \cdot_B\ \mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?k \cdot_B\ P\ \nu$
using *assms P-def B.assoc′-naturality* [*of e ?c μ d ?b*] **by** *simp*
also have
... $= ((e\ ?c \star_B\ \mu) \star_B\ d\ ?b) \cdot_B\ \mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ \nu \cdot_B\ P\ ?h$
using *assms B.comp-arr-dom B.comp-cod-arr B.src-cod B.src-dom B.trg-cod B.trg-dom P.as-nat-trans.naturality*
by *simp*
also have ... $= (((e\ ?c \star_B\ \mu) \star_B\ d\ ?b) \star_B\ P\ \nu) \cdot_B\ (\mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?h)$
using *assms B.interchange* **by** *auto*
finally have $(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?g,\ d\ ?b] \star_B\ P\ ?k) \cdot_B\ (P\ \mu \star_B\ P\ \nu)$
$= (((e\ ?c \star_B\ \mu) \star_B\ d\ ?b) \star_B\ P\ \nu) \cdot_B\ (\mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?h)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B\ \mathrm{a}_B{}^{-1}[?g,\ ?k,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ LUNIT\ (?k \star_B\ d\ ?a)) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ?k \star_B\ d\ ?a]) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ ?g,\ d\ ?b \star_B\ P\ ?k] \cdot_B$
$((e\ ?c \star_B\ \mu) \star_B\ d\ ?b \star_B\ P\ \nu)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B\ ?f,\ d\ ?b,\ P\ ?h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?h)$
using *assms B.assoc-naturality* [*of e ?c* $\star_B$ *μ d ?b P ν*] *B.comp-assoc* **by** *simp*
also have ... $= (e\ ?c \star_B\ \mathrm{a}_B{}^{-1}[?g,\ ?k,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B\ ?g \star_B\ LUNIT\ (?k \star_B\ d\ ?a)) \cdot_B$
$((e\ ?c \star_B\ ?g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ?k \star_B\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B\ \mu \star_B\ d\ ?b \star_B\ P\ \nu)) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ ?f,\ d\ ?b \star_B\ P\ ?h] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B\ ?f,\ d\ ?b,\ P\ ?h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ ?f,\ d\ ?b] \star_B\ P\ ?h)$

using *assms B.assoc-naturality* [*of e ?c* μ *d ?b* $\star_B$ *P* ν] *B.comp-assoc* **by** *simp*
also have ... = (*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?g*, *?k*, *d ?a*]) $\cdot_B$
((*e ?c* $\star_B$ *?g* $\star_B$ *LUNIT* (*?k* $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ μ $\star_B$ *SRC* μ $\star_B$ ν $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*]) $\cdot_B$
a_B[*e ?c*, *?f*, *d ?b* $\star_B$ *P ?h*] $\cdot_B$
a_B[*e ?c* $\star_B$ *?f*, *d ?b*, *P ?h*] $\cdot_B$
($\mathrm{a}_B{}^{-1}$[*e ?c*, *?f*, *d ?b*] $\star_B$ *P ?h*)
proof −
have
(*e ?c* $\star_B$ *?g* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?k* $\star_B$ *d ?a*]) $\cdot_B$ (*e ?c* $\star_B$ μ $\star_B$ *d ?b* $\star_B$ *P* ν)
= (*e ?c* $\star_B$ μ $\star_B$ *TRG* ν $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*])
proof −
have (*e ?c* $\star_B$ *?g* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?k* $\star_B$ *d ?a*]) $\cdot_B$ (*e ?c* $\star_B$ μ $\star_B$ *d ?b* $\star_B$ *P* ν)
= *e ?c* $\star_B$ (*?g* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?k* $\star_B$ *d ?a*]) $\cdot_B$ (μ $\star_B$ *d ?b* $\star_B$ *P* ν)
using *assms P-def B.whisker-left* **by** *simp*
also have ... = *e ?c* $\star_B$ μ $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?k* $\star_B$ *d ?a*] $\cdot_B$ (*d ?b* $\star_B$ *P* ν)
using *assms P-def B.comp-cod-arr*
B.interchange [*of ?g* μ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?k* $\star_B$ *d ?a*] *d ?b* $\star_B$ *P* ν]
by *simp*
also have ... = *e ?c* $\star_B$ μ $\star_B$
(*TRG* ν $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*]
using *assms P-def B.assoc′-naturality* [*of d ?b e ?b* ν $\star_B$ *d ?a*]
by *simp*
also have ... = *e ?c* $\star_B$
(μ $\star_B$ *TRG* ν $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$ (*?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*])
using *assms B.comp-arr-dom*
B.interchange [*of* μ *?f TRG* ν $\star_B$ ν $\star_B$ *d ?a*$\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*]]
by *fastforce*
also have
... = (*e ?c* $\star_B$ μ $\star_B$ *TRG* ν $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*])
using *assms B.whisker-left* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... = ((*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?g*, *?k*, *d ?a*]) $\cdot_B$
(*e ?c* $\star_B$ μ $\star_B$ ν $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ *LUNIT* (*?h* $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*]) $\cdot_B$
a_B[*e ?c*, *?f*, *d ?b* $\star_B$ *P ?h*] $\cdot_B$
a_B[*e ?c* $\star_B$ *?f*, *d ?b*, *P ?h*] $\cdot_B$
($\mathrm{a}_B{}^{-1}$[*e ?c*, *?f*, *d ?b*] $\star_B$ *P ?h*)
proof −
have ((*e ?c* $\star_B$ *?g* $\star_B$ *LUNIT* (*?k* $\star_B$ *d ?a*)) $\cdot_B$ (*e ?c* $\star_B$ μ $\star_B$ *TRG* ν $\star_B$ ν $\star_B$ *d ?a*))
= *e ?c* $\star_B$ μ $\star_B$ *LUNIT* (*B.cod* (ν $\star_B$ *d ?a*)) $\cdot_B$ ((*d ?b* $\star_B$ *e ?b*) $\star_B$ ν $\star_B$ *d ?a*)

using *assms comp-arr-dom B.comp-cod-arr B.whisker-left*
B.interchange [*of ?g* μ *LUNIT* (*?k* $\star_B$ *d ?a*) (*d ?b* $\star_B$ *e ?b*) $\star_B$ ν $\star_B$ *d ?a*]
by *fastforce*
also have ... = *e ?c* $\star_B$ μ $\star_B$ (ν $\star_B$ *d ?a*) $\cdot_B$ *LUNIT* (*?h* $\star_B$ *d ?a*)
using *assms LUNIT-naturality* [*of* ν $\star_B$ *d ?a*] **by** *simp*
also have ... = (*e ?c* $\star_B$ μ $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$ (*e ?c* $\star_B$ *?f* $\star_B$ *LUNIT* (*?h* $\star_B$ *d ?a*))
using *assms B.comp-arr-dom B.comp-cod-arr B.whisker-left*
B.interchange [*of* μ *?f* ν $\star_B$ *d ?a LUNIT* (*?h* $\star_B$ *d ?a*)]
by *simp*
finally have ((*e ?c* $\star_B$ *?g* $\star_B$ *LUNIT* (*?k* $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ μ $\star_B$ *TRG* ν $\star_B$ ν $\star_B$ *d ?a*))
= (*e ?c* $\star_B$ μ $\star_B$ ν $\star_B$ *d ?a*) $\cdot_B$ (*e ?c* $\star_B$ *?f* $\star_B$ *LUNIT* (*?h* $\star_B$ *d ?a*))
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... = *P* (μ $\star_B$ ν) $\cdot_B$
(*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?f*, *?h*, *d ?a*]) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ *LUNIT* (*?h* $\star_B$ *d ?a*)) $\cdot_B$
(*e ?c* $\star_B$ *?f* $\star_B$ $\mathrm{a}_B{}^{-1}$[*d ?b*, *e ?b*, *?h* $\star_B$ *d ?a*]) $\cdot_B$
a_B[*e ?c*, *?f*, *d ?b* $\star_B$ *P ?h*] $\cdot_B$
a_B[*e ?c* $\star_B$ *?f*, *d ?b*, *P ?h*] $\cdot_B$
($\mathrm{a}_B{}^{-1}$[*e ?c*, *?f*, *d ?b*] $\star_B$ *P ?h*)
proof −
have (*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?g*, *?k*, *d ?a*]) $\cdot_B$ (*e ?c* $\star_B$ μ $\star_B$ ν $\star_B$ *d ?a*)
= *P* (μ $\star_B$ ν) $\cdot_B$ (*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?f*, *?h*, *d ?a*])
proof −
have (*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?g*, *?k*, *d ?a*]) $\cdot_B$ (*e ?c* $\star_B$ μ $\star_B$ ν $\star_B$ *d ?a*)
= *e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?g*, *?k*, *d ?a*] $\cdot_B$ (μ $\star_B$ ν $\star_B$ *d ?a*)
using *assms B.whisker-left* **by** *simp*
also have ... = *e ?c* $\star_B$ ((μ $\star_B$ ν) $\star_B$ *d ?a*) $\cdot_B$ $\mathrm{a}_B{}^{-1}$[*?f*, *?h*, *d ?a*]
using *assms B.assoc′-naturality* [*of* μ ν *d ?a*] **by** *simp*
also have ... = *P* (μ $\star_B$ ν) $\cdot_B$ (*e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*?f*, *?h*, *d ?a*])
using *assms P-def B.whisker-left* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = *P* (μ $\star_B$ ν) $\cdot_B$ *CMP ?f ?h*
unfolding *CMP-def LUNIT-def* **using** *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed

interpretation *EV*: *self-evaluation-map* V_B H_B a_B i_B src_B trg_B **..**
notation *EV.eval* (‹⦃-⦄›)

abbreviation (*input*) *SRCt* (‹**SRC**›)
where SRC μ ≡ ⟨*d* (src_B μ)⟩ ★ ⟨*e* (src_B μ)⟩

abbreviation (*input*) *TRGt* (‹**TRG**›)
where TRG $\mu \equiv \langle d\ (trg_B\ \mu)\rangle \star \langle e\ (trg_B\ \mu)\rangle$

abbreviation (*input*) *PRJt* (‹**PRJ**›)
where PRJ $\mu \equiv \langle e\ (trg_B\ \mu)\rangle \star \langle \mu\rangle \star \langle d\ (src_B\ \mu)\rangle$

The *CMP f g* satisfy the coherence conditions with respect to associativity that are required of compositors.

lemma *CMP-coherence*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $B.ide\ h$ **and** $src_B\ f = trg_B\ g$ **and** $src_B\ g = trg_B\ h$
shows $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B CMP\ (f \star_B g)\ h \cdot_B (CMP\ f\ g \star_B P\ h)$
$= CMP\ f\ (g \star_B h) \cdot_B (P\ f \star_B CMP\ g\ h) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −

The overall strategy of the proof is to expand the definition of *CMP* on the left and right-hand sides, then permute the occurrences of *LUNIT* and *RUNIT* to the left ends of both the left-hand side and right-hand side of the equation to be proved, so that the initial portions of these expressions become identical and the remaining parts to the right consist only of canonical isomorphisms. Then the Coherence Theorem is applied to prove syntactically (and automatically) that the canonical parts are equal, which implies equality of the complete expressions. The rest is just grinding through the calculations.

let $?a = trg_B\ f$
let $?b = trg_B\ g$
let $?c = trg_B\ h$
let $?d = src_B\ h$
have $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B CMP\ (f \star_B g)\ h \cdot_B (CMP\ f\ g \star_B P\ h)$
$= P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B LUNIT\ (h \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \cdot_B$
$(e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g)$
$\star_B P\ h)$
unfolding *CMP-def LUNIT-def* **using** *assms B.comp-assoc* **by** *simp*
also have ... $= P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B LUNIT\ (h \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$

$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
using *assms B.whisker-right P-def* **by** *simp*
also have ... $= P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B LUNIT\ h \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $e\ ?a \star_B (f \star_B g) \star_B LUNIT\ (h \star_B d\ ?d)$
$= e\ ?a \star_B (f \star_B g) \star_B (LUNIT\ h \star_B d\ ?d) \cdot_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]$
using *assms LUNIT-hcomp* [*of h d ?d*] *B.invert-side-of-triangle* **by** *simp*
also have ... $= (e\ ?a \star_B (f \star_B g) \star_B LUNIT\ h \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $e\ ?a \star_B (f \star_B g) \star_B LUNIT\ (h \star_B d\ ?d)$
$= (e\ ?a \star_B (f \star_B g) \star_B LUNIT\ h \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B$
$(e\ ?a \star_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$

proof −
have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d]) \cdot_B (e\ ?a \star_B (f \star_B g) \star_B LUNIT\ h \star_B d\ ?d)$
$= e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d] \cdot_B ((f \star_B g) \star_B LUNIT\ h \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B (((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of f* $\star_B$ *g LUNIT h d ?d*] **by** *simp*
also have ... $= (e\ ?a \star_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B LUNIT\ h \star_B d\ ?d)$
$= (e\ ?a \star_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B (e\ ?a \star_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d)$
$= e\ ?a \star_B \mathrm{a}_B[f,\ g,\ h] \cdot_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
also have ... $= e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \cdot_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d$
using *assms B.assoc-naturality* [*of f g LUNIT h*] **by** *simp*
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d)$
using *assms B.whisker-left B.whisker-right* **by** *simp*
finally have $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B (e\ ?a \star_B ((f \star_B g) \star_B LUNIT\ h) \star_B d\ ?d)$
$= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d)$
by *simp*
thus *?thesis* **using** *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$

$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$(((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B RUNIT\ f \star_B g \star_B d\ ?c) \star_B P\ h)) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$

proof −
have $((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h)$
$= (e\ ?a \star_B (RUNIT\ f \star_B g \star_B d\ ?c) \cdot_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B d\ ?c]) \star_B P\ h$
using *assms TRIANGLE* [*of f g* $\star_B$ *d ?c*] *B.invert-side-of-triangle* **by** *simp*
also have ... $= ((e\ ?a \star_B RUNIT\ f \star_B g \star_B d\ ?c) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B d\ ?c]) \star_B P\ h)$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
finally have $((e\ ?a \star_B f \star_B LUNIT\ (g \star_B d\ ?c)) \star_B P\ h)$
$= ((e\ ?a \star_B RUNIT\ f \star_B g \star_B d\ ?c) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B d\ ?c]) \star_B P\ h)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h)) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B TRG\ g,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$

proof −
have $((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B ((e\ ?a \star_B RUNIT\ f \star_B g \star_B d\ ?c) \star_B P\ h)$
$= (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c] \cdot_B (RUNIT\ f \star_B g \star_B d\ ?c)) \star_B P\ h$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
also have ... $= (e\ ?a \star_B$
$((RUNIT\ f \star_B g) \star_B d\ ?c) \cdot_B \mathrm{a}_B{}^{-1}[f \star_B TRG\ g,\ g,\ d\ ?c])$
$\star_B P\ h$

using *assms B.assoc′-naturality* [*of RUNIT f g d ?c*] **by** *simp*
also have ... = $((e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B TRG\ g,\ g,\ d\ ?c]) \star_B P\ h)$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
finally have $((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B RUNIT\ f \star_B g \star_B d\ ?c) \star_B P\ h)$
$= ((e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B TRG\ g,\ g,\ d\ ?c]) \star_B P\ h)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = $(e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(((e\ ?a \star_B (RUNIT\ f \star_B g)) \star_B d\ ?c) \star_B P\ h)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B ((e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h)$
$= \mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \cdot_B (e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
also have ... = $((e\ ?a \star_B (RUNIT\ f \star_B g)) \star_B d\ ?c) \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h$
using *assms B.assoc′-naturality* [*of e ?a RUNIT f* $\star_B$ *g d ?c*] **by** *simp*
also have ... = $(((e\ ?a \star_B (RUNIT\ f \star_B g)) \star_B d\ ?c) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h)$
using *assms B.whisker-left B.whisker-right P-def* **by** *simp*
finally have $(\mathrm{a}_B{}^{-1}[e\ ?a,\ f \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c) \star_B P\ h)$
$= (((e\ ?a \star_B (RUNIT\ f \star_B g)) \star_B d\ ?c) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h)$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... = $(e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$

$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$((e\ ?a \star_B (RUNIT\ f \star_B g)) \star_B d\ ?c \star_B P\ h)) \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B d\ ?b \star_B e\ ?b) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B d\ ?b \star_B e\ ?b) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B d\ ?b \star_B e\ ?b,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ d\ ?b \star_B e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
using *assms B.assoc-naturality* [*of e ?a* $\star_B$ (*RUNIT f* $\star_B$ *g*) *d ?c P h*] *B.comp-assoc*
by *simp*
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c \star_B P\ h)) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
using *assms B.assoc-naturality* [*of e ?a RUNIT f* $\star_B$ *g d ?c* $\star_B$ *P h*] *B.comp-assoc*
by *simp*
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d])) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $((e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c \star_B P\ h))$
$= e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d] \cdot_B (d\ ?c \star_B P\ h)$
using *assms B.comp-cod-arr B.whisker-left P-def*

$B.interchange\ [of\ f \star_B g\ RUNIT\ f \star_B g]$
by *simp*
also have ... $= e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]$
using *assms B.comp-arr-dom P-def* **by** *simp*
finally have $((e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B d\ ?c \star_B P\ h))$
$= e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d])) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B$
$\mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) =$
$e\ ?a \star_B (RUNIT\ f \star_B g) \star_B$
$\mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d] \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]$
using *assms B.comp-arr-dom B.comp-cod-arr B.whisker-left*
$B.interchange$
$[of\ f \star_B g\ RUNIT\ f \star_B g\ \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]\ \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]]$
by *simp*
also have ... $= (e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d])$
using *assms B.comp-arr-dom B.whisker-left*
$B.interchange\ [of\ RUNIT\ f \star_B g\ (f \star_B SRC\ f) \star_B g\ \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]$
$\mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]]$
by *simp*
finally have $(e\ ?a \star_B (f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d])$
$= (e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B$
$\mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d])$
by *simp*
thus *?thesis*

using *B.comp-assoc* **by** *simp*
qed
also have ... = $(e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d])$
$= e\ ?a \star_B (\mathrm{a}_B[f,\ g,\ TRG\ h \star_B h] \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f \star_B g,\ TRG\ h \star_B h,\ d\ ?d]$
using *assms B.whisker-left* **by** *simp*
also have ... = $e\ ?a \star_B$
$\mathrm{a}_B{}^{-1}[f,\ g \star_B (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d] \cdot_B$
$(f \star_B \mathrm{a}_B{}^{-1}[g,\ (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[f,\ g,\ ((d\ ?c \star_B e\ ?c) \star_B h) \star_B d\ ?d]$
proof −
have $(\mathrm{a}_B{}^{-1}[f,\ g,\ (d\ ?c \star_B e\ ?c) \star_B h] \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ g \star_B (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d] \cdot_B$
$(f \star_B \mathrm{a}_B{}^{-1}[g,\ (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d])$
$= \mathrm{a}_B{}^{-1}[f \star_B g,\ (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d] \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ g,\ ((d\ ?c \star_B e\ ?c) \star_B h) \star_B d\ ?d]$
using *assms B.pentagon′ B.comp-assoc* **by** *simp*
moreover have $B.inv\ (\mathrm{a}_B{}^{-1}[f,\ g,\ (d\ ?c \star_B e\ ?c) \star_B h] \star_B d\ ?d)$
$= \mathrm{a}_B[f,\ g,\ (d\ ?c \star_B e\ ?c) \star_B h] \star_B d\ ?d$
using *assms* **by** *simp*
ultimately show *?thesis*
using *assms B.comp-assoc*
B.invert-opposite-sides-of-square
[*of* $\mathrm{a}_B{}^{-1}[f,\ g,\ (d\ ?c \star_B e\ ?c) \star_B h] \star_B d\ ?d$
$\mathrm{a}_B{}^{-1}[f,\ g \star_B (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d] \cdot_B$
$(f \star_B \mathrm{a}_B{}^{-1}[g,\ (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d])$
$\mathrm{a}_B{}^{-1}[f \star_B g,\ (d\ ?c \star_B e\ ?c) \star_B h,\ d\ ?d]$
$\mathrm{a}_B{}^{-1}[f,\ g,\ ((d\ ?c \star_B e\ ?c) \star_B h) \star_B d\ ?d]]$
by *simp*
qed
also have ... = $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$

$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ TRG\ h \star_B\ h] \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B\ g,\ TRG\ h \star_B\ h,\ d\ ?d])$
$= (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ (d\ ?c \star_B\ e\ ?c) \star_B\ h \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B\ SRC\ f) \star_B\ g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c \star_B\ P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B\ SRC\ f,\ g,\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ g] \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B\ P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ g) \star_B\ P\ h)$
proof −
have $(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B\ g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
$= e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d] \cdot_B ((RUNIT\ f \star_B\ g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,$
$h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B$
$(RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ TRG\ h \star_B\ h \star_B\ d\ ?d]$
using *assms B.hseqI′ B.assoc-naturality* [*of RUNIT f g* $\mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]$]
by *simp*
also have ... $= (e\ ?a \star_B RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ TRG\ h \star_B\ h \star_B\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B[f,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (RUNIT\ f \star_B\ g) \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
$= (e\ ?a \star_B RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ TRG\ h \star_B\ h \star_B\ d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$

$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B\ LUNIT\ (g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d))) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B\ e\ ?b,\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c \star_B\ e\ ?c,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ TRG\ h \star_B\ h \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B\ SRC\ f) \star_B\ g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c \star_B\ P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B\ SRC\ f) \star_B\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B\ SRC\ f,\ g,\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ g] \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B\ P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ g) \star_B\ P\ h)$

proof −
have $e\ ?a \star_B\ RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c \star_B\ e\ ?c,\ h,\ d\ ?d]$
$= (e\ ?a \star_B\ RUNIT\ f \star_B\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
using *assms B.whisker-left B.comp-arr-dom B.comp-cod-arr*
B.interchange [*of RUNIT f f* $\star_B$ *SRC f g* $\star_B$ $((TRG\ h \star_B\ h) \star_B\ d\ ?d)$
$g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]$]
by *simp*
also have ... $= (e\ ?a \star_B$
$(f \star_B\ LUNIT\ (g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$\mathrm{a}_B[f,\ d\ ?b \star_B\ e\ ?b,\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
using *assms TRIANGLE* [*of f g* $\star_B$ $(TRG\ h \star_B\ h) \star_B\ d\ ?d$] **by** *simp*
also have ... $= (e\ ?a \star_B f \star_B\ LUNIT\ (g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B\ e\ ?b,\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
using *assms B.whisker-left B.comp-assoc* **by** *simp*
finally have $e\ ?a \star_B\ RUNIT\ f \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]$
$= (e\ ?a \star_B f \star_B\ LUNIT\ (g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B\ e\ ?b,\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B\ LUNIT\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B\ e\ ?b,\ g \star_B\ (TRG\ h \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f) \star_B\ g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B\ SRC\ f,\ g,\ TRG\ h \star_B\ h \star_B\ d\ ?d]) \cdot_B$

$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$

proof −
have $e\ ?a \star_B f \star_B LUNIT\ (g \star_B (TRG\ h \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B f \star_B$
$(LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ ?b \star_B e\ ?b,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]$
using *assms LUNIT-hcomp* [*of* $g\ (TRG\ h \star_B h) \star_B d\ ?d$]
B.invert-side-of-triangle
by *simp*
also have ... $= (e\ ?a \star_B f \star_B LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $e\ ?a \star_B f \star_B LUNIT\ (g \star_B (TRG\ h \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B f \star_B LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B e\ ?b,\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f) \star_B g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B SRC\ f,\ g,\ TRG\ h \star_B h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$

proof −
have $(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d)$

$= e\ ?a \star_B f \star_B$
$\mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B h,\ d\ ?d] \cdot_B (LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B f \star_B$
$((LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of LUNIT g TRG h* $\star_B$ *h d ?d*]
by *simp*
also have ... $= (e\ ?a \star_B f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B LUNIT\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= ((e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B LUNIT\ g \star_B SRC\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B e\ ?b,\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f) \star_B g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B SRC\ f,\ g,\ TRG\ h \star_B h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d] \cdot_B$
$(f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B$
$((f \star_B LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of f LUNIT g* $\star_B$ *TRG h* $\star_B$ *h d ?d*] **by** *simp*
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d])$

using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B (f \star_B LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B e\ ?b,\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f) \star_B g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B SRC\ f,\ g,\ TRG\ h \star_B h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)$
proof −
have $(e\ ?a \star_B (f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B LUNIT\ g \star_B TRG\ h \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B (f \star_B LUNIT\ g \star_B LUNIT\ h) \star_B d\ ?d$
using *assms B.whisker-left B.whisker-right B.comp-arr-dom B.comp-cod-arr B.interchange* [*of g LUNIT g LUNIT h* $(d\ ?c \star_B e\ ?c) \star_B h$]
by *simp*
thus *?thesis*
using *assms* **by** *simp*
qed
finally have *L*: $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B CMP\ (f \star_B g)\ h \cdot_B (CMP\ f\ g \star_B P\ h)$
$= (e\ ?a \star_B (f \star_B LUNIT\ g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B e\ ?b,\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f) \star_B g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B[f \star_B SRC\ f,\ g,\ TRG\ h \star_B h \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B$

$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$((e\ ?a \star_B\ f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B\ d\ ?c]) \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ g] \star_B\ P\ h) \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ g] \star_B\ P\ h) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ g) \star_B\ P\ h)$
by *simp*

have $CMP\ f\ (g \star_B\ h) \cdot_B (P\ f \star_B\ CMP\ g\ h) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
$= (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ LUNIT\ ((g \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B\ f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ (g \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ (g \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ (g \star_B\ h)) \cdot_B$
$(P\ f \star_B$
$(e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(e\ ?b \star_B\ g \star_B\ LUNIT\ (h \star_B\ d\ ?d)) \cdot_B$
$(e\ ?b \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h] \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h)) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
unfolding *CMP-def LUNIT-def* **using** *assms B.comp-assoc* **by** *simp*
also have ... $= (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ LUNIT\ ((g \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B\ f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ (g \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ (g \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ (g \star_B\ h)) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B\ LUNIT\ (h \star_B\ d\ ?d)) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
using *assms B.whisker-left P-def B.comp-assoc* **by** *auto*
also have ... $= ((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ LUNIT\ (g \star_B\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B\ f \star_B \mathrm{a}_B{}^{-1}[d\ ?b \star_B\ e\ ?b,\ g \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ (g \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ (g \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ P\ (g \star_B\ h)) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B\ LUNIT\ (h \star_B\ d\ ?d)) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$

$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $e\ ?a \star_B f \star_B LUNIT\ ((g \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B f \star_B (LUNIT\ (g \star_B h) \star_B d\ ?d) \cdot_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]$
using *assms LUNIT-hcomp* [*of* $g \star_B h\ d\ ?d$] *B.invert-side-of-triangle*
by *simp*
also have ... $= (e\ ?a \star_B f \star_B LUNIT\ (g \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $e\ ?a \star_B f \star_B LUNIT\ ((g \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B f \star_B LUNIT\ (g \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ (g \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ (g \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ (g \star_B h)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B LUNIT\ (h \star_B d\ ?d)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B h,\ d\ ?d]) \cdot_B (e\ ?a \star_B f \star_B LUNIT\ (g \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B h,\ d\ ?d] \cdot_B (f \star_B LUNIT\ (g \star_B h) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B ((f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of* $f\ LUNIT\ (g \star_B h)\ d\ ?d$]
LUNIT-in-hom [*of* $g \star_B h$]
by *auto*
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B LUNIT\ (g \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*

using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ (g \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ (g \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ (g \star_B h)) \cdot_B$
$((P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B RUNIT\ g \star_B h \star_B d\ ?d)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $P\ f \star_B e\ ?b \star_B g \star_B LUNIT\ (h \star_B d\ ?d)$
$= P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h \star_B d\ ?d) \cdot_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h,\ h \star_B d\ ?d]$
using *assms TRIANGLE* [*of g h* $\star_B$ *d ?d*] *B.invert-side-of-triangle* **by** *simp*
also have ... $= (P\ f \star_B e\ ?b \star_B RUNIT\ g \star_B h \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h,\ h \star_B d\ ?d])$
using *assms B.whisker-left P-def* **by** *simp*
finally have $P\ f \star_B e\ ?b \star_B g \star_B LUNIT\ (h \star_B d\ ?d) =$
$(P\ f \star_B e\ ?b \star_B RUNIT\ g \star_B h \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ TRG\ h,\ h \star_B d\ ?d])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ (g \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ (g \star_B h)] \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ (g \star_B h)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B (P\ f \star_B e\ ?b \star_B RUNIT\ g \star_B h \star_B d\ ?d)$
$= P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d] \cdot_B (RUNIT\ g \star_B h \star_B d\ ?d)$

using *assms B.whisker-left P-def* **by** *simp*
also have ... $= P\ f \star_B e\ ?b \star_B$
$((RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of RUNIT g h d ?d*] **by** *auto*
also have ... $= (P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d])$
using *assms B.whisker-left P-def* **by** *simp*
finally have $(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B RUNIT\ g \star_B h \star_B d\ ?d)$
$= (P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ (g \star_B h)] \cdot_B$
$(\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ (g \star_B h)] \cdot_B$
$(((e\ ?a \star_B f) \star_B d\ ?b) \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ (g \star_B h)) \cdot_B (P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= \mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d$
using *assms B.comp-arr-dom B.comp-cod-arr P-def*
B.interchange
[*of* $\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b]\ P\ f\ P\ (g \star_B h)\ e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d$]
by *simp*
also have ... $= (((e\ ?a \star_B f) \star_B d\ ?b) \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d)$
using *assms B.comp-arr-dom B.comp-cod-arr*
B.interchange
[*of* $(e\ ?a \star_B f) \star_B d\ ?b\ \mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b]$
$e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d$
$e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d$]
by *simp*
finally have $(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ (g \star_B h)) \cdot_B$
$(P\ f \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= (((e\ ?a \star_B f) \star_B d\ ?b) \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d)$

by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... = $(e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b \star_B e\ ?b,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$(\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ (g \star_B h)] \cdot_B$
$((e\ ?a \star_B f) \star_B d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
using *assms P-def B.comp-assoc*
B.assoc-naturality [*of* $e\ ?a \star_B f\ d\ ?b\ e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d$]
by *simp*
also have ... = $(e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (d\ ?b \star_B e\ ?b) \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b \star_B e\ ?b,\ g \star_B h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B d\ ?c \star_B e\ ?c) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B d\ ?c \star_B e\ ?c) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B d\ ?c \star_B e\ ?c) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?c \star_B e\ ?c,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?c \star_B e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
using *assms P-def B.comp-assoc*
B.assoc-naturality [*of* $e\ ?a\ f\ d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d$]
by *simp*
also have ... = $(e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B TRG\ g \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B d\ ?c \star_B e\ ?c) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B d\ ?c \star_B e\ ?c) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$

$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$

proof −
have
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B f \star_B$
$\mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d] \cdot_B$
$(d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B f \star_B$
$(TRG\ g \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]$
using *assms B.assoc′-naturality* [*of d ?b e ?b* $(RUNIT\ g \star_B h) \star_B d\ ?d$] **by** *auto*
also have ... $= (e\ ?a \star_B f \star_B SRC\ f \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ (g \star_B h) \star_B d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B d\ ?b \star_B e\ ?b \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B f \star_B SRC\ f \star_B (RUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$

proof −
have $(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B TRG\ g \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= e\ ?a \star_B f \star_B$

$\mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d] \cdot_B (TRG\ g \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B f \star_B$
$((TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]$
using *assms B.assoc′-naturality* [*of TRG g RUNIT g* $\star_B$ *h d ?d*] **by** *simp*
also have ... $= (e\ ?a \star_B f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B TRG\ g \star_B (RUNIT\ g \star_B h) \star_B d\ ?d)$
$= (e\ ?a \star_B f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d])$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d)$
$= e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d] \cdot_B$
$(f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d)$
using *assms B.whisker-left* **by** *simp*
also have ... $= e\ ?a \star_B$
$((f \star_B SRC\ f \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d]$
using *assms*
B.assoc′-naturality [*of f* $(d\ ?b \star_B e\ ?b) \star_B (RUNIT\ g \star_B h)\ d\ ?d$]
by *simp*
also have ... $= (e\ ?a \star_B (f \star_B SRC\ f \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d])$
using *assms B.whisker-left* **by** *simp*
finally have $(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B g \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B (TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d)$

$= (e\ ?a \star_B (f \star_B TRG\ g \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d])$
using *assms* **by** *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ (g \star_B h)) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f \star_B \mathrm{a}_B[g,\ d\ ?c \star_B e\ ?c,\ h]) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $e\ ?a \star_B (f \star_B SRC\ f \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d$
$= e\ ?a \star_B (f \star_B SRC\ f \star_B (g \star_B LUNIT\ h) \cdot_B \mathrm{a}_B[g,\ SRC\ g,\ h]) \star_B d\ ?d$
using *assms TRIANGLE* [*of g h*] **by** *simp*
also have ... $= (e\ ?a \star_B (f \star_B SRC\ f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f \star_B \mathrm{a}_B[g,\ TRG\ h,\ h]) \star_B d\ ?d)$
using *assms B.whisker-left B.whisker-right* **by** *simp*
finally have $e\ ?a \star_B (f \star_B SRC\ f \star_B (RUNIT\ g \star_B h)) \star_B d\ ?d$
$= (e\ ?a \star_B (f \star_B SRC\ f \star_B g \star_B LUNIT\ h) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B SRC\ f \star_B \mathrm{a}_B[g,\ TRG\ h,\ h]) \star_B d\ ?d)$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B LUNIT\ g \star_B h) \star_B d\ ?d) \cdot_B$
$((e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B TRG\ g \star_B g \star_B LUNIT\ h) \star_B d\ ?d)) \cdot_B$
$(e\ ?a \star_B (f \star_B TRG\ g \star_B \mathrm{a}_B[g,\ SRC\ g,\ h]) \star_B d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$

$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$

proof −
have $e\ ?a \star_B (f \star_B\ LUNIT\ (g \star_B\ h)) \star_B\ d\ ?d$
$= e\ ?a \star_B (f \star_B (LUNIT\ g \star_B\ h) \cdot_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B\ d\ ?d$
using *assms LUNIT-hcomp* [*of g h*] *B.invert-side-of-triangle* **by** *simp*
also have ... $= (e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B\ d\ ?d)$
using *assms B.whisker-left B.whisker-right* **by** *simp*
finally have $e\ ?a \star_B (f \star_B\ LUNIT\ (g \star_B\ h)) \star_B\ d\ ?d$
$= (e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B\ d\ ?d)$
by *simp*
thus *?thesis*
using *assms B.comp-assoc* **by** *simp*
qed
also have ... $= ((e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B (TRG\ g \star_B\ g) \star_B\ LUNIT\ h) \star_B\ d\ ?d)) \cdot_B$
$(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ TRG\ g \star_B \mathrm{a}_B[g,\ SRC\ g,\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B\ SRC\ g) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B\ SRC\ g) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B\ SRC\ g) \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ e\ ?b \star_B ((g \star_B\ SRC\ g) \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B\ SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$

proof −
have
$(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ TRG\ g \star_B\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d)$
$= e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h] \cdot_B (TRG\ g \star_B\ g \star_B\ LUNIT\ h)) \star_B\ d\ ?d$
using *assms B.whisker-left B.whisker-right* **by** *auto*
also have ... $= e\ ?a \star_B$
$(f \star_B ((TRG\ g \star_B\ g) \star_B\ LUNIT\ h) \cdot_B$
$\mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ SRC\ g \star_B\ h]) \star_B\ d\ ?d$
using *assms B.assoc′-naturality* [*of TRG g g LUNIT h*] **by** *auto*
also have ... $= (e\ ?a \star_B (f \star_B (TRG\ g \star_B\ g) \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B\ h]) \star_B\ d\ ?d)$
using *assms B.whisker-left B.whisker-right* **by** *auto*
finally have $(e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ h]) \star_B\ d\ ?d) \cdot_B$

$(e\ ?a \star_B (f \star_B\ TRG\ g \star_B\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d)$
$= (e\ ?a \star_B (f \star_B (TRG\ g \star_B\ g) \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B\ h]) \star_B\ d\ ?d)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ TRG\ g \star_B\ \mathrm{a}_B[g,\ d\ ?c \star_B\ e\ ?c,\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B\ \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B\ d\ ?c \star_B\ e\ ?c) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B\ SRC\ g) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B\ SRC\ g) \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ e\ ?b \star_B ((g \star_B\ d\ ?c \star_B\ e\ ?c) \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ \mathrm{a}_B{}^{-1}[g \star_B\ SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
proof −
have $(e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B (TRG\ g \star_B\ g) \star_B\ LUNIT\ h) \star_B\ d\ ?d)$
$= e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d$
using *assms B.whisker-left B.whisker-right B.comp-arr-dom B.comp-cod-arr B.interchange* [*of LUNIT g TRG g* $\star_B$ *g h LUNIT h*]
by *simp*
thus *?thesis*
using *assms* **by** *simp*
qed
finally have *R*: $CMP\ f\ (g \star_B\ h) \cdot_B (P\ f \star_B\ CMP\ g\ h) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
$= (e\ ?a \star_B (f \star_B\ LUNIT\ g \star_B\ LUNIT\ h) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B (f \star_B\ SRC\ f \star_B\ \mathrm{a}_B[g,\ TRG\ h,\ h]) \star_B\ d\ ?d) \cdot_B$
$(e\ ?a \star_B\ \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B\ SRC\ g) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B\ SRC\ g) \star_B\ h,\ d\ ?d]) \cdot_B$
$(e\ ?a \star_B\ f \star_B\ \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B\ SRC\ g) \star_B\ h) \star_B\ d\ ?d]) \cdot_B$
$\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$\mathrm{a}_B[e\ ?a \star_B\ f,\ d\ ?b,\ P\ ((g \star_B\ SRC\ g) \star_B\ h)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B\ e\ ?b \star_B ((g \star_B\ SRC\ g) \star_B\ h) \star_B\ d\ ?d) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ \mathrm{a}_B{}^{-1}[g \star_B\ SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ e\ ?b \star_B\ g \star_B\ \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B\ d\ ?d]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B\ P\ h]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B[e\ ?b \star_B\ g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B\ \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B\ P\ h) \cdot_B$

$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
using *assms* **by** *simp*

The portions of the expressions on the right-hand sides of assertions L and R that are not identical only involve canonical isomorphisms, and thus they can be proved equal automatically by the simplifier, once we have expressed them in the formal language of B.

$$
\begin{aligned}
\textbf{let}\ ?LHS = {} & (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ (TRG\ g \star_B g) \star_B TRG\ h \star_B h,\ d\ ?d]) \cdot_B \\
& (e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g \star_B g,\ TRG\ h \star_B h,\ d\ ?d]) \cdot_B \\
& (e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B \\
& (e\ ?a \star_B \mathrm{a}_B[f,\ d\ ?b \star_B e\ ?b,\ g \star_B (TRG\ h \star_B h) \star_B d\ ?d]) \cdot_B \\
& (e\ ?a \star_B (f \star_B SRC\ f) \star_B g \star_B \mathrm{a}_B{}^{-1}[TRG\ h,\ h,\ d\ ?d]) \cdot_B \\
& (e\ ?a \star_B \mathrm{a}_B[f \star_B SRC\ f,\ g,\ TRG\ h \star_B h \star_B d\ ?d]) \cdot_B \\
& (e\ ?a \star_B ((f \star_B SRC\ f) \star_B g) \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B \\
& \mathrm{a}_B[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c \star_B P\ h] \cdot_B \\
& \mathrm{a}_B[e\ ?a \star_B (f \star_B SRC\ f) \star_B g,\ d\ ?c,\ P\ h] \cdot_B \\
& (\mathrm{a}_B{}^{-1}[e\ ?a,\ (f \star_B SRC\ f) \star_B g,\ d\ ?c] \star_B P\ h) \cdot_B \\
& ((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f \star_B SRC\ f,\ g,\ d\ ?c]) \star_B P\ h) \cdot_B \\
& ((e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ SRC\ f,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B \\
& ((e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ g \star_B d\ ?c]) \star_B P\ h) \cdot_B \\
& (\mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ g] \star_B P\ h) \cdot_B \\
& (\mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ g] \star_B P\ h) \cdot_B \\
& ((\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B P\ g) \star_B P\ h)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{let}\ ?LHSt = {} & (\langle e\ ?a\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ (\mathbf{TRG}\ g \star \langle g\rangle) \star \mathbf{TRG}\ h \star \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star \langle f\rangle \star \mathbf{a}^{-1}[\mathbf{TRG}\ g \star \langle g\rangle,\ \mathbf{TRG}\ h \star \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star \langle f\rangle \star \mathbf{a}^{-1}[\mathbf{TRG}\ g,\ \langle g\rangle,\ (\mathbf{TRG}\ h \star \langle h\rangle) \star \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star \mathbf{a}[\langle f\rangle,\ \langle d\ ?b\rangle \star \langle e\ ?b\rangle,\ \langle g\rangle \star (\mathbf{TRG}\ h \star \langle h\rangle) \star \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star (\langle f\rangle \star \mathbf{SRC}\ f) \star \langle g\rangle \star \mathbf{a}^{-1}[\mathbf{TRG}\ h,\ \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star \mathbf{a}[\langle f\rangle \star \mathbf{SRC}\ f,\ \langle g\rangle,\ \mathbf{TRG}\ h \star \langle h\rangle \star \langle d\ ?d\rangle]) \cdot \\
& (\langle e\ ?a\rangle \star ((\langle f\rangle \star \mathbf{SRC}\ f) \star \langle g\rangle) \star \mathbf{a}^{-1}[\langle d\ ?c\rangle,\ \langle e\ ?c\rangle,\ \langle h\rangle \star \langle d\ ?d\rangle]) \cdot \\
& \mathbf{a}[\langle e\ ?a\rangle,\ (\langle f\rangle \star \mathbf{SRC}\ f) \star \langle g\rangle,\ \langle d\ ?c\rangle \star \mathbf{PRJ}\ h] \cdot \\
& \mathbf{a}[\langle e\ ?a\rangle \star (\langle f\rangle \star \mathbf{SRC}\ f) \star \langle g\rangle,\ \langle d\ ?c\rangle,\ \mathbf{PRJ}\ h] \cdot \\
& (\mathbf{a}^{-1}[\langle e\ ?a\rangle,\ (\langle f\rangle \star \mathbf{SRC}\ f) \star \langle g\rangle,\ \langle d\ ?c\rangle] \star \mathbf{PRJ}\ h) \cdot \\
& ((\langle e\ ?a\rangle \star \mathbf{a}^{-1}[\langle f\rangle \star \mathbf{SRC}\ f,\ \langle g\rangle,\ \langle d\ ?c\rangle]) \star \mathbf{PRJ}\ h) \cdot \\
& ((\langle e\ ?a\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ \mathbf{SRC}\ f,\ \langle g\rangle \star \langle d\ ?c\rangle]) \star \mathbf{PRJ}\ h) \cdot \\
& ((\langle e\ ?a\rangle \star \langle f\rangle \star \mathbf{a}^{-1}[\langle d\ ?b\rangle,\ \langle e\ ?b\rangle,\ \langle g\rangle \star \langle d\ ?c\rangle]) \star \mathbf{PRJ}\ h) \cdot \\
& (\mathbf{a}[\langle e\ ?a\rangle,\ \langle f\rangle,\ \langle d\ ?b\rangle \star \mathbf{PRJ}\ g] \star \mathbf{PRJ}\ h) \cdot \\
& (\mathbf{a}[\langle e\ ?a\rangle \star \langle f\rangle,\ \langle d\ ?b\rangle,\ \mathbf{PRJ}\ g] \star \mathbf{PRJ}\ h) \cdot \\
& ((\mathbf{a}^{-1}[\langle e\ ?a\rangle,\ \langle f\rangle,\ \langle d\ ?b\rangle] \star \mathbf{PRJ}\ g) \star \mathbf{PRJ}\ h)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{let}\ ?RHS = {} & (e\ ?a \star_B (f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ g,\ TRG\ h \star_B h]) \star_B d\ ?d) \cdot_B \\
& (e\ ?a \star_B (f \star_B SRC\ f \star_B \mathrm{a}_B[g,\ TRG\ h,\ h]) \star_B d\ ?d) \cdot_B \\
& (e\ ?a \star_B \mathrm{a}_B{}^{-1}[f,\ TRG\ g \star_B (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B \\
& (e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[TRG\ g,\ (g \star_B SRC\ g) \star_B h,\ d\ ?d]) \cdot_B \\
& (e\ ?a \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d]) \cdot_B \\
& \mathrm{a}_B[e\ ?a,\ f,\ d\ ?b \star_B P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B \\
& \mathrm{a}_B[e\ ?a \star_B f,\ d\ ?b,\ P\ ((g \star_B SRC\ g) \star_B h)] \cdot_B \\
& (\mathrm{a}_B{}^{-1}[e\ ?a,\ f,\ d\ ?b] \star_B e\ ?b \star_B ((g \star_B SRC\ g) \star_B h) \star_B d\ ?d) \cdot_B
\end{aligned}
$$

$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g \star_B SRC\ g,\ h,\ d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B \mathrm{a}_B{}^{-1}[g,\ SRC\ g,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B e\ ?b \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?c,\ e\ ?c,\ h \star_B d\ ?d]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b,\ g,\ d\ ?c \star_B P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B[e\ ?b \star_B g,\ d\ ?c,\ P\ h]) \cdot_B$
$(P\ f \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ g,\ d\ ?c] \star_B P\ h) \cdot_B$
$\mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$

let $?RHSt = (\langle e\ ?a\rangle \star (\langle f\rangle \star \mathbf{a}^{-1}[\mathbf{TRG}\ g,\ \langle g\rangle,\ \mathbf{TRG}\ h \star \langle h\rangle]) \star \langle d\ ?d\rangle) \cdot$
$(\langle e\ ?a\rangle \star (\langle f\rangle \star \mathbf{SRC}\ f \star \mathbf{a}[\langle g\rangle,\ \mathbf{TRG}\ h,\ \langle h\rangle]) \star \langle d\ ?d\rangle) \cdot$
$(\langle e\ ?a\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ \mathbf{TRG}\ g \star (\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot$
$(\langle e\ ?a\rangle \star \langle f\rangle \star \mathbf{a}^{-1}[\mathbf{TRG}\ g,\ (\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot$
$(\langle e\ ?a\rangle \star \langle f\rangle \star \mathbf{a}^{-1}[\langle d\ ?b\rangle,\ \langle e\ ?b\rangle,\ ((\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle) \star \langle d\ ?d\rangle]) \cdot$
$\mathbf{a}[\langle e\ ?a\rangle,\ \langle f\rangle,\ \langle d\ ?b\rangle \star (\langle e\ ?b\rangle \star ((\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle) \star \langle d\ ?d\rangle)] \cdot$
$\mathbf{a}[\langle e\ ?a\rangle \star \langle f\rangle,\ \langle d\ ?b\rangle,\ (\langle e\ ?b\rangle \star ((\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle) \star \langle d\ ?d\rangle)] \cdot$
$(\mathbf{a}^{-1}[\langle e\ ?a\rangle,\ \langle f\rangle,\ \langle d\ ?b\rangle] \star \langle e\ ?b\rangle \star ((\langle g\rangle \star \mathbf{SRC}\ g) \star \langle h\rangle) \star \langle d\ ?d\rangle) \cdot$
$(\mathbf{PRJ}\ f \star \langle e\ ?b\rangle \star \mathbf{a}^{-1}[\langle g\rangle \star \mathbf{SRC}\ g,\ \langle h\rangle,\ \langle d\ ?d\rangle]) \cdot$
$(\mathbf{PRJ}\ f \star \langle e\ ?b\rangle \star \mathbf{a}^{-1}[\langle g\rangle,\ \mathbf{SRC}\ g,\ \langle h\rangle \star \langle d\ ?d\rangle]) \cdot$
$(\mathbf{PRJ}\ f \star \langle e\ ?b\rangle \star \langle g\rangle \star \mathbf{a}^{-1}[\langle d\ ?c\rangle,\ \langle e\ ?c\rangle,\ \langle h\rangle \star \langle d\ ?d\rangle]) \cdot$
$(\mathbf{PRJ}\ f \star \mathbf{a}[\langle e\ ?b\rangle,\ \langle g\rangle,\ \langle d\ ?c\rangle \star \mathbf{PRJ}\ h]) \cdot$
$(\mathbf{PRJ}\ f \star \mathbf{a}[\langle e\ ?b\rangle \star \langle g\rangle,\ \langle d\ ?c\rangle,\ \mathbf{PRJ}\ h]) \cdot$
$(\mathbf{PRJ}\ f \star \mathbf{a}^{-1}[\langle e\ ?b\rangle,\ \langle g\rangle,\ \langle d\ ?c\rangle] \star \mathbf{PRJ}\ h) \cdot$
$\mathbf{a}[\mathbf{PRJ}\ f,\ \mathbf{PRJ}\ g,\ \mathbf{PRJ}\ h]$

have E: $?LHS = ?RHS$
proof −
have $?LHS = \{\!|?LHSt|\!\}$
using *assms B.α-def B.α′.map-ide-simp* $B.VVV.ide\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P-def*
by *simp*
also have ... $= \{\!|?RHSt|\!\}$
using *assms* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= ?RHS$
using *assms B.α-def B.α′.map-ide-simp* $B.VVV.ide\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P-def*
by *simp*
finally show *?thesis* **by** *blast*
qed
show *?thesis*
using *L R E* **by** *argo*
qed

interpretation *CMP*: *transformation-by-components B.VV.comp comp HoPP.map PoH.map* ‹$\lambda\mu\nu.$ *CMP* (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
proof
show *1*: $\bigwedge fg.$ *B.VV.ide fg* $\Longrightarrow$ «*CMP* (*fst fg*) (*snd fg*) : *HoPP.map fg* $\Rightarrow$ *PoH.map fg*»
using *CMP-in-hom*(*2*) $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P.FF-def hcomp-def* $arr\text{-}char_{SbC}$ P_0*-props*(*1*) *P.preserves-arr*
by *auto*

show $\bigwedge fg.\ B.VV.arr\ fg \Longrightarrow$
$CMP\ (fst\ (B.VV.cod\ fg))\ (snd\ (B.VV.cod\ fg)) \cdot HoPP.map\ fg =$
$PoH.map\ fg \cdot CMP\ (fst\ (B.VV.dom\ fg))\ (snd\ (B.VV.dom\ fg))$
proof −
fix *fg*
assume *fg*: $B.VV.arr\ fg$
have $CMP\ (fst\ (B.VV.cod\ fg))\ (snd\ (B.VV.cod\ fg)) \cdot HoPP.map\ fg =$
$CMP\ (fst\ (B.VV.cod\ fg))\ (snd\ (B.VV.cod\ fg)) \cdot_B HoPP.map\ fg$
using *fg 1* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *HoPP.preserves-arr*
P.FF-def hcomp-char comp-char
by *auto*
also have ... $= PoH.map\ fg \cdot_B CMP\ (fst\ (B.VV.dom\ fg))\ (snd\ (B.VV.dom\ fg))$
using *fg CMP-naturality* [*of fst fg snd fg*]
$B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$
P.FF-def
hcomp-def $arr\text{-}char_{SbC}$ P_0*-props*(*1*) *P.preserves-arr*
by *auto*
also have ... $= PoH.map\ fg \cdot CMP\ (fst\ (B.VV.dom\ fg))\ (snd\ (B.VV.dom\ fg))$
proof −
have $P_0\ (src_B\ (snd\ fg)) \in Obj \wedge P_0\ (trg_B\ (fst\ fg)) \in Obj$
using *fg* P_0*-props*(*6*) *B.VV.arrE B.obj-src B.obj-trg* **by** *meson*
moreover have $B.seq\ (P\ (fst\ fg \star_B snd\ fg))$
$(CMP\ (fst\ (B.VV.dom\ fg))\ (snd\ (B.VV.dom\ fg)))$
using *fg 1* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ **by** *simp*
ultimately show *?thesis*
using *fg 1 comp-char* $arr\text{-}char_{SbC}$ $in\text{-}hom\text{-}char_{SbC}$ **by** *simp*
qed
finally show $CMP\ (fst\ (B.VV.cod\ fg))\ (snd\ (B.VV.cod\ fg)) \cdot HoPP.map\ fg =$
$PoH.map\ fg \cdot CMP\ (fst\ (B.VV.dom\ fg))\ (snd\ (B.VV.dom\ fg))$
by *blast*
qed
qed

interpretation *CMP*: *natural-isomorphism B.VV.comp comp HoPP.map PoH.map CMP.map*
using *iso-CMP* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *CMP.map-simp-ide*
by *unfold-locales simp*

definition Φ_P
where $\Phi_P = CMP.map$

interpretation Φ_P: *natural-isomorphism B.VV.comp comp HoPP.map PoH.map* Φ_P
unfolding Φ_P*-def*
using *CMP.natural-isomorphism-axioms* **by** *simp*

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

lemma Φ_P*-in-hom* [*intro*]:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows «$\Phi_P\ (f,\ g) : src\ (P\ g) \rightarrow trg\ (P\ f)$»

and «Φ_P $(f,\ g)$: $P\ f \star P\ g \Rightarrow P\ (f \star_B\ g)$»
proof −
  show *1*: «Φ_P $(f,\ g)$: $P\ f \star P\ g \Rightarrow P\ (f \star_B\ g)$»
    using *assms* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P.FF-def* **by** *auto*
  show «Φ_P $(f,\ g)$: $src\ (P\ g) \rightarrow trg\ (P\ f)$»
    using *1*
    by (*metis* (*no-types*, *lifting*) *hcomp-simps*(*2*) *in-hhom-def* $in\text{-}hom\text{-}char_{SbC}$
      src-hcomp vconn-implies-hpar(*1*−*2*))
qed

lemma Φ_P*-simps* [*simp*]:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $arr\ (\Phi_P\ (f,\ g))$
and $src\ (\Phi_P\ (f,\ g)) = src\ (P\ g)$
and $trg\ (\Phi_P\ (f,\ g)) = trg\ (P\ f)$
and $dom\ (\Phi_P\ (f,\ g)) = P\ f \star P\ g$
and $cod\ (\Phi_P\ (f,\ g)) = P\ (f \star_B\ g)$
  using *assms* Φ_P*-in-hom* **by** *blast*+

sublocale *prj*: *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ comp\ hcomp\ \mathrm{a}\ \mathrm{i}_B\ src\ trg\ P\ \Phi_P$
proof
  fix $f\ g\ h$
  assume f: $B.ide\ f$ **and** g: $B.ide\ g$ **and** h: $B.ide\ h$
  and fg: $src_B\ f = trg_B\ g$ **and** gh: $src_B\ g = trg_B\ h$
  have *1*: $ide\ (P\ f) \wedge ide\ (P\ g) \wedge ide\ (P\ h)$
    using $f\ g\ h$ *P.preserves-ide* **by** *simp*
  have $P\ \mathrm{a}_B[f,\ g,\ h] \cdot \Phi_P\ (f \star_B\ g,\ h) \cdot (\Phi_P\ (f,\ g) \star P\ h) =$
      $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B \Phi_P\ (f \star_B\ g,\ h) \cdot_B (\Phi_P\ (f,\ g) \star_B\ P\ h)$
    using $f\ g\ h\ fg\ gh$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *P.FF-def*
    by (*intro comp-eqI hcomp-eqI seqI hseqI′*) *auto*
  also have ... $= CMP\ f\ (g \star_B\ h) \cdot_B (P\ f \star_B\ CMP\ g\ h) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
  using $f\ g\ h\ fg\ gh$ *CMP-coherence* *CMP.map-simp-ide* $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$
      Φ_P*-def*
    by *simp*
  also have ... $= \Phi_P\ (f,\ g \star_B\ h) \cdot_B (P\ f \star_B\ \Phi_P\ (g,\ h)) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
    using $f\ g\ h\ fg\ gh$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ Φ_P*-def* **by** *simp*
  also have ... $= \Phi_P\ (f,\ g \star_B\ h) \cdot (P\ f \star_B\ \Phi_P\ (g,\ h)) \cdot \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
  proof −
    have *2*: $arr\ ((P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h])$
      using $f\ g\ h\ fg\ gh$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ *P.FF-def* **by** *auto*
    moreover have $(P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h] =$
        $(P\ f \star_B\ \Phi_P\ (g,\ h)) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
      using $f\ g\ h\ fg\ gh$ *2* *comp-eqI hcomp-eqI assoc-simp* **by** *auto*
    moreover have *3*: $arr\ (P\ f \star_B\ \Phi_P\ (g,\ h))$
      using $f\ g\ h\ fg\ gh$ *arr-hcompI*
      by (*intro arr-hcompI hseqI′*) *auto*
    moreover have $B.dom\ (P\ f \star_B\ \Phi_P\ (g,\ h)) = P\ f \star_B\ P\ g \star_B\ P\ h$
    proof −
      have $B.dom\ (P\ f \star_B\ \Phi_P\ (g,\ h)) = P\ f \star_B\ B.dom\ (\Phi_P\ (g,\ h))$

using *f g h fg gh 3 B.hcomp-simps*(*3*)
by (*metis* (*no-types*, *lifting*) *1 arrE ideD*(*1*) *ideD*(*2*) *dom-char*$_{SbC}$)
also have ... = $P\ f \star_B P\ g \star_B P\ h$
using *g h gh dom-char*$_{SbC}$ *hcomp-char* Φ_P*-simps*(*1*−*4*) **by** *auto*
finally show *?thesis* **by** *blast*
qed
moreover have $B.dom\ (\Phi_P\ (f,\ g \star_B h)) =$
$B.cod\ ((P\ f \star_B \Phi_P\ (g,\ h)) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h])$
proof −
have $B.dom\ (\Phi_P\ (f,\ g \star_B h)) = dom\ (\Phi_P\ (f,\ g \star_B h))$
using *f g h fg gh B.VV.arr-char*$_{SbC}$ *dom-char*$_{SbC}$ **by** *simp*
also have ... = $cod\ ((P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h])$
using *f g h fg gh 2 VV.arr-char*$_{SbC}$ Φ_P*-simps* **by** *simp*
also have ... = $B.cod\ ((P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h])$
using *2 cod-char*$_{SbC}$ **by** *presburger*
also have ... = $B.cod\ ((P\ f \star_B \Phi_P\ (g,\ h)) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h])$
proof −
have $(P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h] = (P\ f \star_B \Phi_P\ (g,\ h)) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
using *f g h fg gh 2 comp-eqI hcomp-eqI assoc-simp* **by** *auto*
thus *?thesis* **by** *presburger*
qed
finally show *?thesis* **by** *blast*
qed
moreover have $B.cod\ \mathrm{a}_B[P\ f,\ P\ g,\ P\ h] = P\ f \star_B P\ g \star_B P\ h$
using *f g h fg gh 1 ide-char*$_{SbC}$ **by** *auto*
ultimately show *?thesis*
using *f g h fg gh B.VV.arr-char*$_{SbC}$ *assoc-simp assoc-simps*(*1*) *dom-char*$_{SbC}$ *cod-char*$_{SbC}$
by (*intro comp-eqI′ seqI arr-compI arr-hcompI*) *auto*
qed
also have ... = $\Phi_P\ (f,\ g \star_B h) \cdot (P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h]$
using *f g h fg gh assoc-simp hcomp-eqI* Φ_P*-simps*(*1*) **by** *auto*
finally show $P\ \mathrm{a}_B[f,\ g,\ h] \cdot \Phi_P\ (f \star_B g,\ h) \cdot (\Phi_P\ (f,\ g) \star P\ h) =$
$\Phi_P\ (f,\ g \star_B h) \cdot (P\ f \star \Phi_P\ (g,\ h)) \cdot \mathrm{a}[P\ f,\ P\ g,\ P\ h]$
by *blast*
qed

lemma *pseudofunctor-prj*:
shows *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ comp\ hcomp\ \mathrm{a}\ \mathrm{i}_B\ src\ trg\ P\ \Phi_P$
..

We need an explicit formula for the unit constraints for P.

lemma *prj-unit-char*:
assumes *B.obj a*
shows $prj.unit\ a = (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)$
proof −

We eventually need one of the triangle identities from the following interpretation. However in the meantime, it contains a lot of simps that make an otherwise arduous calculation much easier. So interpret it up front.

interpret *adjoint-equivalence-in-bicategory* V_B H_B a_B i_B src_B trg_B ‹$e\ a$› ‹$d\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
  using *assms chosen-adjoint-equivalence*(*1*) **by** *simp*

let $?x = (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)$
have *x*: «$?x : P.map_0\ a \Rightarrow_B P\ a$»
  using *assms P-def* $P.map_0$*-def* P-map_0*-simp*
  by (*intro B.comp-in-homI′*) *auto*
have $?x = prj.unit\ a$
proof (*intro prj.unit-eqI*)
  show $B.obj\ a$ **by** *fact*
  have *a-da*: «$a \star_B d\ a : P_0\ a \rightarrow_B a$» $\wedge$ $B.ide\ (a \star_B d\ a)$
    using *assms B.obj-simps* **by** *auto*
  show x_S: «$?x : P.map_0\ a \Rightarrow P\ a$»
    using *assms x* P-map_0*-simp* P-$simps_B$(*1*) arr-$char_{SbC}$ *B.arrI*
      equivalence-data-$simps_B$(*6*) *counit-simps*(*1*,*4*) *B.obj-simps*(*1*)
      B.src.preserves-arr B.vconn-implies-hpar(*1*−*4*)
    by (*metis* (*no-types*, *lifting*) *P-simps*(*1*) *in-hom-*$char_{SbC}$)
  show *iso ?x*
    using *assms* x_S *B.isos-compose* iso-$char_{SbC}$ arr-$char_{SbC}$ **by** *auto*
  have $*$: «$?x : P_0\ a \rightarrow P_0\ a$»
    using *assms* x_S P_0*-props vconn-implies-hpar*(*1*−*2*)
    by (*intro in-hhomI*) *auto*
  show $?x \cdot \mathrm{i}_B[P.map_0\ a] = (P\ \mathrm{i}_B[a] \cdot \Phi_P\ (a,\ a)) \cdot (?x \star ?x)$
  proof −
    have *0*: «$d\ a \star_B e\ a \star_B a \star_B d\ a : P_0\ a \rightarrow_B a$»
      using *assms* **by** *force*
    have *1*: $B.seq\ (B.inv\ (\eta\ a) \star_B a \star_B d\ a)\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]$
      using *assms* **by** (*elim B.objE*, *intro B.seqI*) *auto*
    have $(P\ \mathrm{i}_B[a] \cdot \Phi_P\ (a,\ a)) \cdot (?x \star ?x) = P\ \mathrm{i}_B[a] \cdot \Phi_P\ (a,\ a) \cdot (?x \star ?x)$
      using *comp-assoc* **by** *simp*
    also have ... $= P\ \mathrm{i}_B[a] \cdot (CMP\ a\ a \cdot (P\ a \star P\ a)) \cdot (?x \star ?x)$
      unfolding Φ_P*-def CMP.map-def*
      using *assms* $B.VV.arr$-$char_{SbC}$ $B.VV.cod$-$char_{SbC}$ *P.FF-def* **by** *auto*
    also have ... $= P\ \mathrm{i}_B[a] \cdot (CMP\ a\ a \cdot (P\ a \star_B P\ a)) \cdot (?x \star_B ?x)$
      using *assms* x_S *hcomp-char src-def trg-def* **by** *auto*
    also have ... $= P\ \mathrm{i}_B[a] \cdot_B (CMP\ a\ a \cdot_B (P\ a \star_B P\ a)) \cdot_B (?x \star_B ?x)$
    proof −
      have «$P\ \mathrm{i}_B[a] \cdot (CMP\ a\ a \cdot (P\ a \star_B P\ a)) \cdot (?x \star_B ?x) : P_0\ a \star P_0\ a \Rightarrow P\ a$»
      proof (*intro comp-in-homI*)
        show «$?x \star_B ?x : P_0\ a \star P_0\ a \Rightarrow P\ a \star P\ a$»
        proof −
          have «$?x \star ?x : P_0\ a \star P_0\ a \Rightarrow P\ a \star P\ a$»
            using *assms* x_S $*$ P_0*-props* $P.map_0$*-simps*(*3*)
            by (*intro hcomp-in-vhom*) *auto*
          moreover have $?x \star ?x = ?x \star_B ?x$
            using x_S $*$ **by** (*intro hcomp-eqI*) *auto*
          ultimately show *?thesis* **by** *simp*
        qed
        show «$P\ a \star_B P\ a : P\ a \star P\ a \Rightarrow P\ a \star P\ a$»

using *assms hcomp-eqI* **by** *fastforce*
show «$CMP\ a\ a : P\ a \star P\ a \Rightarrow P\ (a \star_B a)$»
using *assms CMP-in-hom(2)* **by** *auto*
show «$P\ \mathrm{i}_B[a] : P\ (a \star_B a) \Rightarrow P\ a$»
using *assms* **by** *auto*
qed
thus *?thesis*
by (*intro comp-eqI hcomp-eqI*) *auto*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a] \cdot_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a)) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$(P\ a \star_B P\ a) \cdot_B$
$(?x \star_B ?x)$
using *assms B.comp-assoc CMP-def* **by** *auto*
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$(e\ a \star_B a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$(P\ a \star_B P\ a) \cdot_B$
$(?x \star_B ?x)$
using *assms 1 B.whisker-left B.comp-assoc* **by** *fastforce*
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$(e\ a \star_B a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((P\ a \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]))) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$
using *assms B.interchange B.comp-assoc* **by** *simp*
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$((e\ a \star_B a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$

$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$

proof −
have $(P\ a \star_B P\ a) \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]))$
$= ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]))$
unfolding *P-def*
using *assms B.comp-cod-arr* [*of* $(e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
$(e\ a \star_B a \star_B d\ a) \star_B (e\ a \star_B a \star_B d\ a)$]
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$((e\ a \star_B a \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$

proof −
have $(e\ a \star_B a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
$= e\ a \star_B (a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B (a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
proof −
have $B.seq\ (a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a)\ (a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
using *assms* **by** (*elim B.objE*, *intro B.seqI*) *auto*
thus *?thesis*
using *assms B.whisker-left* **by** *simp*
qed
also have ... $= e\ a \star_B a \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]$
using *assms 1 B.whisker-left* **by** *fastforce*
finally have $(e\ a \star_B a \star_B B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
$= e\ a \star_B a \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$(\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a)) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$

$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$
proof −
have $(e\ a \star_B a \star_B$
$(B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a]$
$= \mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
using *assms 1 a-da P-def*
B.assoc-naturality
$[of\ e\ a\ a\ (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]]$
by *fastforce*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$(((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a \star_B P\ a]) \cdot_B$
$(e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a] \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$
proof −
have *1*: $B.ide\ (e\ a) \wedge B.ide\ a \wedge B.ide\ (d\ a) \wedge B.ide\ (P\ a)$
using *assms ide-char$_{SbC}$ P.preserves-ide* **by** *auto*
have *2*: $src_B\ (e\ a) = trg_B\ a \wedge src_B\ a = trg_B\ (d\ a) \wedge src_B\ (d\ a) = trg_B\ (P\ a)$
using *assms* **by** *auto*
have $((e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]) \cdot_B (\mathrm{a}_B[e\ a,\ a,\ d\ a] \star_B P\ a)$
$= \mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a]$
using *assms 1 2 B.pentagon B.comp-assoc* **by** *force*
hence $(e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]$
$= (\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a]) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a)$
using *assms 1 2*
B.invert-side-of-triangle(2)
$[of\ \mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a]$
$(e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]$
$\mathrm{a}_B[e\ a,\ a,\ d\ a] \star_B P\ a]$
by *fastforce*
hence $\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B (\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a)$
$= \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B (e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]$
using *assms 1 2 B.invert-side-of-triangle(1) B.comp-assoc* **by** *fastforce*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$

$(e\ a \star_B\ a \star_B\ \mathrm{l}_B[a \star_B\ d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$(e\ a \star_B\ a \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]) \cdot_B$
$((e\ a \star_B\ \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a \star_B\ d\ a,\ P\ a]) \cdot_B$
$((e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B\ B.inv\ (\varepsilon\ a))$

proof −
have $B.arr\ (e\ a) \wedge B.arr\ a$
using *assms* **by** *auto*
moreover have $B.dom\ (e\ a) = e\ a \wedge B.dom\ a = a \wedge B.cod\ a = a \wedge$
$B.cod\ (e\ a) = e\ a$
using *assms* **by** *auto*
moreover have $B.seq\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a)\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]$
using *assms* **by** (*elim B.objE*, *intro B.seqI*) *auto*
moreover have $src_B\ a =$
$trg_B\ ((B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a])$
using *assms a-da trg-vcomp* **by** *fastforce*
moreover have $src_B\ a = a \wedge trg_B\ a = a$
using *assms* **by** *auto*
moreover have $B.dom\ ((B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a])$
$= d\ a \star_B\ e\ a \star_B\ a \star_B\ d\ a$
using *assms a-da* **by** *fastforce*
moreover have $B.cod\ ((B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a])$
$= a \star_B\ a \star_B\ d\ a$
using *assms a-da* **by** *fastforce*
ultimately have $((e\ a \star_B\ a) \star_B$
$(B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]) \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a \star_B\ P\ a]$
$= \mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$(e\ a \star_B\ a \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a])$
unfolding *P-def*
using *assms*
B.assoc′-naturality
[*of* $e\ a\ a\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]$]
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*

qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B\ \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B\ a \star_B\ \mathrm{l}_B[a \star_B\ d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$((e\ a \star_B\ a \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B\ P\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P\ a] \cdot_B$

$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$

proof −
have *1*: $B.ide\ (e\ a) \wedge B.ide\ a \wedge B.ide\ (d\ a) \wedge B.ide\ (P\ a)$
using *assms ide-char*$_{SbC}$ *P.preserves-ide* **by** *auto*
have *2*: $src_B\ (e\ a) = trg_B\ a \wedge src_B\ a = trg_B\ (d\ a) \wedge src_B\ (d\ a) = trg_B\ (P\ a)$
using *assms* **by** *auto*
have $((e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]) \cdot_B (\mathrm{a}_B[e\ a,\ a,\ d\ a] \star_B P\ a)$
$= \mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a]$
using *assms 1 2 B.pentagon B.comp-assoc* **by** *fastforce*
hence $(e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]$
$= \mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a)$
using *assms 1 2 P.preserves-ide B.comp-assoc*
B.invert-side-of-triangle(*2*)
[*of* $\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a] \cdot_B \mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a]$
$(e\ a \star_B \mathrm{a}_B[a,\ d\ a,\ P\ a]) \cdot_B \mathrm{a}_B[e\ a,\ a \star_B d\ a,\ P\ a]$
$\mathrm{a}_B[e\ a,\ a,\ d\ a] \star_B P\ a]$
by *force*
thus *?thesis*
using *B.comp-assoc* **by** *simp*

qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$

proof −
have $(e\ a \star_B a \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ d\ a \star_B P\ a]$
$= \mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B ((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
proof −
have *1*: $B.seq\ (B.inv\ (\eta\ a) \star_B a \star_B d\ a)\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]$
using *assms* **by** (*elim B.objE*, *intro B.seqI*) *auto*
moreover have $src_B\ a =$
$trg_B\ ((B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
using *a-da* **by** *force*
moreover have $B.arr\ a \wedge B.dom\ a = a \wedge B.cod\ a = a \wedge src_B\ a = a \wedge trg_B\ a = a$
using *assms* **by** *auto*
moreover have $B.dom\ ((B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$

$= d\ a \star_B e\ a \star_B a \star_B d\ a$
using *a-da* **by** *auto*
moreover have $B.cod\ ((B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a])$
$= a \star_B a \star_B d\ a$
using *a-da* **by** *auto*
ultimately show *?thesis*
unfolding *P-def*
using *assms*
B.assoc-naturality
$[of\ e\ a\ a\ (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]]$
by *auto*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have $... = P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P\ a] \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B$
$B.inv\ (\varepsilon\ a))) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
proof −
have $((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$
$= ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
proof −
have $((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B B.inv\ (\varepsilon\ a))$
$= (((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$((e\ a \star_B d\ a) \star_B B.inv\ (\varepsilon\ a))) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
using *assms B.comp-arr-dom B.comp-cod-arr B.comp-assoc*
B.interchange $[of\ e\ a \star_B d\ a\ B.inv\ (\varepsilon\ a)\ B.inv\ (\varepsilon\ a)\ P_0\ a]$
by *simp*
also have $... = ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B (e\ a \star_B d\ a) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B$
$B.inv\ (\varepsilon\ a)) \cdot_B (B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
using *assms B.interchange* **by** *simp*
also have $... = ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
using *assms B.comp-arr-dom* **by** *simp*
finally show *?thesis* **by** *simp*

```
  qed
  thus ?thesis
    using B.comp-assoc by simp
qed
also have ... = P i_B[a] ·_B
                (e a ⋆_B a_B^-1[a, a, d a]) ·_B
                (e a ⋆_B a ⋆_B l_B[a ⋆_B d a]) ·_B
                a_B[e a, a, a ⋆_B a ⋆_B d a] ·_B
                a_B^-1[e a, a, a ⋆_B a ⋆_B d a] ·_B
                a_B[e a, a, a ⋆_B a ⋆_B d a] ·_B
                ((e a ⋆_B a) ⋆_B (B.inv (η a) ⋆_B a ⋆_B d a) ·_B
                a_B^-1[d a, e a, a ⋆_B d a]) ·_B
                (a_B[e a ⋆_B a, d a, P a] ·_B
                (((e a ⋆_B a) ⋆_B d a) ⋆_B (e a ⋆_B l_B^-1[d a]) ·_B B.inv (ε a))) ·_B
                (a_B^-1[e a, a, d a] ·_B (e a ⋆_B l_B^-1[d a]) ⋆_B P_0 a) ·_B
                (B.inv (ε a) ⋆_B P_0 a)
proof -
  have (a_B^-1[e a, a, d a] ⋆_B P a) ·_B ((e a ⋆_B l_B^-1[d a]) ⋆_B (e a ⋆_B l_B^-1[d a]) ·_B
      B.inv (ε a))
        = (a_B^-1[e a, a, d a] ·_B (e a ⋆_B l_B^-1[d a]) ⋆_B P a ·_B (e a ⋆_B l_B^-1[d a]) ·_B
          B.inv (ε a))
  proof -
    have B.seq a_B^-1[e a, a, d a] (e a ⋆_B l_B^-1[d a])
      using assms a-da by (elim B.objE, intro B.seqI) auto
    moreover have B.seq (P a) ((e a ⋆_B l_B^-1[d a]) ·_B B.inv (ε a))
      using assms a-da P-def by auto
    ultimately show ?thesis
      using assms B.interchange by simp
  qed
  also have ... = (a_B^-1[e a, a, d a] ·_B (e a ⋆_B l_B^-1[d a]) ⋆_B (e a ⋆_B l_B^-1[d a]) ·_B
                  B.inv (ε a))
  proof -
    have P a ·_B (e a ⋆_B l_B^-1[d a]) ·_B B.inv (ε a) = (e a ⋆_B l_B^-1[d a]) ·_B B.inv (ε a)
      using assms x P-def B.comp-cod-arr by blast
    moreover have a_B^-1[e a, a, d a] ·_B (e a ⋆_B l_B^-1[d a]) = r_B^-1[e a] ⋆_B d a
    proof -
      have a_B^-1[e a, a, d a] ·_B (e a ⋆_B l_B^-1[d a])
            = B.inv ((e a ⋆_B l_B[d a]) ·_B a_B[e a, a, d a])
      proof -
        have B.inv ((e a ⋆_B l_B[d a]) ·_B a_B[e a, a, d a])
              = B.inv a_B[e a, a, d a] ·_B B.inv (e a ⋆_B l_B[d a])
        proof -
          have B.seq (e a ⋆_B l_B[d a]) a_B[e a, a, d a]
          proof -
            have B.iso ((e a ⋆_B l_B[d a]) ·_B a_B[e a, a, d a])
              using assms by (elim B.objE, intro B.isos-compose) auto
            thus ?thesis
              by blast
          qed
```

moreover have $B.iso\ \mathrm{a}_B[e\ a,\ a,\ d\ a]$
using *assms* **by** *force*
ultimately show *?thesis*
using *assms B.inv-comp* **by** *auto*
qed
also have ... $= \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
using *assms*
by (*elim B.objE*) (*simp add*: *assms*)
finally show *?thesis* **by** *simp*
qed
also have ... $= B.inv\ (\mathrm{r}_B[e\ a] \star_B\ d\ a)$
using *assms B.triangle* [*of d a e a*] **by** *simp*
also have ... $= \mathrm{r}_B{}^{-1}[e\ a] \star_B\ d\ a$
using *assms* **by** *simp*
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
qed
also have ... $= (((e\ a \star_B\ a) \star_B\ d\ a) \cdot_B \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]) \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B$
$(e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \cdot_B P_0\ a$
using *assms 0 B.comp-cod-arr B.comp-arr-dom*
by (*elim B.objE*) *auto*
also have ... $= ((e\ a \star_B\ a) \star_B\ d\ a) \cdot_B \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B P_0\ a$
using *B.comp-assoc* **by** *simp*
also have ... $= (((e\ a \star_B\ a) \star_B\ d\ a) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B P_0\ a)$
proof −
have $B.seq\ ((e\ a \star_B\ a) \star_B\ d\ a)\ (\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]))$
using *assms 0 a-da*
by (*elim B.objE*, *intro B.seqI*) *auto*
moreover have $B.seq\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))\ (P_0\ a)$
using *assms 0*
apply (*intro B.seqI*, *auto simp add*: *B.obj-simps*(*5*) P_0*-props*(*5*))
by (*meson B.in-hhomE B.obj-simps*(*1*) *a-da*)
ultimately show *?thesis*
using *assms B.interchange* **by** *blast*
qed
finally have $(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B P\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
$= (((e\ a \star_B\ a) \star_B\ d\ a) \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B P_0\ a)$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B\ a \star_B \mathrm{l}_B[a \star_B\ d\ a]) \cdot_B$

$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$(((e\ a \star_B\ a) \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]) \cdot_B$
$((e\ a \star_B\ a) \star_B\ d\ a \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a))) \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P_0\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \star_B\ P_0\ a) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B\ P_0\ a)$

proof −
have $\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P\ a] \cdot_B$
$(((e\ a \star_B\ a) \star_B\ d\ a) \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a))$
$= ((e\ a \star_B\ a) \star_B\ d\ a \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a)) \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P_0\ a]$
proof −
have $B.hseq\ (e\ a)\ a$
using *assms* **by** *force*
moreover have $src_B\ (d\ a) = trg_B\ ?x$
using *assms B.trg-vcomp* [*of* $e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]$ $B.inv\ (\varepsilon\ a)$] **by** *simp*
moreover have $B.cod\ ((e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a)) = P\ a$
using *assms x* **by** *blast*
ultimately show *?thesis*
using *assms B.assoc-naturality* [*of* $e\ a \star_B\ a\ d\ a$ *?x*] **by** *auto*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B\ \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B\ a \star_B\ \mathrm{l}_B[a \star_B\ d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a] \cdot_B$
$((e\ a \star_B\ a) \star_B$
$(B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a] \cdot_B$
$(d\ a \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a))) \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P_0\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \star_B\ P_0\ a) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B\ P_0\ a)$

proof −
have $((e\ a \star_B\ a) \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a]) \cdot_B$
$((e\ a \star_B\ a) \star_B\ d\ a \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a))$
$= (e\ a \star_B\ a) \star_B\ (B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a] \cdot_B$
$(d\ a \star_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B\ B.inv\ (\varepsilon\ a))$
proof −
have $B.ide\ (e\ a \star_B\ a)$
using *assms* **by** *force*
moreover have $B.seq\ ((B.inv\ (\eta\ a) \star_B\ a \star_B\ d\ a) \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B\ d\ a])$

$(d\ a \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
using *a-da B.whisker-left* **by** *auto*
ultimately show *?thesis*
using *assms B.whisker-left B.comp-assoc* **by** *fastforce*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B$
$(B.inv\ (\eta\ a) \star_B d\ a) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a] \cdot_B$
$(d\ a \star_B B.inv\ (\varepsilon\ a))) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P_0\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B P_0\ a) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B P_0\ a)$
proof −
have $(B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B$
$(d\ a \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
$= (a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B (B.inv\ (\eta\ a) \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a] \cdot_B$
$(d\ a \star_B B.inv\ (\varepsilon\ a))$
proof −
have $(B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B$
$(d\ a \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
$= (B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B (\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B$
$(d\ a \star_B e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B (d\ a \star_B B.inv\ (\varepsilon\ a))$
using *assms B.whisker-left B.comp-assoc* **by** *simp*
also have ... $= ((B.inv\ (\eta\ a) \star_B a \star_B d\ a) \cdot_B ((d\ a \star_B e\ a) \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B$
$\mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a] \cdot_B (d\ a \star_B B.inv\ (\varepsilon\ a))$
using *assms B.assoc′-naturality* [*of d a e a* $\mathrm{l}_B{}^{-1}[d\ a]$] *B.comp-assoc* **by** *simp*
also have ... $= (B.inv\ (\eta\ a) \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a] \cdot_B$
$(d\ a \star_B B.inv\ (\varepsilon\ a))$
using *assms B.interchange* [*of B.inv* $(\eta\ a)$ $d\ a \star_B e\ a\ a \star_B d\ a\ \mathrm{l}_B{}^{-1}[d\ a]$]
B.comp-arr-dom B.comp-cod-arr
by *simp*
also have ... $= (a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B (B.inv\ (\eta\ a) \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a] \cdot_B$
$(d\ a \star_B B.inv\ (\varepsilon\ a))$
using *assms B.interchange* [*of a B.inv* $(\eta\ a)$ $\mathrm{l}_B{}^{-1}[d\ a]$ $d\ a$]
B.comp-arr-dom B.comp-cod-arr B.comp-assoc
by *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed

also have ... $= P\ \mathrm{i}_B[a]\ \cdot_B$
$(e\ a \star_B\ \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a])\ \cdot_B$
$(e\ a \star_B\ a \star_B\ \mathrm{l}_B[a \star_B\ d\ a])\ \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$((e\ a \star_B\ a) \star_B\ (a \star_B\ \mathrm{l}_B{}^{-1}[d\ a])\ \cdot_B\ \mathrm{l}_B{}^{-1}[d\ a]\ \cdot_B\ \mathrm{r}_B[d\ a])\ \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P_0\ a]\ \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]\ \cdot_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \star_B\ P_0\ a)\ \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B\ P_0\ a)$
proof −
have $(B.inv\ (\eta\ a) \star_B\ d\ a)\ \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a]\ \cdot_B\ (d\ a \star_B\ B.inv\ (\varepsilon\ a))$
$= \mathrm{l}_B{}^{-1}[d\ a]\ \cdot_B\ \mathrm{r}_B[d\ a]$
proof −
have $(B.inv\ (\eta\ a) \star_B\ d\ a)\ \cdot_B\ \mathrm{a}_B{}^{-1}[d\ a,\ e\ a,\ d\ a]\ \cdot_B\ (d\ a \star_B\ B.inv\ (\varepsilon\ a))$
$= B.inv\ (\eta\ a \star_B\ d\ a)\ \cdot_B\ B.inv\ \mathrm{a}_B[d\ a,\ e\ a,\ d\ a]\ \cdot_B\ B.inv\ (d\ a \star_B\ \varepsilon\ a)$
using *assms* **by** *simp*
also have ... $= B.inv\ ((d\ a \star_B\ \varepsilon\ a)\ \cdot_B\ \mathrm{a}_B[d\ a,\ e\ a,\ d\ a]\ \cdot_B\ (\eta\ a \star_B\ d\ a))$
proof −
have $B.iso\ ((d\ a \star_B\ \varepsilon\ a)\ \cdot_B\ \mathrm{a}_B[d\ a,\ e\ a,\ d\ a]\ \cdot_B\ (\eta\ a \star_B\ d\ a))$
using *assms* **by** (*intro B.isos-compose*) *auto*
thus *?thesis*
using *assms B.inv-comp-left B.comp-assoc* **by** *auto*
qed
also have ... $= B.inv\ (\mathrm{r}_B{}^{-1}[d\ a]\ \cdot_B\ \mathrm{l}_B[d\ a])$
using *assms triangle-right* **by** *simp*
also have ... $= \mathrm{l}_B{}^{-1}[d\ a]\ \cdot_B\ \mathrm{r}_B[d\ a]$
using *assms B.inv-comp* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (P\ \mathrm{i}_B[a]\ \cdot_B$
$(e\ a \star_B\ \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a])\ \cdot_B$
$(e\ a \star_B\ a \star_B\ \mathrm{l}_B[a \star_B\ d\ a])\ \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B\ a \star_B\ d\ a]\ \cdot_B$
$((e\ a \star_B\ a) \star_B\ (a \star_B\ \mathrm{l}_B{}^{-1}[d\ a])\ \cdot_B\ \mathrm{l}_B{}^{-1}[d\ a]\ \cdot_B\ \mathrm{r}_B[d\ a])\ \cdot_B$
$\mathrm{a}_B[e\ a \star_B\ a,\ d\ a,\ P_0\ a]\ \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]\ \cdot_B\ (e\ a \star_B\ \mathrm{l}_B{}^{-1}[d\ a]) \star_B\ P_0\ a)\ \cdot_B$
$\mathrm{r}_B{}^{-1}[e\ a \star_B\ d\ a])\ \cdot_B$
$B.inv\ (\varepsilon\ a)\ \cdot_B$
$\mathrm{i}_B[P_0\ a]$
proof −
have $\mathrm{r}_B[e\ a \star_B\ d\ a]\ \cdot_B\ (B.inv\ (\varepsilon\ a) \star_B\ P_0\ a) = B.inv\ (\varepsilon\ a)\ \cdot_B\ \mathrm{i}_B[P_0\ a]$
using *assms B.runit-naturality* [*of B.inv* $(\varepsilon\ a)$] *B.unitor-coincidence* P_0*-props*
by *simp*

hence $B.inv\ (\varepsilon\ a) \star_B P_0\ a = \mathrm{r}_B{}^{-1}[e\ a \star_B d\ a] \cdot_B B.inv\ (\varepsilon\ a) \cdot_B \mathrm{i}_B[P_0\ a]$
using *assms* P_0*-props*(*5*) *B.invert-side-of-triangle*(*1*) **by** *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \cdot_B \mathrm{i}_B[P_0\ a]$
proof −
have P_0*-a*: $B.obj\ (P_0\ a) \wedge B.arr\ (P_0\ a)$
using *assms a-da* **by** *fastforce*
have *i-a*: $B.\mathfrak{r}\ a = \mathrm{i}_B[a]$
using *assms B.unitor-coincidence B.*$\mathfrak{r}$*-ide-simp B.obj-simps*
by (*metis* (*no-types*, *lifting*) *B.objE*)
have $P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B \mathrm{l}_B{}^{-1}[d\ a] \cdot_B \mathrm{r}_B[d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P_0\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B P_0\ a) \cdot_B$
$\mathrm{r}_B{}^{-1}[e\ a \star_B d\ a]$
$= e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]$
proof −
have $P\ \mathrm{i}_B[a] \cdot_B$
$(e\ a \star_B \mathrm{a}_B{}^{-1}[a,\ a,\ d\ a]) \cdot_B$
$(e\ a \star_B a \star_B \mathrm{l}_B[a \star_B d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a \star_B a \star_B d\ a] \cdot_B$
$((e\ a \star_B a) \star_B (a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B \mathrm{l}_B{}^{-1}[d\ a] \cdot_B \mathrm{r}_B[d\ a]) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ P_0\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B P_0\ a) \cdot_B$
$\mathrm{r}_B{}^{-1}[e\ a \star_B d\ a]$
$= \{\!\!\{(\langle e\ a\rangle \star \mathbf{r}[\langle a\rangle_0] \star \langle d\ a\rangle) \cdot$
$(\langle e\ a\rangle \star \mathbf{a}^{-1}[\langle a\rangle_0,\ \langle a\rangle_0,\ \langle d\ a\rangle]) \cdot$
$(\langle e\ a\rangle \star \langle a\rangle_0 \star \mathbf{l}[\langle a\rangle_0 \star \langle d\ a\rangle]) \cdot$
$\mathbf{a}[\langle e\ a\rangle,\ \langle a\rangle_0,\ \langle a\rangle_0 \star \langle a\rangle_0 \star \langle d\ a\rangle] \cdot$
$\mathbf{a}^{-1}[\langle e\ a\rangle,\ \langle a\rangle_0,\ \langle a\rangle_0 \star \langle a\rangle_0 \star \langle d\ a\rangle] \cdot$
$\mathbf{a}[\langle e\ a\rangle,\ \langle a\rangle_0,\ \langle a\rangle_0 \star \langle a\rangle_0 \star \langle d\ a\rangle] \cdot$
$((\langle e\ a\rangle \star \langle a\rangle_0) \star (\langle a\rangle_0 \star \mathbf{l}^{-1}[\langle d\ a\rangle]) \cdot \mathbf{l}^{-1}[\langle d\ a\rangle] \cdot \mathbf{r}[\langle d\ a\rangle]) \cdot$
$\mathbf{a}[\langle e\ a\rangle \star \langle a\rangle_0,\ \langle d\ a\rangle,\ \langle P_0\ a\rangle_0] \cdot$
$(\mathbf{a}^{-1}[\langle e\ a\rangle,\ \langle a\rangle_0,\ \langle d\ a\rangle] \cdot (\langle e\ a\rangle \star \mathbf{l}^{-1}[\langle d\ a\rangle]) \star \langle P_0\ a\rangle_0) \cdot$
$\mathbf{r}^{-1}[\langle e\ a\rangle \star \langle d\ a\rangle]\}\!\!\}$
unfolding *P-def*
using *assms B.obj-def* [*of a*] P_0*-a i-a B.*α*-def B.*α'*.map-ide-simp*
B.VVV.ide-char$_{SbC}$ *B.VVV.arr-char*$_{SbC}$ *B.VV.ide-char*$_{SbC}$ *B.VV.arr-char*$_{SbC}$
B.$\mathfrak{l}$*-ide-simp B.*$\mathfrak{r}$*-ide-simp*
by (*elim B.objE*) *simp*

also have ... $= \{\!|\langle e\ a\rangle \star \mathbf{l}^{-1}[\langle d\ a\rangle]|\!\}$
using *assms* P_0*-a* **by** (*intro EV.eval-eqI*) *auto*
also have ... $= e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]$
using *assms P-def B.*ℓ*-ide-simp* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= ?x \cdot \mathrm{i}_B[P.map_0\ a]$
proof −
have $B.arr\ \mathrm{i}_B[P.map_0\ a] \wedge src_B\ \mathrm{i}_B[P.map_0\ a] \in Obj \wedge trg_B\ \mathrm{i}_B[P.map_0\ a] \in Obj \wedge$
$P_0\ a \cdot_B P_0\ a \in Obj \wedge B.cod\ \mathrm{i}_B[P.map_0\ a] = P_0\ a$
using *assms* P_0*-props arr-char*$_{SbC}$ *unit-simps*(*1*)
apply *auto*
using *obj-char*
by (*metis* (*no-types*, *lifting*) *B.comp-ide-self B.objE*)
thus *?thesis*
using *assms comp-def B.comp-assoc* P_0*-props* [*of a*] **by** *simp*
qed
finally show *?thesis* **by** *argo*
qed
qed
thus *?thesis* **by** *simp*
qed

sublocale *PoE*: *composite-pseudofunctor*
comp hcomp a i_B *src trg* V_B H_B a_B i_B src_B trg_B
comp hcomp a i_B *src trg*
$E\ \Phi_E\ P\ \Phi_P$
..
sublocale *EoP*: *composite-pseudofunctor*
V_B H_B a_B i_B src_B trg_B *comp hcomp* a i_B *src trg* V_B H_B a_B i_B src_B trg_B
$P\ \Phi_P\ E\ \Phi_E$
..
sublocale I_C: *identity-pseudofunctor* V_B H_B a_B i_B src_B trg_B **..**
sublocale I_S: *identity-pseudofunctor comp hcomp* a i_B *src trg* **..**

no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

abbreviation (*input*) $unit_0$
where $unit_0 \equiv e$

definition $unit_1$
where $unit_1\ f = (e\ (trg_B\ f) \star_B \mathrm{r}_B[f]) \cdot_B$
$\mathrm{a}_B[e\ (trg_B\ f),\ f,\ src_B\ f] \cdot_B$
$((e\ (trg_B\ f) \star_B f) \star_B B.inv\ (\eta\ (src_B\ f))) \cdot_B$
$\mathrm{a}_B[e\ (trg_B\ f) \star_B f,\ d\ (src_B\ f),\ e\ (src_B\ f)] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ (trg_B\ f),\ f,\ d\ (src_B\ f)] \star_B e\ (src_B\ f))$

abbreviation (*input*) η_0
where $\eta_0 \equiv unit_0$

abbreviation (*input*) η_1
where $\eta_1 \equiv unit_1$

lemma $unit_1$*-in-hom* [*intro*]:
assumes $B.ide\ f$
shows «$\eta_1\ f : src_B\ f \rightarrow_B P_0\ (trg_B\ f)$»
and «$\eta_1\ f : (e\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)) \star_B e\ (src_B\ f) \Rightarrow_B e\ (trg_B\ f) \star_B f$»
proof −
  show «$\eta_1\ f : (e\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)) \star_B e\ (src_B\ f) \Rightarrow_B e\ (trg_B\ f) \star_B f$»
    using *assms* $unit_1$*-def* **by** *force*
  thus «$\eta_1\ f : src_B\ f \rightarrow_B P_0\ (trg_B\ f)$»
    using *assms B.vconn-implies-hpar*(*1*−*2*) **by** *auto*
qed

lemma $unit_1$*-simps* [*simp*]:
assumes $B.ide\ f$
shows $B.arr\ (\eta_1\ f)$
and $src_B\ (\eta_1\ f) = src_B\ f$ **and** $trg_B\ (\eta_1\ f) = P_0\ (trg_B\ f)$
and $B.dom\ (\eta_1\ f) = (e\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)) \star_B e\ (src_B\ f)$
and $B.cod\ (\eta_1\ f) = e\ (trg_B\ f) \star_B f$
and $B.iso\ (\eta_1\ f)$
  using *assms* $unit_1$*-in-hom*
      apply *auto*
  unfolding $unit_1$*-def*
  by (*intro B.isos-compose*) *auto*

lemma $unit_1$*-in-hom*$_S$ [*intro*]:
assumes $ide\ f$
shows «$\eta_1\ f : src\ f \rightarrow P_0\ (trg\ f)$»
and «$\eta_1\ f : PoE.map\ f \star e\ (src\ f) \Rightarrow e\ (trg\ f) \star I_S.map\ f$»
  using *assms ide-char*$_{SbC}$ *arr-char*$_{SbC}$ P_0*-props*(*1*,*6*) *P.preserves-ide hcomp-def*
      src-def trg-def P-def emb.map-def equivalence-data-simps(*2*)
      in-hom-char$_{SbC}$
  by *auto*

lemma $unit_1$*-simps*$_S$ [*simp*]:
assumes $ide\ f$
shows $arr\ (\eta_1\ f)$
and $src\ (\eta_1\ f) = src\ f$ **and** $trg\ (\eta_1\ f) = P_0\ (trg\ f)$
and $dom\ (\eta_1\ f) = PoE.map\ f \star e\ (src\ f)$
and $cod\ (\eta_1\ f) = e\ (trg\ f) \star I_S.map\ f$
and $iso\ (\eta_1\ f)$
using *assms* $unit_1$*-in-hom*$_S$ *iso-char*$_{SbC}$ *ide-char*$_{SbC}$ *arr-char*$_{SbC}$ *P.as-nat-iso.components-are-iso*
  by *auto*

sublocale *unit*: *pseudonatural-equivalence comp hcomp* a i_B *src trg*

comp hcomp a i_B *src trg*
I_S*.map* I_S*.cmp* ‹$P \circ E$› *PoE.cmp* $unit_0$ $unit_1$
proof
show $\bigwedge a.$ *obj a* $\Longrightarrow$ *ide* (*e a*)
using *obj-char ide-char*$_{SbC}$ *arr-char*$_{SbC}$ P_0*-props*(*1*) **by** *force*
show $\bigwedge a.$ *obj a* $\Longrightarrow$ «*e a* : *src* (I_S*.map a*) $\rightarrow$ *src* (($P \circ E$) *a*)»
using *emb.map-def*
apply (*intro in-hhomI*)
apply *auto*
using *obj-char* **by** *auto*
show $\bigwedge a.$ *obj a* $\Longrightarrow$ *equivalence-map* (*e a*)
proof −
fix *a*
assume *a*: *obj a*
interpret *Adj′*: *equivalence-in-bicategory comp hcomp* a i_B *src trg*
‹*e a*› ‹*d a*› ‹η *a*› ‹ε *a*›
using *a* **by** *unfold-locales auto*
show *equivalence-map* (*e a*)
using *Adj′.equivalence-in-bicategory-axioms equivalence-map-def* **by** *auto*
qed
show $\bigwedge f.$ *ide f* $\Longrightarrow$ «η_1 *f* : *PoE.map f* $\star$ *e* (*src f*) $\Rightarrow$ *e* (*trg f*) $\star$ I_S*.map f*»
using $unit_1$*-in-hom*$_S$(*2*) **by** *simp*
show $\bigwedge f.$ *ide f* $\Longrightarrow$ *iso* (η_1 *f*)
by *simp*
show $*$: $\bigwedge \mu.$ *arr* μ $\Longrightarrow$
η_1 (*cod* μ) $\cdot$ (*PoE.map* μ $\star$ *e* (*src* μ)) = (*e* (*trg* μ) $\star$ I_S*.map* μ) $\cdot$ η_1 (*dom* μ)
proof −
fix μ
assume μ: *arr* μ
have *1*: *B.arr* μ
using μ *arr-char*$_{SbC}$ **by** *simp*
let *?f* = *B.dom* μ
let *?g* = *B.cod* μ
let *?a* = src_B μ
let *?b* = trg_B μ
have η_1 (*cod* μ) $\cdot$ (*PoE.map* μ $\star$ *e* (*src* μ))
= ((*e ?b* $\star_B$ r_B[*?g*]) $\cdot_B$
a_B[*e ?b*, *?g*, *?a*] $\cdot_B$
((*e ?b* $\star_B$ *?g*) $\star_B$ *B.inv* (η *?a*)) $\cdot_B$
a_B[*e ?b* $\star_B$ *?g*, *d ?a*, *e ?a*] $\cdot_B$
($\mathrm{a}_B{}^{-1}$[*e ?b*, *?g*, *d ?a*] $\star_B$ *e ?a*)) $\cdot_B$
((*e ?b* $\star_B$ μ $\star_B$ *d ?a*) $\star_B$ *e ?a*)
unfolding $unit_1$*-def P-def emb.map-def hcomp-def src-def*
using μ *comp-char cod-simp arr-char*$_{SbC}$ P_0*-props* [*of ?a*] P_0*-props* [*of ?b*]
by *auto*
also have ... = (*e ?b* $\star_B$ r_B[*?g*]) $\cdot_B$
a_B[*e ?b*, *?g*, *?a*] $\cdot_B$
((*e ?b* $\star_B$ *?g*) $\star_B$ *B.inv* (η *?a*)) $\cdot_B$
a_B[*e ?b* $\star_B$ *?g*, *d ?a*, *e ?a*] $\cdot_B$

$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?g,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?b \star_B \mu \star_B d\ ?a) \star_B e\ ?a)$
using *1 B.comp-assoc* **by** *simp*
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B ?g,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?g,\ d\ ?a] \cdot_B (e\ ?b \star_B \mu \star_B d\ ?a) \star_B e\ ?a \cdot_B e\ ?a)$
using *1 B.interchange* [*of* $\mathrm{a}_B{}^{-1}[e\ ?b,\ ?g,\ d\ ?a]\ e\ ?b \star_B \mu \star_B d\ ?a\ e\ ?a\ e\ ?a$]
by *simp*
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B ?g,\ d\ ?a,\ e\ ?a] \cdot_B$
$(((e\ ?b \star_B \mu) \star_B d\ ?a) \cdot_B \mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B e\ ?a \cdot_B e\ ?a)$
using *1 B.assoc′-naturality* [*of* $e\ ?b\ \mu\ d\ ?a$] **by** *simp*
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(\mathrm{a}_B[e\ ?b \star_B ?g,\ d\ ?a,\ e\ ?a] \cdot_B$
$(((e\ ?b \star_B \mu) \star_B d\ ?a) \star_B e\ ?a)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B e\ ?a)$
using *1 B.comp-assoc*
B.interchange [*of* $(e\ ?b \star_B \mu) \star_B d\ ?a\ \mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a]\ e\ ?a\ e\ ?a$]
by *simp*
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$(((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$((e\ ?b \star_B \mu) \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B e\ ?a)$
using *1 B.assoc-naturality* [*of* $e\ ?b \star_B \mu\ d\ ?a\ e\ ?a$] *B.comp-assoc* **by** *simp*
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$((e\ ?b \star_B \mu) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B e\ ?a)$
proof −
have $((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B ((e\ ?b \star_B \mu) \star_B d\ ?a \star_B e\ ?a)$
$= (e\ ?b \star_B ?g) \cdot_B (e\ ?b \star_B \mu) \star_B B.inv\ (\eta\ ?a) \cdot_B (d\ ?a \star_B e\ ?a)$
using *1 B.interchange* **by** *simp*
also have ... $= (e\ ?b \star_B \mu) \star_B B.inv\ (\eta\ ?a)$
using *1 B.comp-arr-dom B.comp-cod-arr* **by** *simp*
finally have $((e\ ?b \star_B ?g) \star_B B.inv\ (\eta\ ?a)) \cdot_B ((e\ ?b \star_B \mu) \star_B d\ ?a \star_B e\ ?a)$
$= (e\ ?b \star_B \mu) \star_B B.inv\ (\eta\ ?a)$
by *simp*
thus *?thesis* **by** *simp*
qed
also have ... $= (e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$

$(\mathrm{a}_B[e\ ?b,\ ?g,\ ?a] \cdot_B$
$((e\ ?b \star_B \mu) \star_B\ ?a)) \cdot_B$
$((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B\ ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B\ e\ ?a)$
proof −
have $(e\ ?b \star_B \mu) \star_B\ B.inv\ (\eta\ ?a) = (e\ ?b \star_B \mu) \cdot_B (e\ ?b \star_B\ ?f) \star_B\ ?a \cdot_B\ B.inv\ (\eta\ ?a)$
using *1 B.comp-arr-dom B.comp-cod-arr* **by** *simp*
also have ... $= ((e\ ?b \star_B \mu) \star_B\ ?a) \cdot_B ((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a))$
using *1 B.interchange* **by** *simp*
finally have $(e\ ?b \star_B \mu) \star_B\ B.inv\ (\eta\ ?a)$
$= ((e\ ?b \star_B \mu) \star_B\ ?a) \cdot_B ((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a))$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= ((e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B$
$(e\ ?b \star_B \mu \star_B\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?f,\ ?a] \cdot_B$
$((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B\ ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B\ e\ ?a)$
using *1 B.assoc-naturality* [*of e ?b* μ *?a*] *B.comp-assoc* **by** *simp*
also have ... $= (e\ ?b \star_B \mu) \cdot_B$
$(e\ ?b \star_B \mathrm{r}_B[?f]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?f,\ ?a] \cdot_B$
$((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B\ ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ ?f,\ d\ ?a] \star_B\ e\ ?a)$
proof −
have $(e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B (e\ ?b \star_B \mu \star_B\ ?a) = e\ ?b \cdot_B\ e\ ?b \star_B \mathrm{r}_B[?g] \cdot_B (\mu \star_B\ ?a)$
using *1 B.interchange* [*of e ?b e ?b* $\mathrm{r}_B[?g]$ $\mu \star_B\ ?a$] **by** *simp*
also have ... $= e\ ?b \cdot_B\ e\ ?b \star_B \mu \cdot_B \mathrm{r}_B[?f]$
using *1 B.runit-naturality* **by** *simp*
also have ... $= (e\ ?b \star_B \mu) \cdot_B (e\ ?b \star_B \mathrm{r}_B[?f])$
using *1 B.interchange* [*of e ?b e ?b* μ $\mathrm{r}_B[?f]$] **by** *simp*
finally have $(e\ ?b \star_B \mathrm{r}_B[?g]) \cdot_B (e\ ?b \star_B \mu \star_B\ ?a) = (e\ ?b \star_B \mu) \cdot_B (e\ ?b \star_B \mathrm{r}_B[?f])$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ (trg\ \mu) \star I_S.map\ \mu) \cdot \eta_1\ (dom\ \mu)$
proof −
have *arr* $((e\ (trg\ \mu) \star I_S.map\ \mu) \cdot \eta_1\ (dom\ \mu))$
using μ **by** *simp*
hence *2*: *arr* $((e\ ?b \star_B \mu) \cdot$
$((e\ ?b \star_B \mathrm{r}_B[?f]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ ?f,\ ?a] \cdot_B$
$((e\ ?b \star_B\ ?f) \star_B\ B.inv\ (\eta\ ?a)) \cdot_B$

$\mathrm{a}_B[e\ ?b \star_B\ ?f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(B.inv\ \mathrm{a}_B[e\ ?b,\ ?f,\ d\ ?a] \star_B\ e\ ?a)))$
unfolding $unit_1$-*def*
using μ *1 arr-char*$_{SbC}$ P_0-*props trg-def hcomp-def dom-char*$_{SbC}$ **by** *simp*
show *?thesis*
unfolding $unit_1$-*def*
using μ *1 arr-char*$_{SbC}$ P_0-*props trg-def hcomp-def dom-char*$_{SbC}$
apply *simp*
using *2 comp-eqI* **by** *blast*
qed
finally show $\eta_1\ (cod\ \mu) \cdot (PoE.map\ \mu \star e\ (src\ \mu))$
$= (e\ (trg\ \mu) \star I_S.map\ \mu) \cdot \eta_1\ (dom\ \mu)$
by *simp*
qed
show $\bigwedge a.\ obj\ a \Longrightarrow (e\ a \star I_S.unit\ a) \cdot \mathrm{r}^{-1}[e\ a] \cdot \mathrm{l}[e\ a] = unit_1\ a \cdot (PoE.unit\ a \star e\ a)$
proof −
fix *a*
assume *a*: *obj a*
have *0*: *B.obj a*
using *a obj-char* **by** *blast*
interpret *adjoint-equivalence-in-bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ ‹*e a*› ‹*d a*› ‹η *a*› ‹ε *a*›
using *0 chosen-adjoint-equivalence*(*1*) [*of a*] **by** *simp*
have *src*: $src_B\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) = P_0\ a$
using *0 B.src-vcomp* [*of* $e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]$ $B.inv\ (\varepsilon\ a)$] *emb.map-def* **by** *simp*
have *trg*: $trg_B\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) = P_0\ a$
using *0 B.trg-vcomp* [*of* $e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]$ $B.inv\ (\varepsilon\ a)$] *emb.map-def* **by** *simp*
have *1*: $arr\ a \wedge src_B\ a = a$
using *a src-def* **by** *auto*
have *2*: $seq\ (P\ a)\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
proof −
have $B.arr\ (P\ a \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)))$
using *a 0 P-def emb.map-def*
by (*elim B.objE*, *intro B.seqI*) *auto*
moreover have $arr\ (P\ a)$
using *0* **by** *auto*
moreover have $arr\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
apply (*unfold arr-char*$_{SbC}$)
by (*simp add*: *0* P_0-*props*(*6*) *src trg*)
ultimately show *?thesis*
using *seq-char*$_{SbC}$ **by** *simp*
qed
have *3*: $obj\ (P_0\ a)$
using *0* P_0-*props arr-char*$_{SbC}$ *obj-char* **by** *blast*
have *4*: $B.seq\ (P\ a)\ ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a))$
using *2 seq-char*$_{SbC}$ **by** *simp*

have $\eta_1\ a \cdot (PoE.unit\ a \star e\ a) = \eta_1\ a \cdot (P\ a \cdot (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
proof −

have $\eta_1\ a \cdot (PoE.unit\ a \star e\ a) = \eta_1\ a \cdot (PoE.unit\ a \star_B e\ a)$
using *0 a hcomp-eqI* **by** *force*
moreover have $PoE.unit\ a = P\ a \cdot (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)$
using *a obj-char PoE.unit-char′ emb.unit-char′ prj-unit-char* **by** *simp*
ultimately show *?thesis* **by** *argo*
qed
also have ... $= \eta_1\ a \cdot (P\ a \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
using *2 comp-simp* **by** *force*
also have ... $= \eta_1\ a \cdot_B (P\ a \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
proof −
have $arr\ (\eta_1\ a)$
using *a* **by** *auto*
moreover have $arr\ (P\ a \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
by (*metis* (*no-types*, *lifting*) *a 0 2 B.src-vcomp B.vseq-implies-hpar*(*1*) *arr-char*$_{SbC}$ *arr-compI equivalence-data-in-hom*(*3*) *equivalence-data-simps*$_B$(*6*) *hcomp-closed in-hom-char*$_{SbC}$ *src*)
moreover have $B.seq\ (\eta_1\ a)\ (P\ a \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
using *0 3 P-def obj-char*
by (*elim B.objE*, *intro B.seqI*) *auto*
ultimately show *?thesis*
using *2 comp-char* **by** *meson*
qed
also have ... $= (e\ a \star_B \mathrm{r}_B[a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a] \cdot_B$
$((e\ a \star_B a) \star_B B.inv\ (\eta\ a)) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ e\ a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B$
$(((e\ a \star_B a \star_B d\ a) \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
using *0 1 unit*$_1$*-def P-def emb.map*$_0$*-def emb.map-def B.comp-assoc B.obj-def′* **by** *simp*
also have ... $= (e\ a \star_B \mathrm{r}_B[a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a] \cdot_B$
$((e\ a \star_B a) \star_B B.inv\ (\eta\ a)) \cdot_B$
$\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ e\ a] \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a))$
proof −
have $(e\ a \star_B a \star_B d\ a) \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) = e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]$
using *0 B.comp-cod-arr* **by** *simp*
thus *?thesis* **by** *simp*
qed
also have ... $= (e\ a \star_B \mathrm{r}_B[a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a] \cdot_B$
$((e\ a \star_B a) \star_B B.inv\ (\eta\ a)) \cdot_B$
$(\mathrm{a}_B[e\ a \star_B a,\ d\ a,\ e\ a] \cdot_B$
$((\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a) \star_B e\ a)) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B e\ a)$
proof −
have $(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a)$
$= (\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B$
$((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B e\ a) \cdot_B (B.inv\ (\varepsilon\ a) \star_B e\ a)$

using *0 B.whisker-right* **by** *simp*
also have ... = $((\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B e\ a)) \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B e\ a)$
using *B.comp-assoc* **by** *simp*
also have ... = $(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \star_B e\ a) \cdot_B (B.inv\ (\varepsilon\ a) \star_B e\ a)$
proof −
have $B.seq\ \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]\ (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
using *a 0* **by** (*elim B.objE*, *intro B.seqI*) *auto*
thus *?thesis*
using *0 B.whisker-right* **by** *simp*
qed
also have ... = $((\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a) \star_B e\ a) \cdot_B (B.inv\ (\varepsilon\ a) \star_B e\ a)$
proof −
have $\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) = \mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a$
proof −
have $e\ a \star_B \mathrm{l}_B[d\ a] = (\mathrm{r}_B[e\ a] \star_B d\ a) \cdot_B \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]$
using *0 1 B.triangle′* [*of e a d a*] **by** *simp*
moreover have $B.hseq\ (e\ a)\ \mathrm{l}_B[d\ a]$
using *0* **by** *auto*
moreover have $src_B\ (e\ a) = a$
using *0* **by** *simp*
ultimately have $(\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a) \cdot_B (e\ a \star_B \mathrm{l}_B[d\ a]) = \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]$
using *0 B.invert-side-of-triangle*(*1*)
[*of* $e\ a \star_B \mathrm{l}_B[d\ a]\ \mathrm{r}_B[e\ a] \star_B d\ a\ \mathrm{a}_B{}^{-1}[e\ a,\ src_B\ (e\ a),\ d\ a]$]
by *simp*
moreover have $B.arr\ \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]$
using *0 B.assoc′-in-hom* [*of e a a d a*] **by** *fastforce*
ultimately have $\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a = \mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \cdot_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
using *0 B.invert-side-of-triangle*(*2*)
[*of* $\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a]\ \mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a\ e\ a \star_B \mathrm{l}_B[d\ a]$]
by *simp*
thus *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
finally have $(\mathrm{a}_B{}^{-1}[e\ a,\ a,\ d\ a] \star_B e\ a) \cdot_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \star_B e\ a) =$
$((\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a) \star_B e\ a) \cdot_B (B.inv\ (\varepsilon\ a) \star_B e\ a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = $(e\ a \star_B \mathrm{r}_B[a]) \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a] \cdot_B$
$(((e\ a \star_B a) \star_B B.inv\ (\eta\ a)) \cdot_B$
$(\mathrm{r}_B{}^{-1}[e\ a] \star_B d\ a \star_B e\ a)) \cdot_B$
$\mathrm{a}_B[e\ a,\ d\ a,\ e\ a] \cdot_B$
$(B.inv\ (\varepsilon\ a) \star_B e\ a)$
using *0 B.assoc-naturality* [*of* $\mathrm{r}_B{}^{-1}[e\ a]\ d\ a\ e\ a$] *B.comp-assoc* **by** *simp*
also have ... = $(e\ a \star_B \mathrm{r}_B[a]) \cdot_B$

$\mathrm{a}_B[e\ a,\ a,\ a]\ \cdot_B$
$(\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)\ \cdot_B$
$((e\ a\ \star_B\ B.inv\ (\eta\ a))\ \cdot_B$
$\mathrm{a}_B[e\ a,\ d\ a,\ e\ a]\ \cdot_B$
$(B.inv\ (\varepsilon\ a)\ \star_B\ e\ a))$
proof −
have $((e\ a\ \star_B\ a)\ \star_B\ B.inv\ (\eta\ a))\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ d\ a\ \star_B\ e\ a)$
$= (e\ a\ \star_B\ a)\ \cdot_B\ \mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ B.inv\ (\eta\ a)\ \cdot_B\ (d\ a\ \star_B\ e\ a)$
using *0 B.interchange* [*of* $e\ a\ \star_B\ a\ \mathrm{r}_B{}^{-1}[e\ a]\ B.inv\ (\eta\ a)\ d\ a\ \star_B\ e\ a$]
by *force*
also have ... $= \mathrm{r}_B{}^{-1}[e\ a]\ \cdot_B\ e\ a\ \star_B\ a\ \cdot_B\ B.inv\ (\eta\ a)$
using *0 B.comp-arr-dom B.comp-cod-arr* **by** *simp*
also have ... $= (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)\ \cdot_B\ (e\ a\ \star_B\ B.inv\ (\eta\ a))$
using *0 B.interchange* [*of* $\mathrm{r}_B{}^{-1}[e\ a]\ e\ a\ a\ B.inv\ (\eta\ a)$] **by** *fastforce*
ultimately have $((e\ a\ \star_B\ a)\ \star_B\ B.inv\ (\eta\ a))\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ d\ a\ \star_B\ e\ a)$
$= (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)\ \cdot_B\ (e\ a\ \star_B\ B.inv\ (\eta\ a))$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= ((e\ a\ \star_B\ \mathrm{r}_B[a])\ \cdot_B$
$\mathrm{a}_B[e\ a,\ a,\ a]\ \cdot_B$
$(\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a))\ \cdot_B$
$\mathrm{r}_B{}^{-1}[e\ a]\ \cdot_B\ \mathrm{l}_B[e\ a]$
by (*metis B.comp-assoc adjoint-equivalence-in-bicategory-def adjunction-in-bicategory.triangle-right dual-adjoint-equivalence*)
also have ... $= \mathrm{r}_B{}^{-1}[e\ a]\ \cdot_B\ \mathrm{l}_B[e\ a]$
proof −
have $(e\ a\ \star_B\ \mathrm{r}_B[a])\ \cdot_B\ \mathrm{a}_B[e\ a,\ a,\ a]\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a) = e\ a\ \star_B\ a$
proof −
have $(e\ a\ \star_B\ \mathrm{r}_B[a])\ \cdot_B\ \mathrm{a}_B[e\ a,\ a,\ a]\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)$
$= ((e\ a\ \star_B\ \mathrm{r}_B[a])\ \cdot_B\ \mathrm{a}_B[e\ a,\ a,\ a])\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)$
using *B.comp-assoc* **by** *simp*
also have ... $= (\mathrm{r}_B[e\ a]\ \star_B\ a)\ \cdot_B\ (\mathrm{r}_B{}^{-1}[e\ a]\ \star_B\ a)$
using *0 B.runit-char*(*2*) [*of* $e\ a$] *B.unitor-coincidence* **by** *simp*
also have ... $= e\ a\ \star_B\ a$
using *0 B.whisker-right* [*of* $a\ \mathrm{r}_B[e\ a]\ \mathrm{r}_B{}^{-1}[e\ a]$] *B.comp-arr-inv′* **by** *auto*
finally show *?thesis*
by *simp*
qed
thus *?thesis*
using *0 B.comp-cod-arr* **by** *simp*
qed
also have ... $= (e\ a\ \star\ I_S.unit\ a)\ \cdot\ \mathrm{r}^{-1}[e\ a]\ \cdot\ \mathrm{l}[e\ a]$
proof −
have $(e\ a\ \star\ I_S.unit\ a)\ \cdot\ \mathrm{r}^{-1}[e\ a]\ \cdot\ \mathrm{l}[e\ a] = (e\ a\ \star_B\ a)\ \cdot_B\ \mathrm{r}_B{}^{-1}[e\ a]\ \cdot_B\ \mathrm{l}_B[e\ a]$
proof −
have *1*: $arr\ \mathrm{l}_B[e\ a]\ \wedge\ arr\ \mathrm{r}_B{}^{-1}[e\ a]$
using *a lunit-simp runit′-simp lunit-simps runit′-simps* **by** *simp*

moreover have *arr* $(\mathrm{r}_B{}^{-1}[e\ a] \cdot_B \mathrm{l}_B[e\ a])$
using *1 arr-char*$_{SbC}$ *dom-char*$_{SbC}$ *cod-char*$_{SbC}$ *comp-char* **by** *auto*
moreover have *2*: *hseq* $(e\ a)$ $(I_S.unit\ a)$
using *a* I_S*.unit-char* **by** (*intro hseqI′*) *auto*
moreover have *B.seq* $(e\ a \star I_S.unit\ a)$ $(\mathrm{r}_B{}^{-1}[e\ a] \cdot_B \mathrm{l}_B[e\ a])$
using *0 2 hcomp-char arr-char*$_{SbC}$ I_S*.unit-char′*
apply (*intro B.seqI*)
apply *auto*[*4*]
using *a* **by** *force*
ultimately show *?thesis*
using *a comp-char* I_S*.unit-char′ lunit-simp runit′-simp hcomp-char*
by *auto*
qed
also have ... $= \mathrm{r}_B{}^{-1}[e\ a] \cdot_B \mathrm{l}_B[e\ a]$
using *0 B.comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *simp*
qed
finally show $(e\ a \star I_S.unit\ a) \cdot \mathrm{r}^{-1}[e\ a] \cdot \mathrm{l}[e\ a] = \eta_1\ a \cdot (PoE.unit\ a \star e\ a)$
by *argo*
qed
show $\bigwedge f\ g.\ [\![ide\ f;\ ide\ g;\ src\ g = trg\ f]\!] \Longrightarrow$
$(e\ (trg\ g) \star I_S.cmp\ (g, f)) \cdot$
$\mathrm{a}[e\ (trg\ g),\ I_S.map\ g,\ I_S.map\ f] \cdot$
$(\eta_1\ g \star I_S.map\ f) \cdot$
$inv\ \mathrm{a}[PoE.map\ g,\ e\ (src\ g),\ I_S.map\ f] \cdot$
$(PoE.map\ g \star \eta_1\ f) \cdot$
$\mathrm{a}[PoE.map\ g,\ PoE.map\ f,\ e\ (src\ f)]$
$= \eta_1\ (g \star f) \cdot (PoE.cmp\ (g, f) \star e\ (src\ f))$
proof −
fix *f g*
assume *f*: *ide f* **and** *g*: *ide g* **and** *fg*: *src g* = *trg f*
let *?a* = $src_B\ f$
let *?b* = $src_B\ g$
let *?c* = $trg_B\ g$
have *hseq*: *ide f* ∧ *ide g* ∧ *arr f* ∧ *arr g* ∧ *src g* = *trg f*
using *f g fg ide-char*$_{SbC}$ **by** *auto*
have $hseq_B$: *B.ide f* ∧ *B.ide g* ∧ *B.arr f* ∧ *B.arr g* ∧ $src_B\ g = trg_B\ f$
using *f g fg ide-char*$_{SbC}$ *src-def trg-def* **by** *auto*
have *1*: *?a* = *src f* ∧ *?b* = *src g* ∧ *?c* = *trg g*
using *f g fg src-def trg-def* **by** *simp*
have $(e\ (trg\ g) \star I_S.cmp\ (g, f)) \cdot$
$\mathrm{a}[e\ (trg\ g),\ I_S.map\ g,\ I_S.map\ f] \cdot$
$(\eta_1\ g \star I_S.map\ f) \cdot$
$inv\ \mathrm{a}[PoE.map\ g,\ e\ (src\ g),\ I_S.map\ f] \cdot$
$(PoE.map\ g \star \eta_1\ f) \cdot$
$\mathrm{a}[PoE.map\ g,\ PoE.map\ f,\ e\ (src\ f)]$
$= (e\ ?c \star I_S.cmp\ (g, f)) \cdot$
$\mathrm{a}[e\ ?c,\ I_S.map\ g,\ I_S.map\ f] \cdot$
$(\eta_1\ g \star I_S.map\ f) \cdot$

$inv\ \mathrm{a}[PoE.map\ g,\ e\ ?b,\ I_S.map\ f]\ \cdot$
$(PoE.map\ g \star \eta_1\ f)\ \cdot$
$\mathrm{a}[PoE.map\ g,\ PoE.map\ f,\ e\ ?a]$
using *1* **by** *simp*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g]\ \cdot_B\ (g \star_B B.inv\ (\eta\ ?b))$
$\star_B\ \mathrm{r}_B[f]\ \cdot_B\ (f \star_B B.inv\ (\eta\ ?a)))\ \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a]\ \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)\ \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)\ \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a))\ \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a]\ \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a])\ \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a])\ \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a)\ \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $(e\ ?c \star I_S.cmp\ (g,\ f))\ \cdot$
$\mathrm{a}[e\ ?c,\ I_S.map\ g,\ I_S.map\ f]\ \cdot$
$(\eta_1\ g \star I_S.map\ f)\ \cdot$
$inv\ \mathrm{a}[PoE.map\ g,\ e\ ?b,\ I_S.map\ f]\ \cdot$
$(PoE.map\ g \star unit_1\ f)\ \cdot$
$\mathrm{a}[PoE.map\ g,\ PoE.map\ f,\ e\ ?a]$
$= (e\ ?c \star g \star f)\ \cdot$
$\mathrm{a}[e\ ?c,\ g,\ f]\ \cdot$
$(\eta_1\ g \star f)\ \cdot$
$inv\ \mathrm{a}[P\ (E\ g),\ e\ ?b,\ f]\ \cdot$
$(P\ (E\ g) \star \eta_1\ f)\ \cdot$
$\mathrm{a}[P\ (E\ g),\ P\ (E\ f),\ e\ ?a]$
using *f g fg* **by** *simp*
also have ... $= (e\ ?c \star g \star f)\ \cdot_B$
$\mathrm{a}[e\ ?c,\ g,\ f]\ \cdot_B$
$(\eta_1\ g \star f)\ \cdot_B$
$inv\ \mathrm{a}[P\ (E\ g),\ e\ ?b,\ f]\ \cdot_B$
$(P\ (E\ g) \star \eta_1\ f)\ \cdot_B$
$\mathrm{a}[P\ (E\ g),\ P\ (E\ f),\ e\ ?a]$
proof −
have *arr* ...
proof −
have «... : $(P\ (E\ g) \star P\ (E\ f)) \star e\ ?a \Rightarrow e\ ?c \star g \star f$»
proof (*intro comp-in-homI*)
show «$\mathrm{a}[P\ (E\ g),\ P\ (E\ f),\ e\ ?a]$:
$(P\ (E\ g) \star P\ (E\ f)) \star e\ ?a \Rightarrow P\ (E\ g) \star P\ (E\ f) \star e\ ?a$»
using *f g fg VVV.arr-char$_{SbC}$ VV.arr-char$_{SbC}$ src-def obj-src assoc-in-hom*
by *force*
show «$P\ (E\ g) \star \eta_1\ f : P\ (E\ g) \star P\ (E\ f) \star e\ ?a \Rightarrow P\ (E\ g) \star e\ ?b \star f$»
using *f g fg 1 unit$_1$-in-hom$_S$* [*of f*] *vconn-implies-hpar*(*2*)
by (*intro hcomp-in-vhom*) *auto*
show «$inv\ \mathrm{a}[P\ (E\ g),\ e\ ?b,\ f] : P\ (E\ g) \star e\ ?b \star f \Rightarrow (P\ (E\ g) \star e\ ?b) \star f$»
proof −

have «a[P (E g), e ?b, f] : (P (E g) $\star$ e ?b) $\star$ f $\Rightarrow$ P (E g) $\star$ e ?b $\star$ f»
using *f g fg 1 VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *assoc-in-hom* **by** *simp*
moreover have *iso* a[P (E g), e ?b, f]
using *f g fg 1 VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *iso-assoc* **by** *simp*
ultimately show *?thesis*
using *inv-in-hom* **by** *simp*
qed
show «*unit*$_1$ g $\star$ f : (P (E g) $\star$ e ?b) $\star$ f $\Rightarrow$ (e ?c $\star$ g) $\star$ f»
using *f g fg 1 unit*$_1$*-in-hom*$_S$ [*of g*]
by (*intro hcomp-in-vhom*) *auto*
have *2*: *ide* (e (*trg g*))
using *g* **by** *simp*
have *3*: *src* (e (*trg g*)) = *trg g*
using *g trg-def hseqE in-hom-char*$_{SbC}$ *obj-trg* **by** *auto*
show «a[e ?c, g, f] : (e ?c $\star$ g) $\star$ f $\Rightarrow$ e ?c $\star$ g $\star$ f»
using *f g fg 1 2 3 VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *assoc-in-hom* [*of e ?c g f*]
by *simp*
show «e ?c $\star$ g $\star$ f : e ?c $\star$ g $\star$ f $\Rightarrow$ e ?c $\star$ g $\star$ f»
using *f g fg 1 2 3*
by (*intro hcomp-in-vhom*) *auto*
qed
thus *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-eqI* **by** *blast*
qed
also have ... = (e ?c $\star$ g $\star$ f) $\cdot_B$
a[e ?c, g, f] $\cdot_B$
(η_1 g $\star$ f) $\cdot_B$
B.inv a[P (E g), e ?b, f] $\cdot_B$
(P (E g) $\star$ η_1 f) $\cdot_B$
a[P (E g), P (E f), e ?a]
proof −

have *iso* a[P (E g), e ?b, f]
using *f g fg hseq hseq*$_B$ *src-def trg-def emb.map*$_0$*-def emb.map-def arr-char*$_{SbC}$
obj-char P$_0$*-props*(*6*)
by (*intro iso-assoc*) *auto*
thus *?thesis*
using *f g fg inv-char*$_{SbC}$ [*of* a[P (E g), e ?b, f]] **by** *simp*
qed
also have ... = (e ?c $\star_B$ g $\star_B$ f) $\cdot_B$
a[e ?c, g, f] $\cdot_B$
(η_1 g $\star_B$ f) $\cdot_B$
B.inv a[P (E g), e ?b, f] $\cdot_B$
(P (E g) $\star_B$ η_1 f) $\cdot_B$
a[P (E g), P (E f), e ?a]
proof −
have *arr* (*unit*$_0$?c)

using *hseq obj-trg trg-def* **by** *auto*
moreover have *arr* $(g \star_B f)$
using *hseq* $hseq_B$ *arr-char*$_{SbC}$ **by** *auto*
ultimately show *?thesis*
unfolding *hcomp-def*
using *f g fg* $hseq_B$ *src-def trg-def* P_0*-props*(*1*) *emb.map*$_0$*-def emb.map-def*
by *simp*
qed
also have ... $= (e\ ?c \star_B g \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$(\eta_1\ g \star_B f) \cdot_B$
$B.inv\ \mathrm{a}_B[P\ (E\ g),\ e\ ?b,\ f] \cdot_B$
$(P\ (E\ g) \star_B \eta_1\ f) \cdot_B$
$\mathrm{a}_B[P\ (E\ g),\ P\ (E\ f),\ e\ ?a]$
proof −
have $arr\ (e\ ?a) \wedge arr\ (e\ ?b) \wedge arr\ (e\ ?c)$
using *hseq* $hseq_B$ *src-def trg-def obj-char src.preserves-arr trg.preserves-arr*
by *auto*
thus *?thesis*
using *f g fg VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *src-def trg-def* $hseq_B$
emb.map$_0$*-def emb.map-def*
by *simp*
qed
also have ... $= (e\ ?c \star_B g \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$(\eta_1\ g \star_B f) \cdot_B$
$(B.inv\ \mathrm{a}_B[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B \eta_1\ f) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a])$
using *f g fg 1 emb.map-def* **by** *simp*
also have ... $= (e\ ?c \star_B g \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$(\eta_1\ g \star_B f) \cdot_B$
$(\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B \eta_1\ f) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a])$
using $hseq_B$ *P.preserves-ide ide-char*$_{SbC}$ *B.assoc′-eq-inv-assoc* [*of P g e ?b f*]
by *simp*
also have ... $= (e\ ?c \star_B g \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ ?b] \cdot_B$
$((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b)$
$\star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B (e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$\mathrm{a}_B[e\ ?b,\ f,\ ?a] \cdot_B$

$((e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
unfolding *unit*$_1$*-def*
using *hseq*$_B$ *B.comp-assoc* **by** *auto*
also have ... $= ((e\ ?c \star_B g \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ f]) \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g]) \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ ?b] \star_B f) \cdot_B$
$(((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B$
$(P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
using *hseq*$_B$ *ide-char*$_{SbC}$ *P.preserves-ide B.whisker-left B.whisker-right B.comp-assoc*
by *auto*
also have ... $= \mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g]) \star_B f) \cdot_B$
$((\mathrm{a}_B[e\ ?c,\ g,\ ?b] \star_B f) \cdot_B$
$(((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \star_B f)) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B$
$(P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $(e\ ?c \star_B g \star_B f) \cdot_B \mathrm{a}_B[e\ ?c,\ g,\ f] = \mathrm{a}_B[e\ ?c,\ g,\ f]$
using *hseq*$_B$ *B.comp-cod-arr* **by** *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$(((e\ ?c \star_B \mathrm{r}_B[g]) \star_B f) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b)) \star_B f)) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$

$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B$
$(P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $(\mathrm{a}_B[e\ ?c,\ g,\ ?b] \star_B f) \cdot_B (((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \star_B f)$
$= ((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B (\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f)$
proof −
have $(\mathrm{a}_B[e\ ?c,\ g,\ ?b] \star_B f) \cdot_B (((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \star_B f)$
$= \mathrm{a}_B[e\ ?c,\ g,\ ?b] \cdot_B ((e\ ?c \star_B g) \star_B B.inv\ (\eta\ ?b)) \star_B f$
using $hseq_B$ *B.whisker-right* **by** *auto*
also have ... $= (e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b)) \cdot_B \mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f$
using $hseq_B$ *B.assoc-naturality* [*of e ?c g B.inv* $(\eta\ ?b)$] **by** *simp*
also have ... $= ((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f)$
using $hseq_B$ *B.whisker-right* **by** *auto*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_B[e\ ?c,\ g,\ f] \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B$
$(P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
using $hseq_B$ *B.whisker-left B.whisker-right B.comp-assoc* **by** *simp*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$((P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B$
$(P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
using $hseq_B$ *B.comp-assoc*
B.assoc-naturality [*of e ?c* $\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))\ f$]

by *fastforce*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$((P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f]) \cdot_B$
$(P\ g \star_B e\ ?b \star_B f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B (P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a))$
$= (P\ g \star_B e\ ?b \star_B f \star_B B.inv\ (\eta\ ?a)) \cdot_B (P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a])$
proof −
have $(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a]) \cdot_B (P\ g \star_B (e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a))$
$= P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ ?a] \cdot_B ((e\ ?b \star_B f) \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ $ide\text{-}char_{SbC}$ *P.preserves-ide B.whisker-left* **by** *auto*
also have ... $= P\ g \star_B (e\ ?b \star_B f \star_B B.inv\ (\eta\ ?a)) \cdot_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]$
using $hseq_B$ *B.assoc-naturality* [*of e ?b f B.inv* $(\eta\ ?a)$] **by** *auto*
also have ... $= (P\ g \star_B e\ ?b \star_B f \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a])$
using $hseq_B$ $ide\text{-}char_{SbC}$ *P.preserves-ide B.whisker-left* **by** *auto*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$(\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f] \cdot_B$
$(P\ g \star_B e\ ?b \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
using $hseq_B$ $ide\text{-}char_{SbC}$ *P.preserves-ide B.whisker-left B.comp-assoc* **by** *auto*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$(((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$((P\ g \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$

$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
using $hseq_B$ *B.comp-assoc*
B.assoc′-naturality [*of* $P\ g\ e\ ?b\ \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$]
by *simp*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$((P\ g \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
proof −
have $((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B f) \cdot_B$
$((P\ g \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \cdot_B (P\ g \star_B e\ ?b) \star_B$
$(f \cdot_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
proof −
have $B.seq\ (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b)\ (P\ g \star_B e\ ?b)$
using $hseq_B$ *P.preserves-ide P-def src-def trg-def* **by** *auto*
moreover have $B.seq\ f\ (\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
using $hseq_B$ **by** *simp*
ultimately show *?thesis*
using *B.interchange*
[*of* $\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b\ P\ g \star_B e\ ?b$
$f\ \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$]
by *presburger*
qed
also have ... $= (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \cdot_B (e\ ?c \star_B g \star_B d\ ?b) \star_B e\ ?b)$
$\star_B f \cdot_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.whisker-right P-def* **by** *auto*
also have ... $= (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.comp-arr-dom B.comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$

$((\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$

proof −
have $(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B ((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b)$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= (\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a))$
proof −
have $(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \cdot_B (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b)$
$\star_B f \cdot_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.interchange* **by** *simp*
also have ... $= \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \cdot_B (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b)$
$\star_B (\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B (f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.comp-cod-arr B.comp-arr-dom* **by** *simp*
also have ... $= (\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a))$
using $hseq_B$ *B.interchange* **by** *auto*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$

proof −
have $(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= \mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \cdot_B \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b]$
$\star_B f \cdot_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.interchange* **by** *simp*
also have ... $= \mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \cdot_B \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b]$
$\star_B (\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B (f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.comp-cod-arr B.comp-arr-dom* **by** *simp*

also have ... = $(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.interchange* **by** *auto*
finally have $(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f) \cdot_B (\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b]$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= (\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = $((e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$
$= \mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B ((e\ ?c \star_B g \star_B d\ ?b \star_B e\ ?b) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b]$
$\star_B (\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B (f \star_B d\ ?a \star_B e\ ?a))$
using $hseq_B$ *B.comp-arr-dom B.comp-cod-arr* **by** *simp*
also have ... = $\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$((e\ ?c \star_B g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.interchange* **by** *simp*
also have ... = $(\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B$
$((e\ ?c \star_B g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
using *B.comp-assoc* **by** *simp*
also have ... = $(((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a]) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.assoc-naturality* [*of* $e\ ?c\ g \star_B d\ ?b \star_B e\ ?b$
$\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$]
by *auto*
also have ... = $((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
using *B.comp-assoc* **by** *simp*
finally have $\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f] \cdot_B (\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b]$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))$

$= ((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
have $(e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))))$
$= e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \cdot_B (g \star_B d\ ?b \star_B e\ ?b)$
$\star_B f \cdot_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.whisker-left B.interchange* **by** *fastforce*
also have ... $= e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.comp-arr-dom B.comp-cod-arr* **by** *auto*
finally have $(e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \cdot_B$
$((e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))))$
$= e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
by *simp*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ ?a] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −

have $(e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$

$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ ?a] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
$= (e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g \star_B f,\ ?a] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
using *B.comp-assoc* **by** *simp*
also have ... $= ((e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$(e\ ?c \star_B (g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
using $hseq_B$ *B.assoc-naturality* [*of e ?c* $g \star_B f$ *B.inv* $(\eta\ ?a)$] *B.comp-assoc*
by *simp*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a)) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$

using $hseq_B$ *B.whisker-left B.comp-assoc* **by** *auto*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g] \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a)) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −
have $((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a)$
$= ((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{r}_B[g] \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a)$
using $hseq_B$ *B.whisker-right B.whisker-left B.triangle′ comp-assoc* **by** *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{r}_B[g] \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (((g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a))) \star_B e\ ?a)) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −
have $((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a)$
$= (e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ ?b,\ f \star_B d\ ?a] \cdot_B (g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)) \star_B e\ ?a$
using $hseq_B$ *B.whisker-right B.whisker-left* **by** *auto*
also have ... $= (e\ ?c \star_B$
$(((g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])$
$\star_B e\ ?a$
using $hseq_B$ *B.assoc′-naturality* [*of g B.inv* $(\eta\ ?b)$ *f* $\star_B$ *d ?a*] **by** *auto*
also have ... $= ((e\ ?c \star_B (((g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a))) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a)$
using $hseq_B$ *B.whisker-right B.whisker-left* **by** *auto*
finally have $((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$

$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a)$
$= ((e\ ?c \star_B (((g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a))) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f \star_B d\ ?a) \star_B e\ ?a)) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
using $hseq_B$ *B.whisker-right B.whisker-left B.comp-assoc* **by** *auto*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a)) \star_B e\ ?a)) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −
have $((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f \star_B d\ ?a) \star_B e\ ?a)$
$= (e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a] \cdot_B$
$((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f \star_B d\ ?a)) \star_B e\ ?a$
using $hseq_B$ *B.whisker-left B.whisker-right* **by** *auto*
also have ... $= (e\ ?c \star_B$
$(((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a])$
$\star_B e\ ?a$
using $hseq_B$ *B.assoc′-naturality* [*of* $\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))\ f\ d\ ?a$]
by *auto*
also have ... $= ((e\ ?c \star_B (((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a))$
$\star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a)$
using $hseq_B$ *B.whisker-left B.whisker-right* **by** *auto*
finally have $((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f \star_B d\ ?a) \star_B e\ ?a)$
$= ((e\ ?c \star_B (((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a)) \star_B e\ ?a) \cdot_B$

$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a) \star_B e\ ?a)) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −
have $(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a)) \star_B e\ ?a)$
$= \mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \cdot_B$
$(e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \star_B d\ ?a)) \star_B e\ ?a$
using $hseq_B$ *B.whisker-right* **by** *auto*
also have ... $= ((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a$
using $hseq_B$ *B.assoc′-naturality* [*of e ?c* $(\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f\ d\ ?a$]
by *fastforce*
also have ... $= (((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a)$
$\star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a)$
using $hseq_B$ *B.whisker-right* **by** *auto*
finally have $(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B (((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a)) \star_B e\ ?a)$
$= (((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a)$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$

proof −
have $\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a) \star_B e\ ?a)$
$= ((e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f)) \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a,\ e\ ?a]$
using $hseq_B$
B.assoc-naturality [*of* $(e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f))$
$d\ ?a\ e\ ?a$]
by *fastforce*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
using $hseq_B$ *B.comp-assoc*
B.assoc-naturality [*of* $e\ ?c\ (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f$
$d\ ?a \star_B e\ ?a$]
by *force*
also have ... $= (e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B[g,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(e\ ?c \star_B ((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))) \star_B f) \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
proof −
have $e\ ?c \star_B \mathrm{r}_B[g \star_B f] \cdot_B ((g \star_B f) \star_B B.inv\ (\eta\ ?a))$
$= (e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(e\ ?c \star_B \mathrm{a}_B[g,\ f,\ d\ ?a \star_B e\ ?a])$
using $hseq_B$ *B.runit-hcomp B.whisker-left B.comp-assoc*
B.assoc-naturality [*of g f B.inv* $(\eta\ ?a)$]
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed

also have ... $= ((e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a))) \cdot_B$
$(e\ ?c \star_B \mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a \star_B e\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c, (g \star_B d\ ?b \star_B e\ ?b) \star_B f, d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f, d\ ?a, e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c, (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f, d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, d\ ?b \star_B e\ ?b, f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b, e\ ?b, f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c, g, d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g, d\ ?b, P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c, g, d\ ?b] \star_B P\ f) \star_B e\ ?a)$

proof −
have $(e\ ?c \star_B \mathrm{a}_B[g, f, d\ ?a \star_B e\ ?a]) \cdot_B$
$(e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \star_B d\ ?a \star_B e\ ?a)$
$= e\ ?c \star_B \mathrm{a}_B[g, f, d\ ?a \star_B e\ ?a] \cdot_B$
$((\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \star_B d\ ?a \star_B e\ ?a)$
using *hseq*$_B$ *B.whisker-left* **by** *auto*
also have ... $= e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a \star_B e\ ?a]$
using *hseq*$_B$ *B.assoc-naturality* [*of* $\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))\ f\ d\ ?a \star_B e\ ?a$]
by *auto*
also have ... $= (e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$(e\ ?c \star_B \mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a \star_B e\ ?a])$
using *hseq*$_B$ *B.whisker-left* **by** *auto*
finally have $(e\ ?c \star_B \mathrm{a}_B[g, f, d\ ?a \star_B e\ ?a]) \cdot_B$
$(e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f) \star_B d\ ?a \star_B e\ ?a)$
$= (e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$(e\ ?c \star_B \mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a \star_B e\ ?a])$
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*

qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B$
$\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$(e\ ?c \star_B \mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a \star_B e\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c, (g \star_B d\ ?b \star_B e\ ?b) \star_B f, d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f, d\ ?a, e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c, (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f, d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b, f, d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, d\ ?b \star_B e\ ?b, f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b, e\ ?b, f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c, g, d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g, d\ ?b, P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c, g, d\ ?b] \star_B P\ f) \star_B e\ ?a)$

proof −
have $(e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a))$
$= e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$

$(\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.whisker-left* **by** *auto*
also have ... $= e\ ?c \star_B g \cdot_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))$
$\star_B (\mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B (f \star_B d\ ?a \star_B e\ ?a)$
using $hseq_B$ *B.interchange* **by** *auto*
also have ... $= e\ ?c \star_B (g \cdot_B \mathrm{r}_B[g]) \cdot_B (g \star_B B.inv\ (\eta\ ?b))$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)) \cdot_B (f \star_B d\ ?a \star_B e\ ?a)$
using *B.comp-assoc* **by** *simp*
also have ... $= e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
using $hseq_B$ *B.comp-arr-dom B.comp-cod-arr* **by** *auto*
finally have $(e\ ?c \star_B (g \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a)))) \cdot_B$
$(e\ ?c \star_B (\mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B f \star_B d\ ?a \star_B e\ ?a))$
$= e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b)) \star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))$
by *simp*
thus *?thesis*
by *simp*
qed
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g] \cdot_B (g \star_B B.inv\ (\eta\ ?b))$
$\star_B \mathrm{r}_B[f] \cdot_B (f \star_B B.inv\ (\eta\ ?a))) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$
proof −
let $?LHS = (e\ ?c \star_B \mathrm{a}_B[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a \star_B e\ ?a] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B (g \star_B d\ ?b \star_B e\ ?b) \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ (g \star_B (d\ ?b \star_B e\ ?b)) \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g \star_B d\ ?b \star_B e\ ?b,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a])) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$

let $?RHS = \mathrm{a}_B[e\ ?c,\ g \star_B d\ ?b \star_B e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ e\ ?b] \star_B f \star_B d\ ?a \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B e\ ?b) \star_B (f \star_B d\ ?a \star_B e\ ?a)) \cdot_B$
$\mathrm{a}_B{}^{-1}[P\ g,\ e\ ?b,\ f \star_B d\ ?a \star_B e\ ?a] \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b,\ f,\ d\ ?a \star_B e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B[e\ ?b \star_B f,\ d\ ?a,\ e\ ?a]) \cdot_B$
$(P\ g \star_B \mathrm{a}_B{}^{-1}[e\ ?b,\ f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$\mathrm{a}_B[P\ g,\ P\ f,\ e\ ?a]$

have *?LHS* = *?RHS*
proof −
let *?LHSt* = $(\langle e\ ?c\rangle \star \mathbf{a}[\langle g\rangle \star \langle d\ ?b\rangle \star \langle e\ ?b\rangle, \langle f\rangle, \langle d\ ?a\rangle \star \langle e\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle e\ ?c\rangle, (\langle g\rangle \star \langle d\ ?b\rangle \star \langle e\ ?b\rangle) \star \langle f\rangle, \langle d\ ?a\rangle \star \langle e\ ?a\rangle] \cdot$
$\mathbf{a}[\langle e\ ?c\rangle \star (\langle g\rangle \star \langle d\ ?b\rangle \star \langle e\ ?b\rangle) \star \langle f\rangle, \langle d\ ?a\rangle, \langle e\ ?a\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle e\ ?c\rangle, (\langle g\rangle \star (\langle d\ ?b\rangle \star \langle e\ ?b\rangle)) \star \langle f\rangle, \langle d\ ?a\rangle] \star \langle e\ ?a\rangle) \cdot$
$((\langle e\ ?c\rangle \star \mathbf{a}^{-1}[\langle g\rangle \star \langle d\ ?b\rangle \star \langle e\ ?b\rangle, \langle f\rangle, \langle d\ ?a\rangle]) \star \langle e\ ?a\rangle) \cdot$
$((((\langle e\ ?c\rangle \star \mathbf{a}^{-1}[\langle g\rangle, \langle d\ ?b\rangle \star \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle])) \star \langle e\ ?a\rangle) \cdot$
$((\langle e\ ?c\rangle \star \langle g\rangle \star \mathbf{a}^{-1}[\langle d\ ?b\rangle, \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle]) \star \langle e\ ?a\rangle) \cdot$
$(\mathbf{a}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle \star \mathbf{PRJ}\ f] \star \langle e\ ?a\rangle) \cdot$
$(\mathbf{a}[\langle e\ ?c\rangle \star \langle g\rangle, \langle d\ ?b\rangle, \mathbf{PRJ}\ f] \star \langle e\ ?a\rangle) \cdot$
$((\mathbf{a}^{-1}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle\ ] \star \mathbf{PRJ}\ f) \star \langle e\ ?a\rangle)$

let *?RHSt* = $\mathbf{a}[\langle e\ ?c\rangle, \langle g\rangle \star \langle d\ ?b\rangle \star \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle \star \langle e\ ?a\rangle] \cdot$
$(\mathbf{a}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle \star \langle e\ ?b\rangle] \star \langle f\rangle \star \langle d\ ?a\rangle \star \langle e\ ?a\rangle) \cdot$
$(\mathbf{a}[\langle e\ ?c\rangle \star \langle g\rangle, \langle d\ ?b\rangle, \langle e\ ?b\rangle] \star \langle f\rangle \star \langle d\ ?a\rangle \star \langle e\ ?a\rangle) \cdot$
$((\mathbf{a}^{-1}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle] \star \langle e\ ?b\rangle) \star (\ \langle f\rangle \star \langle d\ ?a\rangle \star \langle e\ ?a\rangle)) \cdot$
$\mathbf{a}^{-1}[\mathbf{PRJ}\ g, \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle \star \langle e\ ?a\rangle] \cdot$
$(\mathbf{PRJ}\ g \star \mathbf{a}[\langle e\ ?b\rangle, \langle f\rangle, \langle d\ ?a\rangle \star \langle e\ ?a\rangle]) \cdot$
$(\mathbf{PRJ}\ g \star \mathbf{a}[\langle e\ ?b\rangle \star \langle f\rangle, \langle d\ ?a\rangle, \langle e\ ?a\rangle]) \cdot$
$(\mathbf{PRJ}\ g \star \mathbf{a}^{-1}[\langle e\ ?b\rangle, \langle f\rangle, \langle d\ ?a\rangle] \star \langle e\ ?a\rangle) \cdot$
$\mathbf{a}[\mathbf{PRJ}\ g, \mathbf{PRJ}\ f, \langle e\ ?a\rangle]$

have *?LHS* = $\{\!|$*?LHSt*$|\!\}$
using *f g fg* $hseq_B$ *B.α-def B.α′.map-ide-simp* $B.VVV.ide\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P-def*
by *auto*
also have ... = $\{\!|$*?RHSt*$|\!\}$
using $hseq_B$ **by** (*intro EV.eval-eqI*, *auto*)
also have ... = *?RHS*
using *f g fg* $hseq_B$ *B.α-def B.α′.map-ide-simp* $B.VVV.ide\text{-}char_{SbC}$ $B.VVV.arr\text{-}char_{SbC}$ $B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ *P-def*
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **using** $hseq_B$ **by** *auto*
qed
finally show *?thesis* **by** *simp*
qed
also have ... = $\eta_1\ (g \star f) \cdot (PoE.cmp\ (g, f) \star e\ (src\ f))$
proof −
have $\eta_1\ (g \star f) \cdot (PoE.cmp\ (g, f) \star e\ (src\ f))$
$= \eta_1\ (g \star f) \cdot (P\ (E\ (g \star f)) \cdot P\ (\Phi_E\ (g, f)) \cdot \Phi_P\ (E\ g, E\ f) \star e\ (src\ f))$
using *f g fg PoE.cmp-def* $VV.arr\text{-}char_{SbC}$ $VV.dom\text{-}char_{SbC}$ **by** *simp*
also have ... = $\eta_1\ (g \star f) \cdot (P\ (E\ (g \star f)) \cdot P\ (g \star_B f) \cdot \Phi_P\ (E\ g, E\ f) \star e\ (src\ f))$
using *f g fg emb.cmp-def* $VV.arr\text{-}char_{SbC}$ **by** *simp*
also have ... = $\eta_1\ (g \star f) \cdot (P\ (E\ (g \star f)) \cdot \Phi_P\ (E\ g, E\ f) \star e\ (src\ f))$
using *f g fg* $hseq_B$ *comp-cod-arr emb.map-def hseq-char′ prj.cmp-simps′*(*1*,*5*)
by *auto*

also have ... $= \eta_1\ (g \star f) \cdot (P\ (g \star_B f) \cdot \Phi_P\ (g, f) \star e\ (src\ f))$
using $hseq_B$ *hseq emb.map-def hcomp-char hseqI′* **by** *auto*
also have ... $= \eta_1\ (g \star f) \cdot (\Phi_P\ (g, f) \cdot (P\ g \star P\ f) \star e\ (src\ f))$
using $hseq_B$ $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ $\Phi_P.naturality$ [*of* (g, f)]
P.FF-def
by *auto*
also have ... $= \eta_1\ (g \star f) \cdot (\Phi_P\ (g, f) \star e\ (src\ f))$
proof −
have $\Phi_P\ (g, f) \cdot (P\ g \star P\ f) = \Phi_P\ (g, f)$
using *f g fg comp-arr-dom* [*of* $\Phi_P\ (g, f)\ P\ g \star P\ f$] $VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *prj.cmp-simps′*(*1*) Φ_P*-simps*(*4*) $hseq_B$)
thus *?thesis* **by** *simp*
qed
also have ... $= \eta_1\ (g \star f) \cdot_B (\Phi_P\ (g, f) \star e\ (src\ f))$
proof −

have *seq* $(\eta_1\ (g \star f))\ (\Phi_P\ (g, f) \star e\ (src\ f))$
proof −
have *cod* $(\Phi_P\ (g, f)) = P\ (g \star f)$
using *f g fg* Φ_P*-simps*(*5*) [*of g f*] $VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *hcomp-eqI hseq* $hseq_B$ *hseqI′*)
moreover have *hseq* $(\Phi_P\ (g, f))\ (e\ (src\ f))$
using *f g fg* $hseq_B$ *1* $B.VV.arr\text{-}char_{SbC}$ **by** *auto*
ultimately show *?thesis*
using *f g fg* $hseq_B$ *1* $B.VV.arr\text{-}char_{SbC}$ P_0*-props prj.cmp-simps emb.map-def*
Φ_P*-simps*(*5*) [*of g f*] *hcomp-eqI* $B.VV.dom\text{-}char_{SbC}$ $B.VV.cod\text{-}char_{SbC}$ *P.FF-def*
by *auto*
qed
thus *?thesis*
using *comp-eqI* **by** *simp*
qed
also have ... $= \eta_1\ (g \star_B f) \cdot_B (\Phi_P\ (g, f) \star_B e\ (src\ f))$
using *f g fg* $hseq_B$ *1* $B.VV.arr\text{-}char_{SbC}$ *hcomp-eqI* **by** *simp*
also have ... $= ((e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ ?a] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a)) \cdot_B$
$(\Phi_P\ (g, f) \star_B e\ ?a)$
unfolding $unit_1$*-def*
using $hseq_B$ *1 comp-char* **by** *simp*
also have ... $= ((e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ ?a] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a)) \cdot_B$
$(((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f,\ d\ ?a]) \cdot_B$

$(e\ ?c \star_B g \star_B$
$\quad \mathrm{l}_B[f \star_B d\ ?a] \cdot_B$
$\quad (B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)) \cdot_B$
$\quad (e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\quad \mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\quad \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$\quad (\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f))$
$\quad\quad \star_B\ e\ ?a)$

proof −
have $B.VV.ide\ (g,\ f)$
using *f g fg ide-char*$_{SbC}$ *src-def trg-def* $B.VV.ide\text{-}char_{SbC}$ **by** *auto*
hence $\Phi_P\ (g,\ f) = CMP\ g\ f$
unfolding Φ_P*-def*
using *f g fg ide-char*$_{SbC}$ *CMP.map-simp-ide* **by** *simp*
also have ... $= (e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a] \cdot_B$
$(B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f)$
using $hseq_B$ *CMP-def* **by** *auto*
finally have $\Phi_P\ (g,\ f) = (e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a] \cdot_B$
$(B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f)$
by *blast*
thus *?thesis* **by** *simp*
qed
also have ... $= ((e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ src\ f] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a)) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f)$
$\star_B\ e\ ?a)$
using *f g fg* $hseq_B$ *src-def trg-def B.whisker-left B.comp-assoc* **by** *simp*
also have ... $= (e\ ?c \star_B \mathrm{r}_B[g \star_B f]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g \star_B f,\ src\ f] \cdot_B$
$((e\ ?c \star_B g \star_B f) \star_B B.inv\ (\eta\ ?a)) \cdot_B$

$\mathrm{a}_B[e\ ?c \star_B g \star_B f,\ d\ ?a,\ e\ ?a] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g \star_B f,\ d\ ?a] \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$

proof −
have $B.arr\ ((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f))$
using $hseq_B$ $ide\text{-}char_{SbC}$ $P.preserves\text{-}ide$ $P\text{-}def$ **by** *auto*
hence $(e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f)$
$\star_B e\ ?a$
$= ((e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \star_B e\ ?a) \cdot_B$
$((e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \star_B e\ ?a) \cdot_B$
$(\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \star_B e\ ?a) \cdot_B$
$((\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \star_B e\ ?a)$
using $hseq_B$ $B.whisker\text{-}right$ **by** *fastforce*
thus *?thesis*
using $B.comp\text{-}assoc$ **by** *simp*
qed
finally show *?thesis*
using f $ide\text{-}char_{SbC}$ $src\text{-}def$ **by** *simp*
qed
finally show $(e\ (trg\ g) \star I_S.cmp\ (g,\ f)) \cdot$
$\mathrm{a}[e\ (trg\ g),\ I_S.map\ g,\ I_S.map\ f] \cdot$
$(\eta_1\ g \star I_S.map\ f) \cdot$
$inv\ \mathrm{a}[PoE.map\ g,\ e\ (src\ g),\ I_S.map\ f] \cdot$
$(PoE.map\ g \star \eta_1\ f) \cdot$
$\mathrm{a}[PoE.map\ g,\ PoE.map\ f,\ e\ (src\ f)]$
$= \eta_1\ (g \star f) \cdot (PoE.cmp\ (g,\ f) \star e\ (src\ f))$
by *blast*
qed

qed

abbreviation (*input*) $counit_0$
where $counit_0 \equiv d$

definition $counit_1$
where $counit_1\ f = \mathrm{a}_B[d\ (trg_B\ f),\ e\ (trg_B\ f),\ f \star_B d\ (src_B\ f)] \cdot_B$
$(\eta\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)) \cdot_B$
$\mathrm{l}_B{}^{-1}[f \star_B d\ (src_B\ f)]$

abbreviation (*input*) ε_0
where $\varepsilon_0 \equiv counit_0$

abbreviation (*input*) ε_1
where $\varepsilon_1 \equiv counit_1$

lemma $counit_1$*-in-hom* [*intro*]:
assumes *B.ide f*
shows «$\varepsilon_1\ f : f \star_B d\ (src_B\ f) \Rightarrow_B d\ (trg_B\ f) \star_B e\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)$»
using *assms B.iso-is-arr*
by (*unfold* $counit_1$*-def*, *intro B.comp-in-homI′ B.seqI B.hseqI′ B.hcomp-in-vhom*) *auto*

lemma $counit_1$*-simps* [*simp*]:
assumes *B.ide f*
shows *B.arr* $(\varepsilon_1\ f)$
and $src_B\ (\varepsilon_1\ f) = P_0\ (src_B\ f)$ **and** $trg_B\ (\varepsilon_1\ f) = trg_B\ f$
and $B.dom\ (\varepsilon_1\ f) = f \star_B d\ (src_B\ f)$
and $B.cod\ (\varepsilon_1\ f) = d\ (trg_B\ f) \star_B e\ (trg_B\ f) \star_B f \star_B d\ (src_B\ f)$
and *B.iso* $(\varepsilon_1\ f)$
using *assms* $counit_1$*-in-hom*
apply *auto*
using *B.vconn-implies-hpar*(*1*)
apply *fastforce*
using *B.vconn-implies-hpar*(*2*)
apply *fastforce*
unfolding $counit_1$*-def*
apply (*intro B.isos-compose*)
by *auto*

lemma *technical*:
assumes *B.ide f* **and** *B.ide g* **and** $src_B\ g = trg_B\ f$
shows $(\varepsilon_1\ g \star_B P\ f) \cdot_B \mathrm{a}_B{}^{-1}[g,\ d\ (src_B\ g),\ P\ f] \cdot_B (g \star_B \varepsilon_1\ f)$
$= (\mathrm{a}_B[d\ (trg_B\ g),\ e\ (trg_B\ g),\ g \star_B d\ (src_B\ g)] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ (trg_B\ g) \star_B e\ (trg_B\ g),\ g,\ d\ (src_B\ g)] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ (trg_B\ g) \star_B e\ (trg_B\ g)) \star_B g,\ d\ (src_B\ g),\ P\ f] \cdot_B$
$(((d\ (trg_B\ g) \star_B e\ (trg_B\ g)) \star_B g) \star_B d\ (src_B\ g) \star_B P\ f) \cdot_B$
$(((d\ (trg_B\ g) \star_B e\ (trg_B\ g)) \star_B g)$
$\star_B \mathrm{a}_B[d\ (src_B\ g),\ e\ (src_B\ g),\ f \star_B d\ (src_B\ f)]) \cdot_B$
$((\eta\ (trg_B\ g) \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B (\eta\ (src_B\ g) \star_B f \star_B d\ (src_B\ f))) \cdot_B$

$(g \star_B \mathrm{a}_B[src_B\ g,\ f,\ d\ (src_B\ f)] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B\ d\ (src_B\ f)))$

proof −

let $?a = src_B\ f$

let $?b = src_B\ g$

let $?c = trg_B\ g$

have $(\varepsilon_1\ g \star_B P\ f) \cdot_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] \cdot_B (g \star_B \varepsilon_1\ f)$

$= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ d\ ?b] \cdot_B$
$(\eta\ ?c \star_B\ g \star_B\ d\ ?b) \cdot_B$
$\mathrm{l}_B{}^{-1}[g \star_B\ d\ ?b]$
$\star_B\ P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B\ d\ ?a] \cdot_B$
$(\eta\ ?b \star_B\ f \star_B\ d\ ?a) \cdot_B$
$\mathrm{l}_B{}^{-1}[f \star_B\ d\ ?a])$

using *assms counit*$_1$*-def* **by** *simp*

also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ d\ ?b] \star_B\ P\ f) \cdot_B$
$((\eta\ ?c \star_B\ g \star_B\ d\ ?b) \star_B\ P\ f) \cdot_B$
$(\mathrm{l}_B{}^{-1}[g \star_B\ d\ ?b] \star_B\ P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B\ d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B\ f \star_B\ d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B\ d\ ?a])$

proof −

have $\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ d\ ?b] \cdot_B$
$(\eta\ ?c \star_B\ g \star_B\ d\ ?b) \cdot_B$
$\mathrm{l}_B{}^{-1}[g \star_B\ d\ ?b]$
$\star_B\ P\ f$
$= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ d\ ?b] \star_B\ P\ f) \cdot_B$
$((\eta\ ?c \star_B\ g \star_B\ d\ ?b) \star_B\ P\ f) \cdot_B$
$(\mathrm{l}_B{}^{-1}[g \star_B\ d\ ?b] \star_B\ P\ f)$

using *assms B.iso-is-arr B.whisker-right P-def* **by** *auto*

moreover have $g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B\ d\ ?a] \cdot_B$
$(\eta\ ?b \star_B\ f \star_B\ d\ ?a) \cdot_B$
$\mathrm{l}_B{}^{-1}[f \star_B\ d\ ?a]$
$= (g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B\ d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B\ f \star_B\ d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B\ d\ ?a])$

using *assms B.iso-is-arr B.whisker-left* **by** *simp*

ultimately show *?thesis*

using *B.comp-assoc* **by** *simp*

qed

also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ d\ ?b] \star_B\ P\ f) \cdot_B$
$((\eta\ ?c \star_B\ g \star_B\ d\ ?b) \star_B\ P\ f) \cdot_B$
$(\mathrm{a}_B[?c,\ g,\ d\ ?b] \cdot_B (\mathrm{l}_B{}^{-1}[g] \star_B\ d\ ?b) \star_B\ P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B\ d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B\ f \star_B\ d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B\ d\ ?a])$

using *assms B.lunit-hcomp* [*of g d ?b*] *B.comp-assoc* **by** *simp*

also have ... = $(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(((\eta\ ?c \star_B g \star_B d\ ?b) \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[?c,\ g,\ d\ ?b] \star_B P\ f)) \cdot_B$
$((\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b) \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
using *assms B.whisker-right P-def B.comp-assoc* **by** *simp*
also have ... = $(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$(((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f) \cdot_B$
$(((\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b) \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f]) \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
proof −
have $((\eta\ ?c \star_B g \star_B d\ ?b) \star_B P\ f) \cdot_B (\mathrm{a}_B[?c,\ g,\ d\ ?b] \star_B P\ f) =$
$(\eta\ ?c \star_B g \star_B d\ ?b) \cdot_B \mathrm{a}_B[?c,\ g,\ d\ ?b] \star_B P\ f$
using *assms B.whisker-right P-def B.iso-is-arr* **by** *simp*
also have ... = $\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \cdot_B ((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f$
using *assms B.assoc-naturality* [*of* η *?c g d ?b*] *B.iso-is-arr* **by** *simp*
also have ... = $(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$(((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f)$
using *assms B.whisker-right P-def B.iso-is-arr* **by** *simp*
finally have $((\eta\ ?c \star_B g \star_B d\ ?b) \star_B P\ f) \cdot_B (\mathrm{a}_B[?c,\ g,\ d\ ?b] \star_B P\ f) =$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$(((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f)$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = $(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$((((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[?c \star_B g,\ d\ ?b,\ P\ f]) \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b \star_B P\ f) \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
proof −
have $((\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b) \star_B P\ f) \cdot_B \mathrm{a}_B{}^{-1}[g,\ d\ ?b,\ P\ f] =$
$\mathrm{a}_B{}^{-1}[?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B (\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b \star_B P\ f)$
using *assms B.assoc′-naturality* [*of* $\mathrm{l}_B{}^{-1}[g]$ *d ?b P f*] **by** *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = $(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$

$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((\eta\ ?c \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B d\ ?b \star_B P\ f)) \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$

proof −
have $(((\eta\ ?c \star_B g) \star_B d\ ?b) \star_B P\ f) \cdot_B \mathrm{a}_B{}^{-1}[?c \star_B g,\ d\ ?b,\ P\ f] =$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B ((\eta\ ?c \star_B g) \star_B d\ ?b \star_B P\ f)$
using *assms B.assoc′-naturality* [*of* $\eta\ ?c \star_B g\ d\ ?b\ P\ f$] *B.iso-is-arr*
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B d\ ?b \star_B P\ f) \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a])) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
using *assms B.whisker-right B.iso-is-arr P-def B.comp-assoc* **by** *simp*
also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(g \star_B \eta\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$

proof −
have $((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B d\ ?b \star_B P\ f) \cdot_B$
$(g \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a])$
$= (((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a])$
using *assms B.comp-arr-dom B.comp-cod-arr B.iso-is-arr P-def*
B.interchange [*of* $(\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]\ g$]
B.interchange [*of* $(d\ ?c \star_B e\ ?c) \star_B g\ (\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]$]
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a]))$

proof −
have $((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B (g \star_B \eta\ ?b \star_B f \star_B d\ ?a)$
$= (((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a)$
using *assms B.comp-arr-dom B.comp-cod-arr B.iso-is-arr P-def*
B.interchange [*of* $(\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]\ g$]
B.interchange [*of* $(d\ ?c \star_B e\ ?c) \star_B g\ (\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]$]
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B (\eta\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a))$
proof −
have $((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B (g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
$= (\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B (\eta\ ?b \star_B f \star_B d\ ?a) \cdot_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a]$
using *assms B.comp-arr-dom B.iso-is-arr P-def*
B.interchange [*of* $(\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]\ g\ \eta\ ?b \star_B f \star_B d\ ?a\ \mathrm{l}_B{}^{-1}[f \star_B d\ ?a]$]
by *simp*
also have ... $= (\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B$
$(\eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)$
using *assms B.lunit-hcomp* **by** *simp*
also have ... $= ((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B (\eta\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a))$
using *assms B.comp-arr-dom B.iso-is-arr P-def*
B.interchange [*of* $(\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g]\ g$]
by *simp*
finally have $((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B (g \star_B \mathrm{l}_B{}^{-1}[f \star_B d\ ?a])$
$= ((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B (\eta\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a))$
by *blast*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed

sublocale *counit*: *pseudonatural-equivalence*
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$\langle E \circ P \rangle$ *EoP.cmp* I_C*.map* I_C*.cmp* $counit_0$ $counit_1$
proof
show $\bigwedge a.\ B.obj\ a \Longrightarrow B.ide\ (d\ a)$
by *simp*

show $\bigwedge a.\ B.obj\ a \Longrightarrow B.equivalence\text{-}map\ (d\ a)$
proof −
  fix a
  assume a: $B.obj\ a$
  interpret Adj: $adjoint\text{-}equivalence\text{-}in\text{-}bicategory\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
    ‹$e\ a$› ‹$d\ a$› ‹$\eta\ a$› ‹$\varepsilon\ a$›
    using a $chosen\text{-}adjoint\text{-}equivalence$ **by** $simp$
  show $B.equivalence\text{-}map\ (d\ a)$
    using $Adj.equivalence\text{-}in\text{-}bicategory\text{-}axioms$ $B.equivalence\text{-}map\text{-}def$ $Adj.dual\text{-}equivalence$
    by $blast$
qed
show $\bigwedge a.\ B.obj\ a \Longrightarrow$ «$d\ a : src_B\ ((E \circ P)\ a) \rightarrow_B src_B\ (I_C.map\ a)$»
  unfolding $emb.map\text{-}def$ **by** $fastforce$
show $\bigwedge f.\ B.ide\ f \Longrightarrow B.iso\ (\varepsilon_1\ f)$
  by $simp$
show $\bigwedge f.\ B.ide\ f \Longrightarrow$ «$\varepsilon_1\ f : I_C.map\ f \star_B d\ (src_B\ f) \Rightarrow_B d\ (trg_B\ f) \star_B (E \circ P)\ f$»
  unfolding $counit_1\text{-}def$ $P\text{-}def$ $emb.map\text{-}def$
  using $P\text{-}def$ $P\text{-}simps(1)$
  by ($intro$ $B.comp\text{-}in\text{-}homI'$ $B.seqI$ $B.hseqI'$) ($auto\ simp\ add$: $B.iso\text{-}is\text{-}arr$)
show $\bigwedge \mu.\ B.arr\ \mu \Longrightarrow \varepsilon_1\ (B.cod\ \mu) \cdot_B (I_C.map\ \mu \star_B d\ (src_B\ \mu))$
    $= (d\ (trg_B\ \mu) \star_B (E \circ P)\ \mu) \cdot_B \varepsilon_1\ (B.dom\ \mu)$
proof −
  fix μ
  assume μ: $B.arr\ \mu$
  let $?a = src_B\ \mu$
  let $?b = trg_B\ \mu$
  let $?f = B.dom\ \mu$
  let $?g = B.cod\ \mu$
  have $\varepsilon_1\ ?g \cdot_B (I_C.map\ \mu \star_B d\ ?a)$
    $= (\mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?g \star_B d\ ?a] \cdot_B$
      $(\eta\ ?b \star_B ?g \star_B d\ ?a) \cdot_B$
      $\mathrm{l}_B{}^{-1}[?g \star_B d\ ?a]) \cdot_B$
      $(\mu \star_B d\ ?a)$
    using μ $counit_1\text{-}def$ $P\text{-}def$ $emb.map\text{-}def$ $arr\text{-}char_{SbC}$ $P_0\text{-}props(1)$ **by** $simp$
  also have ... $= (d\ ?b \star_B e\ ?b \star_B \mu \star_B d\ ?a) \cdot_B$
      $\mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?f \star_B d\ ?a] \cdot_B$
      $(\eta\ ?b \star_B ?f \star_B d\ ?a) \cdot_B$
      $\mathrm{l}_B{}^{-1}[?f \star_B d\ ?a]$
  proof −
    have $(\mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?g \star_B d\ ?a] \cdot_B$
      $(\eta\ ?b \star_B ?g \star_B d\ ?a) \cdot_B$
      $\mathrm{l}_B{}^{-1}[?g \star_B d\ ?a]) \cdot_B$
      $(\mu \star_B d\ ?a)$
      $= \mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?g \star_B d\ ?a] \cdot_B$
        $(\eta\ ?b \star_B ?g \star_B d\ ?a) \cdot_B$
        $\mathrm{l}_B{}^{-1}[?g \star_B d\ ?a] \cdot_B$
        $(\mu \star_B d\ ?a)$
      using $B.comp\text{-}assoc$ **by** $simp$
    also have ... $= \mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?g \star_B d\ ?a] \cdot_B$

$((\eta\ ?b \star_B\ ?g \star_B\ d\ ?a) \cdot_B$
$(?b \star_B\ \mu \star_B\ d\ ?a)) \cdot_B$
$\mathrm{l}_B{}^{-1}[?f \star_B\ d\ ?a]$
using μ *B.lunit'-naturality* [*of* $\mu \star_B\ d\ ?a$] *B.comp-assoc* **by** *simp*
also have ... $= (\mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?g \star_B\ d\ ?a] \cdot_B$
$((d\ ?b \star_B\ e\ ?b) \star_B\ \mu \star_B\ d\ ?a)) \cdot_B$
$(\eta\ ?b \star_B\ ?f \star_B\ d\ ?a) \cdot_B$
$\mathrm{l}_B{}^{-1}[?f \star_B\ d\ ?a]$
proof −
have $(\eta\ ?b \star_B\ ?g \star_B\ d\ ?a) \cdot_B (?b \star_B\ \mu \star_B\ d\ ?a) =$
$\eta\ ?b \cdot_B\ ?b \star_B\ ?g \cdot_B\ \mu \star_B\ d\ ?a \cdot_B\ d\ ?a$
using μ *B.iso-is-arr*
B.interchange [*of* $\eta\ ?b\ ?b\ (?g \star_B\ d\ ?a)\ \mu \star_B\ d\ ?a$]
B.interchange [*of* $?g\ \mu\ d\ ?a\ d\ ?a$]
by *simp*
also have ... $= (d\ ?b \star_B\ e\ ?b) \cdot_B\ \eta\ ?b \star_B\ \mu \cdot_B\ ?f \star_B\ d\ ?a \cdot_B\ d\ ?a$
using μ *B.comp-arr-dom B.comp-cod-arr* **apply** *simp*
by (*metis B.arrI equivalence-data-in-hom*$_B$(*7*) *equivalence-data-simps*$_B$(*17*−*18*) *B.obj-trg*)
also have ... $= ((d\ ?b \star_B\ e\ ?b) \star_B\ \mu \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B\ ?f \star_B\ d\ ?a)$
using μ *B.iso-is-arr B.interchange* [*of* $d\ ?b \star_B\ e\ ?b\ \eta\ ?b\ \mu \star_B\ d\ ?a\ ?f \star_B\ d\ ?a$]
B.interchange [*of* $\mu\ ?f\ d\ ?a\ d\ ?a$]
by *simp*
finally have $(\eta\ ?b \star_B\ ?g \star_B\ d\ ?a) \cdot_B (?b \star_B\ \mu \star_B\ d\ ?a)$
$= ((d\ ?b \star_B\ e\ ?b) \star_B\ \mu \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B\ ?f \star_B\ d\ ?a)$
by *blast*
thus *?thesis* **using** *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?b \star_B\ e\ ?b \star_B\ \mu \star_B\ d\ ?a) \cdot_B$
$\mathrm{a}_B[d\ ?b,\ e\ ?b,\ ?f \star_B\ d\ ?a] \cdot_B$
$(\eta\ ?b \star_B\ ?f \star_B\ d\ ?a) \cdot_B$
$\mathrm{l}_B{}^{-1}[?f \star_B\ d\ ?a]$
using μ *B.assoc-naturality* [*of* $d\ ?b\ e\ ?b\ \mu \star_B\ d\ ?a$] *B.comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= (d\ ?b \star_B\ (E \circ P)\ \mu) \cdot_B\ \varepsilon_1\ ?f$
using μ *counit*$_1$*-def P-def emb.map-def arr-char*$_{SbC}$ P_0*-props*(*1*) *P-simps*$_B$(*1*)
by (*simp add*: P_0*-props*(*6*))
finally show $\varepsilon_1\ (?g) \cdot_B (I_C.map\ \mu \star_B\ d\ ?a) = (d\ ?b \star_B\ (E \circ P)\ \mu) \cdot_B\ \varepsilon_1\ ?f$
by *blast*
qed
show $\bigwedge f\ g.\ [\![B.ide\ f;\ B.ide\ g;\ src_B\ g = trg_B\ f]\!] \Longrightarrow$
$(d\ (trg_B\ g) \star_B\ EoP.cmp\ (g, f)) \cdot_B$
$\mathrm{a}_B[d\ (trg_B\ g),\ (E \circ P)\ g,\ (E \circ P)\ f] \cdot_B$
$(\varepsilon_1\ g \star_B\ (E \circ P)\ f) \cdot_B$
$B.inv\ \mathrm{a}_B[I_C.map\ g,\ d\ (src_B\ g),\ (E \circ P)\ f] \cdot_B (I_C.map\ g \star_B\ \varepsilon_1\ f) \cdot_B$
$\mathrm{a}_B[I_C.map\ g,\ I_C.map\ f,\ d\ (src_B\ f)]$
$= \varepsilon_1\ (g \star_B\ f) \cdot_B (I_C.cmp\ (g, f) \star_B\ d\ (src_B\ f))$
proof −

fix f g
assume f: $B.ide\ f$ **and** g: $B.ide\ g$ **and** fg: $src_B\ g = trg_B\ f$
let $?a = src_B\ f$
let $?b = src_B\ g$
let $?c = trg_B\ g$
have $(d\ ?c \star_B EoP.cmp\ (g, f)) \cdot_B$
$\quad \mathrm{a}_B[d\ ?c, (E \circ P)\ g, (E \circ P)\ f] \cdot_B$
$\quad (\varepsilon_1\ g \star_B (E \circ P)\ f) \cdot_B$
$\quad B.inv\ \mathrm{a}_B[I_C.map\ g, d\ ?b, (E \circ P)\ f] \cdot_B$
$\quad (I_C.map\ g \star_B \varepsilon_1\ f) \cdot_B$
$\quad \mathrm{a}_B[I_C.map\ g, I_C.map\ f, d\ ?a]$
$\quad = (d\ ?c \star_B P\ (g \star_B f) \cdot_B CMP.map\ (g, f) \cdot_B (P\ g \star_B P\ f)) \cdot_B$
$\quad\quad \mathrm{a}_B[d\ ?c, P\ g, P\ f] \cdot_B$
$\quad\quad (\varepsilon_1\ g \star_B P\ f) \cdot_B$
$\quad\quad \mathrm{a}_B{}^{-1}[g, d\ ?b, P\ f] \cdot_B$
$\quad\quad (g \star_B \varepsilon_1\ f) \cdot_B$
$\quad\quad \mathrm{a}_B[g, f, d\ ?a]$
proof −
have $src_B\ (e\ ?c \star_B g \star_B d\ ?b) = trg_B\ (e\ ?b \star_B f \star_B d\ ?a)$
using $f\ g\ fg$ *src-def trg-def* $arr\text{-}char_{SbC}$ $P\text{-}simps_B(1)$
by (*simp add*: P_0*-props*(*1*,*6*) $arr\text{-}char_{SbC}$)
moreover have $B.inv\ \mathrm{a}_B[g, d\ ?b, P\ f] = \mathrm{a}_B{}^{-1}[g, d\ ?b, P\ f]$
using $f\ g\ fg$ **by** (*simp add*: *P-def*)
ultimately show *?thesis*
using $f\ g\ fg$ *emb.map-def P.preserves-reflects-arr P.preserves-reflects-arr*
P.preserves-reflects-arr Φ_P*-def emb.cmp-def*
EoP.cmp-def $B.VV.arr\text{-}char_{SbC}$ $B.VV.dom\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
by *simp*
qed
also have ... $= (d\ ?c \star_B CMP.map\ (g, f)) \cdot_B$
$\quad \mathrm{a}_B[d\ ?c, P\ g, P\ f] \cdot_B$
$\quad ((\varepsilon_1\ g \star_B P\ f) \cdot_B$
$\quad \mathrm{a}_B{}^{-1}[g, d\ ?b, P\ f] \cdot_B$
$\quad (g \star_B \varepsilon_1\ f)) \cdot_B$
$\quad \mathrm{a}_B[g, f, d\ ?a]$
proof −
have «$CMP.map\ (g, f) : P\ g \star P\ f \Rightarrow P\ (g \star_B f)$»
using $f\ g\ fg$ $VV.arr\text{-}char_{SbC}$ $VV.cod\text{-}char_{SbC}$ *P.FF-def* Φ_P*-def* Φ_P*-in-hom*(*2*) **by** *auto*
hence «$CMP.map\ (g, f) : P\ g \star_B P\ f \Rightarrow_B P\ (g \star_B f)$»
using $in\text{-}hom\text{-}char_{SbC}$ *hcomp-char* **by** *auto*
hence $P\ (g \star_B f) \cdot_B CMP.map\ (g, f) \cdot_B (P\ g \star_B P\ f) = CMP.map\ (g, f)$
using *B.comp-arr-dom B.comp-cod-arr* **by** *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?c \star_B CMP.map\ (g, f)) \cdot_B$
$\quad \mathrm{a}_B[d\ ?c, P\ g, P\ f] \cdot_B$
$\quad (\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B d\ ?b] \star_B P\ f) \cdot_B$

$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg technical B.comp-assoc* **by** *simp*
also have ... $= (d\ ?c \star_B (e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$(e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f] \cdot_B$
$\mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f)) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ P\ g,\ P\ f] \cdot_B$
$(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg CMP.map-simp-ide CMP-def B.VV.ide-char$_{SbC}$ B.VV.arr-char$_{SbC}$ B.whisker-left B.whisker-left B.comp-assoc*
by *simp*
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$((d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c,\ g,\ d\ ?b \star_B P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c \star_B g,\ d\ ?b,\ P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B{}^{-1}[e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ P\ g,\ P\ f] \cdot_B$
$(\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g,\ d\ ?b,\ P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b,\ e\ ?b,\ f \star_B d\ ?a])) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg B.whisker-left P-def B.comp-assoc* **by** *simp*
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$((d\ ?c \star_B e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a] \cdot_B$

$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b, f, d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g, f, d\ ?a]$

proof −
have $(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b, e\ ?b, f \star_B d\ ?a]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c, g, d\ ?b \star_B P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c \star_B g, d\ ?b, P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B{}^{-1}[e\ ?c, g, d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B[d\ ?c, P\ g, P\ f] \cdot_B$
$(\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g, d\ ?b, P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b, e\ ?b, f \star_B d\ ?a])$
$= \mathrm{a}_B[d\ ?c, e\ ?c, g \star_B (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a] \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a]$

proof −
have $(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{a}_B{}^{-1}[d\ ?b, e\ ?b, f \star_B d\ ?a]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c, g, d\ ?b \star_B P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B[e\ ?c \star_B g, d\ ?b, P\ f]) \cdot_B$
$(d\ ?c \star_B \mathrm{a}_B{}^{-1}[e\ ?c, g, d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B[d\ ?c, P\ g, P\ f] \cdot_B$
$(\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B d\ ?b] \star_B P\ f) \cdot_B$
$(\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, d\ ?b] \star_B P\ f) \cdot_B$
$\mathrm{a}_B{}^{-1}[(d\ ?c \star_B e\ ?c) \star_B g, d\ ?b, P\ f] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B d\ ?b \star_B P\ f) \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B \mathrm{a}_B[d\ ?b, e\ ?b, f \star_B d\ ?a])$
$= \{\!|(\langle d\ ?c\rangle \star \langle e\ ?c\rangle \star \langle g\rangle \star \mathbf{a}^{-1}[\langle d\ ?b\rangle, \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle]) \cdot$
$(\langle d\ ?c\rangle \star \mathbf{a}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle \star \mathbf{PRJ}\ f]) \cdot$
$(\langle d\ ?c\rangle \star \mathbf{a}[\langle e\ ?c\rangle \star \langle g\rangle, \langle d\ ?b\rangle, \mathbf{PRJ}\ f]) \cdot$
$(\langle d\ ?c\rangle \star \mathbf{a}^{-1}[\langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle] \star \mathbf{PRJ}\ f) \cdot$
$\mathbf{a}[\langle d\ ?c\rangle, \mathbf{PRJ}\ g, \mathbf{PRJ}\ f] \cdot$
$(\mathbf{a}[\langle d\ ?c\rangle, \langle e\ ?c\rangle, \langle g\rangle \star \langle d\ ?b\rangle] \star \mathbf{PRJ}\ f) \cdot$
$(\mathbf{a}[\langle d\ ?c\rangle \star \langle e\ ?c\rangle, \langle g\rangle, \langle d\ ?b\rangle] \star \mathbf{PRJ}\ f) \cdot$
$\mathbf{a}^{-1}[(\langle d\ ?c\rangle \star \langle e\ ?c\rangle) \star \langle g\rangle, \langle d\ ?b\rangle, \mathbf{PRJ}\ f] \cdot$
$((((\langle d\ ?c\rangle \star \langle e\ ?c\rangle) \star \langle g\rangle) \star \langle d\ ?b\rangle \star \mathbf{PRJ}\ f) \cdot$
$((((\langle d\ ?c\rangle \star \langle e\ ?c\rangle) \star \langle g\rangle) \star$
$\mathbf{a}[\langle d\ ?b\rangle, \langle e\ ?b\rangle, \langle f\rangle \star \langle d\ ?a\rangle])\}\!|$
using *f g fg B.α-def B.α′.map-ide-simp B.VVV.ide-char$_{SbC}$ B.VVV.arr-char$_{SbC}$ B.VV.ide-char$_{SbC}$ B.VV.arr-char$_{SbC}$ P-def B.ℓ-ide-simp*
by *auto*
also have ... $= \{\!|\mathbf{a}[\langle d\ ?c\rangle, \langle e\ ?c\rangle,$
$\langle g\rangle \star (\langle d\ ?b\rangle \star \langle e\ ?b\rangle) \star \langle f\rangle \star \langle d\ ?a\rangle] \cdot$
$\mathbf{a}[\langle d\ ?c\rangle \star \langle e\ ?c\rangle, \langle g\rangle,$
$(\langle d\ ?b\rangle \star \langle e\ ?b\rangle) \star \langle f\rangle \star \langle d\ ?a\rangle]\}\!|$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= \mathrm{a}_B[d\ ?c, e\ ?c, g \star_B (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a] \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a]$
using *f g fg B.α-def B.α′.map-ide-simp B.VVV.ide-char$_{SbC}$ B.VVV.arr-char$_{SbC}$*

$B.VV.ide\text{-}char_{SbC}$ $B.VV.arr\text{-}char_{SbC}$ $P\text{-}def$ $B.\mathfrak{l}\text{-}ide\text{-}simp$
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f, d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B ?b \star_B f \star_B d\ ?a] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a]) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b, f, d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g, f, d\ ?a]$
proof –
have $(d\ ?c \star_B e\ ?c \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a] =$
$\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B ?b \star_B f \star_B d\ ?a] \cdot_B$
$((d\ ?c \star_B e\ ?c) \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)$
using *f g fg*
B.assoc-naturality
[*of* $d\ ?c\ e\ ?c\ g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a$]
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f, d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c, e\ ?c, g \star_B ?b \star_B f \star_B d\ ?a] \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, ?b \star_B f \star_B d\ ?a] \cdot_B$
$((((d\ ?c \star_B e\ ?c) \star_B g) \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$((\eta\ ?c \star_B g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B \eta\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(g \star_B \mathrm{a}_B[?b, f, d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g, f, d\ ?a]$
proof –
have $((d\ ?c \star_B e\ ?c) \star_B g \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, (d\ ?b \star_B e\ ?b) \star_B f \star_B d\ ?a] =$
$\mathrm{a}_B[d\ ?c \star_B e\ ?c, g, ?b \star_B f \star_B d\ ?a] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g) \star_B B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a)$
using *f g fg*
B.assoc-naturality
[*of* $d\ ?c \star_B e\ ?c\ g\ B.inv\ (\eta\ ?b) \star_B f \star_B d\ ?a$]
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g, f, d\ ?a]) \cdot_B$
$(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$

$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ ?b \star_B f \star_B\ d\ ?a] \cdot_B$
$\mathrm{a}_B[d\ ?c \star_B\ e\ ?c,\ g,\ ?b \star_B f \star_B\ d\ ?a] \cdot_B$
$((\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B\ d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B\ d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
proof −
have $(((d\ ?c \star_B\ e\ ?c) \star_B\ g) \star_B\ B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B$
$((\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B\ \eta\ ?b \star_B f \star_B\ d\ ?a)$
$= (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B\ d\ ?a$
proof −
have $(((d\ ?c \star_B\ e\ ?c) \star_B\ g) \star_B\ B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B$
$((\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B\ \eta\ ?b \star_B f \star_B\ d\ ?a)$
$= ((d\ ?c \star_B\ e\ ?c) \star_B\ g) \cdot_B (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B$
$(B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B f \star_B\ d\ ?a)$
using *f g fg*
$B.interchange\ [of\ (d\ ?c \star_B\ e\ ?c) \star_B\ g\ (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g]$
$B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a$
$\eta\ ?b \star_B f \star_B\ d\ ?a]$
by *fastforce*
also have ... $= (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B$
$(B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B f \star_B\ d\ ?a)$
using *f g fg B.comp-cod-arr* $[of\ (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g]\ (d\ ?c \star_B\ e\ ?c) \star_B\ g]$
by (*auto simp add*: *B.iso-is-arr*)
also have ... $= (\eta\ ?c \star_B\ g) \cdot_B \mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B\ d\ ?a$
proof −
have $(B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B f \star_B\ d\ ?a)$
$= B.inv\ (\eta\ ?b) \cdot_B \eta\ ?b \star_B f \star_B\ d\ ?a$
proof −
have *B.seq* $(B.inv\ (\eta\ ?b))\ (\eta\ ?b)$
using *f g fg chosen-adjoint-equivalence*(*5*) **by** *fastforce*
thus *?thesis*
using *f g fg B.whisker-right* **by** *simp*
qed
also have ... $= ?b \star_B f \star_B\ d\ ?a$
using *f g fg chosen-adjoint-equivalence*(*5*) *B.comp-inv-arr′* **by** *simp*
finally have $(B.inv\ (\eta\ ?b) \star_B f \star_B\ d\ ?a) \cdot_B (\eta\ ?b \star_B f \star_B\ d\ ?a)$
$= ?b \star_B f \star_B\ d\ ?a$
by *blast*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= (d\ ?c \star_B\ e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(d\ ?c \star_B\ e\ ?c \star_B\ g \star_B \mathrm{l}_B[f \star_B\ d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ ?b \star_B f \star_B\ d\ ?a] \cdot_B$

$(\mathrm{a}_B[d\ ?c \star_B e\ ?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$((\eta\ ?c \star_B g) \star_B\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg B.comp-assoc*
B.whisker-right [*of* $?b \star_B f \star_B d\ ?a\ \eta\ ?c \star_B g\ \mathrm{l}_B{}^{-1}[g]$]
by *fastforce*
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$((d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ ?b \star_B f \star_B d\ ?a]) \cdot_B$
$(\eta\ ?c \star_B g \star_B\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg B.iso-is-arr B.comp-assoc*
B.assoc-naturality [*of* $\eta\ ?c\ g\ ?b \star_B f \star_B d\ ?a$]
by *simp*
also have ... $= (d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B f \star_B d\ ?a] \cdot_B$
$(((d\ ?c \star_B e\ ?c) \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$(\eta\ ?c \star_B g \star_B\ ?b \star_B f \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
proof −
have $(d\ ?c \star_B e\ ?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B\ ?b \star_B f \star_B d\ ?a] =$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B f \star_B d\ ?a] \cdot_B$
$((d\ ?c \star_B e\ ?c) \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a])$
using *f g fg B.iso-is-arr*
B.assoc-naturality [*of* $d\ ?c\ e\ ?c\ g \star_B \mathrm{l}_B[f \star_B d\ ?a]$]
by *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... $= ((d\ ?c \star_B e\ ?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ e\ ?c,\ g \star_B f \star_B d\ ?a]) \cdot_B$
$(\eta\ ?c \star_B g \star_B f \star_B d\ ?a) \cdot_B$
$(?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B\ ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
proof −
have $((d\ ?c \star_B e\ ?c) \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B (\eta\ ?c \star_B g \star_B\ ?b \star_B f \star_B d\ ?a)$
$= (\eta\ ?c \star_B g \star_B f \star_B d\ ?a) \cdot_B (?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a])$

using *f g fg B.comp-arr-dom B.comp-cod-arr B.iso-is-arr*
B.interchange [*of d ?c* $\star_B$ *e ?c η ?c*
g $\star_B$ l_B[*f* $\star_B$ *d ?a*] *g* $\star_B$ *?b* $\star_B$ *f* $\star_B$ *d ?a*]
B.interchange [*of η ?c ?c*
g $\star_B$ *f* $\star_B$ *d ?a g* $\star_B$ l_B[*f* $\star_B$ *d ?a*]]
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = a_B[*d ?c*, *e ?c*, (*g* $\star_B$ *f*) $\star_B$ *d ?a*] $\cdot_B$
(((*d ?c* $\star_B$ *e ?c*) $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]) $\cdot_B$
(*η ?c* $\star_B$ *g* $\star_B$ *f* $\star_B$ *d ?a*)) $\cdot_B$
(*?c* $\star_B$ *g* $\star_B$ l_B[*f* $\star_B$ *d ?a*]) $\cdot_B$
a_B[*?c*, *g*, *?b* $\star_B$ *f* $\star_B$ *d ?a*] $\cdot_B$
($\mathrm{l}_B{}^{-1}$[*g*] $\star_B$ *?b* $\star_B$ *f* $\star_B$ *d ?a*) $\cdot_B$
(*g* $\star_B$ a_B[*?b*, *f*, *d ?a*] $\cdot_B$ ($\mathrm{l}_B{}^{-1}$[*f*] $\star_B$ *d ?a*)) $\cdot_B$
a_B[*g*, *f*, *d ?a*]
proof −
have (*d ?c* $\star_B$ *e ?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]) $\cdot_B$
a_B[*d ?c*, *e ?c*, *g* $\star_B$ *f* $\star_B$ *d ?a*] =
a_B[*d ?c*, *e ?c*, (*g* $\star_B$ *f*) $\star_B$ *d ?a*] $\cdot_B$
((*d ?c* $\star_B$ *e ?c*) $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*])
using *f g fg B.assoc-naturality* [*of d ?c e ?c* $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]] **by** *simp*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = a_B[*d ?c*, *e ?c*, (*g* $\star_B$ *f*) $\star_B$ *d ?a*] $\cdot_B$
(*η ?c* $\star_B$ (*g* $\star_B$ *f*) $\star_B$ *d ?a*) $\cdot_B$
(*?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]) $\cdot_B$
(*?c* $\star_B$ *g* $\star_B$ l_B[*f* $\star_B$ *d ?a*]) $\cdot_B$
a_B[*?c*, *g*, *?b* $\star_B$ *f* $\star_B$ *d ?a*] $\cdot_B$
($\mathrm{l}_B{}^{-1}$[*g*] $\star_B$ *?b* $\star_B$ *f* $\star_B$ *d ?a*) $\cdot_B$
(*g* $\star_B$ a_B[*?b*, *f*, *d ?a*] $\cdot_B$ ($\mathrm{l}_B{}^{-1}$[*f*] $\star_B$ *d ?a*)) $\cdot_B$
a_B[*g*, *f*, *d ?a*]
proof −
have ((*d ?c* $\star_B$ *e ?c*) $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]) $\cdot_B$ (*η ?c* $\star_B$ *g* $\star_B$ *f* $\star_B$ *d ?a*)
= (*η ?c* $\star_B$ (*g* $\star_B$ *f*) $\star_B$ *d ?a*) $\cdot_B$ (*?c* $\star_B$ $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*])
using *f g fg B.comp-arr-dom B.comp-cod-arr B.iso-is-arr*
B.interchange [*of d ?c* $\star_B$ *e ?c η ?c* $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*] *g* $\star_B$ *f* $\star_B$ *d ?a*]
B.interchange [*of η ?c ?c* (*g* $\star_B$ *f*) $\star_B$ *d ?a* $\mathrm{a}_B{}^{-1}$[*g*, *f*, *d ?a*]]
by *auto*
thus *?thesis*
using *B.comp-assoc* **by** *simp*
qed
also have ... = ε_1 (*g* $\star_B$ *f*) $\cdot_B$ (I_C*.cmp* (*g*, *f*) $\star_B$ *d ?a*)
proof −
have ε_1 (*g* $\star_B$ *f*) $\cdot_B$ (I_C*.cmp* (*g*, *f*) $\star_B$ *d ?a*)
= a_B[*d ?c*, *e ?c*, (*g* $\star_B$ *f*) $\star_B$ *d ?a*] $\cdot_B$
(*η ?c* $\star_B$ (*g* $\star_B$ *f*) $\star_B$ *d ?a*) $\cdot_B$

$\mathrm{l}_B{}^{-1}[(g \star_B f) \star_B d\ ?a] \cdot_B ((g \star_B f) \star_B d\ ?a)$
using *f g fg counit*$_1$*-def B.comp-assoc* **by** *simp*
also have ... $= \mathrm{a}_B[d\ ?c,\ e\ ?c,\ (g \star_B f) \star_B d\ ?a] \cdot_B$
$(\eta\ ?c \star_B (g \star_B f) \star_B d\ ?a) \cdot_B$
$(?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
proof −
have $\mathrm{l}_B{}^{-1}[(g \star_B f) \star_B d\ ?a] \cdot_B ((g \star_B f) \star_B d\ ?a)$
$= (?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
proof −
have $\mathrm{l}_B{}^{-1}[(g \star_B f) \star_B d\ ?a] \cdot_B ((g \star_B f) \star_B d\ ?a) =$
$\{\!|\mathbf{l}^{-1}[(\langle g\rangle \star \langle f\rangle) \star \langle d\ ?a\rangle] \cdot ((\langle g\rangle \star \langle f\rangle) \star \langle d\ ?a\rangle)|\!\}$
using *f g fg B.*$\mathfrak{l}$*-ide-simp* **by** *auto*
also have ... $= \{\!|(\langle ?c\rangle_0 \star \mathbf{a}^{-1}[\langle g\rangle,\ \langle f\rangle,\ \langle d\ ?a\rangle]) \cdot$
$(\langle ?c\rangle_0 \star \langle g\rangle \star \mathbf{l}[\langle f\rangle \star \langle d\ ?a\rangle]) \cdot$
$\mathbf{a}[\langle ?c\rangle_0,\ \langle g\rangle,\ \langle ?b\rangle_0 \star \langle f\rangle \star \langle d\ ?a\rangle] \cdot$
$(\mathbf{l}^{-1}[\langle g\rangle] \star \langle ?b\rangle_0 \star \langle f\rangle \star \langle d\ ?a\rangle) \cdot$
$(\langle g\rangle \star \mathbf{a}[\langle ?b\rangle_0,\ \langle f\rangle,\ \langle d\ ?a\rangle] \cdot (\mathbf{l}^{-1}[\langle f\rangle] \star \langle d\ ?a\rangle)) \cdot$
$\mathbf{a}[\langle g\rangle,\ \langle f\rangle,\ \langle d\ ?a\rangle]|\!\}$
using *f g fg* **by** (*intro EV.eval-eqI*, *auto*)
also have ... $= (?c \star_B \mathrm{a}_B{}^{-1}[g,\ f,\ d\ ?a]) \cdot_B$
$(?c \star_B g \star_B \mathrm{l}_B[f \star_B d\ ?a]) \cdot_B$
$\mathrm{a}_B[?c,\ g,\ ?b \star_B f \star_B d\ ?a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[g] \star_B ?b \star_B f \star_B d\ ?a) \cdot_B$
$(g \star_B \mathrm{a}_B[?b,\ f,\ d\ ?a] \cdot_B (\mathrm{l}_B{}^{-1}[f] \star_B d\ ?a)) \cdot_B$
$\mathrm{a}_B[g,\ f,\ d\ ?a]$
using *f g fg B.*α*-def B.*α'*.map-ide-simp B.VVV.ide-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$
B.VV.ide-char$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *P-def B.*$\mathfrak{l}$*-ide-simp*
by *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
finally show $(d\ ?c \star_B EoP.cmp\ (g,\ f)) \cdot_B$
$\mathrm{a}_B[d\ ?c,\ (E \circ P)\ g,\ (E \circ P)\ f] \cdot_B$
$(\varepsilon_1\ g \star_B (E \circ P)\ f) \cdot_B$
$B.inv\ \mathrm{a}_B[I_C.map\ g,\ d\ ?b,\ (E \circ P)\ f] \cdot_B (I_C.map\ g \star_B \varepsilon_1\ f) \cdot_B$
$\mathrm{a}_B[I_C.map\ g,\ I_C.map\ f,\ d\ ?a]$

$= \varepsilon_1\ (g \star_B f) \cdot_B (I_C.cmp\ (g, f) \star_B d\ ?a)$
by *blast*
qed
show $\bigwedge a.\ B.obj\ a \Longrightarrow$
$(d\ a \star_B EoP.unit\ a) \cdot_B \mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a] = \varepsilon_1\ a \cdot_B (I_C.unit\ a \star_B d\ a)$
proof −
fix *a*
assume *a*: $B.obj\ a$
interpret *adjoint-equivalence-in-bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ ‹*e a*› ‹*d a*› ‹η *a*› ‹ε *a*›
using *a chosen-adjoint-equivalence* **by** *simp*
have *0*: $src_B\ a = a \wedge trg_B\ a = a$
using *a* **by** *auto*
have *1*: $obj\ (P_0\ a)$
using *a* P_0*-props*(*1*) **by** *simp*
have $(d\ a \star_B EoP.unit\ a) \cdot_B \mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]$
$= (d\ a \star_B ((e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B src_B\ (a \star_B d\ (src_B\ a))) \cdot_B$
$\mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]$
proof −
have $B.hseq\ (e\ (trg_B\ a))\ (a \star_B d\ (src_B\ a))$
using *a* **by** (*elim B.objE*, *intro B.hseqI′*) *auto*
moreover have $arr\ (src_B\ (d\ (src_B\ a)))$
using *a* $arr\text{-}char_{SbC}$ P_0*-props*(*1*) *obj-char B.obj-simps*(*1*) **by** *auto*
moreover have $arr\ (src_B\ (a \star_B d\ (src_B\ a)))$
using *a* $arr\text{-}char_{SbC}$ P_0*-props*(*1*) *obj-char calculation*(*1*) **by** *fastforce*
moreover have $src_B\ (a \star_B d\ (src_B\ a)) \in Obj \wedge trg_B\ (e\ (trg_B\ a)) \in Obj$
using *a B.obj-simps* P_0*-props obj-char* $arr\text{-}char_{SbC}$ **by** *simp*
moreover have $P_0\ a \cdot_B P_0\ a \in Obj$
using *1* $arr\text{-}char_{SbC}$ P_0*-props*(*1*) *obj-char*
by (*metis* (*no-types*, *lifting*) *B.cod-trg B.obj-def′ B.trg.as-nat-trans.naturality2*)
moreover have $emb.unit\ (P_0\ (src_B\ a)) = P_0\ (src_B\ a)$
using *a 0 1 emb.unit-char′* $P.map_0$*-def src-def* **by** *simp*
ultimately show *?thesis*
using *a emb.map-def EoP.unit-char′ prj-unit-char emb.unit-char′* $P.map_0$*-def P-def*
src-def $arr\text{-}char_{SbC}$ *obj-char*
by *simp*
qed
also have ... $= (d\ a \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a) \cdot_B P_0\ a) \cdot_B$
$\mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]$
using *a 1 B.comp-assoc B.obj-simps* **by** *auto*
also have ... $= (d\ a \star_B (e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B B.inv\ (\varepsilon\ a)) \cdot_B \mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]$
using *a B.comp-arr-dom* **by** *simp*
also have ... $= (d\ a \star_B e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B (d\ a \star_B B.inv\ (\varepsilon\ a)) \cdot_B \mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]$
using *a B.whisker-left B.comp-assoc* **by** *simp*
also have ... $= ((d\ a \star_B e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B \mathrm{a}_B[d\ a,\ e\ a,\ d\ a]) \cdot_B (\eta\ a \star_B d\ a)$
using *a triangle-right B.comp-assoc*
B.invert-side-of-triangle(*1*) [*of* $\mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a]\ d\ a \star_B \varepsilon\ a$]
by *simp*
also have ... $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B ((d\ a \star_B e\ a) \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B (\eta\ a \star_B d\ a)$

proof −
  have $(d\ a \star_B e\ a \star_B \mathrm{l}_B{}^{-1}[d\ a]) \cdot_B \mathrm{a}_B[d\ a,\ e\ a,\ d\ a] =$
    $\mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B ((d\ a \star_B e\ a) \star_B \mathrm{l}_B{}^{-1}[d\ a])$
    using *a B.assoc-naturality* [*of d a e a* $\mathrm{l}_B{}^{-1}[d\ a]$] **by** *simp*
  thus *?thesis*
    using *B.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
  using *a B.interchange* [*of d a* $\star_B$ *e a* η *a* $\mathrm{l}_B{}^{-1}[d\ a]$ *d a*]
    B.comp-arr-dom B.comp-cod-arr
  by *simp*
also have ... $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B[a,\ a,\ d\ a] \cdot_B$
    $(\mathrm{l}_B{}^{-1}[a] \star_B d\ a)$
proof −
  have $\mathrm{a}_B[a,\ a,\ d\ a] \cdot_B (\mathrm{l}_B{}^{-1}[a] \star_B d\ a) = a \star_B \mathrm{l}_B{}^{-1}[d\ a]$
  proof −

    have $\mathrm{a}_B[a,\ a,\ d\ a] \cdot_B (\mathrm{l}_B{}^{-1}[a] \star_B d\ a) = \{\!|\mathbf{a}[\langle a\rangle_0,\ \langle a\rangle_0,\ \langle d\ a\rangle] \cdot (\mathbf{l}^{-1}[\langle a\rangle_0] \star \langle d\ a\rangle)|\!\}$
    using *a ide-char B.α-def B.α′.map-ide-simp B.VVV.ide-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$
      B.VV.ide-char$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *P-def B.𝔩-ide-simp*
      by *auto*
    also have ... $= \{\!|\langle a\rangle_0 \star \mathbf{l}^{-1}[\langle d\ a\rangle]|\!\}$
      using *a ide-char*$_{SbC}$ **by** (*intro EV.eval-eqI*, *auto*)
    also have ... $= a \star_B \mathrm{l}_B{}^{-1}[d\ a]$
    using *a ide-char B.α-def B.α′.map-ide-simp B.VVV.ide-char*$_{SbC}$ *B.VVV.arr-char*$_{SbC}$
      B.VV.ide-char$_{SbC}$ *B.VV.arr-char*$_{SbC}$ *P-def B.𝔩-ide-simp*
      by *auto*
    finally show *?thesis* **by** *blast*
  qed
  hence $\mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B[a,\ a,\ d\ a] \cdot_B$
    $(\mathrm{l}_B{}^{-1}[a] \star_B d\ a)$
      $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B (a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
    by *simp*
  also have ... $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B \mathrm{l}_B{}^{-1}[d\ a])$
    using *a B.interchange* [*of* η *a a a* $\star_B$ *d a* $\mathrm{l}_B{}^{-1}[d\ a]$] *B.comp-arr-dom B.comp-cod-arr*
    by *simp*
  finally show *?thesis* **by** *simp*
qed
also have ... $= \varepsilon_1\ a \cdot_B (I_C.unit\ a \star_B d\ a)$
proof −
  have $\varepsilon_1\ a \cdot_B (I_C.unit\ a \star_B d\ a) =$
    $\mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{l}_B{}^{-1}[a \star_B d\ a] \cdot_B (a \star_B d\ a)$
    using *a 0 counit*$_1$*-def* I_C*.unit-char′ B.comp-assoc* **by** *simp*
  also have ... $= \mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{l}_B{}^{-1}[a \star_B d\ a]$
  proof −
    have *B.arr* $\mathrm{l}_B{}^{-1}[a \star_B d\ a]$
      using *a*
    by (*metis B.ide-hcomp B.lunit′-simps*(*1*) *B.objE equivalence-data-simps*$_B$(*8*) *ide-right*)
    moreover have *B.dom* $\mathrm{l}_B{}^{-1}[a \star_B d\ a] = a \star_B d\ a$

using *a*
by (*metis B.ide-hcomp B.lunit′-simps*(*4*) *B.objE antipar*(*1*) *equivalence-data-simps*$_B$(*5*) *ide-right*)
ultimately show *?thesis*
using *a B.comp-arr-dom* [*of* $\mathrm{l}_B{}^{-1}[a \star_B d\ a]$ $a \star_B d\ a$] **by** *simp*
qed
also have ... = $\mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B[a,\ a,\ d\ a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[a] \star_B d\ a)$
using *a B.lunit-hcomp*(*4*) [*of a d a*] **by** *auto*
finally have $\varepsilon_1\ a \cdot_B (I_C.unit\ a \star_B d\ a) =$
$\mathrm{a}_B[d\ a,\ e\ a,\ a \star_B d\ a] \cdot_B (\eta\ a \star_B a \star_B d\ a) \cdot_B \mathrm{a}_B[a,\ a,\ d\ a] \cdot_B$
$(\mathrm{l}_B{}^{-1}[a] \star_B d\ a)$
by *simp*
thus *?thesis* **by** *simp*
qed
finally show $(d\ a \star_B EoP.unit\ a) \cdot_B \mathrm{r}_B{}^{-1}[d\ a] \cdot_B \mathrm{l}_B[d\ a] = \varepsilon_1\ a \cdot_B (I_C.unit\ a \star_B d\ a)$
by *blast*
qed
qed

interpretation *equivalence-of-bicategories* V_B H_B a_B i_B src_B trg_B *comp hcomp* a i_B *src trg*
E Φ_E P Φ_P $unit_0$ $unit_1$ $counit_0$ $counit_1$
..

lemma *induces-equivalence*:
shows *equivalence-of-bicategories* V_B H_B a_B i_B src_B trg_B *comp hcomp* a i_B *src trg*
E Φ_E P Φ_P $unit_0$ $unit_1$ $counit_0$ $counit_1$
..

end

1.16.6 Equivalence Pseudofunctors, Bijective on Objects

Here we carry out the extension of an equivalence pseudofunctor F to an equivalence of bicategories in the special case that the object map of F is bijective. The bijectivity assumption simplifies the construction of the unit and counit of the equivalence (the components at objects are in fact identities), as well as the proofs of the associated coherence conditions.

locale *equivalence-pseudofunctor-bij-on-obj* =
equivalence-pseudofunctor +
assumes *bij-on-obj*: *bij-betw* map_0 (*Collect C.obj*) (*Collect D.obj*)
begin

abbreviation F_0
where $F_0 \equiv map_0$

definition G_0
where G_0 = *inv-into* (*Collect C.obj*) F_0

lemma G_0*-props*:
shows $D.obj\ b \Longrightarrow C.obj\ (G_0\ b) \wedge F_0\ (G_0\ b) = b$
and $C.obj\ a \Longrightarrow D.obj\ (F_0\ a) \wedge G_0\ (F_0\ a) = a$
proof −
  have *surj*: F_0 ‘ (*Collect C.obj*) = *Collect D.obj*
    using *bij-on-obj bij-betw-imp-surj-on* **by** *blast*
  assume *b*: *D.obj b*
  show $C.obj\ (G_0\ b) \wedge F_0\ (G_0\ b) = b$
  proof
    show $C.obj\ (G_0\ b)$
      unfolding G_0*-def*
      using *b* **by** (*metis inv-into-into mem-Collect-eq surj*)
    show $F_0\ (G_0\ b) = b$
      unfolding G_0*-def*
      using *b* **by** (*simp add*: *f-inv-into-f surj*)
  qed
  next
  have *bij*: *bij-betw* G_0 (*Collect D.obj*) (*Collect C.obj*)
    using *bij-on-obj* G_0*-def bij-betw-inv-into* **by** *auto*
  have *surj*: G_0 ‘ (*Collect D.obj*) = *Collect C.obj*
    using *bij-on-obj* G_0*-def*
    by (*metis bij-betw-def bij-betw-inv-into*)
  assume *a*: *C.obj a*
  show $D.obj\ (F_0\ a) \wedge G_0\ (F_0\ a) = a$
    using *a bij surj* G_0*-def bij-betw-imp-inj-on bij-on-obj* **by** *force*
qed

We extend G_0 to all arrows of D using chosen adjoint equivalences, which extend F, between $hom_C\ (a,\ a')$ and $hom_D\ (F\ a,\ F\ a')$. The use of adjoint equivalences restricts choices that prevent us from validating the necessary coherence conditions.

definition *G*
where $G\ \nu$ = (*if D.arr* ν *then*
    equivalence-pseudofunctor-at-hom.G_1 V_C src_C trg_C V_D src_D trg_D *F*
      $(G_0\ (src_D\ \nu))\ (G_0\ (trg_D\ \nu))\ \nu$
  else C.null)

lemma *G-in-hom* [*intro*]:
assumes *D.arr* ν
shows «$G\ \nu : G_0\ (src_D\ \nu) \rightarrow_C G_0\ (trg_D\ \nu)$»
and «$G\ \nu : G\ (D.dom\ \nu) \Rightarrow_C G\ (D.cod\ \nu)$»
proof −
  have *1*: $src_D\ \nu = F_0\ (G_0\ (src_D\ \nu)) \wedge trg_D\ \nu = F_0\ (G_0\ (trg_D\ \nu))$
    using *assms* G_0*-props* **by** *simp*
  interpret hom_C: *subcategory* V_C ‹$\lambda\mu$. «$\mu : G_0\ (src_D\ \nu) \rightarrow_C G_0\ (trg_D\ \nu)$»›
    using *assms C.hhom-is-subcategory* **by** *simp*
  interpret hom_D: *subcategory* V_D ‹$\lambda\nu'$. «$\nu' : F_0\ (G_0\ (src_D\ \nu)) \rightarrow_D F_0\ (G_0\ (trg_D\ \nu))$»›
    using *assms 1 D.in-hhom-def D.hhom-is-subcategory* **by** *simp*
  interpret *Faa'*: *equivalence-pseudofunctor-at-hom*

V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ ‹G_0 (src_D ν)› ‹G_0 (trg_D ν)›
using *assms* G_0*-props*(*1*) **by** *unfold-locales auto*
have *2*: hom_D*.arr* ν
using *assms 1* hom_D*.arr-char*$_{SbC}$ **by** *simp*
show «G ν : G (*D.dom* ν) $\Rightarrow_C$ G (*D.cod* ν)»
unfolding *G-def*
using *assms 2* hom_C*.arr-char*$_{SbC}$ hom_C*.dom-char*$_{SbC}$ hom_C*.cod-char*$_{SbC}$ hom_D*.dom-char*$_{SbC}$ hom_D*.cod-char*$_{SbC}$
Faa'.ηε.F.preserves-arr Faa'.ηε.F.preserves-dom Faa'.ηε.F.preserves-cod
by (*intro C.in-homI*) *auto*
thus *C.in-hhom* (G ν) (G_0 (src_D ν)) (G_0 (trg_D ν))
proof −
have hom_C*.arr* (*Faa'*.G_1 ν)
using *2* **by** *simp*
thus *?thesis*
using *G-def assms* hom_C*.arrE* **by** *presburger*
qed
qed

lemma *G-simps* [*simp*]:
assumes *D.arr* ν
shows *C.arr* (G ν)
and src_C (G ν) = G_0 (src_D ν) **and** trg_C (G ν) = G_0 (trg_D ν)
and *C.dom* (G ν) = G (*D.dom* ν) **and** *C.cod* (G ν) = G (*D.cod* ν)
using *assms G-in-hom* **by** *auto*

interpretation *G*: *functor* V_D V_C G
proof
show $\bigwedge f.$ ¬ *D.arr* f $\Longrightarrow$ G f = *C.null*
unfolding *G-def* **by** *simp*
fix ν
assume ν: *D.arr* ν
show *C.arr* (G ν)
using ν **by** *simp*
show *C.dom* (G ν) = G (*D.dom* ν)
using ν **by** *simp*
show *C.cod* (G ν) = G (*D.cod* ν)
using ν **by** *simp*
next
fix μ ν
assume $\mu\nu$: *D.seq* μ ν
have *1*: *D.arr* μ ∧ *D.arr* ν ∧ src_D μ = src_D ν ∧ trg_D μ = trg_D ν ∧
src_D μ = F_0 (G_0 (src_D μ)) ∧ trg_D μ = F_0 (G_0 (trg_D μ)) ∧
src_D ν = F_0 (G_0 (src_D ν)) ∧ trg_D ν = F_0 (G_0 (trg_D ν))
using $\mu\nu$ G_0*-props D.vseq-implies-hpar* **by** *auto*
interpret hom_C: *subcategory* V_C ‹$\lambda\mu$. «μ : G_0 (src_D ν) $\rightarrow_C$ G_0 (trg_D ν)»›
using *1 C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu'$. «ν' : F_0 (G_0 (src_D ν)) $\rightarrow_D$ F_0 (G_0 (trg_D ν))»›

using *1 D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret $hom_D{}'$: *subcategory* V_D
$\langle\lambda\nu'.\ D.arr\ \nu' \wedge src_D\ \nu' = src_D\ \nu \wedge trg_D\ \nu' = trg_D\ \nu\rangle$
using *1 D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret *Faa'*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi\ \langle G_0\ (src_D\ \nu)\rangle\ \langle G_0\ (trg_D\ \nu)\rangle$
using $\mu\nu$ *1* G_0*-props*(*1*) **by** *unfold-locales auto*
have *2*: $hom_D.seq\ \mu\ \nu$
using $\mu\nu$ *1* $hom_D.arr\text{-}char_{SbC}$ $hom_D.comp\text{-}def$ **by** *fastforce*
have $G\ (hom_D.comp\ \mu\ \nu) = hom_C.comp\ (G\ \mu)\ (G\ \nu)$
unfolding *G-def*
using *1 2* G_0*-props Faa'*.G_1*-props* $hom_D.arr\text{-}char_{SbC}$
Faa'.$\eta\varepsilon$.*F.as-nat-trans.preserves-comp-2*
by *auto*
thus $G\ (\mu \cdot_D \nu) = G\ \mu \cdot_C G\ \nu$
using $\mu\nu$ *1 2 G-def* $hom_C.comp\text{-}simp$ $hom_D.comp\text{-}simp$ *D.src-vcomp D.trg-vcomp*
Faa'.$\eta\varepsilon$.*F.preserves-arr*
by *metis*
qed

lemma *functor-G*:
shows *functor* $V_D\ V_C\ G$
..

interpretation *G*: *faithful-functor* $V_D\ V_C\ G$
proof
fix $f\ f'$
assume *par*: $D.par\ f\ f'$
assume *eq*: $G\ f = G\ f'$
have *1*: $src_D\ f = src_D\ f' \wedge trg_D\ f = trg_D\ f'$
using *par* **by** (*metis D.src-dom D.trg-dom*)
interpret hom_C: *subcategory* V_C $\langle\lambda\mu.\ «\mu : G_0\ (src_D\ f) \rightarrow_C G_0\ (trg_D\ f)»\rangle$
using *par C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D $\langle\lambda\nu'.\ «\nu' : F_0\ (G_0\ (src_D\ f)) \rightarrow_D F_0\ (G_0\ (trg_D\ f))»\rangle$
using *par D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret *F*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi\ \langle G_0\ (src_D\ f)\rangle\ \langle G_0\ (trg_D\ f)\rangle$
using *par* G_0*-props*(*1*) **by** *unfold-locales auto*
interpret *F'*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi\ \langle G_0\ (src_D\ f')\rangle\ \langle G_0\ (trg_D\ f')\rangle$
using *par* G_0*-props*(*1*) **by** *unfold-locales auto*
have *2*: $hom_D.par\ f\ f'$
using *par 1* $hom_D.arr\text{-}char_{SbC}$ $hom_D.dom\text{-}char_{SbC}$ $hom_D.cod\text{-}char_{SbC}$ G_0*-props* **by** *simp*

have $F.G_1\ f = F.G_1\ f'$
using *par eq G-def G-simps*(*2*−*3*) **by** *metis*
thus $f = f'$
using *F.ηε.F-is-faithful*
by (*simp add*: *2 faithful-functor-axioms-def faithful-functor-def*)
qed

interpretation *FG*: *composite-functor* $V_D\ V_C\ V_D\ G\ F$ **..**
interpretation *FG*: *faithful-functor* $V_D\ V_D\ F\ o\ G$
using *faithful-functor-axioms G.faithful-functor-axioms faithful-functors-compose*
by *blast*

interpretation *GF*: *composite-functor* $V_C\ V_D\ V_C\ F\ G$ **..**
interpretation *GF*: *faithful-functor* $V_C\ V_C\ G\ o\ F$
using *faithful-functor-axioms G.faithful-functor-axioms faithful-functors-compose*
by *blast*

interpretation *G*: *weak-arrow-of-homs* $V_D\ src_D\ trg_D\ V_C\ src_C\ trg_C\ G$
proof
have $*$: $\bigwedge b.\ D.obj\ b \Longrightarrow C.isomorphic\ (G\ b)\ (src_C\ (G\ b))$
proof −
fix *b*
assume *b*: *D.obj b*
interpret hom_C: *subcategory* V_C ‹$\lambda\mu.$ «$\mu : G_0\ b \rightarrow_C G_0\ b$»›
using *b C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu'.$ «$\nu' : F_0\ (G_0\ b) \rightarrow_D F_0\ (G_0\ b)$»›
using *b D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret *Faa′*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi$ ‹$G_0\ b$› ‹$G_0\ b$›
using *b* G_0*-props*(*1*) **by** *unfold-locales auto*
have *1*: *C.in-hhom* $(G_0\ b)\ (G_0\ b)\ (G_0\ b)$
using *b* G_0*-props* **by** *force*

Using the unit constraints of F and the fact that $F_0\ (G_0\ b) = b$, we obtain an isomorphism «$?\psi : b \Rightarrow_D F\ (G_0\ b)$», which is also an isomorphism in hom_D.

let $?\psi = unit\ (G_0\ b)$
have ψ: «$?\psi : b \Rightarrow_D F\ (G_0\ b)$» $\wedge$ *D.iso* $?\psi$
using *b* G_0*-props unit-char* **by** *auto*
have ψ*-in-hhom*: *D.in-hhom* $?\psi\ b\ b$
using *b* ψ *1 D.src-dom D.trg-dom* **by** *fastforce*
have *2*: $hom_D.arr\ ?\psi \wedge hom_D.arr\ (D.inv\ ?\psi)$
by (*metis D.in-hhomE D.inv-in-hhom* G_0*-props*(*1*) ψ ψ*-in-hhom* $hom_D.arr\text{-}char_{SbC}$)
have ψ': $hom_D.in\text{-}hom\ ?\psi\ b\ (Faa'.F_1\ (G_0\ b))$
proof
show $hom_D.arr\ ?\psi$
using *2* **by** *simp*
show $hom_D.dom\ ?\psi = b$

using *2 ψ* hom_D*.dom-char*$_{SbC}$ **by** *auto*
show hom_D*.cod ?ψ = Faa′.*F_1 (G_0 *b*)
proof −
have «*?ψ* : *b* ⇒$_D$ *Faa′.*F_1 (G_0 *b*)» ∧ *D.iso ?ψ*
unfolding *Faa′.*F_1*-def* **using** *b ψ 1* G_0*-props* **by** *auto*
thus *?thesis*
using *2 ψ* hom_D*.cod-char*$_{SbC}$ **by** *auto*
qed
qed

Transposing *?ψ* via the adjunction *ηε* yields an isomorphism from *G b* to G_0 *b* in hom_C, hence in *C*.

have ∗∗: hom_C*.isomorphic* (*G b*) (G_0 *b*)
proof −
have hom_C*.in-hom* (*Faa′.ηε.ψ* (G_0 *b*) *?ψ*) (*Faa′.*G_1 *b*) (G_0 *b*)
by (*metis 1 C.ide-src C.in-hhom-def Faa′.ηε.ψ-in-hom ψ′* hom_C*.ideI*$_{SbC}$)
moreover have hom_C*.iso* (*Faa′.ηε.ψ* (G_0 *b*) *?ψ*)
proof (*unfold Faa′.ηε.ψ-def*)
have hom_C*.iso-in-hom* (hom_C*.comp* (*Faa′.ε* (G_0 *b*))
(*Faa′.*G_1 *?ψ*)) (*Faa′.*G_1 *b*) (G_0 *b*)
proof (*intro* hom_C*.comp-iso-in-hom*)
show hom_C*.iso-in-hom* (*Faa′.*G_1 (*unit* (G_0 *b*)))
(*Faa′.*G_1 *b*) (*Faa′.ηε.FG.map* (G_0 *b*))
using *ψ ψ′ 1 2* hom_D*.iso-char*$_{SbC}$ *Faa′.ηε.F.preserves-iso* hom_D*.iso-char*$_{SbC}$
Faa′.ηε.F.preserves-iso hom_C*.in-hom-char*$_{SbC}$ hom_C*.arr-char*$_{SbC}$
by *auto*
show hom_C*.iso-in-hom* (*Faa′.ε* (G_0 *b*)) (*Faa′.ηε.FG.map* (G_0 *b*)) (G_0 *b*)
using *1 C.ide-src C.ide-trg C.in-hhom-def Faa′.ηε.ε.components-are-iso*
Faa′.ηε.ε.preserves-hom hom_C*.arrI* hom_C*.ideI*$_{SbC}$ hom_C*.ide-in-hom*
hom_C*.map-simp* hom_C*.iso-in-hom-def*
by *metis*
qed
thus hom_C*.iso* (hom_C*.comp* (*Faa′.ε* (G_0 *b*)) (*Faa′.*G_1 (*unit* (G_0 *b*)))) **by** *auto*
qed
ultimately show *?thesis*
unfolding *G-def*
using *b* hom_C*.isomorphic-def D.obj-def D.obj-simps*(*3*) **by** *auto*
qed
hence hom_C*.isomorphic* (*G b*) (src_C (*G b*))
using *b G-in-hom*(*1*) **by** *auto*
moreover have ⋀*f g*. hom_C*.isomorphic f g* ⟹ *C.isomorphic f g*
using hom_C*.iso-char*$_{SbC}$ hom_C*.isomorphic-def C.isomorphic-def* hom_C*.in-hom-char*$_{SbC}$
hom_C*.arr-char*$_{SbC}$
by *auto*
ultimately show *C.isomorphic* (*G b*) (src_C (*G b*))
by *simp*
qed
fix *ν*
assume *ν*: *D.arr ν*

show $C.isomorphic\ (G\ (src_D\ \nu))\ (src_C\ (G\ \nu))$
using ν $*$ **by** *force*
show $C.isomorphic\ (G\ (trg_D\ \nu))\ (trg_C\ (G\ \nu))$
using ν $*$ **by** *force*
qed

lemma *weak-arrow-of-homs-G*:
shows $weak\text{-}arrow\text{-}of\text{-}homs\ V_D\ src_D\ trg_D\ V_C\ src_C\ trg_C\ G$
..

sublocale $H_D oGG$: *composite-functor* $D.VV.comp\ C.VV.comp\ V_C$
$G.FF$ ‹$\lambda\mu\nu.\ fst\ \mu\nu \star_C snd\ \mu\nu$›
..
sublocale GoH_D: *composite-functor* $D.VV.comp\ V_D\ V_C$ ‹$\lambda\mu\nu.\ fst\ \mu\nu \star_D snd\ \mu\nu$› G
..

To get the unit η of the equivalence of bicategories, we piece together the units from the local equivalences. The components at objects will in fact be identities.

definition η
where $\eta\ \nu = (if\ D.arr\ \nu\ then$
$equivalence\text{-}pseudofunctor\text{-}at\text{-}hom.\eta\ V_C\ src_C\ trg_C\ V_D\ src_D\ trg_D\ F$
$(G_0\ (src_D\ \nu))\ (G_0\ (trg_D\ \nu))\ \nu$
$else\ D.null)$

lemma η*-in-hom*:
assumes $D.arr\ \nu$
shows [*intro*]: «$\eta\ \nu : src_D\ \nu \rightarrow_D trg_D\ \nu$»
and [*intro*]: «$\eta\ \nu : D.dom\ \nu \Rightarrow_D F\ (G\ (D.cod\ \nu))$»
and $D.ide\ \nu \Longrightarrow D.iso\ (\eta\ \nu)$
proof −
interpret hom_C: *subcategory* V_C ‹$\lambda\mu.$ «$\mu : G_0\ (src_D\ \nu) \rightarrow_C G_0\ (trg_D\ \nu)$»›
using *assms* $C.hhom\text{-}is\text{-}subcategory$ **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu'.$ «$\nu' : F_0\ (G_0\ (src_D\ \nu)) \rightarrow_D F_0\ (G_0\ (trg_D\ \nu))$»›
using *assms* $D.hhom\text{-}is\text{-}subcategory$
by (*simp add*: $D.in\text{-}hhom\text{-}def\ subcategory\text{-}axioms\text{-}def\ subcategory\text{-}def$)
interpret Faa': $equivalence\text{-}pseudofunctor\text{-}at\text{-}hom\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D$
$src_D\ trg_D$
$F\ \Phi$ ‹$G_0\ (src_D\ \nu)$› ‹$G_0\ (trg_D\ \nu)$›
using *assms* $G_0\text{-}props(1)$ **by** *unfold-locales auto*
have *1*: «$\eta\ \nu : D.dom\ \nu \Rightarrow_D F\ (G\ (D.cod\ \nu))$» $\wedge$ $(D.ide\ \nu \longrightarrow D.iso\ (\eta\ \nu))$
proof −
have $src_D\ \nu = F_0\ (G_0\ (src_D\ \nu)) \wedge trg_D\ \nu = F_0\ (G_0\ (trg_D\ \nu))$
using *assms* $G_0\text{-}props$ **by** *simp*
hence $hom_D.arr\ \nu$
using *assms* $hom_D.arr\text{-}char_{SbC}\ hom_D.ide\text{-}char$ **by** *simp*
hence $hom_D.in\text{-}hom\ (Faa'.\eta\ \nu)\ (D.dom\ \nu)\ (F\ (G\ (D.cod\ \nu))) \wedge$
$(D.ide\ \nu \longrightarrow hom_D.iso\ (Faa'.\eta\ \nu))$
using *assms* $Faa'.\eta\varepsilon.\eta.preserves\text{-}hom$ [*of* $\nu\ D.dom\ \nu\ D.cod\ \nu$]
$hom_D.map\text{-}def\ G\text{-}def\ Faa'.F_1\text{-}def$

apply *simp*
by (*simp add*: *D.arr-iff-in-hom* hom_D*.arr-char*$_{SbC}$ hom_D*.cod-closed* hom_D*.dom-closed* hom_D*.ide-char*$_{SbC}$ hom_D*.in-hom-char*$_{SbC}$)
thus *?thesis*
unfolding *η-def*
using hom_D*.in-hom-char*$_{SbC}$ hom_D*.iso-char*$_{SbC}$ **by** *auto*
qed
show «$\eta\ \nu$: *D.dom* $\nu \Rightarrow_D F\ (G\ (D.cod\ \nu))$»
using *1* **by** *simp*
thus *D.in-hhom* $(\eta\ \nu)\ (src_D\ \nu)\ (trg_D\ \nu)$
using *assms D.src-dom D.trg-dom*
by (*metis D.arrI D.in-hhom-def D.vconn-implies-hpar*(*1*−*2*))
show *D.ide* $\nu \Longrightarrow$ *D.iso* $(\eta\ \nu)$
using *1* **by** *simp*
qed

lemma *η-simps* [*simp*]:
assumes *D.arr* ν
shows *D.arr* $(\eta\ \nu)$
and $src_D\ (\eta\ \nu) = src_D\ \nu$ **and** $trg_D\ (\eta\ \nu) = trg_D\ \nu$
and *D.dom* $(\eta\ \nu)$ = *D.dom* ν **and** *D.cod* $(\eta\ \nu) = F\ (G\ (D.cod\ \nu))$
and *D.ide* $\nu \Longrightarrow$ *D.iso* $(\eta\ \nu)$
using *assms η-in-hom* **by** *auto*

lemma *η-naturality*:
assumes *D.arr* ν
shows $\eta\ (D.cod\ \nu) \cdot_D \nu = \eta\ \nu$ **and** $F\ (G\ \nu) \cdot_D \eta\ (D.dom\ \nu) = \eta\ \nu$
and $\nu \cdot_D D.inv\ (\eta\ (D.dom\ \nu)) = D.inv\ (\eta\ (D.cod\ \nu)) \cdot_D F\ (G\ \nu)$
proof −
interpret hom_C: *subcategory* V_C ‹$\lambda\mu'$. «μ' : $G_0\ (src_D\ \nu) \rightarrow_C G_0\ (trg_D\ \nu)$»›
using *assms C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu'$. «ν' : $F_0\ (G_0\ (src_D\ \nu)) \rightarrow_D F_0\ (G_0\ (trg_D\ \nu))$»›
using *D.hhom-is-subcategory* **by** *simp*
interpret *Faa′*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi$ ‹$G_0\ (src_D\ \nu)$› ‹$G_0\ (trg_D\ \nu)$›
using *assms* G_0*-props* **by** *unfold-locales auto*
show *1*: $\eta\ (D.cod\ \nu) \cdot_D \nu = \eta\ \nu$
unfolding *η-def*
using *assms Faa′.ηε.η.naturality2* hom_D*.comp-char G-def* hom_D*.cod-simp G-in-hom*(*1*) hom_C*.arr-char*$_{SbC}$ hom_D*.arr-char*$_{SbC}$ hom_D*.cod-closed*
apply *simp*
by (*metis* (*no-types*, *lifting*) *D.ext Faa′.ηε.F.preserves-reflects-arr Faa′.ηε.η.preserves-reflects-arr*)
show *2*: $F\ (G\ \nu) \cdot_D \eta\ (D.dom\ \nu) = \eta\ \nu$
unfolding *η-def*
using *assms D.src-dom D.src-cod D.trg-dom D.trg-cod Faa′.ηε.η.naturality1* hom_D*.comp-char G-def Faa′.*F_1*-def* hom_D*.dom-simp* hom_D*.cod-simp*
apply *simp*

by (*metis* (*no-types*, *lifting*) *D.not-arr-null Faa'.ηε.η.extensionality*
η-def η-simps(*1*) hom_D*.null-char*)
show $\nu \cdot_D$ *D.inv* (η (*D.dom* ν)) = *D.inv* (η (*D.cod* ν)) $\cdot_D$ *F* (*G* ν)
using *assms 1 2*
by (*simp add*: *D.invert-opposite-sides-of-square*)
qed

The fact that G_0 is chosen to be right-inverse to F implies that η is an ordinary natural isomorphism (with respect to vertical composition) from I_D to FG.

interpretation η: *natural-isomorphism* V_D V_D *D.map FG.map* η
using *η-simps η-naturality η-def*
by *unfold-locales auto*

In view of the bijectivity assumption, we can obtain the counit ε the same way.

definition ε
where ε μ = (*if C.arr* μ *then*
equivalence-pseudofunctor-at-hom.ε V_C src_C trg_C V_D src_D trg_D *F*
(src_C μ) (trg_C μ) μ
else C.null)

lemma *ε-in-hom*:
assumes *C.arr* μ
shows [*intro*]: «ε μ : src_C μ $\rightarrow_C$ trg_C μ»
and [*intro*]: «ε μ : *G* (*F* (*C.dom* μ)) $\Rightarrow_C$ *C.cod* μ»
and *C.ide* μ $\Longrightarrow$ *C.iso* (ε μ)
proof −
interpret hom_C: *subcategory* V_C ‹$\lambda\mu'$. «μ' : src_C μ $\rightarrow_C$ trg_C μ»›
using *C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu'$. «ν' : F_0 (src_C μ) $\rightarrow_D$ F_0 (trg_C μ)»›
using *assms D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret *Faa'*: *equivalence-pseudofunctor-at-hom*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ ‹G_0 (F_0 (src_C μ))› ‹G_0 (F_0 (trg_C μ))›
using *assms* G_0*-props*
by *unfold-locales simp-all*
have *1*: «ε μ : *G* (*F* (*C.dom* μ)) $\Rightarrow_C$ *C.cod* μ» $\wedge$ (*C.ide* μ $\longrightarrow$ *C.iso* (ε μ))
proof −
have hom_C*.arr* μ
using *assms* hom_C*.arr-char*$_{SbC}$ **by** *simp*
hence hom_C*.in-hom* μ (*C.dom* μ) (*C.cod* μ)
using *assms* hom_C*.in-hom-char*$_{SbC}$ hom_C*.dom-char*$_{SbC}$ hom_C*.cod-char*$_{SbC}$ **by** *auto*
hence hom_C*.in-hom* (*Faa'*.ε μ) (*G* (*F* (*C.dom* μ))) (*C.cod* μ) $\wedge$
(*C.ide* μ $\longrightarrow$ hom_C*.iso* (*Faa'*.ε μ))
using *assms Faa'.ηε.ε.preserves-hom* [*of* μ *C.dom* μ *C.cod* μ]
Faa'.ηε.ε.components-are-iso hom_C*.map-def G-def Faa'.*F_1*-def* G_0*-props*
by (*simp add*: hom_C*.ideI*$_{SbC}$)
thus *?thesis*
unfolding *ε-def*

using *assms* hom_C*.in-hom-char*$_{SbC}$ hom_C*.iso-char*$_{SbC}$ G_0*-props*(*2*) **by** *simp*
qed
show «$\varepsilon\ \mu : G\ (F\ (C.dom\ \mu)) \Rightarrow_C C.cod\ \mu$»
using *1* **by** *simp*
thus *C.in-hhom* ($\varepsilon\ \mu$) ($src_C\ \mu$) ($trg_C\ \mu$)
using *assms C.src-cod C.trg-cod*
by (*metis C.arrI C.in-hhom-def C.vconn-implies-hpar*(*1*−*4*))
show $C.ide\ \mu \Longrightarrow C.iso\ (\varepsilon\ \mu)$
using *1* **by** *simp*
qed

lemma *ε-simps* [*simp*]:
assumes $C.arr\ \mu$
shows $C.arr\ (\varepsilon\ \mu)$
and $src_C\ (\varepsilon\ \mu) = src_C\ \mu$ **and** $trg_C\ (\varepsilon\ \mu) = trg_C\ \mu$
and $C.dom\ (\varepsilon\ \mu) = G\ (F\ (C.dom\ \mu))$ **and** $C.cod\ (\varepsilon\ \mu) = C.cod\ \mu$
and $C.ide\ \mu \Longrightarrow C.iso\ (\varepsilon\ \mu)$
using *assms ε-in-hom* **by** *auto*

lemma *ε-naturality*:
assumes $C.arr\ \mu$
shows $\varepsilon\ (C.cod\ \mu) \cdot_C G\ (F\ \mu) = \varepsilon\ \mu$ **and** $\mu \cdot_C \varepsilon\ (C.dom\ \mu) = \varepsilon\ \mu$
and $G\ (F\ \mu) \cdot_C C.inv\ (\varepsilon\ (C.dom\ \mu)) = C.inv\ (\varepsilon\ (C.cod\ \mu)) \cdot_C \mu$
proof −
interpret hom_C: *subcategory* V_C ‹$\lambda\mu'$. «$\mu' : src_C\ \mu \rightarrow_C trg_C\ \mu$»›
using *assms C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu$. «$\nu : F_0\ (src_C\ \mu) \rightarrow_D F_0\ (trg_C\ \mu)$»›
using *assms D.hhom-is-subcategory*
by (*simp add*: *D.in-hhom-def subcategory-axioms-def subcategory-def*)
interpret *Faa′*: *equivalence-pseudofunctor-at-hom*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi$ ‹$src_C\ \mu$› ‹$trg_C\ \mu$›
using *assms* **by** *unfold-locales simp-all*
show *1*: $\varepsilon\ (C.cod\ \mu) \cdot_C G\ (F\ \mu) = \varepsilon\ \mu$
unfolding *ε-def*
using *assms C.src-dom C.src-cod C.trg-dom C.trg-cod Faa′.ηε.ε.naturality2*
hom_C*.comp-char G-def Faa′.*F_1*-def* G_0*-props* hom_C*.dom-char*$_{SbC}$ hom_C*.cod-char*$_{SbC}$
apply *simp*
by (*metis* (*no-types*, *lifting*) *C.in-hhomI Faa′.ηε.ε.preserves-reflects-arr* hom_C*.arr-char*$_{SbC}$
hom_C*.not-arr-null* hom_C*.null-char*)
show *2*: $\mu \cdot_C \varepsilon\ (C.dom\ \mu) = \varepsilon\ \mu$
unfolding *ε-def*
using *assms C.src-dom C.src-cod C.trg-dom C.trg-cod Faa′.ηε.ε.naturality1*
hom_C*.comp-char G-def Faa′.*F_1*-def* G_0*-props* hom_C*.dom-char*$_{SbC}$ hom_C*.cod-char*$_{SbC}$
apply *simp*
by (*metis* (*no-types*, *lifting*) *C.in-hhomI Faa′.ηε.ε.preserves-reflects-arr*
hom_C*.arr-char*$_{SbC}$ hom_C*.not-arr-null* hom_C*.null-char*)
show $G\ (F\ \mu) \cdot_C C.inv\ (\varepsilon\ (C.dom\ \mu)) = C.inv\ (\varepsilon\ (C.cod\ \mu)) \cdot_C \mu$
using *assms 1 2 C.invert-opposite-sides-of-square* **by** *simp*

qed

interpretation ε: *natural-isomorphism* V_C V_C *GF.map C.map* ε
  using *ε-simps ε-naturality ε-def*
  by *unfold-locales auto*

interpretation *GFG*: *composite-functor* V_D V_C V_C G ‹*GF.map*› **..**
interpretation *FGF*: *composite-functor* V_C V_D V_D F ‹*FG.map*› **..**

interpretation *Goη*: *natural-transformation* V_D V_C G *GFG.map* ‹$G \circ \eta$›
proof −
  have $G \circ D.map = G$
    using *G.as-nat-trans.natural-transformation-axioms* **by** *auto*
  moreover have $G \circ FG.map = GFG.map$
    by *auto*
  ultimately show *natural-transformation* V_D V_C G *GFG.map* $(G \circ \eta)$
    using *η.natural-transformation-axioms G.as-nat-trans.natural-transformation-axioms*
      horizontal-composite [*of* V_D V_D *D.map FG.map* η V_C G G G] **by** *simp*
qed

interpretation *ηoF*: *natural-transformation* V_C V_D F *FGF.map* ‹$\eta \circ F$›
  using *η.natural-transformation-axioms as-nat-trans.natural-transformation-axioms*
    horizontal-composite [*of* V_C V_D F F F V_D *D.map FG.map* η]
  by *simp*

interpretation *εoG*: *natural-transformation* V_D V_C *GFG.map* G ‹$\varepsilon \circ G$›
  using *ε.natural-transformation-axioms G.as-nat-trans.natural-transformation-axioms*
    horizontal-composite [*of* V_D V_C G G G V_C *GF.map C.map* ε]
  by *simp*

interpretation *Foε*: *natural-transformation* V_C V_D *FGF.map* F ‹$F \circ \varepsilon$›
proof −
  have $F \circ C.map = F$
    using *as-nat-trans.natural-transformation-axioms* **by** *auto*
  moreover have $F \circ GF.map = FGF.map$
    by *auto*
  ultimately show *natural-transformation* V_C V_D *FGF.map* F $(F \circ \varepsilon)$
    using *ε.natural-transformation-axioms as-nat-trans.natural-transformation-axioms*
      horizontal-composite [*of* V_C V_C *GF.map C.map* ε V_D F F F]
    by *simp*
qed

interpretation *εoG-Goη*: *vertical-composite* V_D V_C G *GFG.map* G ‹$G \circ \eta$› ‹$\varepsilon \circ G$› **..**
interpretation *Foε-ηoF*: *vertical-composite* V_C V_D F *FGF.map* F ‹$\eta \circ F$› ‹$F \circ \varepsilon$› **..**

Bijectivity results in an ordinary adjunction between the vertical categories.

lemma *adjunction-$\eta\varepsilon$*:
shows *unit-counit-adjunction* V_C V_D G F η ε
proof

```
show εoG-Goη.map = G
proof
  fix ν
  have ¬ D.arr ν ⟹ εoG-Goη.map ν = G ν
    unfolding εoG-Goη.map-def
    by (simp add: G.extensionality)
  moreover have D.arr ν ⟹ εoG-Goη.map ν = G ν
  proof −
    assume ν: D.arr ν
    interpret hom_C: subcategory V_C ‹λμ'. «μ' : G_0 (src_D ν) →_C G_0 (trg_D ν)»›
      using ν C.hhom-is-subcategory by simp
    interpret hom_D: subcategory V_D ‹λν'. «ν' : F_0 (G_0 (src_D ν)) →_D F_0 (G_0 (trg_D ν))»›
      using D.hhom-is-subcategory by simp
    interpret Faa': equivalence-pseudofunctor-at-hom
                    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
                    F Φ ‹G_0 (src_D ν)› ‹G_0 (trg_D ν)›
      using ν G_0-props by unfold-locales simp-all
    show εoG-Goη.map ν = G ν
    proof −
      have 1: C.arr (Faa'.G_1 (D.cod ν))
        using ν hom_D.arr-char_SbC G-def
        by (metis Faa'.ηε.F.preserves-reflects-arr G-in-hom(1) hom_C.arr-char_SbC
            hom_C.inclusion hom_D.cod-closed)
      have 2: D.arr (Faa'.η ν)
        using ν hom_D.arr-char_SbC
        by (metis D.in-hhomI D.obj-src D.obj-trg Faa'.ηε.η.preserves-reflects-arr
            G_0-props(1) hom_D.inclusion)
      have 3: src_C (Faa'.G_1 (D.cod ν)) = G_0 (src_D ν)
        using ν
        by (metis D.arr-cod D.src-cod D.trg-cod G-def G-simps(2))
      have 4: trg_C (Faa'.G_1 (D.cod ν)) = G_0 (trg_D ν)
        using ν
        by (metis D.arr-cod D.src-cod D.trg-cod G-def G-simps(3))
      have 5: hom_D.arr ν
        using ν hom_D.arr-char_SbC G_0-props by simp
      have εoG-Goη.map ν = ε (G (D.cod ν)) ·_C G (η ν)
        using ν εoG-Goη.map-def by simp
      also have ... = Faa'.ε (Faa'.G_1 (D.cod ν)) ·_C Faa'.G_1 (Faa'.η ν)
        using ν 1 2 3 4 η-def ε-def G-def G-simps(2−3) η-simps(2−3) by auto
      also have ... = hom_C.comp ((Faa'.ε ∘ Faa'.G_1) (hom_D.cod ν)) ((Faa'.G_1 ∘ Faa'.η) ν)
      proof −
        have Faa'.ε (Faa'.G_1 (D.cod ν)) ·_C Faa'.G_1 (Faa'.η ν) = Faa'.ηε.εFoFη.map ν
        proof −
          have C.seq (Faa'.ε (Faa'.G_1 (hom_D.cod ν))) (Faa'.G_1 (Faa'.η ν))
          proof −
            have hom_C.seq (Faa'.ε (Faa'.G_1 (hom_D.cod ν))) (Faa'.G_1 (Faa'.η ν))
              using ν 5 hom_C.arr-char_SbC hom_C.seq-char_SbC Faa'.ηε.F.preserves-arr
              apply (intro hom_C.seqI)
                apply auto
```

using *Faa′.ηε.ε.preserves-reflects-arr* hom_C*.inclusion* hom_D*.arr-cod*
by *presburger*
thus *?thesis*
using hom_C*.seq-char*$_{SbC}$ **by** *simp*
qed
moreover have *D.in-hhom* (*Faa′.η* ν) (F_0 (G_0 (src_D ν))) (F_0 (G_0 (trg_D ν)))
using ν *5 Faa′.ηε.η.preserves-reflects-arr* hom_D*.arrE* **by** *blast*
moreover have *D.in-hhom* (*D.cod* ν) (F_0 (G_0 (src_D ν))) (F_0 (G_0 (trg_D ν)))
using ν *5* hom_D*.arrE* hom_D*.cod-closed* **by** *blast*
ultimately show *?thesis*
using ν *5 Faa′.ηε.εFoFη.map-def* hom_D*.arr-char*$_{SbC}$ hom_C*.comp-char* hom_D*.cod-char*$_{SbC}$
by *simp*
qed
thus *?thesis*
using ν *5 Faa′.ηε.εFoFη.map-def* **by** *simp*
qed
also have ... = G ν
using ν *5 G-def Faa′.ηε.triangle-F Faa′.ηε.εFoFη.map-simp-1* [*of* ν] **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed
ultimately show *εoG-Goη.map* ν = G ν **by** *auto*
qed
show *Foε-ηoF.map* = F
proof
fix μ
have ¬ *C.arr* μ ⟹ *Foε-ηoF.map* μ = F μ
unfolding *Foε-ηoF.map-def*
by (*simp add*: *extensionality*)
moreover have *C.arr* μ ⟹ *Foε-ηoF.map* μ = F μ
proof −
assume μ: *C.arr* μ
interpret hom_C: *subcategory* V_C ‹$\lambda\mu'$. «μ' : src_C μ $\rightarrow_C$ trg_C μ»›
using μ *C.hhom-is-subcategory* **by** *simp*
interpret hom_D: *subcategory* V_D ‹$\lambda\nu$. «ν : F_0 (src_C μ) $\rightarrow_D$ F_0 (trg_C μ)»›
using μ *D.in-hhom-def D.hhom-is-subcategory* **by** *simp*
interpret *Faa′*: *equivalence-pseudofunctor-at-hom*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
F Φ ‹src_C μ› ‹trg_C μ›
using μ **by** *unfold-locales auto*
show *Foε-ηoF.map* μ = F μ
proof −
have *1*: hom_C*.arr* μ
using μ hom_C*.arr-char*$_{SbC}$ **by** *simp*
have *Foε-ηoF.map* μ = F (ε (*C.cod* μ)) $\cdot_D$ η (F μ)
using μ *Foε-ηoF.map-def* **by** *simp*
also have ... = *Faa′*.F_1 (*Faa′*.ε (*C.cod* μ)) $\cdot_D$ *Faa′*.η (*Faa′*.F_1 μ)
unfolding *η-def ε-def G-def*

```
        using μ G₀-props Faa'.F₁-def
        by auto
      also have ... = hom_D.comp ((Faa'.F₁ ∘ Faa'.ε) (hom_C.cod μ)) ((Faa'.η ∘ Faa'.F₁) μ)
       proof −
        have 2: hom_C.arr (C.cod μ)
      using 1 hom_C.cod-char_SbC [of μ] hom_C.arr-cod hom_C.dom-char_SbC hom_C.cod-char_SbC
          by simp
        moreover have D.seq (Faa'.F₁ (Faa'.ε (C.cod μ))) (Faa'.η (Faa'.F₁ μ))
        proof −
          have hom_D.seq (Faa'.F₁ (Faa'.ε (C.cod μ))) (Faa'.η (Faa'.F₁ μ))
            using μ 1 2 hom_D.arr-char_SbC hom_D.seq-char_SbC Faa'.ηε.G.preserves-arr
                  hom_C.cod-simp hom_C.dom-cod
            apply (intro hom_D.seqI)
            by auto metis
          thus ?thesis
            using hom_D.seq-char_SbC by simp
        qed
        ultimately show ?thesis
          using 1 hom_D.comp-char hom_C.dom-char_SbC hom_C.cod-char_SbC by simp
      qed
      also have ... = Faa'.ηε.GεoηG.map μ
        unfolding Faa'.ηε.GεoηG.map-def
        using 1 by simp
      also have ... = F μ
        using μ Faa'.ηε.triangle-G hom_C.arr-char_SbC Faa'.ηε.εFoFη.map-def
              hom_D.arr-char_SbC hom_C.comp-char hom_D.comp-char Faa'.F₁-def
        by simp
      finally show ?thesis by simp
    qed
  qed
  ultimately show Foε-ηoF.map μ = F μ by auto
 qed
qed

interpretation ηε: unit-counit-adjunction V_C V_D G F η ε
  using adjunction-ηε by simp
```

We now use the adjunction between the vertical categories to define the compositors for G. Without the bijectivity assumption, we would only obtain η and ε as pseudonatural equivalences, rather than natural isomorphisms, which would make everything more complicated.

```
definition Φ_G0
where Φ_G0 f f' = C.inv (ε (G f ⋆_C G f') ·_C G (Φ (G f, G f') ·_D (η f ⋆_D η f')))

lemma Φ_G0-in-hom:
assumes D.ide f and D.ide f' and src_D f = trg_D f'
shows «Φ_G0 f f' : G f ⋆_C G f' ⇒_C G (f ⋆_D f')»
proof (unfold Φ_G0-def, intro C.inv-in-hom)
  show 1: «ε (G f ⋆_C G f') ·_C G (Φ (G f, G f') ·_D (η f ⋆_D η f')) :
```

$G\ (f \star_D f') \Rightarrow_C G\ f \star_C G\ f'$»
proof
show «$\varepsilon\ (G\ f \star_C G\ f') : G\ (F\ (G\ f \star_C G\ f')) \Rightarrow_C G\ f \star_C G\ f'$»
using *assms ε-in-hom* **by** *auto*
show «$G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')) : G\ (f \star_D f') \Rightarrow_C G\ (F\ (G\ f \star_C G\ f'))$»
proof −
have «$\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f') : f \star_D f' \Rightarrow_D F\ (G\ f \star_C G\ f')$»
using *assms η-in-hom*
by (*intro D.comp-in-homI′*) *auto*
thus *?thesis* **by** *auto*
qed
qed
show $C.iso\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
proof (*intro C.isos-compose*)
show $C.iso\ (G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
proof −
have $D.iso\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f'))$
using *assms η-in-hom*
by (*intro D.isos-compose D.seqI*) *auto*
thus *?thesis* **by** *simp*
qed
show $C.iso\ (\varepsilon\ (G\ f \star_C G\ f'))$
using *assms ε-in-hom* **by** *simp*
show $C.seq\ (\varepsilon\ (G\ f \star_C G\ f'))\ (G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
using *1* **by** *auto*
qed
qed

lemma *iso-*Φ_{G0}:
assumes $D.ide\ f$ **and** $D.ide\ f'$ **and** $src_D\ f = trg_D\ f'$
shows $C.iso\ (\Phi_{G0}\ f\ f')$
proof (*unfold* Φ_{G0}*-def*, *intro C.iso-inv-iso C.isos-compose*)
show *1*: $C.iso\ (G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
using *assms η-in-hom D.in-hhom-def cmp-simps′*(*1*) *cmp-simps*(*4*) **by** *auto*
show $C.iso\ (\varepsilon\ (G\ f \star_C G\ f'))$
using *assms ε-in-hom* **by** *simp*
show $C.seq\ (\varepsilon\ (G\ f \star_C G\ f'))\ (G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
using *assms 1* **by** *force*
qed

lemma Φ_{G0}*-naturality*:
assumes $D.arr\ \nu$ **and** $D.arr\ \nu'$ **and** $src_D\ \nu = trg_D\ \nu'$
shows $\Phi_{G0}\ (D.cod\ \nu)\ (D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu') =$
$G\ (\nu \star_D \nu') \cdot_C \Phi_{G0}\ (D.dom\ \nu)\ (D.dom\ \nu')$
proof −
let $?X = \varepsilon\ (G\ (D.cod\ \nu) \star_C G\ (D.cod\ \nu')) \cdot_C$
$G\ (\Phi\ (G\ (D.cod\ \nu),\ G\ (D.cod\ \nu')) \cdot_D (\eta\ (D.cod\ \nu) \star_D \eta\ (D.cod\ \nu')))$
have *iso-X*: $C.iso\ ?X$
using *assms* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def*

by (*intro C.isos-compose*) *auto*
have Φ_{G0} (*D.cod* ν) (*D.cod* ν') $\cdot_C$ (G ν $\star_C$ G ν') = *C.inv ?X* $\cdot_C$ (G ν $\star_C$ G ν')
unfolding Φ_{G0}*-def* **by** *simp*
also have ... = (*C.inv* (G (Φ (G (*D.cod* ν), G (*D.cod* ν')) $\cdot_D$
(η (*D.cod* ν) $\star_D$ η (*D.cod* ν')))) $\cdot_C$
C.inv (ε (G (*D.cod* ν) $\star_C$ G (*D.cod* ν')))) $\cdot_C$
(G ν $\star_C$ G ν')
using *assms iso-X* η*-in-hom* ε*-in-hom*
by (*simp add*: *C.inv-comp-left*(*1*))
also have ... = (*C.inv* (G (Φ (G (*D.cod* ν), G (*D.cod* ν'))) $\cdot_C$
G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))) $\cdot_C$
C.inv (ε (G (*D.cod* ν) $\star_C$ G (*D.cod* ν')))) $\cdot_C$
(G ν $\star_C$ G ν')
proof −
have G (Φ (G (*D.cod* ν), G (*D.cod* ν')) $\cdot_D$ (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))) =
G (Φ (G (*D.cod* ν), G (*D.cod* ν'))) $\cdot_C$ G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))
using *assms iso-X* η*-in-hom* **by** *fast*
thus *?thesis* **by** *simp*
qed
also have ... = *C.inv* (G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))) $\cdot_C$
C.inv (G (Φ (G (*D.cod* ν), G (*D.cod* ν')))) $\cdot_C$
C.inv (ε (G (*D.cod* ν) $\star_C$ G (*D.cod* ν'))) $\cdot_C$
(G ν $\star_C$ G ν')
proof −
have *C.iso* (G (Φ (G (*D.cod* ν), G (*D.cod* ν'))) $\cdot_C$ G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν')))
using *assms iso-X* η*-in-hom* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
FF-def
by (*intro C.isos-compose C.seqI*) *auto*
hence *C.inv* (G (Φ (G (*D.cod* ν), G (*D.cod* ν'))) $\cdot_C$
G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν')))
= *C.inv* (G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))) $\cdot_C$
C.inv (G (Φ (G (*D.cod* ν), G (*D.cod* ν'))))
using *assms* η*-in-hom*
by (*simp add*: *C.inv-comp-right*(*1*))
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... = G (*D.inv* (η (*D.cod* ν)) $\star_D$ *D.inv* (η (*D.cod* ν'))) $\cdot_C$
C.inv (G (Φ (G (*D.cod* ν), G (*D.cod* ν')))) $\cdot_C$
C.inv (ε (G (*D.cod* ν) $\star_C$ G (*D.cod* ν'))) $\cdot_C$
(G ν $\star_C$ G ν')
proof −
have *C.inv* (G (η (*D.cod* ν) $\star_D$ η (*D.cod* ν'))) =
G (*D.inv* (η (*D.cod* ν)) $\star_D$ *D.inv* (η (*D.cod* ν')))
using *assms* η*-in-hom D.ide-cod D.iso-hcomp D.src-cod D.trg-cod G.preserves-inv*
η*-simps*(*2*−*3*) *D.inv-hcomp*
by (*metis D.arr-cod*)
thus *?thesis* **by** *simp*
qed

also have ... $= G\ (D.inv\ (\eta\ (D.cod\ \nu)) \star_D D.inv\ (\eta\ (D.cod\ \nu'))) \cdot_C$
$(C.inv\ (G\ (\Phi\ (G\ (D.cod\ \nu),\ G\ (D.cod\ \nu')))) \cdot_C$
$G\ (F\ (G\ \nu \star_C G\ \nu'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
proof −
have $C.inv\ (\varepsilon\ (G\ (D.cod\ \nu) \star_C G\ (D.cod\ \nu'))) \cdot_C (G\ \nu \star_C G\ \nu')$
$= G\ (F\ (G\ \nu \star_C G\ \nu')) \cdot_C C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
proof −
have $G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu') = C.dom\ (G\ \nu \star_C G\ \nu')$
by (*simp add*: *assms*)
thus *?thesis*
using *assms C.hcomp-simps*(*4*) *C.hseqI′ G.preserves-cod G.preserves-hseq*
ε-naturality(*3*)
by *presburger*
qed
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= (G\ (D.inv\ (\eta\ (D.cod\ \nu)) \star_D D.inv\ (\eta\ (D.cod\ \nu'))) \cdot_C$
$G\ (F\ (G\ \nu) \star_D F\ (G\ \nu'))) \cdot_C$
$G\ (D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
proof −
have $C.inv\ (G\ (\Phi\ (G\ (D.cod\ \nu),\ G\ (D.cod\ \nu')))) \cdot_C G\ (F\ (G\ \nu \star_C G\ \nu')) =$
$G\ (D.inv\ (\Phi\ (C.cod\ (G\ \nu),\ C.cod\ (G\ \nu'))) \cdot_D F\ (G\ \nu \star_C G\ \nu'))$
using *assms* **by** (*simp add*: *G.preserves-inv*)
also have ... $= G\ ((F\ (G\ \nu) \star_D F\ (G\ \nu')) \cdot_D D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu))))$
using *assms* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def*
Φ.inv-naturality [*of* $(G\ \nu,\ G\ \nu')$]
by *simp*
also have ... $= G\ (F\ (G\ \nu) \star_D F\ (G\ \nu')) \cdot_C$
$G\ (D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu))))$
using *assms* **by** *simp*
finally show *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= G\ (\nu \star_D \nu') \cdot_C$
$G\ (D.inv\ (\eta\ (D.dom\ \nu)) \star_D D.inv\ (\eta\ (D.dom\ \nu'))) \cdot_C$
$G\ (D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
proof −
have $G\ (D.inv\ (\eta\ (D.cod\ \nu)) \star_D D.inv\ (\eta\ (D.cod\ \nu'))) \cdot_C G\ (F\ (G\ \nu) \star_D F\ (G\ \nu')) =$
$G\ ((D.inv\ (\eta\ (D.cod\ \nu)) \star_D D.inv\ (\eta\ (D.cod\ \nu'))) \cdot_D (F\ (G\ \nu) \star_D F\ (G\ \nu')))$
using *assms* **by** *simp*
also have ... $= G\ (D.inv\ (\eta\ (D.cod\ \nu)) \cdot_D F\ (G\ \nu)$
$\star_D D.inv\ (\eta\ (D.cod\ \nu')) \cdot_D F\ (G\ \nu'))$
using *assms D.interchange* **by** *simp*
also have ... $= G\ (\nu \cdot_D D.inv\ (\eta\ (D.dom\ \nu)) \star_D \nu' \cdot_D D.inv\ (\eta\ (D.dom\ \nu')))$
using *assms η-naturality* **by** *simp*

also have ... $= G\ ((\nu \star_D \nu') \cdot_D (D.inv\ (\eta\ (D.dom\ \nu)) \star_D D.inv\ (\eta\ (D.dom\ \nu'))))$
using *assms D.interchange* **by** *simp*
also have ... $= G\ (\nu \star_D \nu') \cdot_C G\ (D.inv\ (\eta\ (D.dom\ \nu)) \star_D D.inv\ (\eta\ (D.dom\ \nu')))$
using *assms* **by** *simp*
finally show *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= G\ (\nu \star_D \nu') \cdot_C \Phi_{G0}\ (D.dom\ \nu)\ (D.dom\ \nu')$
proof −
have $G\ (D.inv\ (\eta\ (D.dom\ \nu)) \star_D D.inv\ (\eta\ (D.dom\ \nu'))) \cdot_C$
$G\ (D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
$= C.inv\ (G\ (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu'))) \cdot_C$
$C.inv\ (G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')))$
proof −
have $G\ (D.inv\ (\eta\ (D.dom\ \nu)) \star_D D.inv\ (\eta\ (D.dom\ \nu'))) =$
$C.inv\ (G\ (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu')))$
using *assms cmp-components-are-iso*
by (*metis D.arr-dom D.ide-dom D.inv-hcomp D.iso-hcomp D.src-dom D.trg-dom G.preserves-inv η-simps(2) η-simps(3) η-simps(6)*)
moreover have $G\ (D.inv\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')))) =$
$C.inv\ (G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu))))$
using *assms cmp-components-are-iso*
by (*simp add*: *G.preserves-inv*)
ultimately show *?thesis* **by** *simp*
qed
also have ... $= C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')) \cdot_C$
$G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu'))) \cdot_C$
$G\ (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu')))$
proof −
have $C.iso\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')) \cdot_C$
$G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu'))) \cdot_C$
$G\ (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu')))$
using *assms η-in-hom ε-in-hom cmp-simps′(1,4)* $C.VV.cod\text{-}char_{SbC}$
by (*intro C.isos-compose C.seqI*) *auto*
thus *?thesis*
using *assms cmp-components-are-iso C.comp-assoc C.inv-comp-left* **by** *simp*
qed
also have ... $= C.inv\ (\varepsilon\ (G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')) \cdot_C$
$G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')) \cdot_D$
$(\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu'))))$
proof −
have $G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu'))) \cdot_C G\ (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu')) =$
$G\ (\Phi\ (C.dom\ (G\ \nu),\ C.dom\ (G\ \nu')) \cdot_D (\eta\ (D.dom\ \nu) \star_D \eta\ (D.dom\ \nu')))$
using *assms cmp-simps′(1,4)* **by** *auto*
thus *?thesis* **by** *simp*
qed
also have ... $= \Phi_{G0}\ (D.dom\ \nu)\ (D.dom\ \nu')$

unfolding Φ_{G0}*-def*
using *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed

interpretation Φ_G: *transformation-by-components* $D.VV.comp$ V_C $H_D oGG.map$ $GoH_D.map$
‹$\lambda fg.$ Φ_{G0} (*fst fg*) (*snd fg*)›
using $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ Φ_{G0}*-in-hom* Φ_{G0}*-naturality* *G.FF-def*
$D.VV.dom\text{-}char_{SbC}$ $D.VV.cod\text{-}char_{SbC}$
by *unfold-locales auto*

interpretation Φ_G: *natural-isomorphism* $D.VV.comp$ V_C $H_D oGG.map$ $GoH_D.map$ $\Phi_G.map$
using Φ_G*.map-simp-ide* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ *iso-*Φ_{G0} **by** *unfold-locales simp*

abbreviation Φ_G
where $\Phi_G \equiv \Phi_G.map$

lemma Φ_G*-in-hom* [*intro*]:
assumes $D.arr\ \nu$ **and** $D.arr\ \nu'$ **and** $src_D\ \nu = trg_D\ \nu'$
shows $C.in\text{-}hhom\ (\Phi_G.map\ (\nu, \nu'))\ (src_C\ (G\ (D.dom\ \nu')))\ (trg_C\ (G\ (D.cod\ \nu)))$
and «$\Phi_G.map\ (\nu, \nu') : G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu') \Rightarrow_C G\ (D.cod\ \nu \star_D D.cod\ \nu')$»
proof −
show «$\Phi_G.map\ (\nu, \nu') : G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu') \Rightarrow_C G\ (D.cod\ \nu \star_D D.cod\ \nu')$»
proof −
have $\Phi_G.map\ (\nu, \nu') = \Phi_{G0}\ (D.cod\ \nu)\ (D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu')$
using *assms* $D.VV.arr\text{-}char_{SbC}$ Φ_G*.map-def* Φ_{G0}*-in-hom* *G.FF-def* $D.VV.cod\text{-}char_{SbC}$
by *simp*
moreover have «$\Phi_{G0}\ (D.cod\ \nu)\ (D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu') :$
$G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu') \Rightarrow_C G\ (D.cod\ \nu \star_D D.cod\ \nu')$»
using *assms* Φ_{G0}*-in-hom* **by** (*intro* $C.comp\text{-}in\text{-}homI$) *auto*
ultimately show *?thesis* **by** *simp*
qed
thus $C.in\text{-}hhom\ (\Phi_G.map\ (\nu, \nu'))\ (src_C\ (G\ (D.dom\ \nu')))\ (trg_C\ (G\ (D.cod\ \nu)))$
using *assms* $C.vconn\text{-}implies\text{-}hpar(1-2)$ **by** *auto*
qed

lemma Φ_G*-simps* [*simp*]:
assumes $D.arr\ \nu$ **and** $D.arr\ \nu'$ **and** $src_D\ \nu = trg_D\ \nu'$
shows $C.arr\ (\Phi_G.map\ (\nu, \nu'))$
and $src_C\ (\Phi_G.map\ (\nu, \nu')) = src_C\ (G\ (D.dom\ \nu'))$
and $trg_C\ (\Phi_G.map\ (\nu, \nu')) = trg_C\ (G\ (D.cod\ \nu))$
and $C.dom\ (\Phi_G.map\ (\nu, \nu')) = G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu')$
and $C.cod\ (\Phi_G.map\ (\nu, \nu')) = G\ (D.cod\ \nu \star_D D.cod\ \nu')$
using *assms* Φ_G*-in-hom*
apply *auto*
by *fast+*

lemma Φ_G*-map-simp-ide*:
assumes *D.ide f* **and** *D.ide f'* **and** $src_D\ f = trg_D\ f'$
shows $\Phi_G.map\ (f, f') = G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ f')) \cdot_C G\ (D.inv\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))$
proof −
have $\Phi_G.map\ (f, f') = C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$
using *assms* Φ_G*.map-simp-ide* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ Φ_{G0}*-def G.FF-def*
by *auto*
also have ... $= C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f')) \cdot_C G\ (\eta\ f \star_D \eta\ f'))$
using *assms* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ *cmp-simps(1,4)* **by** *simp*
also have ... $= C.inv\ (G\ (\eta\ f \star_D \eta\ f')) \cdot_C C.inv\ (G\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))$
proof −
have $C.iso\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f')) \cdot_C G\ (\eta\ f \star_D \eta\ f'))$
using *assms* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def*
by (*intro C.isos-compose C.seqI*) *auto*
thus *?thesis*
using *assms C.inv-comp-left(1−2) cmp-components-are-iso C.comp-assoc* **by** *simp*
qed
also have ... $= G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ f')) \cdot_C G\ (D.inv\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))$
using *assms cmp-components-are-iso D.ideD(1) D.inv-hcomp D.iso-hcomp*
G.preserves-ide G.preserves-inv G-simps(2−3) η-simps(2−3,6)
by *metis*
finally show *?thesis* **by** *blast*
qed

lemma *η-hcomp*:
assumes *D.ide f* **and** *D.ide f'* **and** $src_D\ f = trg_D\ f'$
shows $\eta\ (f \star_D f') = F\ (\Phi_G.map\ (f, f')) \cdot_D \Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')$
proof −
have *1*: «$F\ (\Phi_G.map\ (f, f')) \cdot_D \Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f') :$
$f \star_D f' \Rightarrow_D F\ (G\ (f \star_D f'))$»
using *assms* **by** *fastforce*
have *2*: «$\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f')) \cdot_C G\ (\eta\ f \star_D \eta\ f') :$
$G\ (f \star_D f') \Rightarrow_C G\ f \star_C G\ f'$»
using *assms G.preserves-hom cmp-in-hom(2)*
by (*intro C.comp-in-homI*) *auto*
have $F\ (\Phi_G.map\ (f, f')) \cdot_D \Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')$
$= F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))) \cdot_D$
$\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')$
using *assms* Φ_G*.map-simp-ide* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ Φ_{G0}*-def* **by** *simp*
also have ... $= F\ (C.inv\ (G\ (\eta\ f \star_D \eta\ f')) \cdot_C C.inv\ (G\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))) \cdot_D$
$\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')$
proof −
have $C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f'))) =$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f')) \cdot_C G\ (\eta\ f \star_D \eta\ f'))$

using *assms 1* **by** *force*
also have ... $= C.inv\ (G\ (\eta\ f \star_D \eta\ f')) \cdot_C C.inv\ (G\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))$
proof −
have $C.iso\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f')) \cdot_C G\ (\eta\ f \star_D \eta\ f'))$
using *assms 2* **by** (*intro C.isos-compose*) *auto*
thus *?thesis*
using *assms 1 C.inv-comp-left C.comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
also have ... $= F\ (C.inv\ (G\ (\eta\ f \star_D \eta\ f'))) \cdot_D$
$F\ (C.inv\ (G\ (\Phi\ (G\ f,\ G\ f')))) \cdot_D F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))) \cdot_D$
$\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')$
proof −
have $C.arr\ ((C.inv\ (G\ (\eta\ f \star_D \eta\ f')) \cdot_C C.inv\ (G\ (\Phi\ (G\ f,\ G\ f'))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ f'))))$
using *assms 1 2 cmp-components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def*
by (*intro C.seqI*) *auto*
thus *?thesis*
using *C.inv-comp-left D.comp-assoc* **by** *auto*
qed
also have ... $= D.inv\ (F\ (G\ (\eta\ f \star_D \eta\ f'))) \cdot_D$
$D.inv\ (F\ (G\ (\Phi\ (G\ f,\ G\ f')))) \cdot_D$
$D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f'))) \cdot_D$
$\Phi\ (G\ f,\ G\ f') \cdot_D$
$(\eta\ f \star_D \eta\ f')$
using *assms* **by** (*simp add*: *preserves-inv*)
also have ... $= D.inv\ ((\eta\ (F\ (G\ f) \star_D F\ (G\ f'))) \cdot_D (\eta\ f \star_D \eta\ f') \cdot_D$
$D.inv\ (\eta\ (f \star_D f'))) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ f')) \cdot_D \Phi\ (G\ f,\ G\ f') \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f')))) \cdot_D$
$D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f'))) \cdot_D$
$\Phi\ (G\ f,\ G\ f') \cdot_D$
$(\eta\ f \star_D \eta\ f')$
proof −
have $*$: $\bigwedge\nu.\ D.arr\ \nu \Longrightarrow \eta\ (D.cod\ \nu) \cdot_D \nu \cdot_D D.inv\ (\eta\ (D.dom\ \nu)) = F\ (G\ \nu)$
by (*metis* (*full-types*) *D.arr-dom D.comp-assoc D.ide-dom D.invert-side-of-triangle*(*2*)
η-naturality(*1−2*) *η-simps*(*1*,*6*))
have $F\ (G\ (\eta\ f \star_D \eta\ f')) =$
$\eta\ (F\ (G\ f) \star_D F\ (G\ f')) \cdot_D (\eta\ f \star_D \eta\ f') \cdot_D D.inv\ (\eta\ (f \star_D f'))$
using *assms* $*$ [*of* $\eta\ f \star_D \eta\ f'$] **by** *simp*
moreover have $F\ (G\ (\Phi\ (G\ f,\ G\ f'))) =$
$\eta\ (F\ (G\ f \star_C G\ f')) \cdot_D \Phi\ (G\ f,\ G\ f') \cdot_D D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f')))$
using *assms 1* $*$ [*of* $\Phi\ (G\ f,\ G\ f')$] *cmp-simps*(*5*) **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
also have ... $= \eta\ (f \star_D f') \cdot_D$
$D.inv\ (\eta\ f \star_D \eta\ f') \cdot_D$

$((D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f'))))\ \cdot_D\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f'))))\ \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ f'))\ \cdot_D$
$((D.inv\ (\eta\ (F\ (G\ f \star_C G\ f'))))\ \cdot_D\ D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f'))))\ \cdot_D$
$\Phi\ (G\ f,\ G\ f'))\ \cdot_D$
$(\eta\ f \star_D \eta\ f')$

proof −
have $D.inv\ ((\eta\ (F\ (G\ f) \star_D F\ (G\ f')))\ \cdot_D\ (\eta\ f \star_D \eta\ f')\ \cdot_D\ D.inv\ (\eta\ (f \star_D f')))$
$= \eta\ (f \star_D f')\ \cdot_D\ D.inv\ (\eta\ f \star_D \eta\ f')\ \cdot_D\ D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f')))$
proof −
have $D.iso\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f'))\ \cdot_D\ (\eta\ f \star_D \eta\ f')\ \cdot_D\ D.inv\ (\eta\ (f \star_D f')))$
using *assms* **by** (*intro D.isos-compose*) *auto*
thus *?thesis*
using *assms D.inv-comp-left D.comp-assoc* **by** *simp*
qed
moreover have
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ f'))\ \cdot_D\ \Phi\ (G\ f,\ G\ f')\ \cdot_D\ D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f'))))\ =$
$\eta\ (F\ (G\ f) \star_D F\ (G\ f'))\ \cdot_D\ D.inv\ (\Phi\ (G\ f,\ G\ f'))\ \cdot_D\ D.inv\ (\eta\ (F\ (G\ f \star_C G\ f')))$
proof −
have $D.iso\ (\eta\ (F\ (G\ f \star_C G\ f'))\ \cdot_D\ \Phi\ (G\ f,\ G\ f')\ \cdot_D\ D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f))))$
proof −
have *1*: $D.seq\ (\Phi\ (G\ f,\ G\ f'))\ (D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f))))$
using *assms* **by** (*intro D.seqI*) *auto*
moreover have $D.seq\ (\eta\ (F\ (G\ f \star_C G\ f')))$
$(\Phi\ (G\ f,\ G\ f')\ \cdot_D\ D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f))))$
using *assms 1* **by** *auto*
ultimately show *?thesis*
using *assms cmp-components-are-iso η-in-hom*
by (*intro D.isos-compose*) *auto*
qed
thus *?thesis*
using *assms D.comp-inv-arr′ D.comp-assoc D.inv-comp-left cmp-components-are-iso*
by *auto*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $=\ \eta\ (f \star_D f')\ \cdot_D$
$D.inv\ (\eta\ f \star_D \eta\ f')\ \cdot_D$
$(D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f')))\ \cdot_D\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f'))))\ \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ f'))\ \cdot_D\ \Phi\ (G\ f,\ G\ f'))\ \cdot_D$
$(\eta\ f \star_D \eta\ f')$

proof −
have $(D.inv\ (\eta\ (F\ (G\ f \star_C G\ f'))))\ \cdot_D\ D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f')))\ \cdot_D\ \Phi\ (G\ f,\ G\ f')$
$= F\ (G\ f \star_C G\ f')\ \cdot_D\ \Phi\ (G\ f,\ G\ f')$
proof −
have $(D.inv\ (\eta\ (F\ (G\ f \star_C G\ f'))))\ \cdot_D\ D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f')))$
$= D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f'))\ \cdot_D\ \eta\ (F\ (G\ f \star_C G\ f')))$
using *assms* **by** (*simp add*: *D.inv-comp*)
also have ... $= F\ (G\ f \star_C G\ f')$

using *assms* $\eta\varepsilon$*.triangle-G* $Fo\varepsilon$-ηoF*.map-simp-ide* **by** *fastforce*
finally show *?thesis* **using** *D.comp-assoc* **by** *metis*
qed
also have ... $= \Phi\ (G\ f,\ G\ f')$
using *assms D.comp-cod-arr cmp-in-hom(2)* [*of* $G\ f\ G\ f'$] **by** *auto*
finally have $(D.inv\ (\eta\ (F\ (G\ f \star_C G\ f'))))\ \cdot_D\ D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ f')))\ \cdot_D$
$\Phi\ (G\ f,\ G\ f')$
$= \Phi\ (G\ f,\ G\ f')$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \eta\ (f \star_D f')\ \cdot_D$
$D.inv\ (\eta\ f \star_D \eta\ f')\ \cdot_D$
$(D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f')))\ \cdot_D\ (\eta\ (F\ (G\ f) \star_D F\ (G\ f))))\ \cdot_D$
$(\eta\ f \star_D \eta\ f')$
using *assms D.comp-cod-arr D.comp-inv-arr′ cmp-components-are-iso* **by** *simp*
also have ... $= \eta\ (f \star_D f')\ \cdot_D\ D.inv\ (\eta\ f \star_D \eta\ f')\ \cdot_D\ (\eta\ f \star_D \eta\ f')$
using *assms* η*-in-hom D.comp-inv-arr′ D.comp-cod-arr* **by** *simp*
also have ... $= \eta\ (f \star_D f')$
using *assms D.comp-inv-arr′* [*of* $\eta\ f \star_D \eta\ f'$] *D.comp-arr-dom* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma ε*-hcomp*:
assumes $C.ide\ g$ **and** $C.ide\ g'$ **and** $src_C\ g = trg_C\ g'$
shows $\varepsilon\ (g \star_C g') = (\varepsilon\ g \star_C \varepsilon\ g')\ \cdot_C\ C.inv\ (\Phi_G.map\ (F\ g,\ F\ g'))\ \cdot_C\ C.inv\ (G\ (\Phi\ (g,\ g')))$
proof −
have *1*: «$\varepsilon\ (G\ (F\ g) \star_C G\ (F\ g'))\ \cdot_C\ G\ (\Phi\ (G\ (F\ g),\ G\ (F\ g'))\ \cdot_D\ (\eta\ (F\ g) \star_D \eta\ (F\ g'))) :$
$G\ (F\ g \star_D F\ g') \Rightarrow_C G\ (F\ g) \star_C G\ (F\ g')$»
using *assms* **by** *fastforce*
have $(\varepsilon\ g \star_C \varepsilon\ g')\ \cdot_C\ C.inv\ (\Phi_G.map\ (F\ g,\ F\ g'))\ \cdot_C\ C.inv\ (G\ (\Phi\ (g,\ g')))$
$= (\varepsilon\ g \star_C \varepsilon\ g')\ \cdot_C$
$C.inv\ (C.inv\ (\varepsilon\ (G\ (F\ g) \star_C G\ (F\ g'))\ \cdot_C\ G\ (\Phi\ (G\ (F\ g),\ G\ (F\ g'))\ \cdot_D$
$(\eta\ (F\ g) \star_D \eta\ (F\ g')))))\ \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$
using *assms* Φ_G*.map-simp-ide* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ Φ_{G0}*-def* **by** *simp*
also have ... $= ((\varepsilon\ g \star_C \varepsilon\ g')\ \cdot_C\ \varepsilon\ (G\ (F\ g) \star_C G\ (F\ g')))\ \cdot_C$
$G\ (\Phi\ (G\ (F\ g),\ G\ (F\ g'))\ \cdot_D\ (\eta\ (F\ g) \star_D \eta\ (F\ g')))\ \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$
proof −
have $C.iso\ (\varepsilon\ (G\ (F\ g) \star_C G\ (F\ g'))\ \cdot_C\ G\ (\Phi\ (G\ (F\ g),\ G\ (F\ g'))\ \cdot_D$
$(\eta\ (F\ g) \star_D \eta\ (F\ g))))$
using *assms 1*
by (*intro C.isos-compose D.isos-compose G.preserves-iso*) *auto*
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= \varepsilon\ (g \star_C g')\ \cdot_C$

$(G\ (F\ (\varepsilon\ g \star_C \varepsilon\ g')) \cdot_C G\ (\Phi\ (G\ (F\ g),\ G\ (F\ g')) \cdot_D$
$(\eta\ (F\ g) \star_D \eta\ (F\ g')))) \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$

proof −
have $(\varepsilon\ g \star_C \varepsilon\ g') \cdot_C \varepsilon\ (G\ (F\ g) \star_C G\ (F\ g')) = \varepsilon\ (g \star_C g') \cdot_C G\ (F\ (\varepsilon\ g \star_C \varepsilon\ g'))$
using *assms* ε*-naturality* [*of* $\varepsilon\ g \star_C \varepsilon\ g'$] **by** *simp*
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= \varepsilon\ (g \star_C g') \cdot_C$
$G\ ((F\ (\varepsilon\ g \star_C \varepsilon\ g') \cdot_D \Phi\ (G\ (F\ g),\ G\ (F\ g'))) \cdot_D (\eta\ (F\ g) \star_D \eta\ (F\ g'))) \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$
using *assms* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def* *D.comp-assoc* **by** *auto*
also have ... $= \varepsilon\ (g \star_C g') \cdot_C$
$G\ (\Phi\ (g,\ g') \cdot_D (F\ (\varepsilon\ g) \star_D F\ (\varepsilon\ g')) \cdot_D (\eta\ (F\ g) \star_D \eta\ (F\ g'))) \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$
proof −
have $F\ (\varepsilon\ g \star_C \varepsilon\ g') \cdot_D \Phi\ (G\ (F\ g),\ G\ (F\ g')) = \Phi\ (g,\ g') \cdot_D (F\ (\varepsilon\ g) \star_D F\ (\varepsilon\ g'))$
using *assms* $C.VV.arr\text{-}char_{SbC}$ $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ *FF-def*
Φ*.naturality* [*of* $(\varepsilon\ g,\ \varepsilon\ g')$] $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \varepsilon\ (g \star_C g') \cdot_C$
$G\ (\Phi\ (g,\ g') \cdot_D (F\ (\varepsilon\ g) \cdot_D \eta\ (F\ g) \star_D F\ (\varepsilon\ g') \cdot_D \eta\ (F\ g'))) \cdot_C$
$C.inv\ (G\ (\Phi\ (g,\ g')))$
using *assms* *D.interchange* **by** *simp*
also have ... $= \varepsilon\ (g \star_C g') \cdot_C G\ (\Phi\ (g,\ g') \cdot_D (F\ g \star_D F\ g')) \cdot_C C.inv\ (G\ (\Phi\ (g,\ g')))$
using *assms* $\eta\varepsilon$*.triangle-G* $Fo\varepsilon\text{-}\eta oF$*.map-def* [*of g*] $Fo\varepsilon\text{-}\eta oF$*.map-def* [*of g'*] **by** *simp*
also have ... $= \varepsilon\ (g \star_C g') \cdot_C G\ (\Phi\ (g,\ g')) \cdot_C C.inv\ (G\ (\Phi\ (g,\ g')))$
using *assms* *D.comp-arr-dom* *cmp-simps*(*1*) *cmp-simps*(*4*) **by** *auto*
also have ... $= \varepsilon\ (g \star_C g')$
using *assms* *D.comp-arr-dom* *C.comp-arr-inv'* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
cmp-components-are-iso *C.comp-arr-dom*
by *simp*
finally show *?thesis* **by** *simp*
qed

lemma *G-preserves-hcomp*:
assumes $D.hseq\ \nu\ \nu'$
shows $G\ (\nu \star_D \nu') = \Phi_G.map\ (D.cod\ \nu,\ D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu') \cdot_C$
$C.inv\ (\Phi_G.map\ (D.dom\ \nu,\ D.dom\ \nu'))$
proof −
have $G\ (\nu \star_D \nu') \cdot_C \Phi_G.map\ (D.dom\ \nu,\ D.dom\ \nu') =$
$\Phi_G.map\ (D.cod\ \nu,\ D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu')$
using *assms* Φ_G*.naturality* $D.VV.arr\text{-}char_{SbC}$ $D.VV.cod\text{-}char_{SbC}$ *G.FF-def* $D.VV.dom\text{-}char_{SbC}$
by *auto*
moreover have $C.seq\ (\Phi_G.map\ (D.cod\ \nu,\ D.cod\ \nu'))\ (G\ \nu \star_C G\ \nu')$

proof
show «$G\ \nu \star_C G\ \nu'$:
$G\ (D.dom\ \nu) \star_C G\ (D.dom\ \nu') \Rightarrow_C G\ (D.cod\ \nu) \star_C G\ (D.cod\ \nu')$»
using *assms G-in-hom(2) C.hcomp-in-vhom* **by** *auto*
show «$\Phi_G.map\ (D.cod\ \nu,\ D.cod\ \nu')$:
$G\ (D.cod\ \nu) \star_C G\ (D.cod\ \nu') \Rightarrow_C G\ (D.cod\ \nu \star_D D.cod\ \nu')$»
using *assms* Φ_{G0}*-in-hom* Φ_G*.map-simp-ide* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ **by** *auto*
qed
moreover have $C.iso\ (\Phi_G.map\ (D.dom\ \nu,\ D.dom\ \nu'))$
using *assms* Φ_G*.components-are-iso* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ **by** *auto*
ultimately show *?thesis*
using *assms* Φ_G*.components-are-iso C.comp-assoc*
C.invert-side-of-triangle(2)
[*of* $\Phi_G.map\ (D.cod\ \nu,\ D.cod\ \nu') \cdot_C (G\ \nu \star_C G\ \nu')$
$G\ (\nu \star_D \nu')\ \Phi_G.map\ (D.dom\ \nu,\ D.dom\ \nu')$]
by *simp*
qed

lemma *coherence-LHS*:
assumes $D.ide\ f$ **and** $D.ide\ g$ **and** $D.ide\ h$
and $src_D\ f = trg_D\ g$ **and** $src_D\ g = trg_D\ h$
shows $F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h))$
$= (\eta\ (f \star_D g \star_D h) \cdot_D (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$\mathrm{a}_D[F\ (G\ f),\ F\ (G\ g),\ F\ (G\ h)] \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)) \cdot_D D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h))$
proof −
have $F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h))$
$= F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C$
$(G\ (D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h)) \cdot_C$
$(G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ g)) \star_C G\ h))$
using *assms* Φ_G*-map-simp-ide C.comp-assoc* **by** *simp*
also have ... $= F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C$
$(G\ (D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h)) \cdot_C$
$(G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \star_C G\ h) \cdot_C$
$(G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))) \star_C G\ h) \cdot_C$
$(C.inv\ (\varepsilon\ (G\ f \star_C G\ g)) \star_C G\ h))$
using *assms C.whisker-right D.comp-assoc* **by** *simp*
also have ... $= F\ (G\ \mathrm{a}_D[f,\ g,\ h]) \cdot_D$
$F\ (G\ (D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))) \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h)))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \star_C G\ h) \cdot_D$

$F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ g)))\ \star_C\ G\ h)\ \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f\ \star_C\ G\ g))\ \star_C\ G\ h)$

proof −
have $C.arr\ (G\ \mathrm{a}_D[f,\ g,\ h]\ \cdot_C$
$(G\ (D.inv\ (\eta\ (f\ \star_D\ g))\ \star_D\ D.inv\ (\eta\ h)))\ \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ (f\ \star_D\ g),\ G\ h)))\ \cdot_C$
$C.inv\ (\varepsilon\ (G\ (f\ \star_D\ g)\ \star_C\ G\ h))\ \cdot_C$
$(G\ (D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_C\ G\ h)\ \cdot_C$
$(G\ (D.inv\ (\Phi\ (G\ f,\ G\ g)))\ \star_C\ G\ h)\ \cdot_C$
$(C.inv\ (\varepsilon\ (G\ f\ \star_C\ G\ g))\ \star_C\ G\ h))$
using *assms* Φ_G*-simps G.FF-def* G_0*-props* **by** *auto*
thus *?thesis*
by (*metis C.seqE as-nat-trans.preserves-comp-2*)
qed
also have ... $= F\ (G\ \mathrm{a}_D[f,\ g,\ h])\ \cdot_D$
$F\ (G\ (D.inv\ (\eta\ (f\ \star_D\ g))\ \star_D\ D.inv\ (\eta\ h)))\ \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ (f\ \star_D\ g),\ G\ h))))\ \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f\ \star_D\ g)\ \star_C\ G\ h)))\ \cdot_D$
$(\Phi\ (G\ (f\ \star_D\ g),\ G\ h)\ \cdot_D$
$(F\ (G\ (D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g)))\ \star_D\ F\ (G\ h))\ \cdot_D$
$((D.inv\ (\Phi\ (G\ (F\ (G\ f)\ \star_D\ F\ (G\ g)),\ G\ h)))\ \cdot_D$
$(\Phi\ (G\ (F\ (G\ f)\ \star_D\ F\ (G\ g)),\ G\ h))\ \cdot_D$
$(F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))))\ \star_D\ F\ (G\ h)))\ \cdot_D$
$((D.inv\ (\Phi\ (G\ (F\ (G\ f\ \star_C\ G\ g)),\ G\ h)))\ \cdot_D$
$(\Phi\ (G\ (F\ (G\ f\ \star_C\ G\ g)),\ G\ h))\ \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f\ \star_C\ G\ g)))\ \star_D\ F\ (G\ h)))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f\ \star_C\ G\ g,\ G\ h)))$
using *assms* G_0*-props preserves-hcomp FF-def D.comp-assoc* **by** *auto*
also have ... $= F\ (G\ \mathrm{a}_D[f,\ g,\ h])\ \cdot_D$
$F\ (G\ (D.inv\ (\eta\ (f\ \star_D\ g))\ \star_D\ D.inv\ (\eta\ h)))\ \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ (f\ \star_D\ g),\ G\ h))))\ \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f\ \star_D\ g)\ \star_C\ G\ h)))\ \cdot_D$
$(\Phi\ (G\ (f\ \star_D\ g),\ G\ h)\ \cdot_D$
$(F\ (G\ (D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g)))\ \star_D\ F\ (G\ h))\ \cdot_D$
$(F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))))\ \star_D\ F\ (G\ h))\ \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f\ \star_C\ G\ g)))\ \star_D\ F\ (G\ h))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f\ \star_C\ G\ g,\ G\ h)))$
proof −
have $((D.inv\ (\Phi\ (G\ (F\ (G\ f)\ \star_D\ F\ (G\ g)),\ G\ h)))\ \cdot_D$
$(\Phi\ (G\ (F\ (G\ f)\ \star_D\ F\ (G\ g)),\ G\ h)))\ \cdot_D$
$(F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))))\ \star_D\ F\ (G\ h))$
$= F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ g))))\ \star_D\ F\ (G\ h)$
using *assms* Φ*.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ G_0*-props FF-def*
D.comp-inv-arr′ D.comp-cod-arr
by *simp*
moreover have $(D.inv\ (\Phi\ (G\ (F\ (G\ f\ \star_C\ G\ g)),\ G\ h))\ \cdot_D$
$\Phi\ (G\ (F\ (G\ f\ \star_C\ G\ g)),\ G\ h))\ \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f\ \star_C\ G\ g)))\ \star_D\ F\ (G\ h)) =$

$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ g))) \star_D F\ (G\ h)$
using *assms D.comp-cod-arr* Φ*.components-are-iso C.VV.ide-char*$_{SbC}$ *C.VV.arr-char*$_{SbC}$ G_0*-props FF-def D.comp-inv-arr′*
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \eta\ (f \star_D g \star_D h) \cdot_D$
$\mathrm{a}_D[f,\ g,\ h] \cdot_D$
$((D.inv\ (\eta\ ((f \star_D g) \star_D h))) \cdot_D$
$(\eta\ ((f \star_D g) \star_D h)) \cdot_D$
$(D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))) \cdot_D$
$(((D.inv\ (\eta\ (F\ (G\ (f \star_D g)) \star_D F\ (G\ h)))) \cdot_D$
$(\eta\ (F\ (G\ (f \star_D g)) \star_D F\ (G\ h)))) \cdot_D$
$D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h))) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$(\Phi\ (G\ (f \star_D g),\ G\ h) \cdot_D$
$((\eta\ (f \star_D g) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ g)))) \star_D F\ (G\ h)) \cdot_D$
$((\eta\ (F\ (G\ f) \star_D F\ (G\ g)) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ g)))) \star_D F\ (G\ h)) \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ g))) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h)))$
proof −
have $\bigwedge\nu.\ D.arr\ \nu \Longrightarrow F\ (G\ \nu) = \eta\ (D.cod\ \nu) \cdot_D \nu \cdot_D D.inv\ (\eta\ (D.dom\ \nu))$
by (*metis* (*full-types*) *D.arr-dom D.comp-assoc D.ide-dom D.invert-side-of-triangle*(*2*) η*-naturality*(*1*−*2*) η*-simps*(*1*,*6*))
thus *?thesis*
using *assms D.comp-assoc* **by** *simp*
qed
also have ... $= \eta\ (f \star_D g \star_D h) \cdot_D$
$\mathrm{a}_D[f,\ g,\ h] \cdot_D$
$(D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$(\Phi\ (G\ (f \star_D g),\ G\ h) \cdot_D$
$((\eta\ (f \star_D g) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ g))) \star_D F\ (G\ h)) \cdot_D$
$(\eta\ (F\ (G\ f) \star_D F\ (G\ g)) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ g))) \star_D F\ (G\ h))) \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ g))) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h)))$
proof −

have $(D.inv\ (\eta\ ((f \star_D g) \star_D h)) \cdot_D \eta\ ((f \star_D g) \star_D h)) \cdot_D$
$(D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))$
$= D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h)$
using *assms D.comp-inv-arr′ D.comp-cod-arr η-in-hom* **by** *simp*
moreover have $((D.inv\ (\eta\ (F\ (G\ (f \star_D g)) \star_D F\ (G\ h)))) \cdot_D$
$\eta\ (F\ (G\ (f \star_D g)) \star_D F\ (G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h))$
$= D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h))$
using *assms D.comp-inv-arr′ D.comp-cod-arr* **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \eta\ (f \star_D g \star_D h) \cdot_D$
$\mathrm{a}_D[f,\ g,\ h] \cdot_D$
$((D.inv\ (\eta\ (f \star_D g)) \star_D D.inv\ (\eta\ h))) \cdot_D$
$(D.inv\ (\Phi\ (G\ (f \star_D g),\ G\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ (f \star_D g) \star_C G\ h)))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ (f \star_D g) \star_C G\ h))) \cdot_D$
$(\Phi\ (G\ (f \star_D g),\ G\ h) \cdot_D$
$(\eta\ (f \star_D g) \star_D F\ (G\ h)) \cdot_D$
$((D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \star_D F\ (G\ h)) \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)) \cdot_D$
$(((D.inv\ (\eta\ (F\ (G\ f \star_C G\ g))) \star_D F\ (G\ h)) \cdot_D$
$(F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ g))) \star_D F\ (G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h))))$
proof −
have $(\eta\ (f \star_D g) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ g))) \star_D F\ (G\ h)) \cdot_D$
$(\eta\ (F\ (G\ f) \star_D F\ (G\ g)) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ g))) \star_D F\ (G\ h))$
$= (\eta\ (f \star_D g) \star_D F\ (G\ h)) \cdot_D$
$((D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \star_D F\ (G\ h)) \cdot_D$
$(((D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ g))) \star_D F\ (G\ h)) \cdot_D$
$(\eta\ (F\ (G\ f) \star_D F\ (G\ g)) \star_D F\ (G\ h))) \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h))) \cdot_D$
$(D.inv\ (\eta\ (F\ (G\ f \star_C G\ g))) \star_D F\ (G\ h))$
using *assms D.comp-assoc D.whisker-right* **by** *simp*
also have ... $= (\eta\ (f \star_D g) \star_D F\ (G\ h)) \cdot_D$
$((D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g)) \star_D F\ (G\ h)) \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)) \cdot_D$
$(D.inv\ (\eta\ (F\ (G\ f \star_C G\ g))) \star_D F\ (G\ h))$
proof −
have $((D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ g))) \star_D F\ (G\ h)) \cdot_D$
$(\eta\ (F\ (G\ f) \star_D F\ (G\ g)) \star_D F\ (G\ h))) \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h))$
$= D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)$
using *assms D.comp-inv-arr′ D.comp-cod-arr*

$D.interchange$ [*of* $D.inv$ (η (F (G f) $\star_D$ F (G g))) η (F (G f) $\star_D$ F (G g))
F (G h) F (G h)]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = η (f $\star_D$ g $\star_D$ h) $\cdot_D$
$\mathrm{a}_D[f, g, h]$ $\cdot_D$
(($D.inv$ (η (f $\star_D$ g)) $\star_D$ $D.inv$ (η h))) $\cdot_D$
($D.inv$ (Φ (G (f $\star_D$ g), G h)) $\cdot_D$
(($D.inv$ (η (F (G (f $\star_D$ g) $\star_C$ G h)))) $\cdot_D$
F ($C.inv$ (ε (G (f $\star_D$ g) $\star_C$ G h)))) $\cdot_D$
Φ (G (f $\star_D$ g), G h)) $\cdot_D$
(η (f $\star_D$ g) $\star_D$ F (G h)) $\cdot_D$
(($D.inv$ (η f) $\star_D$ $D.inv$ (η g)) $\star_D$ F (G h)) $\cdot_D$
($D.inv$ (Φ (G f, G g)) $\star_D$ F (G h)) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
proof −
have (($D.inv$ (η (F (G f $\star_C$ G g))) $\star_D$ F (G h)) $\cdot_D$
(F ($C.inv$ (ε (G f $\star_C$ G g))) $\star_D$ F (G h))) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
= $D.inv$ (Φ (G f $\star_C$ G g, G h))
proof −
have (($D.inv$ (η (F (G f $\star_C$ G g))) $\star_D$ F (G h)) $\cdot_D$
(F ($C.inv$ (ε (G f $\star_C$ G g))) $\star_D$ F (G h))) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
= (($D.inv$ (η (F (G f $\star_C$ G g))) $\star_D$ F (G h)) $\cdot_D$
($D.inv$ (F (ε (G f $\star_C$ G g))) $\star_D$ F (G h))) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
using *assms* **by** (*simp add*: *preserves-inv*)
also have ... = ($D.inv$ (η (F (G f $\star_C$ G g))) $\cdot_D$
$D.inv$ (F (ε (G f $\star_C$ G g))) $\star_D$ F (G h) $\cdot_D$ F (G h)) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
using *assms* Φ*.components-are-iso*
$D.interchange$ [*of* $D.inv$ (η (F (G f $\star_C$ G g))) $D.inv$ (F (ε (G f $\star_C$ G g)))
F (G h) F (G h)]
by *simp*
also have ... = ($D.inv$ (F (ε (G f $\star_C$ G g)) $\cdot_D$ η (F (G f $\star_C$ G g))) $\star_D$ F (G h)) $\cdot_D$
$D.inv$ (Φ (G f $\star_C$ G g, G h))
proof −
have $D.iso$ (F (ε (G f $\star_C$ G g)) $\cdot_D$ η (F (G f $\star_C$ G g)))
using *assms* **by** *auto*
hence $D.inv$ (η (F (G f $\star_C$ G g))) $\cdot_D$ $D.inv$ (F (ε (G f $\star_C$ G g))) =
$D.inv$ (F (ε (G f $\star_C$ G g)) $\cdot_D$ η (F (G f $\star_C$ G g)))
using *assms* **by** (*simp add*: *D.inv-comp-right*(*1*))
thus *?thesis*

```
      using assms by simp
    qed
    also have ... = (F (G f ⋆C G g) ⋆D F (G h)) ·D D.inv (Φ (G f ⋆C G g, G h))
      using assms ηε.triangle-G Foε-ηoF.map-def [of G f ⋆C G g] by simp
    also have ... = D.inv (Φ (G f ⋆C G g, G h))
      using assms D.comp-cod-arr by simp
    finally show ?thesis by simp
  qed
  thus ?thesis
    using D.comp-assoc by simp
qed
also have ... = η (f ⋆D g ⋆D h) ·D
                aD[f, g, h] ·D
                ((D.inv (η (f ⋆D g)) ⋆D D.inv (η h))) ·D
                ((D.inv (Φ (G (f ⋆D g), G h)) ·D
                Φ (G (f ⋆D g), G h)) ·D
                (η (f ⋆D g) ⋆D F (G h))) ·D
                ((D.inv (η f) ⋆D D.inv (η g)) ⋆D F (G h)) ·D
                (D.inv (Φ (G f, G g)) ⋆D F (G h)) ·D
                D.inv (Φ (G f ⋆C G g, G h))
proof −
  have ((D.inv (η (F (G (f ⋆D g) ⋆C G h))) ·D F (C.inv (ε (G (f ⋆D g) ⋆C G h))))) ·D
        (Φ (G (f ⋆D g), G h))
          = Φ (G (f ⋆D g), G h)
  proof −
    have D.inv (η (F (G (f ⋆D g) ⋆C G h))) ·D F (C.inv (ε (G (f ⋆D g) ⋆C G h))) =
          D.inv (η (F (G (f ⋆D g) ⋆C G h))) ·D D.inv (F (ε (G (f ⋆D g) ⋆C G h)))
      using assms preserves-inv by simp
    also have ... = D.inv (F (ε (G (f ⋆D g) ⋆C G h)) ·D η (F (G (f ⋆D g) ⋆C G h)))
    proof −
      have D.iso (F (ε (G (f ⋆D g) ⋆C G h)) ·D η (F (G (f ⋆D g) ⋆C G h)))
        using assms by auto
      thus ?thesis
        using assms D.inv-comp-right(1) by simp
    qed
    also have ... = F (G (f ⋆D g) ⋆C G h)
      using assms ηε.triangle-G Foε-ηoF.map-def [of G (f ⋆D g) ⋆C G h] by simp
    finally show ?thesis
      using assms D.comp-cod-arr [of Φ (G (f ⋆D g), G h) F (G (f ⋆D g) ⋆C G h)]
      by fastforce
  qed
  thus ?thesis
    using D.comp-assoc by simp
qed
also have ... = η (f ⋆D g ⋆D h) ·D
                aD[f, g, h] ·D
                (((D.inv (η (f ⋆D g)) ⋆D D.inv (η h)) ·D
                (η (f ⋆D g) ⋆D F (G h))) ·D
                ((D.inv (η f) ⋆D D.inv (η g)) ⋆D F (G h))) ·D
```

$(D.inv\ (\Phi\ (G\ f,\ G\ g))\ \star_D\ F\ (G\ h))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f\ \star_C\ G\ g,\ G\ h))$
proof −
have $((D.inv\ (\Phi\ (G\ (f\ \star_D\ g),\ G\ h))\ \cdot_D\ \Phi\ (G\ (f\ \star_D\ g),\ G\ h))\ \cdot_D$
$(\eta\ (f\ \star_D\ g)\ \star_D\ F\ (G\ h)))$
$= \eta\ (f\ \star_D\ g)\ \star_D\ F\ (G\ h)$
using *assms D.comp-inv-arr′ Φ.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
G_0*-props FF-def D.comp-cod-arr*
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \eta\ (f\ \star_D\ g\ \star_D\ h)\ \cdot_D$
$(\mathrm{a}_D[f,\ g,\ h]\ \cdot_D$
$((D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_D\ D.inv\ (\eta\ h)))\ \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g))\ \star_D\ F\ (G\ h))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f\ \star_C\ G\ g,\ G\ h))$
proof −
have $((D.inv\ (\eta\ (f\ \star_D\ g))\ \star_D\ D.inv\ (\eta\ h))\ \cdot_D\ (\eta\ (f\ \star_D\ g)\ \star_D\ F\ (G\ h)))\ \cdot_D$
$((D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_D\ F\ (G\ h))$
$= (D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_D\ D.inv\ (\eta\ h)$
proof −
have $(D.inv\ (\eta\ (f\ \star_D\ g))\ \star_D\ D.inv\ (\eta\ h))\ \cdot_D\ (\eta\ (f\ \star_D\ g)\ \star_D\ F\ (G\ h))$
$= (f\ \star_D\ g)\ \star_D\ D.inv\ (\eta\ h)$
using *assms D.comp-inv-arr′ D.comp-arr-dom*
D.interchange [*of* $D.inv\ (\eta\ (f\ \star_D\ g))\ \eta\ (f\ \star_D\ g)\ D.inv\ (\eta\ h)\ F\ (G\ h)$]
D.invert-side-of-triangle(*2*)
by *simp*
moreover have $((f\ \star_D\ g)\ \star_D\ D.inv\ (\eta\ h))\ \cdot_D$
$((D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_D\ F\ (G\ h))$
$= (D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g))\ \star_D\ D.inv\ (\eta\ h)$
using *assms D.comp-cod-arr D.comp-arr-dom*
D.interchange [*of* $f\ \star_D\ g\ D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g)\ D.inv\ (\eta\ h)\ F\ (G\ h)$]
by *simp*
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\eta\ (f\ \star_D\ g\ \star_D\ h)\ \cdot_D$
$(D.inv\ (\eta\ f)\ \star_D\ D.inv\ (\eta\ g)\ \star_D\ D.inv\ (\eta\ h))\ \cdot_D$
$\mathrm{a}_D[F\ (G\ f),\ F\ (G\ g),\ F\ (G\ h)])\ \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g))\ \star_D\ F\ (G\ h))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f\ \star_C\ G\ g,\ G\ h))$
using *assms D.assoc-naturality* [*of* $D.inv\ (\eta\ f)\ D.inv\ (\eta\ g)\ D.inv\ (\eta\ h)$]
D.comp-assoc
by *simp*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*

qed

lemma *coherence-RHS*:
assumes $D.ide\ f$ **and** $D.ide\ g$ **and** $D.ide\ h$
and $src_D\ f = trg_D\ g$ **and** $src_D\ g = trg_D\ h$
shows $F\ (\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)))$
$= (\eta\ (f \star_D g \star_D h) \cdot_D (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$(F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
proof −
have $F\ (\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)))$
$= F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))) \cdot_C$
$(G\ f \star_C G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h)) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ g,\ G\ h))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
using *assms* Φ_G*-map-simp-ide* C*.comp-assoc* **by** *simp*
also have ... $= F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))) \cdot_C$
$(G\ f \star_C G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_C$
$(G\ f \star_C G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))) \cdot_C$
$(G\ f \star_C C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
using *assms* C*.whisker-left* **by** *simp*
also have ... $= F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h)))) \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h))))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D$
$F\ (G\ f \star_C G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$F\ (G\ f \star_C G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))) \cdot_D$
$F\ (G\ f \star_C C.inv\ (\varepsilon\ (G\ g \star_C G\ h)))$
proof −
have $C.arr\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_C$
$G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_C$
$C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))) \cdot_C$
$(G\ f \star_C G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_C$
$(G\ f \star_C G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))) \cdot_C$
$(G\ f \star_C C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
using *assms* G_0*-props* **by** *auto*
thus *?thesis*
using *assms* **by** (*metis* C*.seqE as-nat-trans.preserves-comp-2*)
qed
also have ... $= F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h)))) \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h))))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D$
$(\Phi\ (G\ f,\ G\ (g \star_D h)) \cdot_D$
$(F\ (G\ f) \star_D F\ (G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h)))) \cdot_D$
$((D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h))))) \cdot_D$
$(\Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$

$(F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h))))))\ \cdot_D$
$((D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h)))))\ \cdot_D$
$(\Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h))))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h)))))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h)))$
using *assms preserves-hcomp* G_0*-props D.comp-assoc* **by** *simp*
also have ... $= F\ (G\ (D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))))\ \cdot_D$
$F\ (G\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))))\ \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))))\ \cdot_D$
$(\Phi\ (G\ f,\ G\ (g \star_D h))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (G\ (D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))\ \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h)))$
proof −
have $D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h))))\ \cdot_D$
$\Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h)))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))))$
$= (F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))))$
proof −
have $D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h))))\ \cdot_D\ \Phi\ (G\ f,\ G\ (F\ (G\ g) \star_D F\ (G\ h)))$
$= F\ (G\ f) \star_D F\ (G\ (F\ (G\ g) \star_D F\ (G\ h)))$
using *assms D.comp-inv-arr D.inv-is-inverse* G_0*-props FF-def* **by** *simp*
moreover have ... $= D.cod\ (F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))))$
using *assms* G_0*-props* **by** *simp*
moreover have $D.hseq\ (F\ (G\ f))\ (F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h)))))$
using *assms* G_0*-props* **by** *auto*
ultimately show *?thesis*
using *assms D.comp-assoc*
$D.comp\text{-}cod\text{-}arr\ [of\ F\ (G\ f) \star_D F\ (G\ (D.inv\ (\Phi\ (G\ g,\ G\ h))))$
$F\ (G\ f) \star_D F\ (G\ (F\ (G\ g) \star_D F\ (G\ h)))]$
by *metis*
qed
moreover have $D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h))))\ \cdot_D$
$(\Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h))))\ \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
$= (F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
proof −
have $D.inv\ (\Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h))))\ \cdot_D\ \Phi\ (G\ f,\ G\ (F\ (G\ g \star_C G\ h)))$
$= F\ (G\ f) \star_D F\ (G\ (F\ (G\ g \star_C G\ h)))$
using *assms D.comp-inv-arr D.inv-is-inverse* G_0*-props* **by** *simp*
moreover have ... $= D.cod\ (F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
using *assms* **by** *simp*
moreover have $D.hseq\ (F\ (G\ f))\ (F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
using *assms* **by** (*intro D.hseqI′*) *auto*
ultimately show *?thesis*
using *assms D.comp-assoc*
$D.comp\text{-}cod\text{-}arr\ [of\ F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h)))$
$F\ (G\ f) \star_D F\ (G\ (F\ (G\ g \star_C G\ h)))]$

by *metis*
qed
ultimately show *?thesis* **by** *simp*
qed
also have ... $= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_D$
$((D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h))))) \cdot_D$
$\eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h)))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_D$
$((D.inv\ (\eta\ (F\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))))) \cdot_D$
$\Phi\ (G\ f,\ G\ (g \star_D h))) \cdot_D$
$(F\ (G\ f) \star_D$
$\eta\ (g \star_D h) \cdot_D$
$(D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D$
$\eta\ (F\ (G\ g) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ g,\ G\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h)))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
proof −
have $\bigwedge\nu.\ D.arr\ \nu \Longrightarrow F\ (G\ \nu) = \eta\ (D.cod\ \nu) \cdot_D \nu \cdot_D D.inv\ (\eta\ (D.dom\ \nu))$
by (*metis* (*full-types*) *D.arr-dom D.comp-assoc D.ide-dom D.invert-side-of-triangle*(*2*) *η-naturality*(*1*−*2*) *η-simps*(*1*,*6*))
thus *?thesis*
using *assms D.comp-assoc* **by** *simp*
qed
also have ... $= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_D$
$\Phi\ (G\ f,\ G\ (g \star_D h)) \cdot_D$
$(F\ (G\ f) \star_D$
$\eta\ (g \star_D h) \cdot_D$
$(D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D$
$\eta\ (F\ (G\ g) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ g,\ G\ h)) \cdot_D$
$D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h)))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
proof −
have $D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h)))) \cdot_D \eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))$
$= D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))$
proof −
have $D.inv\ (\eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h)))) \cdot_D \eta\ (F\ (G\ f) \star_D F\ (G\ (g \star_D h))) =$

$F\ (G\ f) \star_D F\ (G\ (g \star_D h))$
using *assms D.comp-inv-arr D.inv-is-inverse* **by** *simp*
moreover have ... $= D.cod\ (D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h))))$
using *assms* **by** *simp*
ultimately show *?thesis*
using *assms D.comp-cod-arr* [*of D.inv* $(\Phi\ (G\ f,\ G\ (g \star_D h)))$]
by (*metis D.arr-inv D.comp-assoc D.hcomp-simps*(*2*) *D.ideD*(*1*) *D.ide-hcomp G.preserves-ide G-simps*(*2*) *G-simps*(*3*) *cmp-components-are-iso*)
qed
moreover have $(D.inv\ (\eta\ (F\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D$
$F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))))) \cdot_D$
$\Phi\ (G\ f,\ G\ (g \star_D h))$
$= \Phi\ (G\ f,\ G\ (g \star_D h))$
proof −
have $D.inv\ (\eta\ (F\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))))$
$= F\ (G\ f \star_C G\ (g \star_D h))$
proof −
have $D.inv\ (\eta\ (F\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D F\ (C.inv\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h)))) =$
$D.inv\ (\eta\ (F\ (G\ f \star_C G\ (g \star_D h)))) \cdot_D D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))))$
using *assms* **by** (*simp add*: *preserves-inv*)
also have ... $= D.inv\ (F\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))) \cdot_D \eta\ (F\ (G\ f \star_C G\ (g \star_D h))))$
proof −
have $D.iso\ (F\ (\varepsilon\ (G\ f \star_C G\ (g \star_D h))) \cdot_D \eta\ (F\ (G\ f \star_C G\ (g \star_D h))))$
using *assms* **by** *auto*
thus *?thesis*
using *assms* **by** (*simp add*: *D.inv-comp-right*(*1*))
qed
also have ... $= F\ (G\ f \star_C G\ (g \star_D h))$
using *assms Foε-ηoF.map-def* [*of* $G\ f \star_C G\ (g \star_D h)$] *ηε.triangle-G* **by** *simp*
finally show *?thesis* **by** *blast*
qed
moreover have ... $= D.cod\ (\Phi\ (G\ f,\ G\ (g \star_D h)))$
using *assms* **by** *simp*
moreover have $D.arr\ (\Phi\ (G\ f,\ G\ (g \star_D h)))$
using *assms* **by** *auto*
ultimately show *?thesis*
using *D.comp-cod-arr* **by** *simp*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_D$
$((D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h)))) \cdot_D$
$\Phi\ (G\ f,\ G\ (g \star_D h))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (g \star_D h))) \cdot_D$
$(F\ (G\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h)) \cdot_D$
$(((F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$

$(F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h)))) \cdot_D$
$((F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
using *assms D.whisker-left D.comp-assoc* **by** *simp*
also have ... $= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(((D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ (g \star_D h))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (g \star_D h)))) \cdot_D$
$(F\ (G\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$(F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
proof −
have $(D.inv\ (\Phi\ (G\ f,\ G\ (g \star_D h))) \cdot_D \Phi\ (G\ f,\ G\ (g \star_D h))) \cdot_D (F\ (G\ f) \star_D \eta\ (g \star_D h))$
$= (F\ (G\ f) \star_D \eta\ (g \star_D h))$
using *assms D.comp-inv-arr′ G_0-props D.comp-cod-arr* **by** *simp*
moreover have $((F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h)))$
$= F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h))$
proof −
have $(F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (F\ (G\ g) \star_D F\ (G\ h)))$
$= F\ (G\ f) \star_D F\ (G\ g) \star_D F\ (G\ h)$
proof −
have $(F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D \eta\ (F\ (G\ g) \star_D F\ (G\ h)))$
$= F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g) \star_D F\ (G\ h))) \cdot_D \eta\ (F\ (G\ g) \star_D F\ (G\ h))$
using *assms D.interchange* [*of* $F\ (G\ f)\ F\ (G\ f)$] **by** *simp*
also have ... $= F\ (G\ f) \star_D F\ (G\ g) \star_D F\ (G\ h)$
using *assms D.comp-inv-arr′* **by** *simp*
finally show *?thesis* **by** *blast*
qed
moreover have ... $= D.cod\ (F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h)))$
using *assms* **by** *simp*
moreover have $D.arr\ (F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h)))$
using *assms* **by** *simp*
ultimately show *?thesis*
using *D.comp-cod-arr* **by** *simp*
qed
moreover have $((F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
$= D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))$
proof −
have $(F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$
$(F\ (G\ f) \star_D F\ (C.inv\ (\varepsilon\ (G\ g \star_C G\ h))))$
$= F\ (G\ f) \star_D F\ (G\ g \star_C G\ h)$
proof −
have $(F\ (G\ f) \star_D D.inv\ (\eta\ (F\ (G\ g \star_C G\ h)))) \cdot_D$

```
        (F (G f) ⋆D F (C.inv (ε (G g ⋆C G h))))
          = F (G f) ⋆D D.inv (η (F (G g ⋆C G h))) ·D F (C.inv (ε (G g ⋆C G h)))
      using assms D.interchange [of F (G f) F (G f)] by simp
    also have ... = F (G f) ⋆D D.inv (η (F (G g ⋆C G h))) ·D
                              D.inv (F (ε (G g ⋆C G h)))
      using assms preserves-inv by simp
    also have ... = F (G f) ⋆D D.inv (F (ε (G g ⋆C G h)) ·D η (F (G g ⋆C G h)))
    proof −
      have D.iso (F (ε (G g ⋆C G h)) ·D η (F (G g ⋆C G h)))
        using assms by auto
      thus ?thesis
        using assms by (simp add: D.inv-comp-right(1))
    qed
    also have ... = F (G f) ⋆D F (G g ⋆C G h)
      using assms ηε.triangle-G Foε-ηoF.map-def [of G g ⋆C G h] by simp
    finally show ?thesis by blast
  qed
  thus ?thesis using assms D.comp-cod-arr by simp
 qed
 ultimately show ?thesis
  using D.comp-assoc by simp
qed
also have ... = (η (f ⋆D g ⋆D h) ·D
                (D.inv (η f) ⋆D D.inv (η g) ⋆D D.inv (η h))) ·D
                (F (G f) ⋆D D.inv (Φ (G g, G h))) ·D
                D.inv (Φ (G f, G g ⋆C G h))
proof −
 have ((D.inv (η f) ⋆D D.inv (η (g ⋆D h))) ·D
       (F (G f) ⋆D η (g ⋆D h))) ·D
       (F (G f) ⋆D D.inv (η g) ⋆D D.inv (η h))
         = D.inv (η f) ⋆D D.inv (η g) ⋆D D.inv (η h)
  using assms D.comp-cod-arr D.comp-arr-dom D.interchange
 proof −
  have ((D.inv (η f) ⋆D D.inv (η (g ⋆D h))) ·D
        (F (G f) ⋆D η (g ⋆D h))) ·D
        (F (G f) ⋆D D.inv (η g) ⋆D D.inv (η h))
          = (D.inv (η f) ·D F (G f) ⋆D D.inv (η (g ⋆D h)) ·D η (g ⋆D h)) ·D
            (F (G f) ⋆D D.inv (η g) ⋆D D.inv (η h))
    using assms D.interchange by simp
  also have ... = (D.inv (η f) ⋆D g ⋆D h) ·D (F (G f) ⋆D D.inv (η g) ⋆D D.inv (η h))
    using assms D.comp-inv-arr′ D.comp-arr-dom by simp
  also have ... = D.inv (η f) ·D F (G f) ⋆D (g ⋆D h) ·D (D.inv (η g) ⋆D D.inv (η h))
    using assms D.interchange by simp
  also have ... = D.inv (η f) ⋆D D.inv (η g) ⋆D D.inv (η h)
    using assms D.comp-arr-dom D.comp-cod-arr by simp
  finally show ?thesis by simp
 qed
 thus ?thesis
  using D.comp-assoc by simp
```

qed
finally show *?thesis* **by** *blast*
qed

interpretation G: *pseudofunctor*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G.map$
proof
show $\bigwedge f\ g\ h.\ [\![D.ide\ f;\ D.ide\ g;\ D.ide\ h;\ src_D\ f = trg_D\ g;\ src_D\ g = trg_D\ h]\!]$
$\Longrightarrow G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h) =$
$\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)) \cdot_C \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
proof −
fix $f\ g\ h$
assume f: $D.ide\ f$ **and** g: $D.ide\ g$ **and** h: $D.ide\ h$
assume fg: $src_D\ f = trg_D\ g$ **and** gh: $src_D\ g = trg_D\ h$
have $F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h)) =$
$F\ (\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)) \cdot_C \mathrm{a}_C[G\ f,\ G\ g,\ G\ h])$
proof −
have $F\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h))$
$= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$\mathrm{a}_D[F\ (G\ f),\ F\ (G\ g),\ F\ (G\ h)] \cdot_D$
$(D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h))$
using $f\ g\ h\ fg\ gh$ *coherence-LHS* **by** *simp*
also have ... $= (\eta\ (f \star_D g \star_D h) \cdot_D$
$(D.inv\ (\eta\ f) \star_D D.inv\ (\eta\ g) \star_D D.inv\ (\eta\ h))) \cdot_D$
$((F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))) \cdot_D$
$F\ \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
proof −
have $\mathrm{a}_D[F\ (G\ f),\ F\ (G\ g),\ F\ (G\ h)] \cdot_D (D.inv\ (\Phi\ (G\ f,\ G\ g)) \star_D F\ (G\ h)) \cdot_D$
$D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h))$
$= \mathrm{a}_D[F\ (G\ f),\ F\ (G\ g),\ F\ (G\ h)] \cdot_D D.inv\ (\Phi\ (G\ f \star_C G\ g,\ G\ h) \cdot_D$
$(\Phi\ (G\ f,\ G\ g) \star_D F\ (G\ h)))$
proof −
have $D.iso\ (\Phi\ (G\ f \star_C G\ g,\ G\ h) \cdot_D (\Phi\ (G\ f,\ G\ g) \star_D F\ (G\ h)))$
using $f\ g\ h\ fg\ gh$ $\Phi.components\text{-}are\text{-}iso\ C.VV.ide\text{-}char_{SbC}\ C.VV.arr\text{-}char_{SbC}$
FF-def
by (*intro D.iso-hcomp D.isos-compose D.seqI*) *auto*
thus *?thesis*
using $f\ g\ h\ fg\ gh$
by (*simp add*: $C.VV.arr\text{-}char_{SbC}$ *D.inv-comp-right*(*1*))
qed
also have ... $= D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h) \cdot_D (F\ (G\ f) \star_D \Phi\ (G\ g,\ G\ h))) \cdot_D$
$F\ \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
using $f\ g\ h\ fg\ gh$ *cmp-simps assoc-coherence D.comp-assoc D.isos-compose*
$\Phi.components\text{-}are\text{-}iso\ C.VV.ide\text{-}char_{SbC}\ C.VV.arr\text{-}char_{SbC}$ *FF-def*
D.invert-opposite-sides-of-square
by *simp*

also have ... = $((F\ (G\ f) \star_D D.inv\ (\Phi\ (G\ g,\ G\ h))) \cdot_D$
$D.inv\ (\Phi\ (G\ f,\ G\ g \star_C G\ h))) \cdot_D$
$F\ \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
proof −
have $D.iso\ (\Phi\ (G\ f,\ G\ g \star_C G\ h) \cdot_D (F\ (G\ f) \star_D \Phi\ (G\ g,\ G\ h)))$
using *f g h fg gh* Φ*.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def*
by *auto*
thus *?thesis*
using *f g h fg gh* Φ*.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *FF-def*
by (*simp add*: $C.VV.arr\text{-}char_{SbC}$ *D.comp-assoc D.inv-comp-right*(*1*))
qed
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $F\ (\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h))) \cdot_D$
$F\ \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
using *f g h fg gh coherence-RHS D.comp-assoc* **by** *simp*
also have ... = $F\ ((\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h))) \cdot_C$
$\mathrm{a}_C[G\ f,\ G\ g,\ G\ h])$
using *f g h fg gh* Φ_G*-simps* **by** *auto*
also have ... = $F\ (\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)) \cdot_C$
$\mathrm{a}_C[G\ f,\ G\ g,\ G\ h])$
using *C.comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
moreover have $C.par\ (G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C$
$(\Phi_G.map\ (f,\ g) \star_C G\ h))$
$(\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)) \cdot_C \mathrm{a}_C[G\ f,\ G\ g,\ G\ h])$
using *f g h fg gh* Φ_G*-simps* **by** *auto*
ultimately show $G\ \mathrm{a}_D[f,\ g,\ h] \cdot_C \Phi_G.map\ (f \star_D g,\ h) \cdot_C (\Phi_G.map\ (f,\ g) \star_C G\ h) =$
$\Phi_G.map\ (f,\ g \star_D h) \cdot_C (G\ f \star_C \Phi_G.map\ (g,\ h)) \cdot_C \mathrm{a}_C[G\ f,\ G\ g,\ G\ h]$
using *is-faithful* **by** *blast*
qed
qed

interpretation *GF*: *composite-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi\ G\ \Phi_G.map$
..
interpretation *FG*: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G.map\ F\ \Phi$
..
interpretation I_C: *identity-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ **..**
interpretation I_D: *identity-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ **..**

lemma η*-equals-FG-unit*:
assumes *D.obj a*
shows $\eta\ a = FG.unit\ a$

proof (*intro FG.unit-eqI*)
  show *D.obj a* **by** *fact*
  show *D.iso* (η *a*)
    using *assms* **by** *auto*
  show «η *a* : $FG.map_0$ *a* $\Rightarrow_D$ *FG.map a*»
    using *assms* η*-in-hom* [*of a*] $FG.map_0$*-def* G_0*-props D.obj-def*
    by (*metis D.ideD*(*2*) *D.ideD*(*3*) *D.objE D.vconn-implies-hpar*(*3*) η*.preserves-cod*
      η*-simps*(*5*))
  show η *a* $\cdot_D$ $\mathrm{i}_D[FG.map_0\ a] = (FG.map\ \mathrm{i}_D[a] \cdot_D FG.cmp\ (a, a)) \cdot_D (\eta\ a \star_D \eta\ a)$
  proof −
    have $(FG.map\ \mathrm{i}_D[a] \cdot_D FG.cmp\ (a, a)) \cdot_D (\eta\ a \star_D \eta\ a) =$
      $F\ (G\ \mathrm{i}_D[a]) \cdot_D FG.cmp\ (a, a) \cdot_D (\eta\ a \star_D \eta\ a)$
      using *assms D.comp-assoc* **by** *simp*
    also have ... $= F\ (G\ \mathrm{i}_D[a]) \cdot_D F\ (G\ (I_D.cmp\ (a, a))) \cdot_D F\ (\Phi_G.map\ (a, a)) \cdot_D$
      $\Phi\ (G\ a, G\ a) \cdot_D (\eta\ a \star_D \eta\ a)$
    using *assms FG.cmp-def D.comp-assoc* $D.VV.arr\text{-}char_{SbC}$ $D.VV.dom\text{-}char_{SbC}$ $D.VV.cod\text{-}char_{SbC}$
      by *auto*
    also have ... $= F\ (G\ \mathrm{i}_D[a]) \cdot_D F\ (G\ (I_D.cmp\ (a, a))) \cdot_D \eta\ (a \star_D a)$
      using *assms* η*-hcomp* **by** *auto*
    also have ... $= F\ (G\ \mathrm{i}_D[a]) \cdot_D \eta\ (a \star_D a)$
      using *assms D.comp-cod-arr* **by** *auto*
    also have ... $= \eta\ a \cdot_D \mathrm{i}_D[a]$
      using *assms* η*-naturality* [*of* $\mathrm{i}_D[a]$] **by** *simp*
    also have ... $= \eta\ a \cdot_D \mathrm{i}_D[FG.map_0\ a]$
      using *assms* ‹«η *a* : $FG.map_0$ *a* $\Rightarrow_D$ *FG.map a*»› **by** *fastforce*
    finally show *?thesis* **by** *simp*
  qed
qed

lemma ε*-hcomp′*:
assumes *C.ide g* **and** *C.ide f* **and** $src_C\ g = trg_C\ f$
shows $\varepsilon\ (g \star_C f) \cdot_C GF.cmp\ (g, f) = \varepsilon\ g \star_C \varepsilon\ f$
proof −
  have $\varepsilon\ (g \star_C f) \cdot_C GF.cmp\ (g, f)$
    $= (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C C.inv\ (\Phi_G.map\ (F\ g, F\ f)) \cdot_C C.inv\ (G\ (\Phi\ (g, f))) \cdot_C$
      $G\ (F\ (g \star_C f)) \cdot_C G\ (\Phi\ (g, f)) \cdot_C \Phi_G.map\ (F\ g, F\ f)$
    using *assms* ε*-hcomp GF.cmp-def* $C.VV.arr\text{-}char_{SbC}$ *C.comp-cod-arr*
      C.comp-inv-arr′ C.comp-assoc $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
    by *simp*
  also have ... $= (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C C.inv\ (\Phi_G.map\ (F\ g, F\ f)) \cdot_C (C.inv\ (G\ (\Phi\ (g, f))) \cdot_C$
      $G\ (F\ (g \star_C f)) \cdot_C G\ (\Phi\ (g, f))) \cdot_C \Phi_G.map\ (F\ g, F\ f)$
    using *C.comp-assoc* **by** *simp*
  also have ... $= (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C C.inv\ (\Phi_G.map\ (F\ g, F\ f)) \cdot_C (C.inv\ (G\ (\Phi\ (g, f))) \cdot_C$
      $G\ (\Phi\ (g, f))) \cdot_C \Phi_G.map\ (F\ g, F\ f)$
    using *assms C.comp-ide-arr* [*of* $G\ (F\ (g \star_C f))\ G\ (\Phi\ (g, f))$]
      $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
    by *simp*
  also have ... $= (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C C.inv\ (\Phi_G.map\ (F\ g, F\ f)) \cdot_C \Phi_G.map\ (F\ g, F\ f)$
  proof −

```
    have C.inv (G (Φ (g, f))) ·C G (Φ (g, f)) = G (F g ⋆D F f)
      using assms C.comp-inv-arr′ cmp-components-are-iso C.inv-is-inverse
            C.VV.arr-char_SbC C.VV.ide-char_SbC cmp-simps(4)
      by auto
    moreover have ... = C.cod (Φ_G.map (F g, F f))
      using assms by simp
    ultimately have (C.inv (G (Φ (g, f))) ·C G (Φ (g, f))) ·C Φ_G.map (F g, F f) =
                    Φ_G.map (F g, F f)
      using assms C.comp-cod-arr [of Φ_G.map (F g, F f) G (F g ⋆D F f)]
            C.ideD(1) G.cmp-simps(1) preserves-ide preserves-src preserves-trg
      by presburger
    thus ?thesis by simp
  qed
  also have ... = ε g ⋆C ε f
    using assms C.comp-inv-arr′ C.comp-arr-dom [of ε g ⋆C ε f G (F g) ⋆C G (F f)]
    by simp
  finally show ?thesis by simp
qed

lemma ε-inverts-GF-unit:
assumes C.obj a
shows ε a ·C GF.unit a = a
proof −
  have ε a ·C GF.unit a = I_C.unit a
  proof (intro I_C.unit-eqI)
    show C.obj a by fact
    show 1: «ε a ·C GF.unit a : I_C.map_0 a ⇒C I_C.map a»
    proof −
      have src_C (G (F a)) = src_C (I_C.map a)
        using assms G_0-props C.obj-def′ by simp
      thus ?thesis
        using assms I_C.map_0-def GF.map_0-def GF.unit-in-hom
        by (intro C.comp-in-homI′) auto
    qed
    show C.iso (ε a ·C GF.unit a)
      using assms ε-in-hom GF.unit-char(2)
      by (intro C.isos-compose) auto
    show (ε a ·C GF.unit a) ·C i_C[I_C.map_0 a]
           = (I_C.map i_C[a] ·C I_C.cmp (a, a)) ·C (ε a ·C GF.unit a ⋆C ε a ·C GF.unit a)
    proof −
      have (I_C.map i_C[a] ·C I_C.cmp (a, a)) ·C (ε a ·C GF.unit a ⋆C ε a ·C GF.unit a) =
             i_C[a] ·C (a ⋆C a) ·C (ε a ·C GF.unit a ⋆C ε a ·C GF.unit a)
        using assms C.comp-assoc by simp
      also have ... = i_C[a] ·C (ε a ·C GF.unit a ⋆C ε a ·C GF.unit a)
      proof −
        have C.hseq (ε a ·C GF.unit a) (ε a ·C GF.unit a)
          using assms GF.unit-simps C.iso-is-arr ‹C.iso (ε a ·C GF.unit a)›
          by (intro C.hseqI′) auto
        moreover have a ⋆C a = C.cod (ε a ·C GF.unit a ⋆C ε a ·C GF.unit a)
```

proof −
have $C.cod\ (\varepsilon\ a \cdot_C GF.unit\ a) = a$
using *assms 1 C.obj-simps* **by** *auto*
moreover have $C.hseq\ (\varepsilon\ a \cdot_C GF.unit\ a)\ (\varepsilon\ a \cdot_C GF.unit\ a)$
using *assms 1 C.src-dom* [*of* $\varepsilon\ a \cdot_C GF.unit\ a$] *C.trg-dom* [*of* $\varepsilon\ a \cdot_C GF.unit\ a$]
apply (*intro C.hseqI′*)
by *auto force*
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis*
using *C.comp-cod-arr* **by** *simp*
qed
also have ... $= \mathrm{i}_C[a] \cdot_C (\varepsilon\ a \star_C \varepsilon\ a) \cdot_C (GF.unit\ a \star_C GF.unit\ a)$
using *assms C.interchange* [*of* $\varepsilon\ a\ GF.unit\ a\ \varepsilon\ a\ GF.unit\ a$]
by (*simp add*: *C.iso-is-arr* ‹$C.iso\ (\varepsilon\ a \cdot_C GF.unit\ a)$›)
also have ... $= \mathrm{i}_C[a] \cdot_C (\varepsilon\ (a \star_C a) \cdot_C GF.cmp\ (a, a)) \cdot_C (GF.unit\ a \star_C GF.unit\ a)$
using *assms ε-hcomp′* [*of a a*] **by** *auto*
also have ... $= (\mathrm{i}_C[a] \cdot_C \varepsilon\ (a \star_C a)) \cdot_C GF.cmp\ (a, a) \cdot_C (GF.unit\ a \star_C GF.unit\ a)$
using *C.comp-assoc* **by** *simp*
also have ... $= (\varepsilon\ a \cdot_C G\ (F\ \mathrm{i}_C[a])) \cdot_C GF.cmp\ (a, a) \cdot_C (GF.unit\ a \star_C GF.unit\ a)$
using *assms ε-naturality* [*of* $\mathrm{i}_C[a]$] **by** *simp*
also have ... $= \varepsilon\ a \cdot_C (G\ (F\ \mathrm{i}_C[a]) \cdot_C GF.cmp\ (a, a)) \cdot_C (GF.unit\ a \star_C GF.unit\ a)$
using *C.comp-assoc* **by** *simp*
also have ... $= \varepsilon\ a \cdot_C GF.unit\ a \cdot_C \mathrm{i}_C[GF.map_0\ a]$
using *assms GF.unit-char* [*of a*] **by** *simp*
also have ... $= (\varepsilon\ a \cdot_C GF.unit\ a) \cdot_C \mathrm{i}_C[I_C.map_0\ a]$
proof −
have $GF.map_0\ a = I_C.map_0\ a$
using *assms* G_0*-props*(*2*) [*of a*] $GF.map_0$*-def* **by** *auto*
thus *?thesis*
using *assms GF.unit-char* [*of a*] *C.comp-assoc* **by** *simp*
qed
finally show *?thesis*
using *C.comp-assoc* **by** *simp*
qed
qed
thus *?thesis*
using *assms* I_C*.unit-char′* **by** *simp*
qed

lemma *η-respects-comp*:
assumes $D.ide\ f$ **and** $D.ide\ g$ **and** $src_D\ g = trg_D\ f$
shows $(\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D \eta\ (g \star_D f) \cdot_D \mathrm{r}_D[g \star_D f]) \cdot_D ((g \star_D f) \star_D src_D\ f)$
$= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
and $(trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D \mathrm{a}_D[trg_D\ g, F\ (G\ g), F\ (G\ f)] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \cdot_D \eta\ g \cdot_D \mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D D.inv\ \mathrm{a}_D[g, src_D\ g, F\ (G\ f)] \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f]) \cdot_D \mathrm{a}_D[g, f, src_D\ f]$
$= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D$

$\mathrm{r}_D[g \star_D f]$
proof −
show $(\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D \eta\ (g \star_D f) \cdot_D \mathrm{r}_D[g \star_D f]) \cdot_D ((g \star_D f) \star_D src_D\ f)$
$= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
proof −
have $(\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D \eta\ (g \star_D f) \cdot_D \mathrm{r}_D[g \star_D f]) \cdot_D ((g \star_D f) \star_D src_D\ f)$
$= \mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D \eta\ (g \star_D f) \cdot_D \mathrm{r}_D[g \star_D f]$
using *assms D.comp-assoc D.comp-arr-dom* **by** *simp*
also have *1*: ... $= (\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D F\ (\Phi_G.map\ (g, f))) \cdot_D$
$\Phi\ (G\ g,\ G\ f) \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
using *assms η-hcomp D.comp-assoc* **by** *simp*
also have ... $= (trg_D\ g \star_D F\ (\Phi_G.map\ (g, f))) \cdot_D (\mathrm{l}_D{}^{-1}[F\ (G\ g \star_C G\ f)] \cdot_D$
$\Phi\ (G\ g,\ G\ f)) \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
proof −
have $\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D F\ (\Phi_G.map\ (g, f)) =$
$(trg_D\ g \star_D F\ (\Phi_G.map\ (g, f))) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g \star_C G\ f)]$
using *assms* G_0*-props D.lunit′-naturality* [*of F* $(\Phi_G.map\ (g, f))$] Φ_G*-in-hom* [*of g f*]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((trg_D\ g \star_D F\ (\Phi_G.map\ (g, f))) \cdot_D (trg_D\ g \star_D \Phi\ (G\ g,\ G\ f))) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
proof −
have $\mathrm{l}_D{}^{-1}[F\ (G\ g \star_C G\ f)] \cdot_D \Phi\ (G\ g,\ G\ f)$
$= (trg_D\ g \star_D \Phi\ (G\ g,\ G\ f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)]$
using *assms D.lunit′-naturality* [*of* $\Phi\ (G\ g,\ G\ f)$] G_0*-props* **by** *fastforce*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (trg_D\ g \star_D F\ (\Phi_G.map\ (g, f)) \cdot_D \Phi\ (G\ g,\ G\ f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
using *assms 1 D.whisker-left* [*of* $trg_D\ g\ F\ (\Phi_G.map\ (g, f))\ \Phi\ (G\ g,\ G\ f)$]
by *force*
also have ... $= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D (\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
proof −
have *1*: $FG.cmp\ (g, f) = (F\ (G\ (g \star_D f)) \cdot_D F\ (\Phi_G.map\ (g, f))) \cdot_D \Phi\ (G\ g,\ G\ f)$
using *assms FG.cmp-def* $D.VV.arr\text{-}char_{SbC}$ $D.VV.dom\text{-}char_{SbC}$ *D.comp-assoc* **by** *simp*
also have ... $= F\ (\Phi_G.map\ (g, f)) \cdot_D \Phi\ (G\ g,\ G\ f)$
proof −
have $D.cod\ (F\ (\Phi_G\ (g, f))) = F\ (G\ (g \star_D f))$
using *assms 1*
by (*metis* (*mono-tags*, *lifting*) *D.cod-eqI D.ideD*(*1*) *D.ide-hcomp D.seqE*
FG.cmp-simps′(*1*) *G.preserves-ide preserves-ide*)
thus *?thesis*
using *assms D.comp-cod-arr* [*of* $F\ (\Phi_G.map\ (g, f))\ F\ (G\ (g \star_D f))$]

by *fastforce*
qed
finally have $FG.cmp\ (g, f) = F\ (\Phi_G.map\ (g, f)) \cdot_D \Phi\ (G\ g,\ G\ f)$ **by** *simp*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
show $(trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[trg_D\ g,\ F\ (G\ g),\ F\ (G\ f)] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \cdot_D \eta\ g \cdot_D \mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)] \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f]) \cdot_D$
$\mathrm{a}_D[g,\ f,\ src_D\ f]$
$= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D$
$(\eta\ g \star_D \eta\ f) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
proof −
have $(trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[trg_D\ g,\ F\ (G\ g),\ F\ (G\ f)] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \cdot_D \eta\ g \cdot_D \mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)] \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f]) \cdot_D$
$\mathrm{a}_D[g,\ f,\ src_D\ f]$
$= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[trg_D\ g,\ F\ (G\ g),\ F\ (G\ f)] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \star_D F\ (G\ f)) \cdot_D$
$(\eta\ g \star_D F\ (G\ f)) \cdot_D$
$(\mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)] \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)]) \cdot_D$
$(g \star_D \eta\ f) \cdot_D$
$(g \star_D \mathrm{r}_D[f]) \cdot_D$
$\mathrm{a}_D[g,\ f,\ src_D\ f]$
using *assms D.comp-assoc D.whisker-right D.whisker-left* **by** *simp*
also have ... $= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$(\mathrm{a}_D[trg_D\ g,\ F\ (G\ g),\ F\ (G\ f)] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \star_D F\ (G\ f))) \cdot_D$
$(\eta\ g \star_D F\ (G\ f)) \cdot_D$
$(\mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)] \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)]) \cdot_D$
$(g \star_D \eta\ f) \cdot_D$
$((g \star_D \mathrm{r}_D[f]) \cdot_D$
$\mathrm{a}_D[g,\ f,\ src_D\ f])$
using *assms D.comp-cod-arr* $D.VV.ide\text{-}char_{SbC}$ $D.VV.arr\text{-}char_{SbC}$ $D.VV.dom\text{-}char_{SbC}$
FG.FF-def G_0*-props D.comp-assoc*
by *presburger*
also have ... $= (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$

$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D$
$(\eta\ g \star_D F\ (G\ f)) \cdot_D$
$((\mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)]) \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)]) \cdot_D$
$(g \star_D \eta\ f) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
proof −
have $(g \star_D \mathrm{r}_D[f]) \cdot_D \mathrm{a}_D[g,\ f,\ src_D\ f] = \mathrm{r}_D[g \star_D f]$
using *assms D.runit-hcomp* **by** *simp*
moreover have $\mathrm{a}_D[trg_D\ g,\ F\ (G\ g),\ F\ (G\ f)] \cdot_D (\mathrm{l}_D{}^{-1}[F\ (G\ g)] \star_D F\ (G\ f)) =$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)]$
using *assms D.lunit-hcomp* [*of* $F\ (G\ g)\ F\ (G\ f)$] G_0*-props* **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (trg_D\ g \star_D FG.cmp\ (g,\ f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D$
$(\eta\ g \star_D F\ (G\ f)) \cdot_D$
$(((g \star_D \mathrm{l}_D[F\ (G\ f)]) \cdot_D$
$(g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)])) \cdot_D$
$(g \star_D \eta\ f)) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
proof −
have $(\mathrm{r}_D[g] \star_D F\ (G\ f)) \cdot_D D.inv\ \mathrm{a}_D[g,\ src_D\ g,\ F\ (G\ f)] = g \star_D \mathrm{l}_D[F\ (G\ f)]$
using *assms D.triangle′* [*of* $g\ F\ (G\ f)$] G_0*-props* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (trg_D\ g \star_D FG.cmp\ (g,\ f)) \cdot_D$
$\mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D$
$((\eta\ g \star_D F\ (G\ f)) \cdot_D$
$(g \star_D \eta\ f)) \cdot_D$
$\mathrm{r}_D[g \star_D f]$
proof −
have $((g \star_D \mathrm{l}_D[F\ (G\ f)]) \cdot_D (g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)])) \cdot_D (g \star_D \eta\ f) = g \star_D \eta\ f$
using *assms D.interchange* [*of* $g\ g\ \mathrm{l}_D[F\ (G\ f)]\ \mathrm{l}_D{}^{-1}[F\ (G\ f)]$]
D.comp-arr-inv′ D.comp-cod-arr
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (trg_D\ g \star_D FG.cmp\ (g,\ f)) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ g) \star_D F\ (G\ f)] \cdot_D$
$(\eta\ g \star_D \eta\ f) \cdot_D \mathrm{r}_D[g \star_D f]$
using *assms D.interchange* [*of* $\eta\ g\ g\ F\ (G\ f)\ \eta\ f$] *D.comp-arr-dom D.comp-cod-arr*
by *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *η-respects-unit*:
assumes $D.obj\ a$
shows $(a \star_D FG.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[a] \cdot_D \mathrm{l}_D[a] =$
$(\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ a)] \cdot_D \eta\ a \cdot_D \mathrm{r}_D[D.dom\ a]) \cdot_D (I_D.unit\ a \star_D a)$
proof −
have $(\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ a)] \cdot_D \eta\ a \cdot_D \mathrm{r}_D[D.dom\ a]) \cdot_D (I_D.unit\ a \star_D a) =$
$(\mathrm{l}_D{}^{-1}[F\ (G\ a)] \cdot_D \eta\ a) \cdot_D \mathrm{r}_D[a]$
using *assms* $I_D.lunit\text{-}coherence\ I_D.unit\text{-}char'\ D.comp\text{-}arr\text{-}dom\ D.comp\text{-}assoc$ **by** *auto*
also have ... $= ((a \star_D \eta\ a) \cdot_D \mathrm{l}_D{}^{-1}[a]) \cdot_D \mathrm{r}_D[a]$
using *assms* $D.lunit'\text{-}naturality\ [of\ \eta\ a]$ **by** *auto*
also have ... $= (a \star_D \eta\ a) \cdot_D \mathrm{l}_D{}^{-1}[a] \cdot_D \mathrm{r}_D[a]$
using $D.comp\text{-}assoc$ **by** *simp*
also have ... $= (a \star_D \eta\ a) \cdot_D \mathrm{r}_D{}^{-1}[a] \cdot_D \mathrm{l}_D[a]$
using *assms* $D.unitor\text{-}coincidence$ **by** *simp*
also have ... $= (a \star_D FG.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[a] \cdot_D \mathrm{l}_D[a]$
using *assms* $\eta\text{-}equals\text{-}FG\text{-}unit$ **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *ε-respects-comp*:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $src_C\ g = trg_C\ f$
shows $(trg_C\ g \star_C g \star_C f) \cdot_C \mathrm{a}_C[trg_C\ g, g, f] \cdot_C (\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g), src_C\ g, f] \cdot_C (G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g), G\ (F\ f), src_C\ f]$
$= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
and $(\mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C \varepsilon\ (g \star_C f) \cdot_C \mathrm{r}_C[G\ (F\ (g \star_C f))]) \cdot_C (GF.cmp\ (g, f) \star_C src_C\ f)$
$= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
proof −
have $(trg_C\ g \star_C g \star_C f) \cdot_C$
$\mathrm{a}_C[trg_C\ g, g, f] \cdot_C$
$(\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g), src_C\ g, f] \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g), G\ (F\ f), src_C\ f]$
$= ((trg_C\ g \star_C g \star_C f) \cdot_C$
$\mathrm{a}_C[trg_C\ g, g, f]) \cdot_C$
$(\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g), src_C\ g, f] \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g), G\ (F\ f), src_C\ f]$
using *assms* $C.comp\text{-}assoc$ **by** *simp*
also have ... $= \mathrm{a}_C[trg_C\ g, g, f] \cdot_C$
$(\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g), src_C\ g, f] \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g), G\ (F\ f), src_C\ f]$
using *assms* $C.comp\text{-}cod\text{-}arr$ **by** *simp*
also have ... $= (\mathrm{a}_C[trg_C\ g, g, f] \cdot_C$

$(\mathrm{l}_C{}^{-1}[g] \star_C f)) \cdot_C$
$(\varepsilon\ g \star_C f) \cdot_C$
$(\mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g),\ src_C\ g,\ f] \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f]) \cdot_C$
$(G\ (F\ g) \star_C \varepsilon\ f) \cdot_C$
$((G\ (F\ g) \star_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g),\ G\ (F\ f),\ src_C\ f])$
using *assms C.whisker-left C.whisker-right C.comp-assoc* **by** *simp*
also have ... $= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C$
$(\varepsilon\ g \star_C f) \cdot_C$
$((\mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g),\ src_C\ g,\ f]) \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f]) \cdot_C$
$(G\ (F\ g) \star_C \varepsilon\ f) \cdot_C$
$\mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
using *assms* G_0*-props C.lunit-hcomp C.runit-hcomp C.comp-assoc* **by** *simp*
also have ... $= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C$
$(\varepsilon\ g \star_C f) \cdot_C$
$(((G\ (F\ g) \star_C \mathrm{l}_C[f]) \cdot_C$
$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f])) \cdot_C$
$(G\ (F\ g) \star_C \varepsilon\ f)) \cdot_C$
$\mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
using *assms* G_0*-props C.triangle′ C.comp-assoc* **by** *simp*
also have ... $= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C$
$((\varepsilon\ g \star_C f) \cdot_C$
$(G\ (F\ g) \star_C \varepsilon\ f)) \cdot_C$
$\mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
proof −
have $(G\ (F\ g) \star_C \mathrm{l}_C[f]) \cdot_C (G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f]) = G\ (F\ g) \star_C f$
using *assms C.whisker-left* [*of* $G\ (F\ g)\ \mathrm{l}_C[f]\ \mathrm{l}_C{}^{-1}[f]$] *C.comp-arr-inv′*
by *simp*
moreover have ... $= C.cod\ (G\ (F\ g) \star_C \varepsilon\ f)$
using *assms* G_0*-props* **by** *auto*
moreover have $C.hseq\ (G\ (F\ g))\ (\varepsilon\ f)$
using *assms* G_0*-props* **by** *simp*
ultimately have $((G\ (F\ g) \star_C \mathrm{l}_C[f]) \cdot_C (G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f])) \cdot_C (G\ (F\ g) \star_C \varepsilon\ f) =$
$(G\ (F\ g) \star_C \varepsilon\ f)$
using *assms* G_0*-props C.comp-cod-arr* **by** *presburger*
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
using *assms C.interchange* [*of* $\varepsilon\ g\ G\ (F\ g)\ f\ \varepsilon\ f$] *C.comp-cod-arr C.comp-arr-dom*
by *simp*
finally show $(trg_C\ g \star_C g \star_C f) \cdot_C$
$\mathrm{a}_C[trg_C\ g,\ g,\ f] \cdot_C$
$(\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C f) \cdot_C$
$C.inv\ \mathrm{a}_C[G\ (F\ g),\ src_C\ g,\ f] \cdot_C$

$(G\ (F\ g) \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[G\ (F\ g),\ G\ (F\ f),\ src_C\ f]$
$= \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
by *simp*
show $(\mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C \varepsilon\ (g \star_C f) \cdot_C \mathrm{r}_C[G\ (F\ (g \star_C f))]) \cdot_C (GF.cmp\ (g, f) \star_C src_C\ f)$
$= \ldots$
proof −
have $(\mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C \varepsilon\ (g \star_C f) \cdot_C \mathrm{r}_C[G\ (F\ (g \star_C f))]) \cdot_C (GF.cmp\ (g, f) \star_C src_C\ f) =$
$\mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C \varepsilon\ (g \star_C f) \cdot_C \mathrm{r}_C[G\ (F\ (g \star_C f))] \cdot_C (GF.cmp\ (g, f) \star_C src_C\ f)$
using *assms C.comp-assoc* **by** *simp*
also have $\ldots = \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ (g \star_C f) \cdot_C GF.cmp\ (g, f)) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
proof −
have $src_C\ (GF.cmp\ (g, f)) = src_C\ f$
using *assms* G_0*-props* **by** *simp*
hence $\mathrm{r}_C[G\ (F\ (g \star_C f))] \cdot_C (GF.cmp\ (g, f) \star_C src_C\ f) =$
$GF.cmp\ (g, f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
using *assms C.runit-naturality* [*of* $GF.cmp\ (g, f)$] $C.VV.arr\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
GF.cmp-simps′(*1*,*4*−*5*)
by *simp*
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have $\ldots = \mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C (\varepsilon\ g \star_C \varepsilon\ f) \cdot_C \mathrm{r}_C[G\ (F\ g) \star_C G\ (F\ f)]$
using *assms* ε*-hcomp′* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed

lemma ε*-respects-unit*:
assumes *C.obj a*
shows $(a \star_C I_C.unit\ a) \cdot_C \mathrm{r}_C{}^{-1}[a] \cdot_C \mathrm{l}_C[a] =$
$(\mathrm{l}_C{}^{-1}[C.cod\ a] \cdot_C \varepsilon\ a \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ a)]) \cdot_C (GF.unit\ a \star_C a)$
proof −
have $(\mathrm{l}_C{}^{-1}[C.cod\ a] \cdot_C \varepsilon\ a \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ a)]) \cdot_C (GF.unit\ a \star_C a) =$
$\mathrm{l}_C{}^{-1}[a] \cdot_C \varepsilon\ a \cdot_C \mathrm{r}_C[G\ (F\ a)] \cdot_C (GF.unit\ a \star_C a)$
using *assms C.comp-assoc* **by** *auto*
also have $\ldots = (\mathrm{l}_C{}^{-1}[a] \cdot_C \varepsilon\ a) \cdot_C GF.unit\ a \cdot_C \mathrm{r}_C[a]$
proof −
have $src_C\ (GF.unit\ a) = a$
using *assms GF.unit-simps*(*2*) $GF.map_0$*-def* [*of a*] G_0*-props*
by (*simp add*: *C.obj-simps*(*1*−*2*))
thus *?thesis*
using *assms C.runit-naturality* [*of GF.unit a*] *C.comp-assoc* **by** *simp*
qed
also have $\ldots = (a \star_C \varepsilon\ a) \cdot_C (\mathrm{l}_C{}^{-1}[G\ (F\ a)] \cdot_C GF.unit\ a) \cdot_C \mathrm{r}_C[a]$
proof −
have $\mathrm{l}_C{}^{-1}[a] \cdot_C \varepsilon\ a = (a \star_C \varepsilon\ a) \cdot_C \mathrm{l}_C{}^{-1}[G\ (F\ a)]$
using *assms C.lunit′-naturality* [*of* ε *a*] **by** *auto*
thus *?thesis*

using *C.comp-assoc* **by** *simp*
qed
also have ... = $((a \star_C \varepsilon\ a) \cdot_C (a \star_C GF.unit\ a)) \cdot_C \mathrm{l}_C{}^{-1}[a] \cdot_C \mathrm{r}_C[a]$
proof −
have $\mathrm{l}_C{}^{-1}[G\ (F\ a)] \cdot_C GF.unit\ a = (a \star_C GF.unit\ a) \cdot_C \mathrm{l}_C{}^{-1}[a]$
using *assms C.lunit′-naturality* [*of GF.unit a*] *G*$_0$*-props C.obj-simps*
by (*simp add*: *GF.map*$_0$*-def*)
thus *?thesis*
using *C.comp-assoc* **by** *simp*
qed
also have ... = $(a \star_C \varepsilon\ a \cdot_C GF.unit\ a) \cdot_C \mathrm{l}_C{}^{-1}[a] \cdot_C \mathrm{r}_C[a]$
using *assms C.interchange* [*of a a ε a GF.unit a*] **by** *force*
also have ... = $(a \star_C \varepsilon\ a \cdot_C GF.unit\ a) \cdot_C \mathrm{r}_C{}^{-1}[a] \cdot_C \mathrm{l}_C[a]$
using *assms C.unitor-coincidence* **by** *simp*
also have ... = $(a \star_C I_C.unit\ a) \cdot_C \mathrm{r}_C{}^{-1}[a] \cdot_C \mathrm{l}_C[a]$
using *assms ε-inverts-GF-unit* I_C*.unit-char′* **by** *simp*
finally show *?thesis* **by** *simp*
qed

abbreviation *counit*$_0$*′*
where *counit*$_0$*′* $\equiv \lambda b.\ b$

abbreviation *counit*$_1$*′*
where *counit*$_1$*′* $\equiv \lambda g.\ \mathrm{l}_D{}^{-1}[F\ (G\ g)] \cdot_D \eta\ g \cdot_D \mathrm{r}_D[g]$

interpretation *ε*: *pseudonatural-equivalence*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
FG.map FG.cmp I_D*.map* I_D*.cmp counit*$_0$*′ counit*$_1$*′*
proof
show $\bigwedge a.\ D.obj\ a \Longrightarrow D.ide\ a$
by *auto*
show $\bigwedge f.\ D.ide\ f \Longrightarrow D.iso\ (\mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f])$
using *D.iso-lunit′ D.iso-runit η-in-hom*
by (*intro D.isos-compose D.seqI*) *auto*
show $\bigwedge a.\ D.obj\ a \Longrightarrow$ «$a : src_D\ (FG.map\ a) \rightarrow_D src_D\ (I_D.map\ a)$»
using *D.obj-def G*$_0$*-props*(*1*) **by** *auto*
show $\bigwedge f.\ D.ide\ f \Longrightarrow$ «$\mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f] :$
$I_D.map\ f \star_D src_D\ f \Rightarrow_D trg_D\ f \star_D FG.map\ f$»
using *G*$_0$*-props*
by (*intro D.comp-in-homI′*) *auto*
show $\bigwedge \mu.\ D.arr\ \mu \Longrightarrow$
$(\mathrm{l}_D{}^{-1}[F\ (G\ (D.cod\ \mu))] \cdot_D \eta\ (D.cod\ \mu) \cdot_D \mathrm{r}_D[D.cod\ \mu]) \cdot_D (I_D.map\ \mu \star_D src_D\ \mu)$
$= (trg_D\ \mu \star_D FG.map\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ (D.dom\ \mu))] \cdot_D \eta\ (D.dom\ \mu) \cdot_D \mathrm{r}_D[D.dom\ \mu]$
proof −
fix μ
assume μ: $D.arr\ \mu$

have $(\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D \eta\ (D.cod\ \mu) \cdot_D \mathrm{r}_D[D.cod\ \mu]) \cdot_D$
$(I_D.map\ \mu \star_D src_D\ \mu)$
$= \mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D \eta\ \mu \cdot_D \mathrm{r}_D[D.dom\ \mu]$
proof −
have $(\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D \eta\ (D.cod\ \mu) \cdot_D \mathrm{r}_D[D.cod\ \mu]) \cdot_D$
$(I_D.map\ \mu \star_D src_D\ \mu)$
$= \mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D (\eta\ (D.cod\ \mu) \cdot_D \mu) \cdot_D \mathrm{r}_D[D.dom\ \mu]$
using μ *D.runit-naturality D.comp-assoc* **by** *simp*
also have ... $= \mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D \eta\ \mu \cdot_D \mathrm{r}_D[D.dom\ \mu]$
using μ *η-naturality(1)* **by** *simp*
finally show *?thesis* **by** *blast*
qed
also have ... $= (trg_D\ \mu \star_D FG.map\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[FG.map\ (D.dom\ \mu)] \cdot_D \eta\ (D.dom\ \mu) \cdot_D$
$\mathrm{r}_D[D.dom\ \mu]$
proof −
have $\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D \eta\ \mu \cdot_D \mathrm{r}_D[D.dom\ \mu] =$
$\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D (FG.map\ \mu \cdot_D \eta\ (D.dom\ \mu)) \cdot_D \mathrm{r}_D[D.dom\ \mu]$
using μ *η-naturality(2) D.comp-assoc* **by** *simp*
also have ... $= (\mathrm{l}_D{}^{-1}[FG.map\ (D.cod\ \mu)] \cdot_D FG.map\ \mu) \cdot_D \eta\ (D.dom\ \mu) \cdot_D$
$\mathrm{r}_D[D.dom\ \mu]$
using *D.comp-assoc* **by** *simp*
also have ... $= ((trg_D\ \mu \star_D FG.map\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[FG.map\ (D.dom\ \mu)]) \cdot_D$
$\eta\ (D.dom\ \mu) \cdot_D \mathrm{r}_D[D.dom\ \mu]$
using μ *D.lunit′-naturality* [*of FG.map* μ] *G_0-props* **by** *simp*
also have ... $= (trg_D\ \mu \star_D FG.map\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[FG.map\ (D.dom\ \mu)] \cdot_D$
$\eta\ (D.dom\ \mu) \cdot_D \mathrm{r}_D[D.dom\ \mu]$
using *D.comp-assoc* **by** *simp*
finally show *?thesis* **by** *blast*
qed
finally show $(\mathrm{l}_D{}^{-1}[F\ (G\ (D.cod\ \mu))] \cdot_D \eta\ (D.cod\ \mu) \cdot_D \mathrm{r}_D[D.cod\ \mu]) \cdot_D$
$(I_D.map\ \mu \star_D src_D\ \mu)$
$= (trg_D\ \mu \star_D FG.map\ \mu) \cdot_D \mathrm{l}_D{}^{-1}[F\ (G\ (D.dom\ \mu))] \cdot_D \eta\ (D.dom\ \mu) \cdot_D$
$\mathrm{r}_D[D.dom\ \mu]$
by *simp*
qed
show $\bigwedge f\ g.\ [\![D.ide\ f;\ D.ide\ g;\ src_D\ g = trg_D\ f]\!]$
$\Longrightarrow (trg_D\ g \star_D FG.cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[trg_D\ g,\ FG.map\ g,\ FG.map\ f] \cdot_D$
$(\mathrm{l}_D{}^{-1}[F\ (G\ g)] \cdot_D \eta\ g \cdot_D \mathrm{r}_D[g] \star_D FG.map\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[I_D.map\ g,\ src_D\ g,\ FG.map\ f] \cdot_D$
$(I_D.map\ g \star_D \mathrm{l}_D{}^{-1}[F\ (G\ f)] \cdot_D \eta\ f \cdot_D \mathrm{r}_D[f]) \cdot_D$
$\mathrm{a}_D[I_D.map\ g,\ I_D.map\ f,\ src_D\ f]$
$= (\mathrm{l}_D{}^{-1}[F\ (G\ (g \star_D f))] \cdot_D \eta\ (g \star_D f) \cdot_D$
$\mathrm{r}_D[g \star_D f]) \cdot_D$
$(I_D.cmp\ (g, f) \star_D src_D\ f)$
using *η-respects-comp* **by** *simp*
show $\bigwedge a.\ D.obj\ a \Longrightarrow (a \star_D FG.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[a] \cdot_D \mathrm{l}_D[a] =$
$(\mathrm{l}_D{}^{-1}[F\ (G\ a)] \cdot_D \eta\ a \cdot_D \mathrm{r}_D[a]) \cdot_D (I_D.unit\ a \star_D a)$
using *η-respects-unit* **by** *auto*

show $\bigwedge a.\ D.obj\ a \Longrightarrow D.equivalence\text{-}map\ a$
using $D.obj\text{-}is\text{-}equivalence\text{-}map$ **by** *simp*
qed

abbreviation $unit_0{}'$
where $unit_0{}' \equiv \lambda a.\ a$

abbreviation $unit_1{}'$
where $unit_1{}' \equiv \lambda f.\ \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]$

interpretation η: *pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$I_C.map\ I_C.cmp\ GF.map\ GF.cmp\ unit_0{}'\ unit_1{}'$
proof
show $\bigwedge a.\ C.obj\ a \Longrightarrow C.ide\ a$
by *auto*
show $\bigwedge f.\ C.ide\ f \Longrightarrow C.iso\ (\mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)])$
using $C.iso\text{-}runit\ C.iso\text{-}lunit'$
by (*intro* $C.isos\text{-}compose$) *auto*
show $\bigwedge a.\ C.obj\ a \Longrightarrow$ «$a : src_C\ (I_C.map\ a) \rightarrow_C src_C\ (GF.map\ a)$»
by (*simp-all add*: $C.obj\text{-}simps(1-3)\ G_0\text{-}props(2)$)
show $\bigwedge f.\ C.ide\ f \Longrightarrow$ «$\mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)] :$
$GF.map\ f \star_C src_C\ f \Rightarrow_C trg_C\ f \star_C I_C.map\ f$»
using $G_0\text{-}props$ **by** *auto*
show $\bigwedge \mu.\ C.arr\ \mu \Longrightarrow$
$(\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ (C.cod\ \mu) \cdot_C \mathrm{r}_C[G\ (F\ (C.cod\ \mu))]) \cdot_C$
$(GF.map\ \mu \star_C src_C\ \mu)$
$= (trg_C\ \mu \star_C I_C.map\ \mu) \cdot_C \mathrm{l}_C{}^{-1}[C.dom\ \mu] \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C$
$\mathrm{r}_C[G\ (F\ (C.dom\ \mu))]$
proof −
fix μ
assume μ: $C.arr\ \mu$
have $(\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ (C.cod\ \mu) \cdot_C \mathrm{r}_C[GF.map\ (C.cod\ \mu)]) \cdot_C$
$(GF.map\ \mu \star_C src_C\ \mu)$
$= \mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ \mu \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
proof −
have $(\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ (C.cod\ \mu) \cdot_C \mathrm{r}_C[GF.map\ (C.cod\ \mu)]) \cdot_C$
$(GF.map\ \mu \star_C src_C\ \mu)$
$= \mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ (C.cod\ \mu) \cdot_C \mathrm{r}_C[GF.map\ (C.cod\ \mu)] \cdot_C$
$(GF.map\ \mu \star_C src_C\ \mu)$
using $C.comp\text{-}assoc$ **by** *simp*
also have ... $= \mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C (\varepsilon\ (C.cod\ \mu) \cdot_C GF.map\ \mu) \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using $\mu\ C.runit\text{-}naturality$ [*of* $GF.map\ \mu$] $G_0\text{-}props\ C.comp\text{-}assoc$ **by** *simp*
also have ... $= \mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ \mu \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using $\mu\ \varepsilon\text{-}naturality(1)$ [*of* μ] **by** *simp*
finally show *?thesis* **by** *blast*
qed
also have ... $= (trg_C\ \mu \star_C I_C.map\ \mu) \cdot_C \mathrm{l}_C{}^{-1}[C.dom\ \mu] \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C$
$\mathrm{r}_C[GF.map\ (C.dom\ \mu)]$

proof −
have ... = $\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C (\mu \cdot_C \varepsilon\ (C.dom\ \mu)) \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using μ *ε-naturality(2)* [*of* μ] **by** *simp*
also have ... = $(\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \mu) \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C \mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using *C.comp-assoc* **by** *simp*
also have ... = $((trg_C\ \mu \star_C I_C.map\ \mu) \cdot_C \mathrm{l}_C{}^{-1}[C.dom\ \mu]) \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C$
$\mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using μ *C.lunit′-naturality* [*of* μ] **by** *simp*
also have ... = $(trg_C\ \mu \star_C I_C.map\ \mu) \cdot_C \mathrm{l}_C{}^{-1}[C.dom\ \mu] \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C$
$\mathrm{r}_C[GF.map\ (C.dom\ \mu)]$
using *C.comp-assoc* **by** *simp*
finally show *?thesis* **by** *blast*
qed
finally show $(\mathrm{l}_C{}^{-1}[C.cod\ \mu] \cdot_C \varepsilon\ (C.cod\ \mu) \cdot_C \mathrm{r}_C[G\ (F\ (C.cod\ \mu))]) \cdot_C$
$(GF.map\ \mu \star_C src_C\ \mu)$
$= (trg_C\ \mu \star_C I_C.map\ \mu) \cdot_C \mathrm{l}_C{}^{-1}[C.dom\ \mu] \cdot_C \varepsilon\ (C.dom\ \mu) \cdot_C$
$\mathrm{r}_C[G\ (F\ (C.dom\ \mu))]$
by *simp*
qed
show $\bigwedge f\ g.\ [\![C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f]\!] \Longrightarrow$
$(trg_C\ g \star_C I_C.cmp\ (g, f)) \cdot_C$
$\mathrm{a}_C[trg_C\ g,\ I_C.map\ g,\ I_C.map\ f] \cdot_C$
$(\mathrm{l}_C{}^{-1}[g] \cdot_C \varepsilon\ g \cdot_C \mathrm{r}_C[G\ (F\ g)] \star_C I_C.map\ f) \cdot_C$
$C.inv\ \mathrm{a}_C[GF.map\ g,\ src_C\ g,\ I_C.map\ f] \cdot_C$
$(GF.map\ g \star_C \mathrm{l}_C{}^{-1}[f] \cdot_C \varepsilon\ f \cdot_C \mathrm{r}_C[G\ (F\ f)]) \cdot_C$
$\mathrm{a}_C[GF.map\ g,\ GF.map\ f,\ src_C\ f]$
$= (\mathrm{l}_C{}^{-1}[g \star_C f] \cdot_C \varepsilon\ (g \star_C f) \cdot_C \mathrm{r}_C[G\ (F\ (g \star_C f))]) \cdot_C$
$(GF.cmp\ (g, f) \star_C src_C\ f)$
using *ε-respects-comp* **by** *simp*
show $\bigwedge a.\ C.obj\ a \Longrightarrow (a \star_C I_C.unit\ a) \cdot_C \mathrm{r}_C{}^{-1}[a] \cdot_C \mathrm{l}_C[a] =$
$(\mathrm{l}_C{}^{-1}[a] \cdot_C \varepsilon\ a \cdot_C \mathrm{r}_C[G\ (F\ a)]) \cdot_C (GF.unit\ a \star_C a)$
using *ε-respects-unit* **by** *auto*
show $\bigwedge a.\ C.obj\ a \Longrightarrow C.equivalence\text{-}map\ a$
using *C.obj-is-equivalence-map* **by** *simp*
qed

interpretation *EQ*: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi\ G\ \Phi_G.map\ unit_0{}'\ unit_1{}'\ counit_0{}'\ counit_1{}'$
..

lemma *extends-to-equivalence-of-bicategories*:
shows *equivalence-of-bicategories* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi\ G\ \Phi_G.map\ unit_0{}'\ unit_1{}'\ counit_0{}'\ counit_1{}'$
..

end

1.16.7 Equivalence Pseudofunctors Extend to Equivalences of Bicategories

Now we put the pieces together and prove that an arbitrary equivalence pseudofunctor extends to an equivalence of bicategories.

context *equivalence-pseudofunctor*
begin

Define a set of objects U of C by choosing a representative of each equivalence class of objects having the same image under the object map of the given equivalence pseudofunctor. Then U is obviously dense, because every object of C belongs to such an equivalence class.

definition U
where $U = \{a.\ C.obj\ a \wedge a = (SOME\ a'.\ C.obj\ a' \wedge map_0\ a' = map_0\ a)\}$

lemma *U-dense*:
assumes $C.obj\ a$
shows $\exists\, a' \in U.\ C.equivalent\text{-}objects\ a\ a'$
proof −
  let $?a' = SOME\ a'.\ C.obj\ a' \wedge map_0\ a' = map_0\ a$
  have $\exists\, a'.\ C.obj\ a' \wedge map_0\ a' = map_0\ a$
    using *assms* **by** *auto*
  hence *1*: $?a' \in U \wedge map_0\ ?a' = map_0\ a$
    using *assms U-def someI-ex* [*of* $\lambda a'.\ C.obj\ a' \wedge map_0\ a' = map_0\ a$] **by** *simp*
  hence $C.equivalent\text{-}objects\ ?a'\ a$
    using *D.equivalent-objects-reflexive reflects-equivalent-objects* [*of* $?a'\ a$]
    by (*simp add*: *U-def assms*)
  thus *?thesis*
    using *1 C.equivalent-objects-symmetric* **by** *auto*
qed

Take V to be the collection of images of all objects of C under the given equivalence pseudofunctor. Since equivalence pseudofunctors are biessentially surjective on objects, V is dense. Moreover, by construction, the object map of the given equivalence pseudofunctor is a bijection from U to V.

definition V
where $V = map_0$ ‘ $Collect\ C.obj$

lemma *V-dense*:
assumes $D.obj\ b$
shows $\exists\, b'.\ b' \in map_0$ ‘ $Collect\ C.obj \wedge D.equivalent\text{-}objects\ b\ b'$
  using *assms biessentially-surjective-on-objects D.equivalent-objects-symmetric*
  by *blast*

lemma *bij-betw-U-V*:
shows $bij\text{-}betw\ map_0\ U\ V$
proof −
  have $inj\text{-}on\ map_0\ U$

```
    using U-def by (intro inj-onI) simp
  moreover have map₀ ‘ U = V
  proof
    show map₀ ‘ U ⊆ V
      using U-def V-def by auto
    show V ⊆ map₀ ‘ U
    proof
      fix b
      assume b: b ∈ V
      obtain a where a: C.obj a ∧ map₀ a = b
        using b V-def by auto
      let ?a′ = SOME a′. C.obj a′ ∧ map₀ a′ = map₀ a
      have 1: C.obj ?a′ ∧ map₀ ?a′ = b
        using a someI-ex [of λa′. C.obj a′ ∧ map₀ a′ = map₀ a] by auto
      moreover have ?a′ = (SOME a″. C.obj a″ ∧ map₀ a″ = map₀ ?a′)
        using a 1 by simp
      ultimately have ?a′ ∈ U
        unfolding U-def by simp
      thus b ∈ map₀ ‘ U
         using a 1 U-def
         by (metis (no-types, lifting) image-eqI)
    qed
  qed
  ultimately show ?thesis
    using bij-betw-def [of map₀ U V] by simp
qed

abbreviation (input) Arr_U
where Arr_U ≡ λμ. C.arr μ ∧ src_C μ ∈ U ∧ trg_C μ ∈ U

interpretation C_U: subbicategory V_C H_C a_C i_C src_C trg_C Arr_U
  using C.𝔩-ide-simp C.𝔯-ide-simp
  apply unfold-locales
            apply auto
   apply (metis C.comp-ide-self C.ide-src C.src-cod C.src-dom)
  by (metis C.trg.as-nat-trans.naturality2 C.trg.as-nat-trans.naturality C.trg-cod)

interpretation C_U: dense-subbicategory V_C H_C a_C i_C src_C trg_C U
proof
  show ⋀a. C.obj a ⟹ ∃ a′. a′ ∈ U ∧ C.equivalent-objects a′ a
    using U-dense C.equivalent-objects-symmetric by blast

qed

abbreviation (input) Arr_V
where Arr_V ≡ λμ. D.arr μ ∧ src_D μ ∈ V ∧ trg_D μ ∈ V

interpretation D_V: subbicategory V_D H_D a_D i_D src_D trg_D Arr_V
  using D.𝔩-ide-simp D.𝔯-ide-simp
```

apply *unfold-locales*
apply *auto*
apply (*metis D.comp-ide-self D.ide-src D.src-cod D.src-dom*)
by (*metis D.trg.as-nat-trans.naturality2 D.trg.as-nat-trans.naturality D.trg-cod*)

interpretation D_V: *dense-subbicategory* V_D H_D a_D i_D src_D trg_D V
using *V-dense D.equivalent-objects-def D.equivalent-objects-symmetric V-def*
by *unfold-locales metis*

interpretation F_U: *restricted-pseudofunctor*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ
‹$\lambda\mu.\ C.arr\ \mu \wedge src_C\ \mu \in U \wedge trg_C\ \mu \in U$›
..

interpretation F_{UV}: *corestricted-pseudofunctor*
$C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ i_C $C_U.src$ $C_U.trg$
V_D H_D a_D i_D src_D trg_D $F_U.map$ $F_U.cmp$ ‹$\lambda a.\ a \in V$›
proof
show $\bigwedge\mu.\ C_U.arr\ \mu \Longrightarrow D_V.arr\ (F_U.map\ \mu)$
proof −
fix μ
assume μ: $C_U.arr\ \mu$
have $src_C\ \mu \in U \wedge trg_C\ \mu \in U$
using μ $C_U.arr\text{-}char_{SbC}$ **by** *simp*
moreover have $\bigwedge a.\ a \in U \Longrightarrow F_U.map_0\ a = map_0\ a$
proof −
fix $a :: 'a$
assume a: $a \in U$
hence *1*: $C.obj\ a$
using *U-def* **by** *blast*
have $src_C\ a = a$
using *a U-def* **by** *blast*
thus $F_U.map_0\ a = map_0\ a$
using *a 1 C.obj-simps*(*1*,*3*) $C_U.arr\text{-}char_{SbC}$ $F_U.map_0\text{-}def$ $map_0\text{-}def$ **by** *presburger*
qed
ultimately have $F_U.map_0\ (src_C\ \mu) \in V \wedge F_U.map_0\ (trg_C\ \mu) \in V$
using μ *bij-betw-U-V bij-betw-def*
by (*metis* (*no-types*, *lifting*) *image-eqI*)
hence $src_D\ (F_U.map\ \mu) \in V \wedge trg_D\ (F_U.map\ \mu) \in V$
using μ $F_U.map_0\text{-}def$ $C_U.src\text{-}def$ $C_U.trg\text{-}def$ $F_U.preserves\text{-}src$ $F_U.preserves\text{-}trg$
by *auto*
thus $D_V.arr\ (F_U.map\ \mu)$
using μ $D_V.arr\text{-}char_{SbC}$ [*of* $F_U.map\ \mu$] **by** *blast*
qed
qed

interpretation F_{UV}: *equivalence-pseudofunctor*
$C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ i_C $C_U.src$ $C_U.trg$
$D_V.comp$ $D_V.hcomp$ $D_V.\mathrm{a}$ i_D $D_V.src$ $D_V.trg$

$F_{UV}.map\ F_{UV}.cmp$
proof
show $\bigwedge f\ f'.\ [\![C_U.par\ f\ f';\ F_U.map\ f = F_U.map\ f']\!] \Longrightarrow f = f'$
using $C_U.arr\text{-}char_{SbC}$ $C_U.dom\text{-}char_{SbC}$ $C_U.cod\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *is-faithful*)
show $\bigwedge a\ b\ g.\ [\![C_U.obj\ a;\ C_U.obj\ b;\ D_V.in\text{-}hhom\ g\ (F_{UV}.map_0\ a)\ (F_{UV}.map_0\ b);$
$D_V.ide\ g]\!]$
$\Longrightarrow \exists f.\ C_U.in\text{-}hhom\ f\ a\ b \wedge C_U.ide\ f \wedge D_V.isomorphic\ (F_U.map\ f)\ g$
proof −
fix $a\ b\ g$
assume a: $C_U.obj\ a$ **and** b: $C_U.obj\ b$
assume g: $D_V.in\text{-}hhom\ g\ (F_{UV}.map_0\ a)\ (F_{UV}.map_0\ b)$ **and** *ide-g*: $D_V.ide\ g$
have *1*: $\exists f.$ «$f : a \rightarrow_C b$» $\wedge$ $C.ide\ f \wedge D.isomorphic\ (F\ f)\ g$
proof −
have $C.obj\ a \wedge C.obj\ b$
using $a\ b\ C_U.obj\text{-}char$ **by** *simp*
moreover have «$g : map_0\ a \rightarrow_D map_0\ b$»
using $a\ b\ g\ D_V.in\text{-}hhom\text{-}def\ D_V.arr\text{-}char_{SbC}\ D_V.src\text{-}def\ D_V.trg\text{-}def$
by (*intro* $D.in\text{-}hhomI$) *auto*
moreover have $D.ide\ g$
using *ide-g* $D_V.ide\text{-}char_{SbC}\ D_V.arr\text{-}char_{SbC}$ **by** *simp*
ultimately show *?thesis*
using *locally-essentially-surjective* **by** *simp*
qed
obtain f **where** f: «$f : a \rightarrow_C b$» $\wedge$ $C.ide\ f \wedge D.isomorphic\ (F\ f)\ g$
using *1* **by** *blast*
have *2*: $C_U.in\text{-}hhom\ f\ a\ b$
using $f\ a\ b\ C_U.arr\text{-}char_{SbC}\ C_U.obj\text{-}char\ C_U.src\text{-}def\ C_U.trg\text{-}def$ **by** *fastforce*
moreover have $C_U.ide\ f$
using f *2* $C_U.ide\text{-}char_{SbC}\ C_U.arr\text{-}char_{SbC}$ **by** *auto*
moreover have $D_V.isomorphic\ (F_{UV}.map\ f)\ g$
proof −
obtain φ **where** φ: «$\varphi : F\ f \Rightarrow_D g$» $\wedge$ $D.iso\ \varphi$
using f $D.isomorphic\text{-}def$ **by** *auto*
have *3*: $D_V.in\text{-}hom\ \varphi\ (F_{UV}.map\ f)\ g$
proof −
have $D_V.arr\ \varphi$
using $f\ g\ \varphi$ *2* $D_V.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) *D.arrI D.vconn-implies-hpar*(*1*−*4*) $D_V.ideD(1)$ *ide-g*)
thus *?thesis*
using φ *2* $D_V.dom\text{-}char_{SbC}\ D_V.cod\text{-}char_{SbC}$
by (*intro* $D_V.in\text{-}homI$) *auto*
qed
moreover have $D_V.iso\ \varphi$
using φ $D_V.iso\text{-}char_{SbC}\ D_V.arr\text{-}char_{SbC}$ *3* **by** *fastforce*
ultimately show *?thesis*
using $D_V.isomorphic\text{-}def$ **by** *auto*
qed
ultimately show $\exists f.\ C_U.in\text{-}hhom\ f\ a\ b \wedge C_U.ide\ f \wedge D_V.isomorphic\ (F_{UV}.map\ f)\ g$

by *auto*
qed
show $\bigwedge f\, f'\, \nu.\ [\![C_U.ide\ f;\ C_U.ide\ f';\ C_U.src\ f = C_U.src\ f';\ C_U.trg\ f = C_U.trg\ f';$
$D_V.in\text{-}hom\ \nu\ (F_U.map\ f)\ (F_U.map\ f')]\!]$
$\Longrightarrow \exists\, \mu.\ C_U.in\text{-}hom\ \mu\ f\, f' \wedge F_U.map\ \mu = \nu$
proof −
fix $f\, f'\, \nu$
assume *f*: $C_U.ide\ f$ **and** *f'*: $C_U.ide\ f'$
and *eq-src*: $C_U.src\ f = C_U.src\ f'$ **and** *eq-trg*: $C_U.trg\ f = C_U.trg\ f'$
and ν: $D_V.in\text{-}hom\ \nu\ (F_U.map\ f)\ (F_U.map\ f')$
have *1*: $C.ide\ f \wedge C.ide\ f'$
using $f\, f'$ $C_U.ide\text{-}char_{SbC}$ $C_U.arr\text{-}char_{SbC}$ **by** *simp*
have *2*: $\exists\, \mu.$ «$\mu : f \Rightarrow_C f'$» $\wedge\ F\ \mu = \nu$
proof −
have $src_C\ f = src_C\ f' \wedge trg_C\ f = trg_C\ f'$
using $f\, f'$ *1 eq-src eq-trg* $C_U.src\text{-}def$ $C_U.trg\text{-}def$ **by** *simp*
moreover have «$\nu : F\ f \Rightarrow_D F\ f'$»
using ν $D_V.in\text{-}hom\text{-}char_{SbC}$ $D_V.arr\text{-}char_{SbC}$ **by** *auto*
ultimately show *?thesis*
using *1 locally-full* **by** *simp*
qed
obtain μ **where** μ: «$\mu : f \Rightarrow_C f'$» $\wedge\ F\ \mu = \nu$
using *2* **by** *auto*
have *3*: $C_U.in\text{-}hom\ \mu\ f\, f'$
using $\mu\ f\, f'$ $C_U.in\text{-}hom\text{-}char_{SbC}$ $C_U.ide\text{-}char_{SbC}$ $C_U.arr\text{-}char_{SbC}$ **by** *auto*
moreover have $F_U.map\ \mu = \nu$
using μ *3* **by** *auto*
ultimately show $\exists\, \mu.\ C_U.in\text{-}hom\ \mu\ f\, f' \wedge F_U.map\ \mu = \nu$
by *auto*
qed
show $\bigwedge b.\ D_V.obj\ b \Longrightarrow \exists\, a.\ C_U.obj\ a \wedge D_V.equivalent\text{-}objects\ (F_{UV}.map_0\ a)\ b$
proof −
fix *b*
assume *b*: $D_V.obj\ b$
obtain *a* **where** *a*: $C.obj\ a \wedge map_0\ a = b$
using *b* $D_V.obj\text{-}char$ $D_V.arr\text{-}char_{SbC}$ *V-def* **by** *auto*
have *1*: $D.obj\ b \wedge b \in V$
using *a b* $D_V.obj\text{-}char$ *V-def* **by** *auto*
obtain *a'* **where** *a'*: $a' \in U \wedge C.equivalent\text{-}objects\ a'\ a$
using *a U-dense* $C.equivalent\text{-}objects\text{-}symmetric$ **by** *blast*
have *obj-a'*: $C_U.obj\ a'$
using *a' U-def* $C_U.obj\text{-}char$ $C_U.arr\text{-}char_{SbC}$ **by** *fastforce*
moreover have $D_V.equivalent\text{-}objects\ (F_{UV}.map_0\ a')\ b$
proof −
have $D.equivalent\text{-}objects\ (map_0\ a')\ (map_0\ a)$
using *a' preserves-equivalent-objects* **by** *simp*
hence *2*: $D.equivalent\text{-}objects\ (map_0\ a')\ b$
using *a* $D.equivalent\text{-}objects\text{-}transitive$ **by** *simp*
obtain *e* **where** *e*: «$e : map_0\ a' \rightarrow_D b$» $\wedge\ D.equivalence\text{-}map\ e$

using *2 D.equivalent-objects-def* **by** *auto*
have *3*: $D.obj\ (map_0\ a') \wedge map_0\ a' \in V$
using a' *e U-def V-def* **by** *simp*
moreover have *e-in-hhom*: $D_V.in\text{-}hhom\ e\ (F_{UV}.map_0\ a')\ b$
proof
show *4*: $D_V.arr\ e$
using *1 3 a e b* $D_V.arr\text{-}char_{SbC}$ $D_V.obj\text{-}char$ *V-def*
by (*metis* (*no-types*, *lifting*) *D.in-hhomE*)
show $D_V.src\ e = F_{UV}.map_0\ a'$
using *e 4* $D_V.src\text{-}def$ $F_{UV}.map_0\text{-}def$ *obj-a'* $map_0\text{-}def$ **by** *auto*
show $D_V.trg\ e = b$
using *e 4* $D_V.trg\text{-}def$ **by** *auto*
qed
moreover have $D_V.equivalence\text{-}map\ e$
proof −
obtain $d\ \eta\ \varepsilon$ **where** *d*: *equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ e\ d\ \eta\ \varepsilon$
using *e D.equivalence-map-def* **by** *auto*
interpret *e*: *equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ e\ d\ \eta\ \varepsilon$
using *d* **by** *simp*
have *equivalence-in-bicategory* $D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg\ e\ d\ \eta\ \varepsilon$
proof
show *ide-e*: $D_V.ide\ e$
using *e e-in-hhom* $D_V.ide\text{-}char_{SbC}$ **by** *auto*
show *ide-d*: $D_V.ide\ d$
using *d ide-e e.antipar* $D_V.ide\text{-}char_{SbC}$ $D_V.arr\text{-}char_{SbC}$ **by** *simp*
show *4*: $D_V.in\text{-}hom\ \eta\ (D_V.src\ e)\ (D_V.hcomp\ d\ e)$
proof −
have $D_V.hseq\ d\ e$
using *ide-d ide-e e.antipar* $D_V.src\text{-}def$ $D_V.trg\text{-}def$ **by** *simp*
thus *?thesis*
using *ide-d ide-e e.unit-in-hom* $D_V.arr\text{-}char_{SbC}$ $D_V.ide\text{-}char_{SbC}$
$D_V.dom\text{-}char_{SbC}$ $D_V.cod\text{-}char_{SbC}$ $D_V.src\text{-}def$ $D_V.trg\text{-}def$
by (*intro* $D_V.in\text{-}homI$) *auto*
qed
show *5*: $D_V.in\text{-}hom\ \varepsilon\ (D_V.hcomp\ e\ d)\ (D_V.src\ d)$
proof −
have $D_V.hseq\ e\ d$
using *ide-d ide-e e.antipar* $D_V.src\text{-}def$ $D_V.trg\text{-}def$ **by** *simp*
thus *?thesis*
using *ide-d ide-e e.antipar e.counit-in-hom* $D_V.arr\text{-}char_{SbC}$ $D_V.ide\text{-}char_{SbC}$
$D_V.dom\text{-}char_{SbC}$ $D_V.cod\text{-}char_{SbC}$ $D_V.src\text{-}def$ $D_V.trg\text{-}def$
by (*intro* $D_V.in\text{-}homI$) *auto*
qed
show $D_V.iso\ \eta$
using *4* $D_V.iso\text{-}char_{SbC}$ $D_V.arr\text{-}char_{SbC}$ **by** *fastforce*
show $D_V.iso\ \varepsilon$
using *5* $D_V.iso\text{-}char_{SbC}$ $D_V.arr\text{-}char_{SbC}$ **by** *fastforce*
qed
thus *?thesis*

using D_V.*equivalence-map-def* **by** *auto*
qed
ultimately show *?thesis*
using D_V.*equivalent-objects-def* **by** *auto*
qed
ultimately show $\exists\, a.\ C_U.obj\ a \wedge D_V.equivalent\text{-}objects\ (F_{UV}.map_0\ a)\ b$ **by** *auto*
qed
qed

interpretation F_{UV}: *equivalence-pseudofunctor-bij-on-obj*
$C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$F_{UV}.map\ F_{UV}.cmp$
proof
have $Collect\ C_U.obj = U$
using $C_U.obj\text{-}char\ C_U.arr\text{-}char_{SbC}\ U\text{-}def$ **by** *fastforce*
moreover have $Collect\ D_V.obj = V$
using $D_V.obj\text{-}char\ D_V.arr\text{-}char_{SbC}\ V\text{-}def\ D.obj\text{-}def'\ map_0\text{-}simps(1)$ **by** *auto*
moreover have $\bigwedge a.\ a \in Collect\ C_U.obj \Longrightarrow F_{UV}.map_0\ a = map_0\ a$
using $F_{UV}.map_0\text{-}def\ C_U.obj\text{-}char\ C_U.arr\text{-}char_{SbC}\ D_V.src\text{-}def$
$F_U.map_0\text{-}simp\ F_{UV}.map_0\text{-}simp$ **by** *auto*
ultimately show $bij\text{-}betw\ F_{UV}.map_0\ (Collect\ C_U.obj)\ (Collect\ D_V.obj)$
using *bij-betw-U-V*
by (*simp add*: *bij-betw-U-V bij-betw-cong*)
qed

interpretation EQ_{UV}: *equivalence-of-bicategories*
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$F_{UV}.map\ F_{UV}.cmp\ F_{UV}.G\ F_{UV}.\Phi_G$
$F_{UV}.unit_0'\ F_{UV}.unit_1'\ F_{UV}.counit_0'\ F_{UV}.counit_1'$
using F_{UV}.*extends-to-equivalence-of-bicategories* **by** *blast*

Now compose EQ_{UV} with the equivalence from D_V to D and the converse of the equivalence from C_U to C. The result is an equivalence of bicategories from C to D.

interpretation EQ_C: *equivalence-of-bicategories*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$C_U.E\ C_U.\Phi_E\ C_U.P\ C_U.\Phi_P$
$C_U.unit_0\ C_U.unit_1\ C_U.counit_0\ C_U.counit_1$
using C_U.*induces-equivalence* **by** *simp*

interpretation EQ_C': *converse-equivalence-of-bicategories*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$C_U.E\ C_U.\Phi_E\ C_U.P\ C_U.\Phi_P$
$C_U.unit_0\ C_U.unit_1\ C_U.counit_0\ C_U.counit_1$
..

interpretation EQ_D: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$

$D_V.E\ D_V.\Phi_E\ D_V.P\ D_V.\Phi_P$
$D_V.unit_0\ D_V.unit_1\ D_V.counit_0\ D_V.counit_1$
using $D_V.induces\text{-}equivalence$ **by** *simp*

interpretation $EQ_{UV}oEQ_C'$: *composite-equivalence-of-bicategories*
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F_{UV}.map\ F_{UV}.cmp\ F_{UV}.G\ F_{UV}.\Phi_G$
$C_U.P\ C_U.\Phi_P\ C_U.E\ C_U.\Phi_E$
$F_{UV}.unit_0'\ F_{UV}.unit_1'\ F_{UV}.counit_0'\ F_{UV}.counit_1'$
$EQ_C'.unit_0\ EQ_C'.unit_1\ EQ_C'.counit_0\ EQ_C'.counit_1$
..

interpretation $EQ_D oEQ_{UV} oEQ_C'$: *composite-equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$D_V.E\ D_V.\Phi_E\ D_V.P\ D_V.\Phi_P$
$EQ_{UV}oEQ_C'.left\text{-}map\ EQ_{UV}oEQ_C'.left\text{-}cmp$
$EQ_{UV}oEQ_C'.right\text{-}map\ EQ_{UV}oEQ_C'.right\text{-}cmp$
$D_V.unit_0\ D_V.unit_1\ D_V.counit_0\ D_V.counit_1$
$EQ_{UV}oEQ_C'.unit_0\ EQ_{UV}oEQ_C'.unit_1$
$EQ_{UV}oEQ_C'.counit_0\ EQ_{UV}oEQ_C'.counit_1$
..

lemma *induces-equivalence-of-bicategories*:
shows *equivalence-of-bicategories* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.right\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.right\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.unit_0\ EQ_D oEQ_{UV} oEQ_C'.unit_1$
$EQ_D oEQ_{UV} oEQ_C'.counit_0\ EQ_D oEQ_{UV} oEQ_C'.counit_1$
..

lemma *left-map-simp*:
assumes $C.arr\ \mu$
shows $EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ \mu = D_V.E\ (F\ (C_U.P\ \mu))$
using *assms* **by** *simp*

lemma *right-map-simp*:
assumes $D.arr\ \nu$
shows $EQ_D oEQ_{UV} oEQ_C'.right\text{-}map\ \nu = C_U.E\ (F_{UV}.G\ (D_V.P\ \nu))$
using *assms* **by** *simp*

lemma $unit_0\text{-}simp$:
assumes $C.obj\ a$
shows $EQ_D oEQ_{UV} oEQ_C'.unit_0\ a =$
$C_U.E\ (F_{UV}.G\ (D_V.\eta_0\ (D_V.src\ (F\ (C_U.P\ a))))) \star_C C_U.E\ (C_U.P_0\ (src_C\ a))$
$\star_C EQ_C'.unit_0\ a$

using *assms* $EQ_D oEQ_{UV} oEQ_C{}'.unit_0$-*simp* $EQ_{UV} oEQ_C{}'.unit_0$-*simp*
$EQ_{UV} oEQ_C{}'.FH.map_0$-*def* $C_U.src$-*def*
by *auto*

We've now got an equivalence of bicategories between C and D, but it involves $EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map$ and not the originally given equivalence pseudofunctor F. However, we can patch things up by showing that $EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map$ is pseudonaturally equivalent to F. From this, we may conclude, using the fact that equivalences of bicategories respect pseudonatural equivalence, that there is an equivalence of bicategories between C and D that involves F and $EQ_D oEQ_{UV} oEQ_C{}'.right\text{-}map$, rather than $EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map$ and $EQ_D oEQ_{UV} oEQ_C{}'.right\text{-}map$.

abbreviation τ_0
where $\tau_0\ a \equiv F\ (C_U.\varepsilon_0\ a)$

abbreviation τ_1
where $\tau_1\ f \equiv D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ f),\ C_U.P\ f)) \cdot_D F\ (C_U.\varepsilon_1\ f) \cdot_D \Phi\ (f,\ C_U.\varepsilon_0\ (src_C\ f))$

lemma τ_0-*in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\tau_0\ a : map_0\ (C_U.P_0\ a) \rightarrow_D map_0\ a$»
and «$\tau_0\ a : \tau_0\ a \Rightarrow_D \tau_0\ a$»
proof −
  show «$\tau_0\ a : map_0\ (C_U.P_0\ a) \rightarrow_D map_0\ a$»
  proof
    show $D.arr\ (\tau_0\ a)$
      using *assms* **by** *simp*
    show $src_D\ (\tau_0\ a) = map_0\ (C_U.P_0\ a)$
    using *assms* map_0-*def* $C_U.counit.ide\text{-}map_0\text{-}obj$ $C_U.equivalence\text{-}data\text{-}simps_B(7)$ $C.ideD(1)$
      preserves-src
      by *presburger*
    show $trg_D\ (\tau_0\ a) = map_0\ a$
      using *assms* **by** *auto*
  qed
  show «$\tau_0\ a : \tau_0\ a \Rightarrow_D \tau_0\ a$»
    using *assms* **by** *auto*
qed

lemma τ_0-*simps* [*simp*]:
assumes $C.obj\ a$
shows $D.ide\ (\tau_0\ a)$
and $src_D\ (\tau_0\ a) = map_0\ (C_U.P_0\ a)$ **and** $trg_D\ (\tau_0\ a) = map_0\ a$
  using *assms* τ_0-*in-hom*(*1*) [*of a*] **by** *auto*

lemma τ_1-*in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\tau_1\ f : map_0\ (C_U.P_0\ (src_C\ f)) \rightarrow_D map_0\ (trg_C\ f)$»
and «$\tau_1\ f : F\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ (C_U.P\ f)$»
proof −
  show *1*: «$\tau_1\ f : F\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ (C_U.P\ f)$»

proof (*intro D.comp-in-homI*)
show «Φ (f, $C_U.d$ (src_C f)) : $F\ f \star_D \tau_0$ (src_C f) $\Rightarrow_D$ F ($f \star_C C_U.\varepsilon_0$ (src_C f))»
using *assms* **by** *auto*
show «F ($C_U.\varepsilon_1$ f) : F ($f \star_C C_U.\varepsilon_0$ (src_C f)) $\Rightarrow_D$ F ($C_U.\varepsilon_0$ (trg_C f) $\star_C C_U.P\ f$)»
using *assms* $C_U.counit_1$*-in-hom* [*of f*] C_U*.P-def* **by** *auto*
show «*D.inv* (Φ ($C_U.\varepsilon_0$ (trg_C f), $C_U.P\ f$)) :
F ($C_U.\varepsilon_0$ (trg_C f) $\star_C C_U.P\ f$) $\Rightarrow_D \tau_0$ (trg_C f) $\star_D$ F ($C_U.P\ f$)»
using *assms* $C.VV.arr\text{-}char_{SbC}$ $C.VV.ide\text{-}char_{SbC}$ Φ*.components-are-iso*
by (*metis* (*no-types*, *lifting*) C_U*.P-def* $C_U.counit.ide\text{-}map_0\text{-}obj$ $C_U.counit_1\text{-}simps(1,5)$
$C_U.equivalence\text{-}data\text{-}simps_B(2)$ *C.hseqE C.ide-hcomp C.obj-trg D.arr-cod D.inv-in-homI*
cmp-components-are-iso cmp-in-hom(*2*) *preserves-cod preserves-reflects-arr*)
qed
show «τ_1 f : map_0 ($C_U.P_0$ (src_C f)) $\rightarrow_D$ map_0 (trg_C f)»
using *assms 1* map_0*-def* [*of* $C_U.P_0$ (src_C f)] $C_U.emb.map_0\text{-}simp$ $C_U.P_0$*-props*
apply (*intro D.in-hhomI*)
apply *auto*
using *D.src-dom* [*of* τ_1 f]
apply (*elim D.in-homE*)
apply *auto*
using *D.trg-dom* [*of* τ_1 f]
apply (*elim D.in-homE*)
by *auto*
qed

lemma τ_1*-simps* [*simp*]:
assumes *C.ide f*
shows *D.arr* (τ_1 f)
and src_D (τ_1 f) = map_0 ($C_U.P_0$ (src_C f)) **and** trg_D (τ_1 f) = map_0 (trg_C f)
and *D.dom* (τ_1 f) = $F\ f \star_D \tau_0$ (src_C f) **and** *D.cod* (τ_1 f) = τ_0 (trg_C f) $\star_D$ F ($C_U.P\ f$)
using *assms* τ_1*-in-hom* **by** *blast+*

lemma *iso-*τ_1:
assumes *C.ide f*
shows *D.iso* (τ_1 f)
proof −
have $C.VV.ide$ ($C_U.\varepsilon_0$ (trg_C f), $C_U.P\ f$)
proof −
have $C.VV.arr$ ($C_U.\varepsilon_0$ (trg_C f), $C_U.P\ f$)
using *assms* $C_U.equivalence\text{-}data\text{-}simps_B(7)$ *C.ideD*(*1*) **by** *auto*
moreover have $C.VxV.ide$ ($C_U.\varepsilon_0$ (trg_C f), $C_U.P\ f$)
using *assms* $C_U.prj.preserves\text{-}ide$ [*of f*] $C_U.ide\text{-}char_{SbC}$ **by** *simp*
ultimately show *?thesis*
using *assms* $C.VV.ide\text{-}char_{SbC}$ [*of* ($C_U.\varepsilon_0$ (trg_C f), $C_U.P\ f$)] **by** *blast*
qed
hence *D.iso* (Φ ($C_U.d$ (trg_C f), $C_U.P\ f$))
by *simp*
thus *?thesis*
using *assms* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ Φ*.components-are-iso*
by (*intro D.isos-compose*) *auto*

```
qed

interpretation τ: pseudonatural-equivalence
                  V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
                  EQ_D oEQ_UV oEQ_C'.left-map EQ_D oEQ_UV oEQ_C'.left-cmp F Φ
                  τ_0 τ_1
proof
  show ⋀a. C.obj a ⟹ D.ide (τ_0 a)
    by simp
  show ⋀a. C.obj a ⟹ D.equivalence-map (τ_0 a)
    using C_U.counit.components-are-equivalences preserves-equivalence-maps by blast
  show ⋀a. C.obj a ⟹ «τ_0 a : src_D (EQ_D oEQ_UV oEQ_C'.left-map a) →_D src_D (F a)»
  proof −
    fix a
    assume a: C.obj a
    show «τ_0 a : src_D (EQ_D oEQ_UV oEQ_C'.left-map a) →_D src_D (F a)»
    proof
      show D.arr (τ_0 a)
        using a by simp
      show src_D (τ_0 a) = src_D (EQ_D oEQ_UV oEQ_C'.left-map a)
      proof −
        have D_V.arr (F (C_U.P a))
        proof −
          have src_D (F (C_U.P a)) ∈ V
          proof −
            have src_D (F (C_U.P a)) = map_0 (C_U.prj.map_0 a)
              using a by auto
            moreover have C.obj (C_U.prj.map_0 a)
              using a C_U.obj-char [of C_U.prj.map_0 a] C_U.prj.map_0-simps(1) by auto
            ultimately show ?thesis
              using V-def by simp
          qed
          moreover have trg_D (F (C_U.P a)) ∈ V
          proof −
            have trg_D (F (C_U.P a)) = map_0 (C_U.prj.map_0 a)
              using a by auto
            moreover have C.obj (C_U.prj.map_0 a)
              using a C_U.obj-char [of C_U.prj.map_0 a] C_U.prj.map_0-simps(1) by auto
            ultimately show ?thesis
              using V-def by simp
          qed
          ultimately show ?thesis
            using a D_V.arr-char_SbC by fastforce
        qed
        thus ?thesis
      using a D_V.emb.map_0-def D_V.emb.map-def D_V.src-def C.obj-simps(1−2) C_U.P-simps_B(2)
          by (simp add: C_U.P_0-props(1))
      qed
      show trg_D (F (C_U.d a)) = src_D (F a)
```

using a **by** *auto*
qed
qed
show $\bigwedge f.\ C.ide\ f \Longrightarrow$
$\ll \tau_1\ f : F\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f \gg$
proof −
fix f
assume f: $C.ide\ f$
show $\ll \tau_1\ f : F\ f \star_D \tau_0\ (src_C\ f) \Rightarrow_D \tau_0\ (trg_C\ f) \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f \gg$
proof −
have $D_V.arr\ (F\ (C_U.P\ f))$
using $f\ F_{UV}.preserves\text{-}arr$ **by** *auto*
hence $EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f = F\ (C_U.P\ f)$
using $f\ EQ_{UV} oEQ_C{}'.FH.map_0\text{-}def\ D_V.emb.map\text{-}def$ **by** *simp*
thus *?thesis*
using f **by** *auto*
qed
qed
show $\bigwedge f.\ C.ide\ f \Longrightarrow D.iso\ (\tau_1\ f)$
using $iso\text{-}\tau_1$ **by** *simp*
show $\bigwedge \mu.\ C.arr\ \mu \Longrightarrow$
$\tau_1\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0\ (src_C\ \mu)) =$
$(\tau_0\ (trg_C\ \mu) \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ \mu) \cdot_D \tau_1\ (C.dom\ \mu)$
proof −
fix μ
assume μ: $C.arr\ \mu$
show $\tau_1\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0\ (src_C\ \mu))$
$= (\tau_0\ (trg_C\ \mu) \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ \mu) \cdot_D \tau_1\ (C.dom\ \mu)$
proof −
have $\tau_1\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0\ (src_C\ \mu))$
$= D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.cod\ \mu))) \cdot_D$
$F\ (C_U.\varepsilon_1\ (C.cod\ \mu)) \cdot_D$
$\Phi\ (C.cod\ \mu,\ C_U.\varepsilon_0\ (src_C\ \mu)) \cdot_D$
$(F\ \mu \star_D \tau_0\ (src_C\ \mu))$
using μ $D.comp\text{-}assoc$ **by** *simp*
also have ... $= D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.cod\ \mu))) \cdot_D$
$(F\ (C_U.\varepsilon_1\ (C.cod\ \mu)) \cdot_D$
$F\ (\mu \star_C C_U.\varepsilon_0\ (src_C\ \mu))) \cdot_D$
$\Phi\ (C.dom\ \mu,\ C_U.\varepsilon_0\ (src_C\ \mu))$
using μ $\Phi.naturality\ [of\ (\mu,\ C_U.\varepsilon_0\ (src_C\ \mu))]$ $D.comp\text{-}assoc$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $FF\text{-}def$
by *simp*
also have ... $= (D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.cod\ \mu))) \cdot_D$
$F\ (C_U.\varepsilon_0\ (trg_C\ \mu) \star_C C_U.EoP.map\ \mu)) \cdot_D$
$F\ (C_U.\varepsilon_1\ (C.dom\ \mu)) \cdot_D$
$\Phi\ (C.dom\ \mu,\ C_U.\varepsilon_0\ (src_C\ \mu))$
proof −
have $F\ (C_U.\varepsilon_1\ (C.cod\ \mu)) \cdot_D F\ (\mu \star_C C_U.\varepsilon_0\ (src_C\ \mu))$
$= F\ (C_U.\varepsilon_1\ (C.cod\ \mu) \cdot_C (\mu \star_C C_U.\varepsilon_0\ (src_C\ \mu)))$

using μ **by** *simp*
also have ... $= F\ ((C_U.\varepsilon_0\ (trg_C\ \mu) \star_C C_U.EoP.map\ \mu) \cdot_C C_U.\varepsilon_1\ (C.dom\ \mu))$
using μ $C_U.counit.naturality$ [*of* μ] **by** *simp*
also have ... $= F\ (C_U.\varepsilon_0\ (trg_C\ \mu) \star_C C_U.EoP.map\ \mu) \cdot_D$
$F\ (C_U.\varepsilon_1\ (C.dom\ \mu))$
using μ **by** *simp*
finally have $F\ (C_U.\varepsilon_1\ (C.cod\ \mu)) \cdot_D F\ (\mu \star_C C_U.\varepsilon_0\ (src_C\ \mu))$
$= F\ (C_U.\varepsilon_0\ (trg_C\ \mu) \star_C C_U.EoP.map\ \mu) \cdot_D F\ (C_U.\varepsilon_1\ (C.dom\ \mu))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ (C_U.\varepsilon_0\ (trg_C\ \mu)) \star_D F\ (C_U.EoP.map\ \mu)) \cdot_D$
$D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.dom\ \mu))) \cdot_D$
$F\ (C_U.\varepsilon_1\ (C.dom\ \mu)) \cdot_D$
$\Phi\ (C.dom\ \mu,\ C_U.\varepsilon_0\ (src_C\ \mu))$
proof −
have $D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.cod\ \mu))) \cdot_D$
$F\ (C_U.\varepsilon_0\ (trg_C\ \mu) \star_C C_U.EoP.map\ \mu)$
$= (F\ (C_U.\varepsilon_0\ (trg_C\ \mu)) \star_D F\ (C_U.EoP.map\ \mu)) \cdot_D$
$D.inv\ (\Phi\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ (C.dom\ \mu)))$
proof −
have $C.VV.arr\ (C_U.\varepsilon_0\ (trg_C\ \mu),\ C_U.P\ \mu)$
proof (*unfold* $C.VV.arr\text{-}char_{SbC}$, *intro conjI*)
show $C.arr\ (fst\ (C_U.d\ (trg_C\ \mu),\ C_U.P\ \mu))$
using μ **by** *simp*
show $C.arr\ (snd\ (C_U.d\ (trg_C\ \mu),\ C_U.P\ \mu))$
using μ **by** *simp*
show $src_C\ (fst\ (C_U.d\ (trg_C\ \mu),\ C_U.P\ \mu)) = trg_C\ (snd\ (C_U.d\ (trg_C\ \mu),\ C_U.P\ \mu))$
using μ $C_U.emb.map_0\text{-}def$ $C_U.emb.map\text{-}def$ **apply** *simp*
using $C_U.P_0\text{-}props(1)$ $C_U.P\text{-}simps_B(3)$ $C_U.src\text{-}def$ **by** *fastforce*
qed
moreover have $C_U.EoP.map\ \mu = C_U.P\ \mu$
using μ **by** (*simp add*: $C_U.emb.map\text{-}def$)
moreover have $C_U.emb.map_0\ (C_U.P_0\ (trg_C\ \mu)) = C_U.P_0\ (trg_C\ \mu)$
using μ $C_U.P_0\text{-}props(1)$ $C_U.emb.map_0\text{-}simp$ **by** *blast*
ultimately show *?thesis*
using μ $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C.VV.ide\text{-}char_{SbC}$
FF-def
$\Phi.inv\text{-}naturality$ [*of* $(C_U.counit_0\ (trg_C\ \mu),\ C_U.P\ \mu)$]
by *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0\ (trg_C\ \mu) \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ \mu) \cdot_D \tau_1\ (C.dom\ \mu)$
using μ $F_{UV}.preserves\text{-}arr$ $D.comp\text{-}assoc$ $D_V.emb.map\text{-}def$ $C_U.emb.map\text{-}def$ **by** *simp*
finally show $\tau_1\ (C.cod\ \mu) \cdot_D (F\ \mu \star_D \tau_0\ (src_C\ \mu))$
$= (\tau_0\ (trg_C\ \mu) \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ \mu) \cdot_D \tau_1\ (C.dom\ \mu)$

```
      by blast
  qed
qed
show ⋀a. C.obj a ⟹
        (τ0 a ⋆D EQDoEQUVoEQC'.FH.unit a) ·D r_D^-1[τ0 a] ·D l_D[τ0 a] =
        τ1 a ·D (unit a ⋆D τ0 a)
proof −
  fix a
  assume a: C.obj a
  have 1: C_U.obj (C_U.prj.map0 a)
    using a C_U.prj.map0-simps(1) by simp
  have 2: «C_U.prj.unit a : C_U.prj.map0 a ⇒C C_U.P a»
    using a C_U.in-hom-char_SbC by blast
  have 3: «C_U.prj.unit a : C_U.prj.map0 a →C C_U.prj.map0 a»
  proof (intro C.in-hhomI)
    show 4: C.arr (C_U.prj.unit a)
      using 2 by auto
    show src_C (C_U.prj.unit a) = C_U.prj.map0 a
    proof −
      have src_C (C_U.prj.unit a) = src_C (C_U.P a)
        using 2 4 C.src-cod [of C_U.prj.unit a] C.vconn-implies-hpar(1,3) by auto
      also have ... = C_U.prj.map0 a
        using a C.obj-simps(1) C_U.prj.map0-def [of a] C_U.src-def [of C_U.P a]
        by simp
      finally show ?thesis by blast
    qed
    show trg_C (C_U.prj.unit a) = C_U.prj.map0 a
    proof −
      have trg_C (C_U.prj.unit a) = trg_C (C_U.P a)
        using 2 4 C.trg-cod [of C_U.prj.unit a] C.vconn-implies-hpar(2,4) by auto
      also have ... = C_U.prj.map0 a
        using a C.obj-simps(1,3) C_U.prj.map0-def [of a] C_U.trg-def [of C_U.P a]
        by simp
      finally show ?thesis by blast
    qed
  qed
  have [simp]: C.dom (C_U.prj.unit a) = C_U.prj.map0 a
    using 2 by auto
  have [simp]: C.arr (C_U.prj.unit a)
    using 2 by auto
  have [simp]: C.cod (C_U.prj.unit a) = C_U.P a
    using 2 by auto
  have [simp]: src_C (C_U.prj.unit a) = C_U.prj.map0 a
    using 3 by auto
  have [simp]: trg_C (C_U.prj.unit a) = C_U.prj.map0 a
    using 3 by auto
  have [simp]: C.arr (C_U.d a)
    using a by simp
  have [simp]: C.ide (C_U.d a)
```

```
  using a by simp
have [simp]: C.dom (C_U.d a) = C_U.d a
  using a by auto
have [simp]: C.cod (C_U.d a) = C_U.d a
  using a by auto
have [simp]: src_C (C_U.d a) = C_U.prj.map_0 a
  using a C_U.equivalence-data-simps_B(7) by auto
have [simp]: trg_C (C_U.d a) = a
  using a C_U.equivalence-data-simps_B(7) by auto

have seq: D.seq (F (C_U.prj.unit a)) (F_UV.unit (C_U.prj.map_0 a))
  using a
  apply (intro D.seqI)
  using C_U.prj.map_0-simps(1) D_V.arr-char_SbC F_UV.unit-simps(1) apply blast
  using C_U.prj.unit-simps(1) C_U.arr-char_SbC apply blast
proof -
  have D.dom (F (C_U.prj.unit a)) = F (C.dom (C_U.prj.unit a))
    using a C_U.prj.unit-simps(1) C_U.arr-char_SbC by simp
  also have ... = F (C_U.prj.map_0 a)
    using a C_U.prj.unit-simps [of a] C_U.dom-char_SbC by simp
  also have ... = D.cod (F_UV.unit (C_U.prj.map_0 a))
 using a 1 C_U.prj.unit-simps [of a] F_UV.unit-simps(5) [of C_U.prj.map_0 a] D_V.cod-char_SbC
    by simp
  finally show D.dom (F (C_U.prj.unit a)) = D.cod (F_UV.unit (C_U.prj.map_0 a))
    by blast
qed

have 4: EQ_D oEQ_UV oEQ_C'.FH.unit a = F (C_U.prj.unit a) ·_D unit (C_U.prj.map_0 a)
proof -
  have EQ_UV oEQ_C'.FH.map_0 a = map_0 (C_U.P_0 a)
    using a EQ_UV oEQ_C'.FH.map_0-def D_V.src-def C.obj-def F_UV.preserves-arr by auto
  have seq': D_V.arr (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
  proof -
    have 5: D.arr (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
      using seq by simp
    moreover have src_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a)) ∈ V
    proof -
      have src_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
             = map_0 (C_U.prj.map_0 a)
      proof -
        have src_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
               = src_D (F_UV.unit (C_U.prj.map_0 a))
          using 5 D.src-vcomp D.vseq-implies-hpar by simp
        also have ... = map_0 (C_U.prj.map_0 a)
          using a F_UV.unit-simps(1−2) [of C_U.prj.map_0 a] D_V.src-def F_UV.map_0-def
          apply simp
          using a C_U.prj.map_0-simps(1) F_UV.preserves-arr map_0-def by force
        finally show ?thesis by blast
      qed
```

```
    moreover have C.obj (C_U.prj.map_0 a)
      using a C_U.prj.map_0-simps(1) C_U.obj-char by blast
    ultimately show ?thesis
      using V-def by blast
  qed
  moreover have trg_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a)) ∈ V
  proof −
    have trg_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
          = map_0 (C_U.prj.map_0 a)
    proof −
      have trg_D (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a)) =
            trg_D (F_UV.unit (C_U.prj.map_0 a))
        using 5 D.trg-vcomp D.vseq-implies-hpar by simp
      also have ... = map_0 (C_U.prj.map_0 a)
        using a F_UV.unit-simps(1,3) [of C_U.prj.map_0 a] D_V.src-def D_V.trg-def
              F_UV.map_0-def
        apply simp
        using a C_U.prj.map_0-simps(1) F_UV.preserves-arr map_0-def by force
      finally show ?thesis by blast
    qed
    moreover have C.obj (C_U.prj.map_0 a)
      using a C_U.prj.map_0-simps(1) C_U.obj-char by blast
    ultimately show ?thesis
      using V-def by blast
  qed
  ultimately show ?thesis
    using a D_V.arr-char_SbC by simp
qed

have EQ_D oEQ_UV oEQ_C'.FH.unit a =
      F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a) ·_D map_0 (C_U.P_0 a)
proof −
  have 5: D_V.obj (src_D (F (C_U.P_0 a)))
    using a C_U.P_0-props [of a]
    by (metis (no-types, lifting) EQ_UV oEQ_C'.FH.map_0-simps(1)
        ‹EQ_UV oEQ_C'.FH.map_0 a = map_0 (C_U.P_0 a)› map_0-def)
  have 6: D_V.emb.unit (map_0 (C_U.P_0 a)) = map_0 (C_U.P_0 a)
    using a 5 D_V.emb.unit-char' EQ_UV oEQ_C'.FH.map_0-simps(1)
          ‹EQ_UV oEQ_C'.FH.map_0 a = map_0 (C_U.P_0 a)› map_0-def
    by simp
  moreover have D_V.emb.unit (map_0 (C_U.P_0 a)) = map_0 (C_U.prj.map_0 a)
    using 6 a C.obj-simps C_U.equivalence-data-simps_B(7) by simp
  moreover have D_V.arr (F (C_U.prj.unit a) ·_D F_UV.unit (C_U.prj.map_0 a))
    using a C.obj-simps seq' by blast
  moreover have D_V.arr (F_UV.unit (C_U.P_0 a))
    using a 1 by simp
  moreover have D_V.arr (F (C_U.prj.unit a))
    using a F_UV.preserves-arr by auto
  moreover have D_V.obj (F_UV.map_0 (C_U.P_0 a))
```

using *a 1* F_{UV}*.map*$_0$*-simps(1)* **by** *auto*
moreover have F_{UV}*.map*$_0$ (C_U*.P*$_0$ *a*) = *map*$_0$ (C_U*.P*$_0$ *a*)
using *a* ‹EQ_{UV} *o* EQ_C*'.FH.map*$_0$ *a* = *map*$_0$ (C_U*.P*$_0$ *a*)› *map*$_0$*-def* F_{UV}*.map*$_0$*-simps*
by *auto*
ultimately show *?thesis*
using *a seq* EQ_D *o* EQ_{UV} *o* EQ_C*'.FH.unit-char'* [*of a*] EQ_{UV} *o* EQ_C*'.FH.unit-char'* [*of a*]
D_V*.emb.map-def* D_V*.seq-char*$_{SbC}$ D_V*.comp-char D.comp-assoc*
by *simp*
qed
also have ... = *F* (C_U*.prj.unit a*) $\cdot_D$ F_{UV}*.unit* (C_U*.prj.map*$_0$ *a*)
proof −
have *D.dom* (F_{UV}*.unit* (C_U*.prj.map*$_0$ *a*)) = *map*$_0$ (C_U*.P*$_0$ *a*)
using *a 1* F_{UV}*.unit-simps(4)* [*of* C_U*.prj.map*$_0$ *a*] D_V*.dom-char*$_{SbC}$ F_{UV}*.map*$_0$*-def*
D_V*.src-def map*$_0$*-def C.obj-def* F_{UV}*.preserves-arr*
by *simp*
moreover have *D.arr* (F_{UV}*.unit* (C_U*.prj.map*$_0$ *a*))
using *a* ‹*D.seq* (*F* (C_U*.prj.unit a*)) (F_{UV}*.unit* (C_U*.prj.map*$_0$ *a*))› **by** *blast*
ultimately show *?thesis*
using *a D.comp-arr-dom* **by** *auto*
qed
also have ... = *F* (C_U*.prj.unit a*) $\cdot_D$ *unit* (C_U*.prj.map*$_0$ *a*)
using *a 1* F_{UV}*.unit-char'* F_U*.unit-char'* **by** *simp*
finally show *?thesis* **by** *blast*
qed

have (τ_0 *a* $\star_D$ EQ_D *o* EQ_{UV} *o* EQ_C*'.FH.unit a*) $\cdot_D$ $\mathrm{r}_D{}^{-1}[\tau_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\tau_0\ a]$ =
(τ_0 *a* $\star_D$ *F* (C_U*.prj.unit a*) $\cdot_D$ *unit* (C_U*.prj.map*$_0$ *a*)) $\cdot_D$ $\mathrm{r}_D{}^{-1}[\tau_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\tau_0\ a]$
using *4* **by** *simp*
also have (τ_0 *a* $\star_D$ *F* (C_U*.prj.unit a*) $\cdot_D$ *unit* (C_U*.prj.map*$_0$ *a*)) $\cdot_D$
$\mathrm{r}_D{}^{-1}[\tau_0\ a]$ $\cdot_D$ $\mathrm{l}_D[\tau_0\ a]$
= (τ_0 *a* $\star_D$ *F* (C_U*.prj.unit a*) $\cdot_D$
unit (C_U*.prj.map*$_0$ *a*)) $\cdot_D$
(τ_0 *a* $\star_D$ *D.inv* (*unit* (*src*$_C$ (C_U*.d a*)))) $\cdot_D$
D.inv (Φ (C_U*.d a*, *src*$_C$ (C_U*.d a*))) $\cdot_D$
F $\mathrm{r}_C{}^{-1}[C_U.d\ a]$ $\cdot_D$
F $\mathrm{l}_C[C_U.d\ a]$ $\cdot_D$
Φ (*trg*$_C$ (C_U*.d a*), C_U*.d a*) $\cdot_D$
(*unit* (*trg*$_C$ (C_U*.d a*)) $\star_D$ τ_0 *a*)
proof −
have $\mathrm{l}_D[\tau_0\ a]$ = *F* $\mathrm{l}_C[C_U.d\ a]$ $\cdot_D$ Φ (*trg*$_C$ (C_U*.d a*), C_U*.d a*) $\cdot_D$
(*unit* (*trg*$_C$ (C_U*.d a*)) $\star_D$ τ_0 *a*)
using *a preserves-lunit* [*of* $C_U.\varepsilon_0$ *a*] C_U*.counit.ide-map*$_0$*-obj lunit-coherence*
by *blast*
moreover
have $\mathrm{r}_D{}^{-1}[\tau_0\ a]$ = (τ_0 *a* $\star_D$ *D.inv* (*unit* (*src*$_C$ (C_U*.d a*)))) $\cdot_D$
D.inv (Φ (C_U*.d a*, *src*$_C$ (C_U*.d a*))) $\cdot_D$ *F* $\mathrm{r}_C{}^{-1}[C_U.d\ a]$
proof −
have *F* $\mathrm{r}_C{}^{-1}[C_U.d\ a]$ =

$\Phi\ (C_U.d\ a,\ src_C\ (C_U.d\ a)) \cdot_D (\tau_0\ a \star_D\ unit\ (src_C\ (C_U.d\ a))) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a]$
using *preserves-runit*(*2*) [*of* $C_U.\varepsilon_0\ a$] **by** *simp*
moreover have *1*: *D.iso* $(\Phi\ (C_U.d\ a,\ src_C\ (C_U.d\ a)))$
using *a* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$ *C.obj-src* [*of* $C_U.d\ a$]
Φ.*components-are-iso* [*of* $(C_U.d\ a,\ src_C\ (C_U.d\ a))$]
by *auto*
ultimately have *D.inv* $(\Phi\ (C_U.d\ a,\ src_C\ (C_U.d\ a))) \cdot_D F\ \mathrm{r}_C{}^{-1}[C_U.d\ a] =$
$(\tau_0\ a \star_D\ unit\ (src_C\ (C_U.d\ a))) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a]$
using *D.invert-side-of-triangle*(*1*)
[*of* $F\ \mathrm{r}_C{}^{-1}[C_U.d\ a]\ \Phi\ (C_U.d\ a,\ src_C\ (C_U.d\ a))$
$(\tau_0\ a \star_D\ unit\ (src_C\ (C_U.d\ a))) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a]$]
by *fastforce*
thus *?thesis*
using *D.invert-side-of-triangle*(*1*)
[*of* *D.inv* $(\Phi\ (C_U.d\ a,\ src_C\ (C_U.d\ a))) \cdot_D F\ \mathrm{r}_C{}^{-1}[C_U.d\ a]$
$\tau_0\ a \star_D\ unit\ (src_C\ (C_U.d\ a))\ \mathrm{r}_D{}^{-1}[\tau_0\ a]$]
using *C.obj-src* [*of* $C_U.d\ a$] *unit-char*(*2*) **by** *auto*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a)) \cdot_D$
$((\tau_0\ a \star_D\ unit\ (C_U.prj.map_0\ a)) \cdot_D$
$(\tau_0\ a \star_D\ D.inv\ (unit\ (C_U.prj.map_0\ a))))) \cdot_D$
$D.inv\ (\Phi\ (C_U.d\ a,\ C_U.prj.map_0\ a)) \cdot_D$
$F\ \mathrm{r}_C{}^{-1}[C_U.d\ a] \cdot_D F\ \mathrm{l}_C[C_U.d\ a] \cdot_D$
$\Phi\ (a,\ C_U.d\ a) \cdot_D$
$(unit\ a \star_D\ \tau_0\ a)$
proof −
have $\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a) \cdot_D\ unit\ (C_U.prj.map_0\ a) =$
$(\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a)) \cdot_D (\tau_0\ a \star_D\ unit\ (C_U.prj.map_0\ a))$
using *1 seq D.whisker-left* [*of* $\tau_0\ a$]
by (*simp add*: $F_U.unit\text{-}char'\ F_{UV}.unit\text{-}char'$)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a)) \cdot_D$
$D.inv\ (\Phi\ (C_U.d\ a,\ C_U.prj.map_0\ a))) \cdot_D$
$F\ \mathrm{r}_C{}^{-1}[C_U.d\ a] \cdot_D F\ \mathrm{l}_C[C_U.d\ a] \cdot_D$
$\Phi\ (a,\ C_U.d\ a) \cdot_D$
$(unit\ a \star_D\ \tau_0\ a)$
proof −
have $(\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a)) \cdot_D$
$((\tau_0\ a \star_D\ unit\ (C_U.prj.map_0\ a)) \cdot_D$
$(\tau_0\ a \star_D\ D.inv\ (unit\ (C_U.prj.map_0\ a))))$
$= (\tau_0\ a \star_D\ F\ (C_U.prj.unit\ a)) \cdot_D$
$(\tau_0\ a \star_D\ unit\ (C_U.prj.map_0\ a) \cdot_D$
$D.inv\ (unit\ (C_U.prj.map_0\ a)))$
proof −

have $D.seq$ $(unit$ $(C_U.prj.map_0$ $a))$ $(D.inv$ $(unit$ $(C_U.prj.map_0$ $a)))$
using a 1 $unit\text{-}char(1-2)$ $C_U.obj\text{-}char$
by $(intro$ $D.seqI)$ $auto$
thus $?thesis$
using a $D.whisker\text{-}left$ $[of$ τ_0 $a]$ **by** $simp$
qed
also have ... $= (\tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a))$ $\cdot_D$ $(\tau_0$ a $\star_D$ F $(C_U.prj.map_0$ $a))$
using a 1 $unit\text{-}char(1-2)$ $C_U.obj\text{-}char$ $unit\text{-}simps$ $[of$ $C_U.prj.map_0$ $a]$
by $(simp$ add: $D.comp\text{-}arr\text{-}inv')$
also have ... $= \tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a)$ $\cdot_D$ F $(C_U.prj.map_0$ $a)$
using seq $D.whisker\text{-}left$ **by** $auto$
also have ... $= \tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a)$
using $D.comp\text{-}arr\text{-}dom$ **by** $simp$
finally have $(\tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a))$ $\cdot_D$
$((\tau_0$ a $\star_D$ $unit$ $(C_U.prj.map_0$ $a))$ $\cdot_D$
$(\tau_0$ a $\star_D$ $D.inv$ $(unit$ $(C_U.prj.map_0$ $a))))$
$= \tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a)$
by $blast$
thus $?thesis$
using $D.comp\text{-}assoc$ **by** $simp$
qed
also have ... $= D.inv$ $(\Phi$ $(C_U.d$ $a,$ $C_U.P$ $a))$ $\cdot_D$
$(F$ $(C_U.d$ a $\star_C$ $C_U.prj.unit$ $a)$ $\cdot_D$
F $\mathrm{r}_C{}^{-1}[C_U.d$ $a]$ $\cdot_D$ F $\mathrm{l}_C[C_U.d$ $a])$ $\cdot_D$
Φ $(a,$ $C_U.d$ $a)$ $\cdot_D$
$(unit$ a $\star_D$ τ_0 $a)$
proof $-$
have $(\tau_0$ a $\star_D$ F $(C_U.prj.unit$ $a))$ $\cdot_D$ $D.inv$ $(\Phi$ $(C_U.d$ $a,$ $C_U.prj.map_0$ $a))$ $=$
$D.inv$ $(\Phi$ $(C_U.d$ $a,$ $C_U.P$ $a))$ $\cdot_D$ F $(C_U.d$ a $\star_C$ $C_U.prj.unit$ $a)$
proof $-$
have $C.VV.arr$ $(C_U.d$ $a,$ $C_U.prj.unit$ $a)$
using a $C.VV.arr\text{-}char_{SbC}$ **by** $simp$
thus $?thesis$
using a $\Phi.inv\text{-}naturality$ $[of$ $(C_U.d$ $a,$ $C_U.prj.unit$ $a)]$ $C_U.prj.unit\text{-}simps$ $[of$ $a]$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $FF\text{-}def$
by $simp$
qed
thus $?thesis$
using $D.comp\text{-}assoc$ **by** $simp$
qed
also have ... $= (D.inv$ $(\Phi$ $(C_U.d$ $a,$ $C_U.P$ $a))$ $\cdot_D$
F $(C_U.counit_1$ a $\cdot_C$ $(a$ $\star_C$ $C_U.d$ $a))$ $\cdot_D$
Φ $(a,$ $C_U.d$ $a))$ $\cdot_D$
$(unit$ a $\star_D$ τ_0 $a)$
proof $-$
have F $(C_U.d$ a $\star_C$ $C_U.prj.unit$ $a)$ $\cdot_D$ F $\mathrm{r}_C{}^{-1}[C_U.d$ $a]$ $\cdot_D$ F $\mathrm{l}_C[C_U.d$ $a]$ $=$
F $((C_U.d$ a $\star_C$ $C_U.prj.unit$ $a)$ $\cdot_C$ $\mathrm{r}_C{}^{-1}[C_U.d$ $a]$ $\cdot_C$ $\mathrm{l}_C[C_U.d$ $a])$
using a **by** $simp$
also have ... $= F$ $(C_U.counit_1$ a $\cdot_C$ $(a$ $\star_C$ $C_U.d$ $a))$

proof −
have C_U.*EoP.unit a* = C_U.*prj.unit a*
using *a 1* C_U.*emb.map-def* C_U.*EoP.unit-char′* C_U.*emb.unit-char′ C.comp-arr-dom*
by *simp*
thus *?thesis*
using *a* C_U.*counit.respects-unit* [*of a*] C_U.I_C.*unit-char′* [*of a*] **by** *simp*
qed
finally have $F\ (C_U.d\ a \star_C C_U.prj.unit\ a) \cdot_D F\ \mathrm{r}_C{}^{-1}[C_U.d\ a] \cdot_D F\ \mathrm{l}_C[C_U.d\ a] =$
$F\ (C_U.counit_1\ a \cdot_C (a \star_C C_U.d\ a))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $\tau_1\ a \cdot_D (unit\ a \star_D \tau_0\ a)$
proof −
have $D.inv\ (\Phi\ (C_U.d\ a,\ C_U.P\ a)) \cdot_D F\ (C_U.\varepsilon_1\ a \cdot_C (a \star_C C_U.d\ a)) \cdot_D \Phi\ (a,\ C_U.d\ a)$
$= D.inv\ (\Phi\ (C_U.d\ a,\ C_U.P\ a)) \cdot_D F\ (C_U.\varepsilon_1\ a) \cdot_D \Phi\ (a,\ C_U.d\ a)$
using *a C.obj-def′ C.comp-arr-dom*
by (*metis* $C_U.counit_1$*-simps(1)* $C_U.counit_1$*-simps(4) C.objE*)
also have ... = $\tau_1\ a$
using *a* **by** *auto*
finally have $D.inv\ (\Phi\ (C_U.d\ a,\ C_U.P\ a)) \cdot_D F\ (C_U.\varepsilon_1\ a \cdot_C (a \star_C C_U.d\ a)) \cdot_D \Phi\ (a,$
$C_U.d\ a)$
$= \tau_1\ a$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally show $(\tau_0\ a \star_D EQ_D oEQ_{UV} oEQ_C{}'.FH.unit\ a) \cdot_D \mathrm{r}_D{}^{-1}[\tau_0\ a] \cdot_D \mathrm{l}_D[\tau_0\ a]$
$= \tau_1\ a \cdot_D (unit\ a \star_D \tau_0\ a)$
by *blast*
qed
show $\bigwedge f\ g.\ [\![C.ide\ f;\ C.ide\ g;\ src_C\ g = trg_C\ f]\!] \Longrightarrow$
$(\tau_0\ (trg_C\ g) \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}cmp\ (g,\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ g,$
$EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f] \cdot_D$
$(\tau_1\ g \star_D EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (src_C\ g),\ EQ_D oEQ_{UV} oEQ_C{}'.left\text{-}map\ f] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D \mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
$= \tau_1\ (g \star_C f) \cdot_D (\Phi\ (g,\ f) \star_D \tau_0\ (src_C\ f))$
proof −
fix *f g*
assume *f*: *C.ide f* **and** *g*: *C.ide g* **and** *fg*: $src_C\ g = trg_C\ f$
have *1*: $C.ide\ f \wedge C.ide\ g \wedge C.ide\ (C_U.P\ f) \wedge C.ide\ (C_U.P\ g) \wedge$
$C.ide\ (C_U.d\ (trg_C\ f)) \wedge C.ide\ (C_U.d\ (trg_C\ g)) \wedge$
$src_C\ (C_U.d\ (trg_C\ f)) = trg_C\ (C_U.P\ f) \wedge$
$src_C\ (C_U.d\ (trg_C\ g)) = trg_C\ (C_U.P\ g) \wedge$
$trg_C\ (C_U.d\ (trg_C\ f)) = src_C\ g$
using *f g fg* C_U.*emb.map*$_0$*-def* C_U.*emb.map-def* C_U.*obj-char* $C_U.P_0$*-props(1) C.obj-simps(2)*

C_U.*prj.preserves-ide* C_U.*ide-char*$_{SbC}$
by *auto*
have τ_1 $(g \star_C f) \cdot_D (\Phi\ (g, f) \star_D \tau_0\ (src_C\ f))$
$= (D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$F\ ((C_U.d\ (trg_C\ g) \star_C C_U.EoP.cmp\ (g, f)) \cdot_C$
$\mathrm{a}_C[C_U.d\ (trg_C\ g),\ C_U.P\ g,\ C_U.P\ f] \cdot_C$
$(C_U.\varepsilon_1\ g \star_C C_U.P\ f) \cdot_C$
$C.inv\ \mathrm{a}_C[g,\ C_U.d\ (src_C\ g),\ C_U.P\ f] \cdot_C$
$(g \star_C C_U.\varepsilon_1\ f) \cdot_C$
$\mathrm{a}_C[g, f,\ C_U.d\ (src_C\ f)]) \cdot_D$
$\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D$
$(\Phi\ (g, f) \star_D \tau_0\ (src_C\ f))$
using *f g fg* C_U.*emb.map-def* C_U.*counit.respects-hcomp* [*of f g*] *D.comp-arr-dom*
by *simp*
also have ... $= ((D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(\tau_1\ g \star_D F\ (C_U.P\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$(D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D$
$\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D$
$(\Phi\ (g, f) \star_D \tau_0\ (src_C\ f))$
proof −
have $F\ ((C_U.d\ (trg_C\ g) \star_C C_U.EoP.cmp\ (g, f)) \cdot_C$
$\mathrm{a}_C[C_U.d\ (trg_C\ g),\ C_U.P\ g,\ C_U.P\ f] \cdot_C$
$(C_U.\varepsilon_1\ g \star_C C_U.P\ f) \cdot_C$
$C.inv\ \mathrm{a}_C[g,\ C_U.d\ (src_C\ g),\ C_U.P\ f] \cdot_C$
$(g \star_C C_U.\varepsilon_1\ f) \cdot_C$
$\mathrm{a}_C[g, f,\ C_U.d\ (src_C\ f)])$
$= F\ (C_U.d\ (trg_C\ g) \star_C C_U.EoP.cmp\ (g, f)) \cdot_D$
$F\ \mathrm{a}_C[C_U.d\ (trg_C\ g),\ C_U.P\ g,\ C_U.P\ f] \cdot_D$
$F\ (C_U.\varepsilon_1\ g \star_C C_U.P\ f) \cdot_D$
$F\ (C.inv\ \mathrm{a}_C[g,\ C_U.d\ (src_C\ g),\ C_U.P\ f]) \cdot_D$
$F\ (g \star_C C_U.\varepsilon_1\ f) \cdot_D$
$F\ \mathrm{a}_C[g, f,\ C_U.d\ (src_C\ f)]$
proof −
have $C.arr\ ((C_U.d\ (trg_C\ g) \star_C C_U.EoP.cmp\ (g, f)) \cdot_C$
$\mathrm{a}_C[C_U.d\ (trg_C\ g),\ C_U.P\ g,\ C_U.P\ f] \cdot_C$
$(C_U.\varepsilon_1\ g \star_C C_U.P\ f) \cdot_C$
$C.inv\ \mathrm{a}_C[g,\ C_U.d\ (src_C\ g),\ C_U.P\ f] \cdot_C$
$(g \star_C C_U.\varepsilon_1\ f) \cdot_C$
$\mathrm{a}_C[g, f,\ C_U.d\ (src_C\ f)])$
using *f g fg 1* C_U.*emb.map-def* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ C_U.*EoP.FF-def*
by (*intro C.seqI C.hseqI′*) *auto*

thus *?thesis*
using *f g fg* **by** *fastforce*
qed
also have ... $= \Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$((D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g \star_C C_U.P\ f)) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g \star_C C_U.P\ f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f))) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g)) \star_D F\ (C_U.P\ f)) \cdot_D$
$((D.inv\ (\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$(F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f))) \cdot_D$
$((D.inv\ (\Phi\ (g \star_C C_U.d\ (trg_C\ f),\ C_U.P\ f)) \cdot_D$
$\Phi\ (g \star_C C_U.d\ (trg_C\ f),\ C_U.P\ f)) \cdot_D$
$(\Phi\ (g,\ C_U.d\ (trg_C\ f)) \star_D F\ (C_U.P\ f))) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D D.inv\ (\Phi\ (C_U.d\ (trg_C\ f),\ C_U.P\ f))) \cdot_D$
$((D.inv\ (\Phi\ (g,\ C_U.d\ (trg_C\ f) \star_C C_U.P\ f)) \cdot_D$
$\Phi\ (g,\ C_U.d\ (trg_C\ f) \star_C C_U.P\ f)) \cdot_D$
$(F\ g \star_D F\ (C_U.\varepsilon_1\ f))) \cdot_D$
$((D.inv\ (\Phi\ (g, f \star_C C_U.d\ (src_C\ f))) \cdot_D$
$\Phi\ (g, f \star_C C_U.d\ (src_C\ f))) \cdot_D$
$(F\ g \star_D \Phi\ (f,\ C_U.d\ (src_C\ f)))) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f)))$
using *1 f g fg preserves-hcomp preserves-assoc* C_U*.emb.map-def*
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ C_U*.EoP.FF-def*
D.comp-assoc
by *simp*
also have ... $= \Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$((D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g)) \star_D F\ (C_U.P\ f)) \cdot_D$
$(F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f)) \cdot_D$
$(\Phi\ (g,\ C_U.d\ (trg_C\ f)) \star_D F\ (C_U.P\ f))) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$((F\ g \star_D D.inv\ (\Phi\ (C_U.d\ (trg_C\ f),\ C_U.P\ f))) \cdot_D$
$(F\ g \star_D F\ (C_U.\varepsilon_1\ f)) \cdot_D$
$(F\ g \star_D \Phi\ (f,\ C_U.d\ (src_C\ f)))) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f)))$
proof −
have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g \star_C C_U.P\ f)) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g \star_C C_U.P\ f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$

$= \tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f)$
using *f g fg 1* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
moreover have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$(F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f))$
$= F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f)$
proof −
have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$(F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f))$
$= (F\ (C_U.d\ (trg_C\ g) \star_C C_U.P\ g) \star_D F\ (C_U.P\ f)) \cdot_D$
$(F\ (C_U.\varepsilon_1\ g) \star_D F\ (C_U.P\ f))$
using *f g fg 1* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
$C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
also have ... $= F\ (C_U.counit_1\ g) \star_D F\ (C_U.P\ f)$
using *g D.comp-cod-arr f g fg* $C_U.P_0\text{-}props(1)$ $C_U.emb.map\text{-}def$ **by** *simp*
finally show *?thesis* **by** *blast*
qed
moreover have $(D.inv\ (\Phi\ (g \star_C C_U.d\ (trg_C\ f),\ C_U.P\ f)) \cdot_D$
$(\Phi\ (g \star_C C_U.d\ (trg_C\ f),\ C_U.P\ f))) \cdot_D$
$(\Phi\ (g,\ C_U.d\ (trg_C\ f)) \star_D F\ (C_U.P\ f))$
$= (\Phi\ (g,\ C_U.d\ (trg_C\ f)) \star_D F\ (C_U.P\ f))$
using *f g fg 1* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
moreover have $(D.inv\ (\Phi\ (g,\ C_U.d\ (trg_C\ f) \star_C C_U.P\ f)) \cdot_D$
$\Phi\ (g,\ C_U.d\ (trg_C\ f) \star_C C_U.P\ f)) \cdot_D$
$(F\ g \star_D F\ (C_U.counit_1\ f))$
$= F\ g \star_D F\ (C_U.counit_1\ f)$
using *f g fg 1* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
moreover have $(D.inv\ (\Phi\ (g,\ f \star_C C_U.d\ (src_C\ f))) \cdot_D$
$\Phi\ (g,\ f \star_C C_U.d\ (src_C\ f))) \cdot_D$
$(F\ g \star_D \Phi\ (f,\ C_U.d\ (src_C\ f)))$
$= F\ g \star_D \Phi\ (f,\ C_U.d\ (src_C\ f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
ultimately show *?thesis*

using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(\tau_1\ g \star_D F\ (C_U.P\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f)))$
proof −
have $(F\ g \star_D D.inv\ (\Phi\ (C_U.d\ (trg_C\ f),\ C_U.P\ f))) \cdot_D$
$(F\ g \star_D F\ (C_U.counit_1\ f)) \cdot_D$
$(F\ g \star_D \Phi\ (f,\ C_U.d\ (src_C\ f)))$
$= F\ g \star_D \tau_1\ f$
using *f g fg 1 D.whisker-left* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
by *simp*
moreover have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ g)) \star_D F\ (C_U.P\ f)) \cdot_D$
$(F\ (C_U.counit_1\ g) \star_D F\ (C_U.P\ f)) \cdot_D$
$(\Phi\ (g,\ C_U.d\ (trg_C\ f)) \star_D F\ (C_U.P\ f))$
$= \tau_1\ g \star_D F\ (C_U.P\ f)$
using *f g fg 1 D.whisker-right* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
$C.VV.ide\text{-}char_{SbC}$ *Φ.components-are-iso* $C_U.emb.map\text{-}def$
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally have $F\ ((C_U.d\ (trg_C\ g) \star_C C_U.EoP.cmp\ (g, f)) \cdot_C$
$\mathrm{a}_C[C_U.d\ (trg_C\ g),\ C_U.P\ g,\ C_U.P\ f] \cdot_C$
$(C_U.\varepsilon_1\ g \star_C C_U.P\ f) \cdot_C$
$C.inv\ \mathrm{a}_C[g,\ C_U.d\ (src_C\ g),\ C_U.P\ f] \cdot_C$
$(g \star_C C_U.\varepsilon_1\ f) \cdot_C$
$\mathrm{a}_C[g, f,\ C_U.d\ (src_C\ f)])$
$= \Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f)) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(\tau_1\ g \star_D F\ (C_U.P\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f)))$
by *blast*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed

also have ... $= ((\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f))) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(\tau_1\ g \star_D F\ (C_U.P\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
proof −
have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)))$
$= \tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))$
proof −
have $(D.inv\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)))$
$= (\tau_0\ (trg_C\ g) \star_D F\ (C_U.P\ (g \star_C f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)))$
proof −
have $D.iso\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f)))$
using *f g fg* Φ*.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
by (*metis* (*no-types*, *lifting*) C_U*.prj.preserves-ide* $C_U.P\text{-}simps_B(3)$ $C_U.ide\text{-}char_{SbC}$ $C_U.counit.ide\text{-}map_0\text{-}obj$ $C_U.equivalence\text{-}data\text{-}simps_B(7)$ *C.hcomp-simps*(*2*) *C.ideD*(*1*) *C.ide-hcomp C.obj-trg cmp-components-are-iso*)
moreover have $D.dom\ (\Phi\ (C_U.d\ (trg_C\ g),\ C_U.P\ (g \star_C f))) =$
$\tau_0\ (trg_C\ g) \star_D F\ (C_U.P\ (g \star_C f))$
using *f g fg* $C_U.ide\text{-}char_{SbC}$ $C_U.equivalence\text{-}data\text{-}simps_B(7)$ C_U*.prj.preserves-ide cmp-simps*(*4*)
by *simp*
ultimately show *?thesis*
using *D.comp-inv-arr′* **by** *auto*
qed
also have ... $= \tau_0\ (trg_C\ g) \star_D F\ (C_U.P\ (g \star_C f)) \cdot_D F\ (C_U.EoP.cmp\ (g, f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$ $C_U.emb.map\text{-}def$ *D.whisker-left* [*of* $\tau_0\ (trg_C\ g)$]
by *simp*
also have ... $= \tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$ $C_U.emb.map\text{-}def$ *D.comp-cod-arr*
by *simp*
finally show *?thesis* **by** *blast*
qed
moreover have $\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D$
$(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D$
$(D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D$
$\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D$
$(\Phi\ (g, f) \star_D \tau_0\ (src_C\ f))$
$= \mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
proof −

have $D.inv\ (\Phi\ (g \star_C f,\ C_U.d\ (src_C\ f))) \cdot_D \Phi\ (g \star_C f,\ C_U.d\ (src_C\ f)) =$
$F\ (g \star_C f) \star_D F\ (C_U.d\ (src_C\ f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
moreover have $(F\ (g \star_C f) \star_D F\ (C_U.d\ (src_C\ f))) \cdot_D (\Phi\ (g, f) \star_D \tau_0\ (src_C\ f)) =$
$\Phi\ (g, f) \star_D \tau_0\ (src_C\ f)$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
by *simp*
moreover have $(D.inv\ (\Phi\ (g, f)) \star_D \tau_0\ (src_C\ f)) \cdot_D (\Phi\ (g, f) \star_D \tau_0\ (src_C\ f)) =$
$(F\ g \star_D F\ f) \star_D \tau_0\ (src_C\ f)$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
$\Phi.components\text{-}are\text{-}iso$ $C.VV.ide\text{-}char_{SbC}$ *FF-def*
$D.whisker\text{-}right$ [*of* $\tau_0\ (src_C\ f)$ $D.inv\ (\Phi\ (g, f))$ $\Phi\ (g, f)$]
by *simp*
moreover have $\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)] \cdot_D ((F\ g \star_D F\ f) \star_D \tau_0\ (src_C\ f)) =$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$ *D.comp-inv-arr′ D.comp-arr-dom D.comp-cod-arr*
by *simp*
ultimately show *?thesis* **by** *argo*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)) \cdot_D$
$\Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ F\ (C_U.P\ g),\ F\ (C_U.P\ f)] \cdot_D$
$(\tau_1\ g \star_D F\ (C_U.P\ f)) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (trg_C\ f),\ F\ (C_U.P\ f)] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
proof −
have $(\tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f))) \cdot_D$
$(\tau_0\ (trg_C\ g) \star_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$
$= \tau_0\ (trg_C\ g) \star_D F\ (C_U.EoP.cmp\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f)$
proof −
have $D.arr\ (F\ (C_U.EoP.cmp\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ $C_U.EoP.FF\text{-}def$
$C_U.emb.map\text{-}def$
by (*intro D.seqI*) *auto*
thus *?thesis*
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$
$D.whisker\text{-}left$ [*of* $\tau_0\ (trg_C\ g)$]
by *simp*

qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(\tau_0\ (trg_C\ g) \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp\ (g, f)) \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ g,$
$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f] \cdot_D$
$(\tau_1\ g \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f) \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (src_C\ g),\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f] \cdot_D$
$(F\ g \star_D \tau_1\ f) \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
proof −
have $EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp\ (g, f) =$
$F\ (C_U.EoP.cmp\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f)$
proof −
have $EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp\ (g, f) =$
$F\ (C_U.P\ (g \star_C f)) \cdot_D ((F\ (C_U.P\ (g \star_C f)) \cdot_D F\ (C_U.\Phi_P\ (g, f))) \cdot_D$
$\Phi\ (C_U.P\ g,\ C_U.P\ f)) \cdot_D D_V.\Phi_E\ (F\ (C_U.P\ g),\ F\ (C_U.P\ f))$
proof −
have *3*: $D.arr\ (F\ (C_U.P\ (g \star_C f)) \cdot_D F\ (C_U.\Phi_P\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *f g fg* $C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C.VV.cod\text{-}char_{SbC}$ *FF-def* $C_U.prj.FF\text{-}def$
$C_U.\Phi_P\text{-}in\text{-}hom$ [*of g f*] $C_U.hcomp\text{-}def$ $C_U.in\text{-}hom\text{-}char_{SbC}$
by (*intro D.seqI*) *auto*
have *4*: $D_V.arr\ (\Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *f g fg 1 cmp-in-hom*(*1*) [*of* $C_U.P\ g\ C_U.P\ f$] $C_U.prj.map_0\text{-}simps(1)$
$C_U.obj\text{-}char$ $D_V.arr\text{-}char_{SbC}$ *V-def*
by *auto*
moreover have *5*: $D_V.arr\ (F\ (C_U.\Phi_P\ (g, f)))$
using *f g fg 1* $C_U.\Phi_P\text{-}simps$ [*of g f*] $F_{UV}.preserves\text{-}arr$ **by** *presburger*
moreover have *6*: $D_V.arr\ (F\ (C_U.P\ (g \star_C f)))$
using *f g fg* $C_U.P\text{-}simps(1)$ [*of* $g \star_C f$] $F_{UV}.preserves\text{-}arr$ **by** *simp*
moreover have *7*: $D_V.arr\ (F\ (C_U.\Phi_P\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *3 4 5* $D_V.seq\text{-}char_{SbC}$ [*of* $F\ (C_U.\Phi_P\ (g, f))\ \Phi\ (C_U.P\ g,\ C_U.P\ f)$]
$D_V.comp\text{-}char$ [*of* $F\ (C_U.\Phi_P\ (g, f))\ \Phi\ (C_U.P\ g,\ C_U.P\ f)$]
by *auto*
moreover have $D.seq\ (F\ (C_U.\Phi_P\ (g, f)))\ (\Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *3* **by** *blast*
moreover have $D.seq\ (F\ (C_U.P\ (g \star_C f)))$
$(F\ (C_U.\Phi_P\ (g, f)) \cdot_D \Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *3* **by** *blast*
moreover have $D_V.arr\ (F\ (C_U.P\ (g \star_C f)) \cdot_D F\ (C_U.\Phi_P\ (g, f)) \cdot_D$
$\Phi\ (C_U.P\ g,\ C_U.P\ f))$
using *f g fg 3 6 7 D.vseq-implies-hpar V-def*
by (*metis* (*no-types*, *lifting*) $D_V.comp\text{-}char$ $D_V.seq\text{-}char_{SbC}$)
ultimately show *?thesis*
using *f g fg* $EQ_D oEQ_{UV} oEQ_C'.FH.cmp\text{-}def$ $EQ_{UV} oEQ_C'.FH.cmp\text{-}def$
$C.VV.arr\text{-}char_{SbC}$ $C.VV.dom\text{-}char_{SbC}$ $C_U.VV.arr\text{-}char_{SbC}$ $D_V.comp\text{-}char$
$D_V.emb.map\text{-}def$ *D.comp-assoc*

```
    by simp
qed
also have ... = F (C_U.P (g ⋆_C f)) ·_D F (C_U.Φ_P (g, f)) ·_D
                Φ (C_U.P g, C_U.P f) ·_D D_V.Φ_E (F (C_U.P g), F (C_U.P f))
proof −
  have F (C_U.P (g ⋆_C f)) ·_D F (C_U.Φ_P (g, f)) = F (C_U.Φ_P (g, f))
  proof −
    have 1: C.arr (C_U.Φ_P (g, f))
      using f g fg C.VV.arr-char_SbC
      by (metis (no-types, lifting) C_U.prj.preserves-hcomp C_U.prj.preserves-ide
          C_U.ide-char_SbC C_U.seq-char_SbC C.ideD(1) C.ideD(3) C.ide-hcomp C.seqE)
    moreover have C.cod (C_U.Φ_P (g, f)) = C_U.P (g ⋆_C f)
      using f g fg C.VV.arr-char_SbC C_U.Φ_P-in-hom(2) [of g f]
            C_U.hcomp-def C_U.in-hom-char_SbC by auto
    ultimately have D.cod (F (C_U.Φ_P (g, f))) = F (C_U.P (g ⋆_C f))
      using f g fg C.VV.arr-char_SbC preserves-cod [of C_U.Φ_P (g, f)]
      by simp
    thus ?thesis
      using 1 f g fg D.comp-cod-arr [of F (C_U.Φ_P (g, f)) F (C_U.P (g ⋆_C f))]
      by simp
  qed
  thus ?thesis
    using D.comp-assoc by simp
qed
also have ... = F (C_U.P (g ⋆_C f)) ·_D F (C_U.Φ_P (g, f)) ·_D
                F (C_U.Φ_E (C_U.P g, C_U.P f)) ·_D Φ (C_U.P g, C_U.P f)
proof −
  have D_V.Φ_E (F (C_U.P g), F (C_U.P f)) = F (C_U.P g) ⋆_D F (C_U.P f)
    using f g fg D_V.emb.cmp-def D_V.VV.arr-char_SbC D_V.src-def D_V.trg-def
          F_UV.preserves-arr
    by auto
  moreover have C_U.Φ_E (C_U.P g, C_U.P f) = C_U.P g ⋆_C C_U.P f
    using f g fg C_U.VV.arr-char_SbC C_U.emb.cmp-def by simp
  ultimately
  have Φ (C_U.P g, C_U.P f) ·_D D_V.Φ_E (F (C_U.P g), F (C_U.P f)) =
          F (C_U.Φ_E (C_U.P g, C_U.P f)) ·_D Φ (C_U.P g, C_U.P f)
    using f g fg C.VV.arr-char_SbC C.VV.dom-char_SbC C.VV.cod-char_SbC FF-def
          Φ.naturality [of (C_U.P g, C_U.P f)]
    by simp
  thus ?thesis
    using D.comp-assoc by simp
qed
also have ... = F (C_U.P (g ⋆_C f) ·_C C_U.Φ_P (g, f) ·_C C_U.Φ_E (C_U.P g, C_U.P f)) ·_D
                Φ (C_U.P g, C_U.P f)
proof −
  have C.arr (C_U.P (g ⋆_C f) ·_C C_U.Φ_P (g, f) ·_C C_U.Φ_E (C_U.P g, C_U.P f))
  proof (intro C.seqI)
    show C.arr (C_U.Φ_E (C_U.P g, C_U.P f))
      using f g fg by auto
```

```
  show 1: C.arr (C_U.Φ_P (g, f))
    using f g fg C_U.arr-char_SbC C_U.Φ_P-simps(1) [of g f] by auto
  show C.dom (C_U.Φ_P (g, f)) = C.cod (C_U.Φ_E (C_U.P g, C_U.P f))
  proof −
    have C.dom (C_U.Φ_P (g, f)) = C_U.hcomp (C_U.P g) (C_U.P f)
      using f g fg C_U.VV.arr-char_SbC C_U.Φ_P-simps(4) [of g f]
            C_U.dom-char_SbC [of C_U.Φ_P (g, f)]
      by fastforce
    also have ... = C_U.P g ⋆_C (C_U.P f)
      using f g fg by auto
    also have ... = C.cod (C_U.Φ_E (C_U.P g, C_U.P f))
      using f g fg C_U.emb.cmp-def [of (C_U.P g, C_U.P f)] by auto
    finally show ?thesis by blast
  qed
  show C.arr (C_U.P (g ⋆_C f))
    using f g fg by simp
  show C.dom (C_U.P (g ⋆_C f)) =
        C.cod (C_U.Φ_P (g, f) ·_C C_U.Φ_E (C_U.P g, C_U.P f))
  proof −
    have C.dom (C_U.P (g ⋆_C f)) = C_U.P (g ⋆_C f)
      using f g fg by simp
    also have ... = C.cod (C_U.Φ_P (g, f))
      using f g fg C_U.Φ_P-simps(5) [of g f] C_U.cod-char_SbC [of C_U.Φ_P (g, f)]
      by fastforce
    also have ... = C.cod (C_U.Φ_P (g, f) ·_C C_U.Φ_E (C_U.P g, C_U.P f))
    proof −
      have 2: C_U.hcomp (C_U.P g) (C_U.P f) = C_U.P g ⋆_C C_U.P f
        using f g fg by auto
      have C_U.arr (C_U.P g ⋆_C C_U.P f)
      proof −
        have C_U.arr (C_U.hcomp (C_U.P g) (C_U.P f))
          using f g fg by simp
        thus ?thesis
          using 2 by simp
      qed
      moreover have C.dom (C_U.Φ_P (g, f)) = C_U.P g ⋆_C C_U.P f
        using f g fg 2 C_U.VV.arr-char_SbC C_U.Φ_P-simps(4) [of g f]
              C_U.dom-char_SbC [of C_U.Φ_P (g, f)]
        by fastforce
      ultimately have C.arr (C_U.Φ_P (g, f) ·_C C_U.Φ_E (C_U.P g, C_U.P f))
        using f g fg 1 C_U.VV.arr-char_SbC C_U.VV.cod-char_SbC C_U.hcomp-char
              C_U.emb.map-def
        by auto
      thus ?thesis by simp
    qed
    finally show ?thesis by blast
  qed
qed
thus ?thesis
```

using $D.comp\text{-}assoc$ **by** *auto*
qed
also have ... $= F\ (C_U.EoP.cmp\ (g,\ f))\ \cdot_D\ \Phi\ (C_U.P\ g,\ C_U.P\ f)$
using $f\ g\ fg\ C_U.EoP.cmp\text{-}def\ C.VV.arr\text{-}char_{SbC}\ C.VV.dom\text{-}char_{SbC}\ C_U.emb.map\text{-}def$
by *simp*
finally show *?thesis* **by** *blast*
qed
moreover have $EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ g = F\ (C_U.P\ g)$
proof −
have $D_V.arr\ (F\ (C_U.P\ g))$
using $g\ D_V.arr\text{-}char_{SbC}\ C_U.P\text{-}simps(1)\ C.ideD(1)\ F_{UV}.preserves\text{-}arr$ **by** *presburger*
thus *?thesis*
using $g\ D_V.emb.map\text{-}def$ **by** *simp*
qed
moreover have $EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f = F\ (C_U.P\ f)$
proof −
have $D_V.arr\ (F\ (C_U.P\ f))$
using $f\ D_V.arr\text{-}char_{SbC}\ C_U.P\text{-}simps(1)\ C.ideD(1)\ F_{UV}.preserves\text{-}arr$ **by** *presburger*
thus *?thesis*
using $f\ D_V.emb.map\text{-}def$ **by** *simp*
qed
ultimately show *?thesis*
using *fg* **by** *argo*
qed
finally show $(\tau_0\ (trg_C\ g) \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp\ (g,\ f))\ \cdot_D$
$\mathrm{a}_D[\tau_0\ (trg_C\ g),\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ g,$
$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f]\ \cdot_D$
$(\tau_1\ g \star_D EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f)\ \cdot_D$
$D.inv\ \mathrm{a}_D[F\ g,\ \tau_0\ (src_C\ g),\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ f]\ \cdot_D$
$(F\ g \star_D \tau_1\ f)\ \cdot_D\ \mathrm{a}_D[F\ g,\ F\ f,\ \tau_0\ (src_C\ f)]$
$= \tau_1\ (g \star_C f)\ \cdot_D\ (\Phi\ (g,\ f) \star_D \tau_0\ (src_C\ f))$
by *argo*
qed
qed

interpretation *EQ*: *equivalence-of-bicategories-and-pseudonatural-equivalence-left*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.right\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.right\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.unit_0\ EQ_D oEQ_{UV} oEQ_C'.unit_1$
$EQ_D oEQ_{UV} oEQ_C'.counit_0\ EQ_D oEQ_{UV} oEQ_C'.counit_1$
$F\ \Phi\ \tau_0\ \tau_1$
proof −
interpret *E*: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.right\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.right\text{-}cmp$
$EQ_D oEQ_{UV} oEQ_C'.unit_0\ EQ_D oEQ_{UV} oEQ_C'.unit_1$
$EQ_D oEQ_{UV} oEQ_C'.counit_0\ EQ_D oEQ_{UV} oEQ_C'.counit_1$

using *induces-equivalence-of-bicategories* **by** *blast*
interpret τ: *pseudonatural-equivalence*
V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
$EQ_D oEQ_{UV} oEQ_C'$.*left-map* $EQ_D oEQ_{UV} oEQ_C'$.*left-cmp* F Φ τ_0 τ_1
..
show *equivalence-of-bicategories-and-pseudonatural-equivalence-left*
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
$EQ_D oEQ_{UV} oEQ_C'$.*left-map* $EQ_D oEQ_{UV} oEQ_C'$.*left-cmp*
$EQ_D oEQ_{UV} oEQ_C'$.*right-map* $EQ_D oEQ_{UV} oEQ_C'$.*right-cmp*
$EQ_D oEQ_{UV} oEQ_C'$.$unit_0$ $EQ_D oEQ_{UV} oEQ_C'$.$unit_1$
$EQ_D oEQ_{UV} oEQ_C'$.$counit_0$ $EQ_D oEQ_{UV} oEQ_C'$.$counit_1$
F Φ τ_0 τ_1
..
qed

definition *right-map*
where *right-map* $\equiv$ $EQ_D oEQ_{UV} oEQ_C'$.*right-map*

definition *right-cmp*
where *right-cmp* $\equiv$ $EQ_D oEQ_{UV} oEQ_C'$.*right-cmp*

definition $unit_0$
where $unit_0 \equiv EQ.unit.map_0$

definition $unit_1$
where $unit_1 \equiv EQ.unit.map_1$

definition $counit_0$
where $counit_0 \equiv EQ.counit.map_0$

definition $counit_1$
where $counit_1 \equiv EQ.counit.map_1$

theorem *extends-to-equivalence-of-bicategories*:
shows *equivalence-of-bicategories* V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F Φ *right-map right-cmp* $unit_0$ $unit_1$ $counit_0$ $counit_1$
unfolding *right-map-def right-cmp-def* $unit_0$*-def* $unit_1$*-def* $counit_0$*-def* $counit_1$*-def*
..

end

locale *converse-equivalence-pseudofunctor* =
C: *bicategory* V_C H_C a_C i_C src_C trg_C +
D: *bicategory* V_D H_D a_D i_D src_D trg_D +
F: *equivalence-pseudofunctor* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F
for V_C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and H_C :: $'c$ *comp* (**infixr** ‹$\star_C$› *53*)
and a_C :: $'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and i_C :: $'c \Rightarrow 'c$ (‹i_C[-]›)

and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and V_D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and H_D :: $'d$ *comp* (**infixr** ‹$\star_D$› *53*)
and a_D :: $'d \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and i_D :: $'d \Rightarrow 'd$ (‹i_D[-]›)
and src_D :: $'d \Rightarrow 'd$
and trg_D :: $'d \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$
and Φ_F :: $'c * 'c \Rightarrow 'd$
begin

interpretation E: *equivalence-of-bicategories*
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F Φ_F *F.right-map* *F.right-cmp* $F.unit_0$ $F.unit_1$ $F.counit_0$ $F.counit_1$
using *F.extends-to-equivalence-of-bicategories* **by** *simp*
interpretation E': *converse-equivalence-of-bicategories*
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F Φ_F *F.right-map* *F.right-cmp* $F.unit_0$ $F.unit_1$ $F.counit_0$ $F.counit_1$
..

sublocale *equivalence-pseudofunctor* V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F.right-map *F.right-cmp*
using E'*.equivalence-pseudofunctor-left* **by** *simp*

lemma *is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
F.right-map *F.right-cmp*
..

end

definition *equivalent-bicategories*
where *equivalent-bicategories* V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C $\equiv$
$\exists F\ \Phi.$ *equivalence-pseudofunctor*
V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C F Φ

lemma *equivalent-bicategories-reflexive*:
assumes *bicategory* V_C H_C a_C i_C src_C trg_C
shows *equivalent-bicategories* V_C H_C a_C i_C src_C trg_C V_C H_C a_C i_C src_C trg_C
proof −
interpret *bicategory* V_C H_C a_C i_C src_C trg_C
using *assms* **by** *simp*
interpret I: *identity-pseudofunctor* V_C H_C a_C i_C src_C trg_C **..**
show *?thesis*
using I*.equivalence-pseudofunctor-axioms* *equivalent-bicategories-def* **by** *blast*
qed

lemma *equivalent-bicategories-symmetric*:
assumes *equivalent-bicategories* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
shows *equivalent-bicategories* V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
proof −
  obtain F Φ_F **where** F: *equivalence-pseudofunctor*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F
    using *assms equivalent-bicategories-def* **by** *blast*
  interpret F: *equivalence-pseudofunctor*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F
    using F **by** *simp*
  interpret G: *converse-equivalence-pseudofunctor*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F
    ..
  show *?thesis*
    using *G.is-equivalence-pseudofunctor equivalent-bicategories-def* **by** *blast*
qed

lemma *equivalent-bicategories-transitive*:
assumes *equivalent-bicategories* V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C
and *equivalent-bicategories* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
shows *equivalent-bicategories* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
proof −
  obtain F Φ_F **where** F: *equivalence-pseudofunctor*
    V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F
    using *assms(1) equivalent-bicategories-def* **by** *blast*
  obtain G Φ_G **where** G: *equivalence-pseudofunctor*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G
    using *assms(2) equivalent-bicategories-def* **by** *blast*
  interpret F: *equivalence-pseudofunctor*
    V_B H_B a_B i_B src_B trg_B V_C H_C a_C i_C src_C trg_C F Φ_F
    using F **by** *simp*
  interpret G: *equivalence-pseudofunctor*
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D G Φ_G
    using G **by** *simp*
  interpret GF: *composite-equivalence-pseudofunctor* V_B H_B a_B i_B src_B trg_B
    V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ_F G Φ_G **..**
  show *equivalent-bicategories* V_B H_B a_B i_B src_B trg_B V_D H_D a_D i_D src_D trg_D
    using *equivalent-bicategories-def GF.equivalence-pseudofunctor-axioms* **by** *blast*
qed

end

Chapter 2

Bicategories of Spans

2.1 Span Bicategories

In this section we construct the bicategory *Span*(C), where C is a category with pullbacks. The 0-cells of *Span*(C) are the objects of C, the 1-cells of *Span*(C) are pairs (f_0, f_1) of arrows of C having a common domain, and the 2-cells of *Span*(C) are "arrows of spans". An arrow of spans from (f_0, f_1) to (g_0, g_1) is an arrow «u: $dom\ f_0 \to dom\ g_0$» of C, such that $g_0 \cdot u = f_0$ and $g_1 \cdot u = f_1$.

In the present development, a *span* is formalized as a structure $(\!|Leg0 = f_0,\ Leg1 = f_1|\!)$, where f_0 and f_1 are arrows of C with a common domain, which we call the *apex* of the span. An *arrow of spans* is formalized as a structure $(\!|Chn = u,\ Dom = S,\ Cod = T|\!)$, where S and T are spans, and «$u : S.apex \to T.apex$» satisfies $Leg0\ T \cdot u = Leg0\ S$ and $Leg1\ T \cdot u = Leg1\ S$. We refer to the arrow u as the *chine* of the arrow of spans.

Arrows of spans inherit a composition from that of C; this is "vertical composition". Spans may be composed via pullback in C; this "horizontal composition" extends to arrows of spans, so that it is functorial with respect to vertical composition. These two compositions determine a bicategory, as we shall show.

theory *SpanBicategory*
imports *Bicategory InternalAdjunction Category3.FreeCategory Category3.CategoryWithPullbacks*
begin

2.1.1 Spans

```
record 'a span-data =
  Leg0 :: 'a
  Leg1 :: 'a

locale span-in-category =
  C: category +
fixes S :: 'a span-data (structure)
assumes is-span: C.span (Leg0 S) (Leg1 S)
begin
```

abbreviation *leg0*
where *leg0* ≡ *Leg0 S*

abbreviation *leg1*
where *leg1* ≡ *Leg1 S*

abbreviation *src*
where *src* ≡ *C.cod leg0*

abbreviation *trg*
where *trg* ≡ *C.cod leg1*

definition *apex*
where *apex* ≡ *C.dom leg0*

lemma *ide-apex* [*simp*]:
shows *C.ide apex*
  using *is-span apex-def* **by** *simp*

lemma *leg-in-hom* [*intro*]:
shows «*leg0* : *apex* → *src*»
and «*leg1* : *apex* → *trg*»
  using *is-span apex-def* **by** *auto*

lemma *leg-simps* [*simp*]:
shows *C.arr leg0* **and** *C.dom leg0* = *apex*
and *C.arr leg1* **and** *C.dom leg1* = *apex*
  using *leg-in-hom* **by** *auto*

end

record *'a arrow-of-spans-data* =
  Chn :: *'a*
  Dom :: *'a span-data*
  Cod :: *'a span-data*

locale *arrow-of-spans* =
  C: *category C* +
  dom: *span-in-category C* ‹*Dom* μ› +
  cod: *span-in-category C* ‹*Cod* μ›
for *C* :: *'a comp* (**infixr** ‹·› *55*)
and μ :: *'a arrow-of-spans-data* (**structure**) +
assumes *chine-in-hom* [*intro*]: «*Chn* μ : *dom.apex* → *cod.apex*»
and *leg0-commutes* [*simp*]: *cod.leg0* · *Chn* μ = *dom.leg0*
and *leg1-commutes* [*simp*]: *cod.leg1* · (*Chn* μ) = *dom.leg1*
begin

abbreviation *chine*
where *chine* ≡ *Chn* μ

lemma *chine-simps* [*simp*]:
shows *C.arr chine* **and** *C.dom chine* = *dom.apex* **and** *C.cod chine* = *cod.apex*
using *chine-in-hom* **by** *auto*

lemma *cod-src-eq-dom-src* [*simp*]:
shows *cod.src* = *dom.src*
using *dom.is-span cod.is-span*
by (*metis C.cod-comp leg0-commutes*)

lemma *cod-trg-eq-dom-trg* [*simp*]:
shows *cod.trg* = *dom.trg*
using *dom.is-span cod.is-span*
by (*metis C.cod-comp leg1-commutes*)

abbreviation *dsrc*
where *dsrc* ≡ *dom.src*

abbreviation *dtrg*
where *dtrg* ≡ *dom.trg*

end

locale *identity-arrow-of-spans* =
arrow-of-spans +
assumes *chine-is-identity* [*simp*]: *C.ide* (*Chn* μ)
begin

abbreviation *apex*
where *apex* ≡ *dom.apex*

abbreviation *leg0*
where *leg0* ≡ *dom.leg0*

abbreviation *leg1*
where *leg1* ≡ *dom.leg1*

lemma *chine-eq-apex* [*simp*]:
shows *chine* = *apex*
using *chine-is-identity C.ideD*(*2*) *chine-simps*(*2*) **by** *presburger*

lemma *cod-simps* [*simp*]:
shows *cod.apex* = *apex* **and** *cod.leg0* = *leg0* **and** *cod.leg1* = *leg1*
using *chine-is-identity chine-simps*(*3*) *C.comp-arr-ide leg0-commutes leg1-commutes*
by *force*+

end

2.1.2 The Vertical Category of Spans

The following locale constructs the category of spans and arrows of spans in an underlying category C, which is not yet assumed to have pullbacks. The composition is vertical composition of arrows of spans, to which we will later add horizontal composition to obtain a bicategory.

locale *span-vertical-category* =
  C: *category*
begin

  abbreviation *Null*
  where *Null* ≡ ⦇*Chn* = *C.null*,
              Dom = ⦇*Leg0* = *C.null*, *Leg1* = *C.null*⦈,
              Cod = ⦇*Leg0* = *C.null*, *Leg1* = *C.null*⦈⦈

  lemma *not-arr-Null*:
  shows ¬ *arrow-of-spans C Null*
    unfolding *arrow-of-spans-def arrow-of-spans-axioms-def*
    by *auto*

Arrows of spans are composed simply by composing their chines.

  definition *vcomp*
  where *vcomp* ν μ ≡ *if arrow-of-spans C* μ ∧ *arrow-of-spans C* ν ∧ *Dom* ν = *Cod* μ
              then ⦇*Chn* = *Chn* ν · *Chn* μ, *Dom* = *Dom* μ, *Cod* = *Cod* ν⦈
              else Null

  notation *vcomp* (**infixr** ‹•› *55*)

  interpretation *V*: *partial-composition vcomp*
    using *not-arr-Null vcomp-def*
    apply *unfold-locales*
    by (*metis* (*no-types*, *opaque-lifting*))

  lemma *is-partial-composition*:
  shows *partial-magma vcomp*
    ..

  lemma *null-char*:
  shows *V.null* = *Null*
    using *V.null-def vcomp-def not-arr-Null*
    by (*metis* (*no-types*, *lifting*) *V.null-is-zero*(*2*))

Identities are arrows of spans whose chines are identities of C.

  lemma *ide-char*:
  shows *V.ide* μ ⟷ *arrow-of-spans C* μ ∧ *C.ide* (*Chn* μ)
  proof
    show *V.ide* μ ⟹ *arrow-of-spans C* μ ∧ *C.ide* (*Chn* μ)
    proof

```
  assume 0: V.ide μ
  have 1: vcomp μ μ ≠ Null ∧ (∀ν. (ν · μ ≠ Null ⟶ ν · μ = ν) ∧
                                (μ · ν ≠ Null ⟶ μ · ν = ν))
    using 0 V.ide-def null-char by simp
  show μ: arrow-of-spans C μ
    using 1 vcomp-def by metis
  interpret μ: arrow-of-spans C μ
    using μ by auto
  show C.ide (Chn μ)
  proof −
    have μ.chine · μ.chine ≠ C.null
      using 1 vcomp-def
      by (metis C.in-homE C.not-arr-null C.seqI μ.chine-in-hom)
    moreover have ⋀f. f · Chn μ ≠ C.null ⟹ f · Chn μ = f
    proof −
      fix f
      assume f · μ.chine ≠ C.null
      hence f: «f : μ.cod.apex → C.cod f»
        using C.ext C.in-homI by force
      let ?cod-μ = (|Chn = C.cod μ.chine, Dom = Cod μ, Cod = Cod μ|)
      interpret cod-μ: arrow-of-spans C ?cod-μ
        using C.ide-in-hom μ.cod.ide-apex μ.chine-in-hom C.comp-arr-dom
        by (unfold-locales, auto)
      have ?cod-μ · μ = ?cod-μ
        by (metis (no-types, opaque-lifting) 1 C.not-arr-null
            μ.cod.span-in-category-axioms arrow-of-spans-data.select-convs(2)
            cod-μ.arrow-of-spans-axioms span-data.select-convs(1)
            span-in-category.leg-simps(1) vcomp-def)
      thus f · μ.chine = f
        using C.comp-arr-ide C.comp-cod-arr μ cod-μ.arrow-of-spans-axioms f vcomp-def
        by auto
    qed
    moreover have ⋀f. μ.chine · f ≠ C.null ⟹ μ.chine · f = f
      by (metis C.comp-cod-arr C.comp-ide-arr C.ext C.ide-char' calculation(1−2))
    ultimately show C.ide μ.chine
      unfolding C.ide-def by simp
  qed
qed
show arrow-of-spans C μ ∧ C.ide (Chn μ) ⟹ V.ide μ
proof −
  assume μ: arrow-of-spans C μ ∧ C.ide (Chn μ)
  interpret μ: arrow-of-spans C μ
    using μ by auto
  have 1: Dom μ = Cod μ
    using μ identity-arrow-of-spans.cod-simps(2) identity-arrow-of-spans.cod-simps(3)
          identity-arrow-of-spans.intro identity-arrow-of-spans-axioms.intro
    by fastforce
  show V.ide μ
  proof −
```

```
      have μ · μ ≠ V.null
        using μ 1 vcomp-def by (simp add: C.ide-def null-char)
      moreover have ⋀ν. vcomp ν μ ≠ V.null ⟹ vcomp ν μ = ν
      proof −
        fix ν :: 'a arrow-of-spans-data
        assume ν: ν · μ ≠ V.null
        have 2: arrow-of-spans C ν ∧ Dom ν = Cod μ
          using ν 1 vcomp-def by (metis V.null-is-zero(2))
        interpret ν: arrow-of-spans C ν
          using 2 by auto
        show ν · μ = ν
          unfolding vcomp-def
          using μ 1 2 C.comp-arr-ide by simp
      qed
      moreover have ⋀ν. μ · ν ≠ V.null ⟹ μ · ν = ν
      proof −
        fix ν :: 'a arrow-of-spans-data
        assume ν: μ · ν ≠ V.null
        have 2: arrow-of-spans C ν ∧ Dom μ = Cod ν
          using ν 1 vcomp-def by (metis V.null-is-zero(1))
        interpret ν: arrow-of-spans C ν
          using 2 by auto
        show μ · ν = ν
          unfolding vcomp-def
          using μ 1 2 C.comp-ide-arr by simp
      qed
      ultimately show ?thesis
        unfolding V.ide-def by blast
    qed
  qed
qed

lemma has-domain-char:
shows V.domains μ ≠ {} ⟷ arrow-of-spans C μ
proof
  show V.domains μ ≠ {} ⟹ arrow-of-spans C μ
    using V.domains-def null-char vcomp-def by fastforce
  show arrow-of-spans C μ ⟹ V.domains μ ≠ {}
  proof −
    assume μ: arrow-of-spans C μ
    interpret μ: arrow-of-spans C μ
      using μ by auto
    let ?dom-μ = (|Chn = μ.dom.apex, Dom = Dom μ, Cod = Dom μ|)
    interpret dom-μ: arrow-of-spans C ?dom-μ
      using C.comp-arr-dom by (unfold-locales, auto)
    have ?dom-μ ∈ V.domains μ
    proof −
      have V.ide ?dom-μ
        using ide-char dom-μ.arrow-of-spans-axioms by simp
```

```
      moreover have μ · ?dom-μ ≠ V.null
        using μ vcomp-def μ.cod.span-in-category-axioms dom-μ.arrow-of-spans-axioms
              null-char span-in-category.leg-simps(1)
        by fastforce
      ultimately show ?thesis
        unfolding V.domains-def by blast
    qed
    thus V.domains μ ≠ {} by blast
  qed
qed

lemma has-codomain-char:
shows V.codomains μ ≠ {} ⟷ arrow-of-spans C μ
proof
  show V.codomains μ ≠ {} ⟹ arrow-of-spans C μ
    using V.codomains-def null-char vcomp-def by fastforce
  show arrow-of-spans C μ ⟹ V.codomains μ ≠ {}
  proof −
    assume μ: arrow-of-spans C μ
    interpret μ: arrow-of-spans C μ
      using μ by auto
    let ?cod-f = (|Chn = μ.cod.apex, Dom = Cod μ, Cod = Cod μ|)
    interpret cod-f: arrow-of-spans C ?cod-f
      using C.comp-arr-dom by (unfold-locales, auto)
    have ?cod-f ∈ V.codomains μ
    proof −
      have V.ide ?cod-f
        using ide-char cod-f.arrow-of-spans-axioms by simp
      moreover have ?cod-f · μ ≠ V.null
        using μ vcomp-def μ.cod.span-in-category-axioms cod-f.arrow-of-spans-axioms
              null-char span-in-category.leg-simps(1)
        by fastforce
      ultimately show ?thesis
        unfolding V.codomains-def by blast
    qed
    thus V.codomains μ ≠ {} by blast
  qed
qed

lemma arr-char:
shows V.arr μ ⟷ arrow-of-spans C μ
  unfolding V.arr-def
  using has-domain-char has-codomain-char by simp

lemma seq-char:
shows V.seq ν μ ⟷ arrow-of-spans C μ ∧ arrow-of-spans C ν ∧ Dom ν = Cod μ
proof
  show V.seq ν μ ⟹ arrow-of-spans C μ ∧ arrow-of-spans C ν ∧ Dom ν = Cod μ
    using vcomp-def by (metis V.not-arr-null null-char)
```

```
  show arrow-of-spans C μ ∧ arrow-of-spans C ν ∧ Dom ν = Cod μ ⟹ V.seq ν μ
  proof −
    assume 1: arrow-of-spans C μ ∧ arrow-of-spans C ν ∧ Dom ν = Cod μ
    interpret μ: arrow-of-spans C μ
      using 1 by auto
    interpret ν: arrow-of-spans C ν
      using 1 by auto
    show V.seq ν μ
    proof −
      let ?νμ = (|Chn = Chn ν · Chn μ, Dom = Dom μ, Cod = Cod ν|)
      have ν · μ = ?νμ
        using 1 vcomp-def by metis
      moreover have V.arr ?νμ
      proof −
        interpret Dom: span-in-category C ‹Dom ?νμ›
          by (simp add: μ.dom.span-in-category-axioms)
        interpret Cod: span-in-category C ‹Cod ?νμ›
          by (simp add: ν.cod.span-in-category-axioms)
        have arrow-of-spans C ?νμ
          using 1 μ.chine-in-hom ν.chine-in-hom C.comp-reduce
          by (unfold-locales, cases ?νμ, auto)
        thus ?thesis
          using arr-char by blast
      qed
      ultimately show ?thesis by simp
    qed
  qed
qed

interpretation V: category vcomp
proof
  show ⋀μ. (V.domains μ ≠ {}) = (V.codomains μ ≠ {})
    using has-domain-char has-codomain-char by simp
  show ⋀ν μ. ν · μ ≠ V.null ⟹ V.seq ν μ
    using seq-char vcomp-def null-char by metis
  show ⋀π ν μ. V.seq π ν ⟹ V.seq (π · ν) μ ⟹ V.seq ν μ
    using seq-char vcomp-def by (metis arrow-of-spans-data.select-convs(2))
  show ⋀π ν μ. V.seq π (ν · μ) ⟹ V.seq ν μ ⟹ V.seq π ν
    using seq-char vcomp-def by (metis arrow-of-spans-data.select-convs(3))
  show ⋀ν μ π. V.seq ν μ ⟹ V.seq π ν ⟹ V.seq (π · ν) μ
    using seq-char vcomp-def by (metis arr-char arrow-of-spans-data.select-convs(2))
  show ⋀ν μ π. V.seq ν μ ⟹ V.seq π ν ⟹ (π · ν) · μ = π · ν · μ
  proof −
    fix μ ν π
    assume μν: V.seq ν μ and νπ: V.seq π ν
    interpret μ: arrow-of-spans C μ
      using μν seq-char by auto
    interpret ν: arrow-of-spans C ν
      using μν seq-char by auto
```

```
    interpret π: arrow-of-spans C π
      using νπ seq-char by auto
    show (π · ν) · μ = π · ν · μ
      unfolding vcomp-def
      using μν νπ seq-char μ.chine-in-hom ν.chine-in-hom π.chine-in-hom
      by (simp add: C.comp-assoc, metis arr-char vcomp-def)
  qed
qed

lemma is-category:
shows category vcomp
  ..

lemma dom-char:
shows V.dom = (λμ. if V.arr μ then
                  ⦇Chn = span-in-category.apex C (Dom μ), Dom = Dom μ, Cod = Dom μ⦈
                else V.null)
proof
  fix μ
  have ¬ V.arr μ ⟹ V.dom μ = V.null
    by (simp add: V.arr-def V.dom-def)
  moreover have V.arr μ ⟹ V.dom μ = ⦇Chn = span-in-category.apex C (Dom μ),
                                     Dom = Dom μ, Cod = Dom μ⦈
  by (metis V.comp-arr-dom V.comp-ide-self V.ideD(1) V.ide-dom arrow-of-spans-data.cases
        arrow-of-spans-data.select-convs(1−3) ide-char identity-arrow-of-spans.chine-eq-apex
        identity-arrow-of-spans-axioms.intro identity-arrow-of-spans-def seq-char)
  ultimately show V.dom μ = (if V.arr μ then
                              ⦇Chn = span-in-category.apex C (Dom μ),
                               Dom = Dom μ, Cod = Dom μ⦈
                            else V.null)
    by argo
qed

lemma cod-char:
shows V.cod = (λμ. if V.arr μ then
                  ⦇Chn = span-in-category.apex C (Cod μ), Dom = Cod μ, Cod = Cod μ⦈
                else V.null)
proof
  fix μ
  have ¬ V.arr μ ⟹ V.cod μ = V.null
    by (simp add: V.arr-def V.cod-def)
  moreover have V.arr μ ⟹ V.cod μ = ⦇Chn = span-in-category.apex C (Cod μ),
                                     Dom = Cod μ, Cod = Cod μ⦈
    by (metis V.arr-cod V.comp-cod-arr V.dom-cod dom-char span-vertical-category.seq-char
          span-vertical-category-axioms)
  ultimately show V.cod μ = (if V.arr μ then
                              ⦇Chn = span-in-category.apex C (Cod μ),
                               Dom = Cod μ, Cod = Cod μ⦈
                            else V.null)
```

by *argo*
qed

lemma *vcomp-char*:
shows $vcomp = (\lambda\nu\ \mu.\ if\ V.seq\ \nu\ \mu\ then$
$(\!|Chn = Chn\ \nu \cdot Chn\ \mu,\ Dom = Dom\ \mu,\ Cod = Cod\ \nu|\!)$
$else\ V.null)$
by (*meson V.ext seq-char vcomp-def*)

lemma *vcomp-eq*:
assumes $V.seq\ \nu\ \mu$
shows $\nu \cdot \mu = (\!|Chn = Chn\ \nu \cdot Chn\ \mu,\ Dom = Dom\ \mu,\ Cod = Cod\ \nu|\!)$
using *assms vcomp-char* **by** *meson*

lemma *Chn-vcomp*:
assumes $V.seq\ \nu\ \mu$
shows $Chn\ (\nu \cdot \mu) = Chn\ \nu \cdot Chn\ \mu$
using *assms vcomp-eq* [*of* $\nu\ \mu$] **by** *simp*

lemma *ide-char'*:
shows $V.ide\ \mu \longleftrightarrow identity\text{-}arrow\text{-}of\text{-}spans\ C\ \mu$
using *arr-char ide-char identity-arrow-of-spans-axioms-def identity-arrow-of-spans-def identity-arrow-of-spans.axioms*(*1*) *identity-arrow-of-spans.chine-is-identity*
by *metis*

lemma *Chn-in-hom*:
assumes $V.in\text{-}hom\ \tau\ f\ g$
shows $C.in\text{-}hom\ (Chn\ \tau)\ (Chn\ f)\ (Chn\ g)$
by (*metis arr-char arrow-of-spans.chine-in-hom arrow-of-spans-data.select-convs*(*1*) *assms category.in-homE is-category span-vertical-category.cod-char span-vertical-category.dom-char span-vertical-category-axioms*)

abbreviation *mkIde*
where $mkIde\ f0\ f1 \equiv$
$(\!|Chn = C.dom\ f0,\ Dom = (\!|Leg0 = f0,\ Leg1 = f1|\!),\ Cod = (\!|Leg0 = f0,\ Leg1 = f1|\!)|\!)$

lemma *ide-mkIde*:
assumes $C.span\ f0\ f1$
shows $V.ide\ (mkIde\ f0\ f1)$
proof −
interpret *f*: *span-in-category C* ‹$(\!|Leg0 = f0,\ Leg1 = f1|\!)$›
using *assms* **by** (*unfold-locales*, *auto*)
interpret *ff*: *arrow-of-spans C* ‹*mkIde f0 f1*›
using *assms f.apex-def C.comp-arr-dom*
by (*unfold-locales*, *auto*)
show *?thesis*
using *assms ff.arrow-of-spans-axioms ide-char* **by** *simp*
qed

abbreviation *mkObj*
where *mkObj a* ≡ *mkIde a a*

lemma *ide-mkObj*:
assumes *C.ide a*
shows *V.ide* (*mkObj a*)
  using *assms ide-mkIde* [*of a a*] **by** *auto*

lemma *inverse-arrows*:
assumes *V.arr* μ **and** *C.iso* (*Chn* μ)
shows *V.inverse-arrows* μ (|*Chn* = *C.inv* (*Chn* μ), *Dom* = *Cod* μ, *Cod* = *Dom* μ|)
proof −
  interpret μ: *arrow-of-spans C* μ
    using *assms arr-char* **by** *auto*
  let *?*ν = (|*Chn* = *C.inv* (*Chn* μ), *Dom* = *Cod* μ, *Cod* = *Dom* μ|)
  interpret ν: *arrow-of-spans C ?*ν
    using *assms C.invert-side-of-triangle*(*2*) [*of* μ.*dom.leg0* μ.*cod.leg0* μ.*chine*]
        C.invert-side-of-triangle(*2*) [*of* μ.*dom.leg1* μ.*cod.leg1* μ.*chine*]
    by (*unfold-locales*, *auto*)
  show *V.inverse-arrows* μ *?*ν
  proof
    show *V.ide* (*?*ν · μ)
      by (*metis C.invert-side-of-triangle*(*1*) *Chn-vcomp* μ.*arrow-of-spans-axioms*
          μ.*chine-simps*(*1*) μ.*chine-simps*(*2*) μ.*dom.ide-apex* ν.*arrow-of-spans-axioms*
          arr-char select-convs(*1*−*2*) *assms*(*2*) *C.comp-arr-dom ide-char seq-char*)
    thus *V.ide* (μ · *?*ν)
      by (*metis C.comp-inv-arr′ C.inv-inv C.iso-inv-iso V.ide-compE* μ.*cod.ide-apex*
          ν.*chine-simps*(*2*) *arr-char select-convs*(*1*−*2*) *assms*(*2*) *ide-char*
          Chn-vcomp)
  qed
qed

lemma *iso-char*:
shows *V.iso* μ ⟷ *V.arr* μ ∧ *C.iso* (*Chn* μ)
proof
  show *V.iso* μ ⟹ *V.arr* μ ∧ *C.iso* (*Chn* μ)
    using *vcomp-eq ide-char*
    by (*metis C.iso-iff-section-and-retraction C.retractionI C.sectionI Chn-vcomp*
      V.arr-cod V.arr-dom V.comp-arr-inv′ V.comp-inv-arr′ V.ide-cod V.ide-dom
      V.iso-is-arr)
  show *V.arr* μ ∧ *C.iso* (*Chn* μ) ⟹ *V.iso* μ
    using *inverse-arrows* **by** *auto*
qed

lemma *inv-eq*:
assumes *V.iso* μ
shows *V.inv* μ = (|*Chn* = *C.inv* (*Chn* μ), *Dom* = *Cod* μ, *Cod* = *Dom* μ|)
  using *assms inverse-arrows iso-char* **by** (*simp add*: *V.inverse-unique*)

end

2.1.3 Putting Spans in Homs

context *span-vertical-category*
begin

interpretation V: *category vcomp*
using *is-category* **by** *simp*

definition *src*
where $src\ \mu \equiv if\ V.arr\ \mu\ then\ mkObj\ (C.cod\ (Leg0\ (Dom\ \mu)))\ else\ V.null$

lemma *ide-src* [*simp*]:
assumes $V.arr\ \mu$
shows $V.ide\ (src\ \mu)$
using *assms src-def arr-char ide-mkObj C.ide-cod*
by (*simp add*: *arrow-of-spans-def span-in-category.leg-simps*(*1*))

interpretation *src*: *endofunctor vcomp src*
proof
show $\bigwedge\mu.\ \neg\ V.arr\ \mu \Longrightarrow src\ \mu = V.null$
using *arr-char* **by** (*simp add*: *src-def null-char*)
show *1*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.arr\ (src\ \mu)$
using *ide-src* **by** *simp*
show *2*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.dom\ (src\ \mu) = src\ (V.dom\ \mu)$
using *1 arr-char src-def dom-char ide-src V.arr-dom V.ideD*(*2*) **by** *force*
show *3*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.cod\ (src\ \mu) = src\ (V.cod\ \mu)$
using *1 arr-char src-def cod-char ide-src V.arr-cod V.ideD*(*3*)
arrow-of-spans.cod-src-eq-dom-src
by *force*
show $\bigwedge\mu\ \nu.\ V.seq\ \nu\ \mu \Longrightarrow src\ (\nu \cdot \mu) = src\ \nu \cdot src\ \mu$
by (*metis* (*no-types*, *lifting*) *1 2 3 V.comp-ide-self V.dom-comp V.ideD*(*2*)
V.seqE span-vertical-category.ide-src span-vertical-category-axioms)
qed

lemma *src-is-endofunctor*:
shows *endofunctor vcomp src*
..

lemma *src-vcomp*:
assumes $V.seq\ \nu\ \mu$
shows $src\ (\nu \cdot \mu) = src\ \nu \cdot src\ \mu$
using *assms src.preserves-comp* **by** *simp*

definition *trg*
where $trg\ \mu \equiv if\ V.arr\ \mu\ then\ mkObj\ (C.cod\ (Leg1\ (Dom\ \mu)))\ else\ V.null$

lemma *ide-trg* [*simp*]:

assumes $V.arr\ \mu$
shows $V.ide\ (trg\ \mu)$
  using *assms trg-def arr-char ide-mkObj C.ide-cod*
  by (*simp add*: *arrow-of-spans-def span-in-category.leg-simps*(*3*))

interpretation *trg*: *endofunctor vcomp trg*
proof
  show $\bigwedge\mu.\ \neg\ V.arr\ \mu \Longrightarrow trg\ \mu = V.null$
    using *arr-char* **by** (*simp add*: *trg-def null-char*)
  show *1*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.arr\ (trg\ \mu)$
    using *ide-trg* **by** *simp*
  show *2*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.dom\ (trg\ \mu) = trg\ (V.dom\ \mu)$
    using *1 arr-char trg-def dom-char ide-trg V.arr-dom V.ideD*(*2*) **by** *force*
  show *3*: $\bigwedge\mu.\ V.arr\ \mu \Longrightarrow V.cod\ (trg\ \mu) = trg\ (V.cod\ \mu)$
    using *1 arr-char trg-def cod-char ide-trg V.arr-cod V.ideD*(*3*)
        arrow-of-spans.cod-trg-eq-dom-trg
    by *force*
  show $\bigwedge\mu\ \nu.\ V.seq\ \nu\ \mu \Longrightarrow trg\ (\nu \cdot \mu) = trg\ \nu \cdot trg\ \mu$
    by (*metis 2 3 V.comp-ide-self V.dom-comp V.ide-char V.seqE ide-trg*)
qed

lemma *trg-is-endofunctor*:
shows *endofunctor vcomp trg*
  ..

lemma *trg-vcomp*:
assumes $V.seq\ \nu\ \mu$
shows $trg\ (\nu \cdot \mu) = trg\ \nu \cdot trg\ \mu$
  using *assms trg.preserves-comp* **by** *simp*

lemma *src-trg-simps* [*simp*]:
assumes $V.arr\ \mu$
shows $src\ (src\ \mu) = src\ \mu$
and $src\ (trg\ \mu) = trg\ \mu$
and $trg\ (src\ \mu) = src\ \mu$
and $trg\ (trg\ \mu) = trg\ \mu$
proof −
  interpret μ: *arrow-of-spans C* μ
    using *assms arr-char* **by** *auto*
  have *1*: $V.arr\ (\!|Chn = \mu.dsrc,\ Dom = (\!|Leg0 = \mu.dsrc,\ Leg1 = \mu.dsrc|\!),$
        $Cod = (\!|Leg0 = \mu.dsrc,\ Leg1 = \mu.dsrc|\!)|\!)$
    using *ide-mkObj* **by** *auto*
  have *2*: $V.arr\ (\!|Chn = \mu.dtrg,\ Dom = (\!|Leg0 = \mu.dtrg,\ Leg1 = \mu.dtrg|\!),$
        $Cod = (\!|Leg0 = \mu.dtrg,\ Leg1 = \mu.dtrg|\!)|\!)$
    using *ide-mkObj* **by** *auto*
  show $src\ (src\ \mu) = src\ \mu$
    using *assms 1 src-def* **by** *simp*
  show $trg\ (src\ \mu) = src\ \mu$
    using *assms 1 src-def trg-def* **by** *simp*

show *src* (*trg* μ) = *trg* μ
using *assms 2 src-def trg-def* **by** *simp*
show *trg* (*trg* μ) = *trg* μ
using *assms 2 trg-def* **by** *simp*
qed

sublocale *horizontal-homs vcomp src trg*
by (*unfold-locales*, *simp-all*)

lemma *has-horizontal-homs*:
shows *horizontal-homs vcomp src trg*
..

lemma *obj-char*:
shows *obj a* $\longleftrightarrow$ *V.ide a* $\wedge$ *a* = *mkObj* (*Chn a*)
by (*metis C.dom-cod V.comp-ide-self V.ide-char arrow-of-spans.chine-simps*(*3*) *arrow-of-spans-data.select-convs*(*1*,*3*) *objE objI-trg span-data.select-convs*(*2*) *cod-char seq-char trg-def*)

end

2.1.4 Horizontal Composite of Spans

We now define the horizontal composite $S \star T$ of spans S and T, assuming that C is a category with chosen pullbacks. We think of Leg0 as an input and Leg1 as an output. The following then defines the composite span $S \star T$, with T on the "input side" of S. The notation is such that the p_0 projections of C are used for legs on the input (*i.e.* the "0") side and the p_1 projections are used for legs on the output (*i.e.* the "1") side.

locale *composite-span* =
C: *elementary-category-with-pullbacks* +
S: *span-in-category C S* +
T: *span-in-category C T*
for *S* (**structure**)
and *T* (**structure**) +
assumes *composable*: *C.cod* (*Leg0 S*) = *C.cod* (*Leg1 T*)
begin

abbreviation *this*
where *this* $\equiv$ (|*Leg0* = *T.leg0* $\cdot$ p_0[*S.leg0*, *T.leg1*], *Leg1* = *S.leg1* $\cdot$ p_1[*S.leg0*, *T.leg1*]|)

lemma *leg0-prj-in-hom*:
shows «*T.leg0* $\cdot$ p_0[*S.leg0*, *T.leg1*] : *S.leg0* $\downarrow\downarrow$ *T.leg1* $\rightarrow$ *C.cod* (*Leg0 T*)»
using *S.is-span T.is-span C.prj0-in-hom* [*of Leg0 S Leg1 T*] *composable* **by** *auto*

lemma *leg1-prj-in-hom*:
shows «*S.leg1* $\cdot$ p_1[*S.leg0*, *T.leg1*] : *S.leg0* $\downarrow\downarrow$ *T.leg1* $\rightarrow$ *C.cod* (*Leg1 S*)»
using *S.is-span T.is-span C.prj1-in-hom* [*of Leg0 S Leg1 T*] *composable* **by** *auto*

lemma *is-span* [*simp*]:
shows *span-in-category C this*
  using *leg0-prj-in-hom leg1-prj-in-hom*
  by (*unfold-locales*, *fastforce*)

sublocale *span-in-category C this*
  using *is-span* **by** *auto*

end

locale *span-bicategory* =
C: *elementary-category-with-pullbacks* +
  span-vertical-category
begin

definition *chine-hcomp*
where *chine-hcomp* ν μ $\equiv$
  $\langle$*Chn* ν $\cdot$ p_1[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]
    $[\![$*Leg0* (*Cod* ν), *Leg1* (*Cod* μ)$]\!]$
    Chn μ $\cdot$ p_0[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]$\rangle$

$$\begin{tikzcd}
 & \mathrm{src}(\mathrm{Dom}\ \nu) = \mathrm{trg}(\mathrm{Dom}\ \mu) & \\
 & \bullet \arrow[ul, "\mathrm{Leg0}(\mathrm{Dom}\ \nu)"'] \arrow[dl, "\mathrm{Leg1}(\mathrm{Dom}\ \nu)"'] \arrow[dd, "\mathrm{Chn}\ \nu"] & \bullet \arrow[ur, "\mathrm{Leg1}(\mathrm{Dom}\ \mu)"] \arrow[dr, "\mathrm{Leg0}(\mathrm{Dom}\ \mu)"] \arrow[dd, "\mathrm{Chn}\ \mu"'] \\
\mathrm{trg}\ \nu & & \mathrm{src}\ \mu \\
 & \bullet \arrow[ul, "\mathrm{Leg1}(\mathrm{Cod}\ \nu)"] \arrow[ur, "p_1"'] \arrow[dr, "p_0"] & \bullet \arrow[ur, "\mathrm{Leg0}(\mathrm{Cod}\ \mu)"'] \arrow[dl, "\mathrm{Leg1}(\mathrm{Cod}\ \mu)"'] \\
 & \mathrm{src}(\mathrm{Cod}\ \nu) = \mathrm{trg}(\mathrm{Cod}\ \mu) & \arrow[uu, dotted, "\mathrm{chn_hcomp}\ \mu\ \nu"]
\end{tikzcd}$$

definition *hcomp*
where *hcomp* ν μ $\equiv$ *if arr* μ $\wedge$ *arr* ν $\wedge$ *src* ν = *trg* μ *then*
    $(\!|$*Chn* = *chine-hcomp* ν μ,
      Dom = *composite-span.this C prj0 prj1* (*Dom* ν) (*Dom* μ),
      Cod = *composite-span.this C prj0 prj1* (*Cod* ν) (*Cod* μ)$|\!)$
  else
    null

notation *hcomp* (**infixr** ‹$\star$› *53*)

lemma *chine-hcomp-props*:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows «*chine-hcomp* ν μ : *Leg0* (*Dom* ν) $\downarrow\downarrow$ *Leg1* (*Dom* μ) $\rightarrow_C$ *Leg0* (*Cod* ν) $\downarrow\downarrow$ *Leg1* (*Cod* μ)»
and *C.commutative-square* (*Leg0* (*Cod* ν)) (*Leg1* (*Cod* μ))
(*Chn* ν · p_1[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)])
(*Chn* μ · p_0[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)])
and *C.commutative-square* p_1[*Leg0* (*Cod* ν), *Leg1* (*Cod* μ)] (*Chn* ν)
(*chine-hcomp* ν μ) p_1[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]
and *C.commutative-square* p_0[*Leg0* (*Cod* ν), *Leg1* (*Cod* μ)] (*Chn* μ)
(*chine-hcomp* ν μ) p_0[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]
and p_0[*Leg0* (*Cod* ν), *Leg1* (*Cod* μ)] · *chine-hcomp* ν μ =
Chn μ · p_0[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]
and p_1[*Leg0* (*Cod* ν), *Leg1* (*Cod* μ)] · *chine-hcomp* ν μ =
Chn ν · p_1[*Leg0* (*Dom* ν), *Leg1* (*Dom* μ)]
proof −
interpret μ: *arrow-of-spans C* μ
using *assms arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ν
using *assms arr-char* **by** *auto*
show *0*: *C.commutative-square* ν*.cod.leg0* μ*.cod.leg1*
(ν*.chine* · p_1[ν*.dom.leg0*, μ*.dom.leg1*]) (μ*.chine* · p_0[ν*.dom.leg0*, μ*.dom.leg1*])
using *assms src-def trg-def C.pullback-commutes C.comp-reduce C.commutative-square-def*
by *auto*
show *1*: p_1[ν*.cod.leg0*, μ*.cod.leg1*] · *chine-hcomp* ν μ = ν*.chine* · p_1[ν*.dom.leg0*, μ*.dom.leg1*]
unfolding *chine-hcomp-def*
using *0* **by** *simp*
show *2*: p_0[ν*.cod.leg0*, μ*.cod.leg1*] · *chine-hcomp* ν μ = μ*.chine* · p_0[ν*.dom.leg0*, μ*.dom.leg1*]
unfolding *chine-hcomp-def*
using *0* **by** *simp*
show *3*: «*chine-hcomp* ν μ : ν*.dom.leg0* $\downarrow\downarrow$ μ*.dom.leg1* $\rightarrow_C$ ν*.cod.leg0* $\downarrow\downarrow$ μ*.cod.leg1*»
unfolding *chine-hcomp-def*
using *assms 0 src-def trg-def C.tuple-in-hom* **by** *auto*
show *C.commutative-square* p_1[ν*.cod.leg0*, μ*.cod.leg1*] ν*.chine*
(*chine-hcomp* ν μ) p_1[ν*.dom.leg0*, μ*.dom.leg1*]
using *assms src-def trg-def 1 3* **by** *auto*
show *C.commutative-square* p_0[ν*.cod.leg0*, μ*.cod.leg1*] μ*.chine*
(*chine-hcomp* ν μ) p_0[ν*.dom.leg0*, μ*.dom.leg1*]
using *assms src-def trg-def 2 3* **by** *auto*
qed

lemma *chine-hcomp-in-hom* [*intro*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows «*chine-hcomp* ν μ : *Leg0* (*Dom* ν) $\downarrow\downarrow$ *Leg1* (*Dom* μ) $\rightarrow_C$ *Leg0* (*Cod* ν) $\downarrow\downarrow$ *Leg1* (*Cod* μ)»
using *assms chine-hcomp-props*(*1*) **by** *simp*

lemma *arrow-of-spans-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ

shows *arrow-of-spans C* $(\nu \star \mu)$
proof −
  interpret μ: *arrow-of-spans C* μ
    using *assms arr-char* **by** *auto*
  interpret ν: *arrow-of-spans C* ν
    using *assms arr-char* **by** *auto*
  show *?thesis*
  proof
    show *span-Dom*: *C.span* (*Leg0* (*Dom* $(\nu \star \mu)$)) (*Leg1* (*Dom* $(\nu \star \mu)$))
      using *assms src-def trg-def hcomp-def C.seqI′* **by** *auto*
    interpret *Dom*: *span-in-category C* ‹*Dom* $(\nu \star \mu)$›
      using *span-Dom* **by** (*unfold-locales*, *auto*)
    show *span-Cod*: *C.span* (*Leg0* (*Cod* $(\nu \star \mu)$)) (*Leg1* (*Cod* $(\nu \star \mu)$))
      using *assms hcomp-def src-def trg-def* **by** *auto*
    interpret *Cod*: *span-in-category C* ‹*Cod* $(\nu \star \mu)$›
      using *span-Cod* **by** (*unfold-locales*, *auto*)
    show *map*: «*Chn* $(\nu \star \mu)$: *Dom.apex* $\rightarrow_C$ *Cod.apex*»
      using *assms src-def trg-def chine-hcomp-props hcomp-def Cod.apex-def Dom.apex-def*
      by *auto*
    show *Cod.leg0* · *Chn* $(\nu \star \mu)$ = *Dom.leg0*
      by (*metis C.comp-assoc* μ*.leg0-commutes arrow-of-spans-data.select-convs*(*1*−*3*)
        assms(*1*−*3*) *chine-hcomp-props*(*5*) *hcomp-def span-data.select-convs*(*1*))
    show *Cod.leg1* · *Chn* $(\nu \star \mu)$ = *Dom.leg1*
    proof −
      have (ν*.cod.leg1* · p_1[ν*.cod.leg0*, μ*.cod.leg1*]) · *chine-hcomp* ν μ =
          ν*.dom.leg1* · p_1[ν*.dom.leg0*, μ*.dom.leg1*]
        by (*metis C.comp-assoc* ν*.leg1-commutes assms*(*1*−*3*) *chine-hcomp-props*(*6*))
      thus *?thesis*
      using *assms src-def trg-def hcomp-def chine-hcomp-props* ν*.chine-in-hom C.comp-reduce*
        by *auto*
    qed
  qed
qed

lemma *chine-hcomp-ide-arr*:
assumes *ide f* **and** *arr* μ **and** *src f* = *trg* μ
shows *chine-hcomp f* μ =
    ⟨p_1[*Leg0* (*Dom f*), *Leg1* (*Dom* μ)]
      ⟦*Leg0* (*Cod f*), *Leg1* (*Cod* μ)⟧
     Chn μ · p_0[*Leg0* (*Dom f*), *Leg1* (*Dom* μ)]⟩
proof −
  interpret μ: *arrow-of-spans C* μ
    using *assms arr-char* **by** *auto*
  interpret *f*: *arrow-of-spans C f*
    using *assms ide-char* **by** *auto*
  have *1*: *C.cospan f.dom.leg0* μ*.dom.leg1*
    using *assms src-def trg-def* **by** *auto*
  have *chine-hcomp f* μ = ⟨*f.chine* · p_1[*f.dom.leg0*, μ*.dom.leg1*]
                ⟦*f.cod.leg0*, μ*.cod.leg1*⟧

$\mu.chine \cdot \mathrm{p}_0[f.dom.leg0,\ \mu.dom.leg1]\rangle$
unfolding *chine-hcomp-def*
using *assms ide-char* **by** *simp*
moreover have $f.chine \cdot \mathrm{p}_1[f.dom.leg0,\ \mu.dom.leg1] = \mathrm{p}_1[f.dom.leg0,\ \mu.dom.leg1]$
using *assms 1 C.comp-ide-arr ide-char* **by** *auto*
ultimately show *?thesis* **by** *argo*
qed

lemma *chine-hcomp-arr-ide*:
assumes $arr\ \mu$ **and** $ide\ f$ **and** $src\ \mu = trg\ f$
shows $chine\text{-}hcomp\ \mu\ f =$
$\langle Chn\ \mu \cdot \mathrm{p}_1[Leg0\ (Dom\ \mu),\ Leg1\ (Dom\ f)]$
$[\![Leg0\ (Cod\ \mu),\ Leg1\ (Cod\ f)]\!]$
$\mathrm{p}_0[Leg0\ (Dom\ \mu),\ Leg1\ (Dom\ f)]\rangle$
proof −
interpret μ: $arrow\text{-}of\text{-}spans\ C\ \mu$
using *assms arr-char* **by** *auto*
interpret f: $arrow\text{-}of\text{-}spans\ C\ f$
using *assms ide-char* **by** *auto*
have *1*: $C.cospan\ \mu.dom.leg0\ f.dom.leg1$
using *assms src-def trg-def* **by** *auto*
have $chine\text{-}hcomp\ \mu\ f = \langle \mu.chine \cdot \mathrm{p}_1[\mu.dom.leg0,\ f.dom.leg1]$
$[\![\mu.cod.leg0,\ f.cod.leg1]\!]$
$f.chine \cdot \mathrm{p}_0[\mu.dom.leg0,\ f.dom.leg1]\rangle$
unfolding *chine-hcomp-def*
using *assms ide-char* **by** *simp*
moreover have $f.chine \cdot \mathrm{p}_0[\mu.dom.leg0,\ f.dom.leg1] = \mathrm{p}_0[\mu.dom.leg0,\ f.dom.leg1]$
using *assms 1 C.comp-ide-arr ide-char* **by** *auto*
ultimately show *?thesis* **by** *argo*
qed

lemma *chine-hcomp-ide-ide*:
assumes $ide\ g$ **and** $ide\ f$ **and** $src\ g = trg\ f$
shows $chine\text{-}hcomp\ g\ f = Leg0\ (Dom\ g) \downarrow\downarrow Leg1\ (Dom\ f)$
proof −
interpret g: $identity\text{-}arrow\text{-}of\text{-}spans\ C\ g$
using *assms ide-char′* **by** *auto*
interpret f: $identity\text{-}arrow\text{-}of\text{-}spans\ C\ f$
using *assms ide-char′* **by** *auto*
have *1*: $C.cospan\ g.dom.leg0\ f.dom.leg1$
using *assms src-def trg-def* **by** *auto*
have $chine\text{-}hcomp\ g\ f = \langle g.chine \cdot \mathrm{p}_1[g.dom.leg0,\ f.dom.leg1]$
$[\![g.cod.leg0,\ f.cod.leg1]\!]$
$\mathrm{p}_0[g.dom.leg0,\ f.dom.leg1]\rangle$
using *assms chine-hcomp-arr-ide* **by** *simp*
moreover have $g.chine \cdot \mathrm{p}_1[g.dom.leg0,\ f.dom.leg1] = \mathrm{p}_1[g.dom.leg0,\ f.dom.leg1]$
using *assms 1 C.comp-ide-arr ide-char* **by** *auto*
ultimately have $chine\text{-}hcomp\ g\ f = \langle \mathrm{p}_1[g.dom.leg0,\ f.dom.leg1]$
$[\![g.cod.leg0,\ f.cod.leg1]\!]$

$\mathrm{p}_0[g.dom.leg0,\ f.dom.leg1]\rangle$
by *simp*
also have ... =
$\langle \mathrm{p}_1[g.dom.leg0,\ f.dom.leg1] \cdot (g.dom.leg0 \downarrow\downarrow f.dom.leg1)$
$[\![g.cod.leg0,\ f.cod.leg1]\!]$
$\mathrm{p}_0[g.dom.leg0,\ f.dom.leg1] \cdot (g.dom.leg0 \downarrow\downarrow f.dom.leg1)\rangle$
using *assms 1 C.comp-arr-dom* **by** *simp*
also have ... = $g.dom.leg0 \downarrow\downarrow f.dom.leg1$
using *1 C.pullback-commutes C.tuple-prj* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *chine-hcomp-trg-arr*:
assumes *arr* μ
shows *chine-hcomp* (*trg* μ) μ =
$\langle \mathrm{p}_1[C.cod\ (Leg1\ (Dom\ \mu)),\ Leg1\ (Dom\ \mu)]$
$[\![C.cod\ (Leg1\ (Dom\ \mu)),\ Leg1\ (Cod\ \mu)]\!]$
$Chn\ \mu \cdot \mathrm{p}_0[C.cod\ (Leg1\ (Dom\ \mu)),\ Leg1\ (Dom\ \mu)]\rangle$
using *assms chine-hcomp-ide-arr ide-trg src-trg trg-def* **by** *force*

lemma *chine-hcomp-trg-ide*:
assumes *ide f*
shows *chine-hcomp* (*trg f*) $f = C.cod\ (Leg1\ (Dom\ f)) \downarrow\downarrow Leg1\ (Dom\ f)$
using *assms chine-hcomp-ide-ide ide-trg src-trg trg-def* **by** *force*

lemma *chine-hcomp-arr-src*:
assumes *arr* μ
shows *chine-hcomp* μ (*src* μ) =
$\langle Chn\ \mu \cdot \mathrm{p}_1[Leg0\ (Dom\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]$
$[\![Leg0\ (Cod\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]\!]$
$\mathrm{p}_0[Leg0\ (Dom\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]\rangle$
using *assms chine-hcomp-arr-ide ide-src src-def trg-src* **by** *force*

lemma *chine-hcomp-ide-src*:
assumes *ide f*
shows *chine-hcomp f* (*src f*) = $Leg0\ (Dom\ f) \downarrow\downarrow C.cod\ (Leg0\ (Dom\ f))$
using *assms chine-hcomp-ide-ide src.preserves-ide src-def trg-src* **by** *force*

lemma *src-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *src* ($\nu \star \mu$) = *src* μ
proof −
interpret μ: *arrow-of-spans C* μ
using *assms arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ν
using *assms arr-char* **by** *auto*
have $C.cod\ (\mu.dom.leg0 \cdot \mathrm{p}_0[\nu.dom.leg0,\ \mu.dom.leg1]) = C.cod\ \mu.dom.leg0$
using *assms C.commutative-squareE chine-hcomp-props*(*2*)
by (*metis* (*mono-tags*, *lifting*) *C.cod-comp C.match-3* μ*.leg0-commutes* μ*.dom.is-span*)

thus *?thesis*
using *assms arr-char hcomp-def src-def C.comp-cod-arr C.comp-arr-dom arrow-of-spans-hcomp*
by *simp*
qed

lemma *trg-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *trg* ($\nu \star \mu$) = *trg* ν
proof −
interpret μ: *arrow-of-spans C* μ
using *assms arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ν
using *assms arr-char* **by** *auto*
have *C.cod* (ν.*dom.leg1* · p_1[ν.*dom.leg0*, μ.*dom.leg1*]) = ν.*dtrg*
using *assms C.commutative-squareE chine-hcomp-props*(*2*)
by (*metis* (*mono-tags*, *lifting*) *C.cod-comp C.match-3* ν.*leg1-commutes* ν.*dom.is-span*)
thus *?thesis*
using *assms arr-char hcomp-def trg-def C.comp-cod-arr C.comp-arr-dom arrow-of-spans-hcomp*
by *simp*
qed

lemma *dom-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *dom* ($\nu \star \mu$) = *dom* $\nu \star$ *dom* μ
proof −
interpret μ: *arrow-of-spans C* μ
using *assms arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ν
using *assms arr-char* **by** *auto*
interpret $\nu\mu$: *arrow-of-spans C* ‹*hcomp* ν μ›
using *assms arr-char arrow-of-spans-hcomp* **by** *simp*
have *1*: *C.cospan* μ.*dom.leg1* ν.*dom.leg0*
using *assms* μ.*dom.is-span* ν.*dom.is-span src-def trg-def* **by** *auto*
have *dom* ($\nu \star \mu$) =
⦇*Chn* = ν.*dom.leg0* ↓↓ μ.*dom.leg1*,
Dom = ⦇*Leg0* = μ.*dom.leg0* · p_0[ν.*dom.leg0*, μ.*dom.leg1*],
Leg1 = ν.*dom.leg1* · p_1[ν.*dom.leg0*, μ.*dom.leg1*]⦈,
Cod = ⦇*Leg0* = μ.*dom.leg0* · p_0[ν.*dom.leg0*, μ.*dom.leg1*],
Leg1 = ν.*dom.leg1* · p_1[ν.*dom.leg0*, μ.*dom.leg1*]⦈⦈
using *assms* $\nu\mu$.*arrow-of-spans-axioms* $\nu\mu$.*dom.leg-simps*(*2*) $\nu\mu$.*dom.is-span*
arr-char dom-char hcomp-def
by *auto*
also have ... =
⦇*Chn* = *chine-hcomp* (*dom* ν) (*dom* μ),
Dom = ⦇*Leg0* = μ.*dom.leg0* · p_0[ν.*dom.leg0*, μ.*dom.leg1*],
Leg1 = ν.*dom.leg1* · p_1[ν.*dom.leg0*, μ.*dom.leg1*]⦈,
Cod = ⦇*Leg0* = μ.*dom.leg0* · p_0[ν.*dom.leg0*, μ.*dom.leg1*],
Leg1 = ν.*dom.leg1* · p_1[ν.*dom.leg0*, μ.*dom.leg1*]⦈⦈
using *assms src-dom trg-dom ide-dom dom-char chine-hcomp-ide-ide*

apply *auto*
by (*metis* (*no-types*, *lifting*) *arrow-of-spans-data.select-convs*(*2*))
also have ... = *dom* ν $\star$ *dom* μ
using *assms src-dom trg-dom arr-dom dom-char hcomp-def*
apply *auto*
by (*metis* (*no-types*, *lifting*))
finally show *?thesis* **by** *blast*
qed

lemma *cod-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *cod* (ν $\star$ μ) = *cod* ν $\star$ *cod* μ
proof −
interpret μ: *arrow-of-spans C* μ
using *assms arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ν
using *assms arr-char* **by** *auto*
interpret $\nu\mu$: *arrow-of-spans C* ‹*hcomp* ν μ›
using *assms arr-char arrow-of-spans-hcomp* **by** *simp*
have *1*: *C.cospan* μ*.cod.leg1* ν*.cod.leg0*
using *assms* μ*.cod.is-span* ν*.cod.is-span src-def trg-def* **by** *simp*
have *2*: *cod* (ν $\star$ μ) =
⦇*Chn* = ν*.cod.leg0* ↓↓ μ*.cod.leg1*,
Dom = ⦇*Leg0* = μ*.cod.leg0* · p_0[ν*.cod.leg0*, μ*.cod.leg1*],
Leg1 = ν*.cod.leg1* · p_1[ν*.cod.leg0*, μ*.cod.leg1*]⦈,
Cod = ⦇*Leg0* = μ*.cod.leg0* · p_0[ν*.cod.leg0*, μ*.cod.leg1*],
Leg1 = ν*.cod.leg1* · p_1[ν*.cod.leg0*, μ*.cod.leg1*]⦈⦈
using *assms* $\nu\mu$*.arrow-of-spans-axioms* $\nu\mu$*.cod.leg-simps*(*2*) $\nu\mu$*.cod.is-span arr-char cod-char hcomp-def*
by *auto*
also have ... =
⦇*Chn* = *chine-hcomp* (*cod* ν) (*cod* μ),
Dom = ⦇*Leg0* = μ*.cod.leg0* · p_0[ν*.cod.leg0*, μ*.cod.leg1*],
Leg1 = ν*.cod.leg1* · p_1[ν*.cod.leg0*, μ*.cod.leg1*]⦈,
Cod = ⦇*Leg0* = μ*.cod.leg0* · p_0[ν*.cod.leg0*, μ*.cod.leg1*],
Leg1 = ν*.cod.leg1* · p_1[ν*.cod.leg0*, μ*.cod.leg1*]⦈⦈
using *assms src-cod trg-cod ide-cod cod-char chine-hcomp-ide-ide*
apply *auto*
by (*metis* (*no-types*, *lifting*) *arrow-of-spans-data.select-convs*(*2*))
also have ... = *cod* ν $\star$ *cod* μ
using *assms src-cod trg-cod arr-cod cod-char hcomp-def*
apply *auto*
by (*metis* (*no-types*, *lifting*))
finally show *?thesis* **by** *simp*
qed

lemma *hcomp-vcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
and *arr* μ' **and** *arr* ν' **and** *src* ν' = *trg* μ'

and *seq* μ' μ **and** *seq* ν' ν
shows $(\nu' \cdot \nu) \star (\mu' \cdot \mu) = (\nu' \star \mu') \cdot (\nu \star \mu)$
proof −
  interpret μ: *arrow-of-spans C* μ **using** *assms arr-char* **by** *auto*
  interpret ν: *arrow-of-spans C* ν **using** *assms arr-char* **by** *auto*
  interpret μ': *arrow-of-spans C* μ' **using** *assms arr-char* **by** *auto*
  interpret ν': *arrow-of-spans C* ν' **using** *assms arr-char* **by** *auto*
  interpret $\nu\mu$: *arrow-of-spans C* ‹*hcomp* ν μ›
    using *assms arr-char arrow-of-spans-hcomp* **by** *auto*
  interpret $\nu'\mu'$: *arrow-of-spans C* ‹*hcomp* ν' μ'›
    using *assms arr-char arrow-of-spans-hcomp* **by** *auto*

  have *1*: $Dom\ \nu' = Cod\ \nu \wedge Dom\ \mu' = Cod\ \mu$
      using *assms src-def trg-def seq-char* **by** *blast*
  have *2*: $Dom\ (\mu' \cdot \mu) = Dom\ \mu \wedge Dom\ (\nu' \cdot \nu) = Dom\ \nu \wedge$
        $Cod\ (\mu' \cdot \mu) = Cod\ \mu' \wedge Cod\ (\nu' \cdot \nu) = Cod\ \nu'$
    using *assms seq-char arr-char vcomp-def*
    by (*metis arrow-of-spans-data.select-convs*(*2*) *arrow-of-spans-data.select-convs*(*3*))
  have *3*: *chine-hcomp* $(\nu' \cdot \nu)$ $(\mu' \cdot \mu) =$
        $\langle Chn\ (\nu' \cdot \nu) \cdot \mathrm{p}_1[\nu.dom.leg0,\ \mu.dom.leg1]$
          $[\![\nu'.cod.leg0,\ \mu'.cod.leg1]\!]$
         $Chn\ (\mu' \cdot \mu) \cdot \mathrm{p}_0[\nu.dom.leg0,\ \mu.dom.leg1]\rangle$
    unfolding *chine-hcomp-def* **using** *2* **by** *simp*

  have *C1*: *C.commutative-square* $\nu'.cod.leg0$ $\mu'.cod.leg1$
          $(Chn\ \nu' \cdot \mathrm{p}_1[\nu'.dom.leg0,\ \mu'.dom.leg1])$
          $(Chn\ \mu' \cdot \mathrm{p}_0[\nu'.dom.leg0,\ \mu'.dom.leg1])$
     using *assms 1 vcomp-def seq-char arr-char chine-hcomp-props*(*2*) **by** *blast*
  have *C2*: *C.commutative-square* $\nu'.cod.leg0$ $\mu'.cod.leg1$
          $(Chn\ (\nu' \cdot \nu) \cdot \mathrm{p}_1[\nu.dom.leg0,\ \mu.dom.leg1])$
          $(Chn\ (\mu' \cdot \mu) \cdot \mathrm{p}_0[\nu.dom.leg0,\ \mu.dom.leg1])$
    by (*metis 2 assms*(*3*,*6*−*8*) *chine-hcomp-props*(*2*) *src.as-nat-trans.preserves-comp-2*
          trg.as-nat-trans.preserves-comp-2 vseq-implies-hpar(*2*))
  have $(\nu' \cdot \nu) \star (\mu' \cdot \mu) =$
      $(\!|Chn = chine\text{-}hcomp\ (\nu' \cdot \nu)\ (\mu' \cdot \mu),$
       $Dom = (\!|Leg0 = \mu.dom.leg0 \cdot \mathrm{p}_0[\nu.dom.leg0,\ \mu.dom.leg1],$
            $Leg1 = \nu.dom.leg1 \cdot \mathrm{p}_1[\nu.dom.leg0,\ \mu.dom.leg1]|\!),$
       $Cod = (\!|Leg0 = \mu'.cod.leg0 \cdot \mathrm{p}_0[\nu'.cod.leg0,\ \mu'.cod.leg1],$
            $Leg1 = \nu'.cod.leg1 \cdot \mathrm{p}_1[\nu'.cod.leg0,\ \mu'.cod.leg1]|\!)|\!)$
    using *2 assms*(*3*,*6*−*8*) *hcomp-def src-vcomp trg-vcomp* **by** *presburger*
  moreover have $(\nu' \star \mu') \cdot (\nu \star \mu) =$
          $(\!|Chn = chine\text{-}hcomp\ \nu'\ \mu' \cdot chine\text{-}hcomp\ \nu\ \mu,$
           $Dom = (\!|Leg0 = \mu.dom.leg0 \cdot \mathrm{p}_0[\nu.dom.leg0,\ \mu.dom.leg1],$
                $Leg1 = \nu.dom.leg1 \cdot \mathrm{p}_1[\nu.dom.leg0,\ \mu.dom.leg1]|\!),$
           $Cod = (\!|Leg0 = \mu'.cod.leg0 \cdot \mathrm{p}_0[\nu'.cod.leg0,\ \mu'.cod.leg1],$
                $Leg1 = \nu'.cod.leg1 \cdot \mathrm{p}_1[\nu'.cod.leg0,\ \mu'.cod.leg1]|\!)|\!)$
    using *1* $\nu'\mu'$*.arrow-of-spans-axioms* $\nu\mu$*.arrow-of-spans-axioms assms*(*1*−*6*)
        hcomp-def span-vertical-category.seq-char span-vertical-category-axioms vcomp-eq
    by *fastforce*

moreover have *chine-hcomp* $(\nu' \cdot \nu)$ $(\mu' \cdot \mu)$ = *chine-hcomp* ν' μ' · *chine-hcomp* ν μ
proof −
have *C.cospan* ν'*.cod.leg0* μ'*.cod.leg1*
using *assms src-def trg-def* **by** *simp*
moreover have *C.seq* p_1[ν'*.cod.leg0*, μ'*.cod.leg1*] (*chine-hcomp* $(\nu' \cdot \nu)$ $(\mu' \cdot \mu)$)
using *assms 2 C2 chine-hcomp-props* [*of* $\mu' \cdot \mu$ $\nu' \cdot \nu$] **by** *auto*
moreover have *C.seq* p_1[ν'*.cod.leg0*, μ'*.cod.leg1*] (*chine-hcomp* ν' μ' · *chine-hcomp* ν μ)
using *assms 1 chine-hcomp-props* [*of* μ ν] *chine-hcomp-props* [*of* μ' ν'] **by** *auto*
moreover have p_0[ν'*.cod.leg0*, μ'*.cod.leg1*] · *chine-hcomp* $(\nu' \cdot \nu)$ $(\mu' \cdot \mu)$ =
p_0[ν'*.cod.leg0*, μ'*.cod.leg1*] · *chine-hcomp* ν' μ' · *chine-hcomp* ν μ
by (*metis* (*no-types*, *lifting*) *1 2 C.comp-assoc Chn-vcomp assms*(*1*−*8*)
chine-hcomp-props(*5*) *src-vcomp trg-vcomp*)
moreover have p_1[ν'*.cod.leg0*, μ'*.cod.leg1*] · *chine-hcomp* $(\nu' \cdot \nu)$ $(\mu' \cdot \mu)$ =
p_1[ν'*.cod.leg0*, μ'*.cod.leg1*] · *chine-hcomp* ν' μ' · *chine-hcomp* ν μ
by (*metis* (*no-types*, *lifting*) *1 3 C.comp-assoc C.prj-tuple*(*2*) *C1 C2 Chn-vcomp*
assms(*1*−*3*,*8*) *chine-hcomp-def chine-hcomp-props*(*6*))
ultimately show *?thesis*
using *C.prj-joint-monic*
[*of* ν'*.cod.leg0* μ'*.cod.leg1*
chine-hcomp $(\nu' \cdot \nu)$ $(\mu' \cdot \mu)$ *chine-hcomp* ν' μ' · *chine-hcomp* ν μ]
by *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

interpretation *H*: *functor VV.comp vcomp* ‹$\lambda\nu\mu$. *fst* $\nu\mu$ ⋆ *snd* $\nu\mu$›
proof
show $\bigwedge\nu\mu$. ¬ *VV.arr* $\nu\mu$ ⟹ *fst* $\nu\mu$ ⋆ *snd* $\nu\mu$ = *null*
using *hcomp-def* $VV.arr\text{-}char_{SbC}$ *null-char* **by** *auto*
show $\bigwedge\nu\mu$. *VV.arr* $\nu\mu$ ⟹ *arr* (*fst* $\nu\mu$ ⋆ *snd* $\nu\mu$)
using *arr-char arrow-of-spans-hcomp* $VV.arr\text{-}char_{SbC}$ **by** *simp*
show $\bigwedge\nu\mu$. *VV.arr* $\nu\mu$ ⟹
dom (*fst* $\nu\mu$ ⋆ *snd* $\nu\mu$) = *fst* (*VV.dom* $\nu\mu$) ⋆ *snd* (*VV.dom* $\nu\mu$)
using $VV.arr\text{-}char_{SbC}$ $VV.dom\text{-}char_{SbC}$ *dom-hcomp* **by** *auto*
show $\bigwedge\nu\mu$. *VV.arr* $\nu\mu$ ⟹ *cod* (*fst* $\nu\mu$ ⋆ *snd* $\nu\mu$) = *fst* (*VV.cod* $\nu\mu$) ⋆ *snd* (*VV.cod* $\nu\mu$)
using $VV.arr\text{-}char_{SbC}$ $VV.cod\text{-}char_{SbC}$ *cod-hcomp* **by** *auto*
show $\bigwedge\nu\mu'$ $\nu\mu$. *VV.seq* $\nu\mu'$ $\nu\mu$ ⟹ *fst* (*VV.comp* $\nu\mu'$ $\nu\mu$) ⋆ *snd* (*VV.comp* $\nu\mu'$ $\nu\mu$) =
(*fst* $\nu\mu'$ ⋆ *snd* $\nu\mu'$) · (*fst* $\nu\mu$ ⋆ *snd* $\nu\mu$)
proof −
fix $\nu\mu'$ $\nu\mu$
assume *1*: *VV.seq* $\nu\mu'$ $\nu\mu$
have *VV.comp* $\nu\mu'$ $\nu\mu$ = (*fst* $\nu\mu'$ · *fst* $\nu\mu$, *snd* $\nu\mu'$ · *snd* $\nu\mu$)
by (*metis* (*no-types*, *lifting*) *1 VV.comp-simp* $VV.seq\text{-}char_{SbC}$ *VxV.comp-char* $VxV.seqE_{PC}$)
thus *fst* (*VV.comp* $\nu\mu'$ $\nu\mu$) ⋆ *snd* (*VV.comp* $\nu\mu'$ $\nu\mu$) =
(*fst* $\nu\mu'$ ⋆ *snd* $\nu\mu'$) · (*fst* $\nu\mu$ ⋆ *snd* $\nu\mu$)
using *1 hcomp-vcomp* $VV.seq\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *VV.comp-char*
by (*metis* (*no-types*, *lifting*) *fst-conv snd-conv*)
qed
qed

lemma *hcomp-is-functor*:
shows *functor VV.comp vcomp* ($\lambda\nu\mu$. *fst* $\nu\mu$ $\star$ *snd* $\nu\mu$)
..

lemma *ide-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows *ide* ($f \star g$)
using *assms* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *H.preserves-ide* [*of* (f, g)] **by** *auto*

sublocale *horizontal-composition vcomp hcomp src trg*
using *src-hcomp trg-hcomp* $VV.arr\text{-}char_{SbC}$ *not-arr-null hcomp-def null-char*
by (*unfold-locales*, *auto*)

lemma *has-horizontal-composition*:
shows *horizontal-composition vcomp hcomp src trg*
..

end

2.1.5 The Bicategory Span(C)

context *span-bicategory*
begin

lemma *arr-eqI*:
assumes *par* μ μ' **and** *Chn* μ = *Chn* μ'
shows $\mu = \mu'$
using *assms dom-char cod-char* **by** *auto*

abbreviation l
where l f $\equiv$ (| *Chn* = p_0[*C.cod* (*Leg1* (*Dom f*)), *Leg1* (*Dom f*)],
Dom = *Dom* (*L f*), *Cod* = *Cod f* |)

interpretation l: *transformation-by-components vcomp vcomp L map* l
proof
have $*$: $\bigwedge f$. *ide f* $\Longrightarrow$ *arrow-of-spans C* (l f)
proof −
fix f
assume f: *ide f*
interpret f: *identity-arrow-of-spans C f*
using *f ide-char′* **by** *auto*
interpret lf: *arrow-of-spans C* ‹l f›
proof
show *Dom*: *C.span* (*Leg0* (*Dom* (l f))) (*Leg1* (*Dom* (l f)))
using f
by (*simp add*: *arrow-of-spans-hcomp arrow-of-spans.axioms*(*2*)
span-in-category.is-span)
interpret *Dom*: *span-in-category C* ‹*Dom* (l f)›

```
      using Dom by (unfold-locales, auto)
    show Cod: C.span (Leg0 (Cod (l f))) (Leg1 (Cod (l f)))
      using f hcomp-def trg-def src-def ide-mkObj C.pullback-commutes by force
    interpret Cod: span-in-category C ‹Cod (l f)›
      using Cod by (unfold-locales, auto)
    show «Chn (l f) : Dom.apex →C Cod.apex»
    proof −
      have C.dom Dom.leg0 = C.cod f.dom.leg1 ↓↓ f.dom.leg1
      proof −
        have arr (trg f)
          using f by simp
        hence Dom (l f) = ⦇Leg0 = f.dom.leg0 · p0[C.cod f.dom.leg1, f.dom.leg1],
                           Leg1 = C.cod f.dom.leg1 · p1[C.cod f.dom.leg1, f.dom.leg1]⦈
          using f src-def trg-def hcomp-def by simp
        thus ?thesis
          using f Dom hcomp-def by auto
      qed
      thus ?thesis
        using f ide-char Dom.apex-def Cod.apex-def by simp
    qed
    show Cod.leg0 · Chn (l f) = Dom.leg0
      using f ide-char hcomp-def src-def trg-def C.comp-arr-ide ide-mkObj by simp
    show Cod.leg1 · Chn (l f) = Dom.leg1
      using f ide-char hcomp-def src-def trg-def C.pullback-commutes ide-mkObj
            C.comp-arr-ide
      by (simp add: C.commutative-square-def)
  qed
  show arrow-of-spans C (l f) ..
qed
show 0: ⋀f. ide f ⟹ «l f : L f ⇒ map f»
proof −
  fix f
  assume f: ide f
  interpret f: identity-arrow-of-spans C f
    using f ide-char' by auto
  interpret lf: arrow-of-spans C ‹l f›
    using f * by blast
  show «l f : L f ⇒ map f»
  proof
    show 1: arr (l f)
      using f * arr-char by blast
    show dom (l f) = L f
      using f 1 dom-char ideD(2) by auto
    show cod (l f) = map f
      using f 1 cod-char ideD(3) by auto
   qed
qed
show ⋀μ. arr μ ⟹ l (cod μ) · L μ = map μ · l (dom μ)
proof −
```

```
    fix μ
    assume μ: arr μ
    interpret μ: arrow-of-spans C μ
      using μ arr-char by auto
    interpret l-dom-μ: arrow-of-spans C ‹l (dom μ)›
      using μ ∗ [of dom μ] by fastforce
    interpret l-cod-μ: arrow-of-spans C ‹l (cod μ)›
      using μ ∗ [of cod μ] by fastforce
    interpret Lμ: arrow-of-spans C ‹L μ›
      using μ arr-char by blast
    show l (cod μ) · L μ = map μ · l (dom μ)
    proof (intro arr-eqI)
      show par: par (l (cod μ) · L μ) (map μ · l (dom μ))
        using μ 0 [of dom μ] 0 [of cod μ] by fastforce
      show Chn (l (cod μ) · L μ) = Chn (map μ · l (dom μ))
      proof −
        have Chn (l (cod μ) · L μ) =
               p0[μ.dtrg, μ.cod.leg1] ·
                 ⟨p1[μ.dtrg, μ.dom.leg1] ⟦μ.dtrg, μ.cod.leg1⟧ μ.chine · p0[μ.dtrg, μ.dom.leg1]⟩
        proof −
          have Chn (l (cod μ) · L μ) = p0[μ.dtrg, μ.cod.leg1] · Chn (trg μ ⋆ μ)
            using Chn-vcomp μ cod-char par by force
          moreover
          have Chn (trg μ ⋆ μ) = ⟨p1[μ.dtrg, μ.dom.leg1]
                                  ⟦μ.dtrg, μ.cod.leg1⟧
                                  μ.chine · p0[μ.dtrg, μ.dom.leg1]⟩
            using μ hcomp-def chine-hcomp-trg-arr by simp
          ultimately show ?thesis
            using μ by (auto simp add: cod-char)
        qed
        also have ... = μ.chine · p0[C.cod μ.dom.leg1, μ.dom.leg1]
          using μ C.in-homE C.pullback-commutes [of C.cod μ.dom.leg1 μ.dom.leg1]
                C.comp-reduce ide-char C.prj-tuple(1)
          by auto
        also have ... = Chn (map μ · l (dom μ))
          using μ par seq-char dom-char vcomp-eq map-simp by simp
        finally show ?thesis by blast
      qed
    qed
  qed
qed

interpretation ℓ: natural-isomorphism vcomp vcomp L map ℓ.map
proof
  fix f
  assume f: ide f
  show iso (ℓ.map f)
  proof −
    interpret f: identity-arrow-of-spans C f
```

```
  using f ide-char′ by auto
have 1: ⦇.map f = ⦇Chn = p₀[f.dtrg, f.leg1], Dom = Dom (trg f ⋆ f), Cod = Dom f⦈
  using f ide-char cod-char by simp
interpret ⦇f: arrow-of-spans C ‹⦇.map f›
  using f arr-char ⦇.preserves-reflects-arr by fastforce
let ?⦇f′ = ⦇Chn = ⟨f.leg1 ⟦f.dtrg, f.leg1⟧ C.dom f.leg1⟩,
          Dom = Dom f, Cod = Dom (trg f ⋆ f)⦈
have 2: C.inverse-arrows ⦇f.chine (Chn ?⦇f′)
  using 1 C.pullback-arr-cod(2) [of f.leg1] by simp
interpret ⦇f′: arrow-of-spans C ?⦇f′
proof
  show Dom: C.span (Leg0 (Dom ?⦇f′)) (Leg1 (Dom ?⦇f′))
    using f 1 by auto
  interpret Dom: span-in-category C ‹Dom ?⦇f′›
    using Dom by (unfold-locales, auto)
  show Cod: C.span (Leg0 (Cod ?⦇f′)) (Leg1 (Cod ?⦇f′))
    using f 1 ⦇f.dom.is-span by auto
  interpret Cod: span-in-category C ‹Cod ?⦇f′›
    using Cod by (unfold-locales, auto)
  show «Chn ?⦇f′ : Dom.apex →C Cod.apex»
    using f src-def trg-def hcomp-def ide-mkObj Cod.apex-def Dom.apex-def
          C.comp-arr-dom C.comp-cod-arr
    by auto
  show Cod.leg0 · Chn ?⦇f′ = Dom.leg0
    using 2 ⦇f.leg0-commutes C.invert-side-of-triangle
    by (metis (no-types, lifting) 1 C.inverse-unique C.isoI ⦇f.dom.is-span
        arrow-of-spans-data.select-convs(2) arrow-of-spans-data.select-convs(3))
  show Cod.leg1 · Chn ?⦇f′ = Dom.leg1
    using 2 ⦇f.leg1-commutes C.invert-side-of-triangle
    by (metis (no-types, lifting) 1 C.inverse-unique C.isoI ⦇f.dom.is-span
        arrow-of-spans-data.select-convs(2) arrow-of-spans-data.select-convs(3))
qed
have inverse-arrows (⦇.map f) ?⦇f′
proof
  show ide (?⦇f′ · ⦇.map f)
  proof −
    have ?⦇f′ · ⦇.map f = dom (⦇.map f)
    proof −
      have ?⦇f′ · ⦇.map f =
            ⦇Chn = f.dtrg ↓↓ f.leg1, Dom = Dom (⦇.map f), Cod = Dom (⦇.map f)⦈
        using f 1 2 f.arrow-of-spans-axioms ⦇f.arrow-of-spans-axioms
              ⦇f′.arrow-of-spans-axioms vcomp-def ide-char arr-char
        by (simp add: vcomp-def C.comp-inv-arr)
      also have ... = dom (⦇.map f)
        by (metis (no-types, lifting) 1 C.pbdom-def ⦇f.chine-simps(2)
            ⦇.preserves-reflects-arr arrow-of-spans-data.select-convs(1)
            dom-char f ideD(1))
      finally show ?thesis by blast
    qed
```

```
      thus ?thesis
        using ⦇f.arrow-of-spans-axioms arr-char by simp
    qed
    show ide (⦇.map f · ?⦇f′)
    proof −
      have ⦇.map f · ?⦇f′ = dom ?⦇f′
      proof −
        have ⦇.map f · ?⦇f′ = ⦇Chn = Chn f, Dom = Dom ?⦇f′, Cod = Dom ?⦇f′⦈
          using f 1 2 f.arrow-of-spans-axioms ⦇f.arrow-of-spans-axioms
                ⦇f′.arrow-of-spans-axioms vcomp-def ide-char arr-char
          by fastforce
        also have ... = dom ?⦇f′
          using 1 ⦇f′.arrow-of-spans-axioms arr-char dom-char by simp
        finally show ?thesis by blast
      qed
      thus ?thesis
        using ⦇f′.arrow-of-spans-axioms arr-char by simp
    qed
  qed
  thus ?thesis by auto
 qed
qed

lemma ⦇-is-natural-isomorphism:
shows natural-isomorphism vcomp vcomp L map ⦇.map
  ..

sublocale L: equivalence-functor vcomp vcomp L
  using L.isomorphic-to-identity-is-equivalence ⦇.natural-isomorphism-axioms by simp

lemma equivalence-functor-L:
shows equivalence-functor vcomp vcomp L
  ..

abbreviation r
where r f ≡ ⦇Chn = p₁[Leg0 (Dom f), C.cod (Leg0 (Dom f))],
           Dom = Dom (R f), Cod = Cod f⦈

interpretation ϱ: transformation-by-components vcomp vcomp R map r
proof
  have ∗: ⋀f. ide f ⟹ arrow-of-spans C (r f)
  proof −
    fix f
    assume f: ide f
    interpret f: identity-arrow-of-spans C f
      using f ide-char′ by auto
    interpret rf: arrow-of-spans C ‹r f›
    proof
      show Dom: C.span (Leg0 (Dom (r f))) (Leg1 (Dom (r f)))
```

```
      using f
      by (simp add: arrow-of-spans-hcomp arrow-of-spans.axioms(2)
          span-in-category.is-span)
    interpret Dom: span-in-category C ‹Dom (r f)›
      using Dom by (unfold-locales, auto)
    show Cod: C.span (Leg0 (Cod (r f))) (Leg1 (Cod (r f)))
      using f hcomp-def trg-def src-def ide-mkObj C.pullback-commutes by force
    interpret Cod: span-in-category C ‹Cod (r f)›
      using Cod by (unfold-locales, auto)
    show «Chn (r f) : Dom.apex →C Cod.apex»
    proof −
      have C.dom Dom.leg0 = f.dom.leg0 ↓↓ C.cod f.dom.leg0
      proof −
        have arr (src f)
          using f by simp
        hence Dom (r f) = ⦇Leg0 = C.cod f.dom.leg0 · p0[f.dom.leg0, C.cod f.dom.leg0],
                           Leg1 = f.dom.leg1 · p1[f.dom.leg0, C.cod f.dom.leg0]⦈
          using f src-def trg-def by (simp add: hcomp-def)
        thus ?thesis
          using f ide-char Dom.apex-def Cod.apex-def by simp
      qed
      thus ?thesis
        using f ide-char Dom.apex-def Cod.apex-def by simp
    qed
    show Cod.leg0 · Chn (r f) = Dom.leg0
      using f ide-char hcomp-def src-def trg-def C.pullback-commutes
            ide-mkObj C.comp-arr-ide
      by (simp add: C.commutative-square-def)
    show Cod.leg1 · Chn (r f) = Dom.leg1
      using f ide-char hcomp-def src-def trg-def ide-mkObj C.comp-arr-ide
      by (simp add: C.commutative-square-def)
  qed
  show arrow-of-spans C (r f) ..
qed
show 0: ⋀f. ide f ⟹ «r f : R f ⇒ map f»
proof −
  fix f
  assume f: ide f
  interpret f: identity-arrow-of-spans C f
    using f ide-char' by auto
  interpret rf: arrow-of-spans C ‹r f›
    using f ∗ by blast
  show «r f : R f ⇒ map f»
  proof
    show 1: arr (r f)
      using f ∗ arr-char by blast
    show dom (r f) = R f
      using f 1 dom-char ideD(2) by auto
    show cod (r f) = map f
```

```
      using f 1 cod-char ideD(3) by auto
    qed
  qed
  show ⋀μ. arr μ ⟹ r (cod μ) · R μ = map μ · r (dom μ)
  proof −
    fix μ
    assume μ: arr μ
    interpret μ: arrow-of-spans C μ
      using μ arr-char by auto
    interpret r-dom-μ: arrow-of-spans C ‹r (dom μ)›
      using μ ∗ [of dom μ] by fastforce
    interpret r-cod-μ: arrow-of-spans C ‹r (cod μ)›
      using μ ∗ [of cod μ] by fastforce
    interpret Rμ: arrow-of-spans C ‹R μ›
      using μ arr-char by blast
    show r (cod μ) · R μ = map μ · r (dom μ)
    proof (intro arr-eqI)
      show par: par (r (cod μ) · R μ) (map μ · r (dom μ))
        using μ 0 [of dom μ] 0 [of cod μ] by force
      show Chn (r (cod μ) · R μ) = Chn (map μ · r (dom μ))
      proof −
        have Chn (r (cod μ) · R μ) =
              p₁[μ.cod.leg0, μ.cod.src] ·
                ⟨μ.chine · p₁[μ.dom.leg0, μ.dsrc] ⟦μ.cod.leg0, μ.cod.src⟧ p₀[μ.dom.leg0, μ.dsrc]⟩
        proof −
          have Chn (r (cod μ) · R μ) = p₁[μ.cod.leg0, μ.cod.src] · Chn (μ ⋆ src μ)
            using Chn-vcomp μ cod-char par by force
          moreover
          have Chn (μ ⋆ src μ) = ⟨μ.chine · p₁[μ.dom.leg0, μ.dsrc]
                                  ⟦μ.cod.leg0, μ.dsrc⟧
                                  p₀[μ.dom.leg0, μ.dsrc]⟩
            using μ hcomp-def chine-hcomp-arr-src by simp
          ultimately show ?thesis
            using μ by (auto simp add: cod-char)
        qed
        also have ... = μ.chine · p₁[μ.dom.leg0, C.cod μ.dom.leg0]
          using μ ide-char C.prj-tuple(2)
                C.in-homE C.pullback-commutes [of μ.dom.leg0 C.cod μ.dom.leg0]
                C.comp-reduce
          by auto
        also have ... = Chn (map μ · r (dom μ))
          using μ par seq-char dom-char vcomp-eq map-simp by simp
        finally show ?thesis by blast
      qed
    qed
  qed
qed

interpretation ϱ: natural-isomorphism vcomp vcomp R map ϱ.map
```

```
proof
  fix f
  assume f: ide f
  show iso (ϱ.map f)
  proof −
    interpret f: identity-arrow-of-spans C f
      using f ide-char′ by auto
    have 1: ϱ.map f = ⦇Chn = p₁[f.leg0, f.dsrc], Dom = Dom (f ⋆ src f), Cod = Dom f⦈
      using f ide-char by auto
    interpret ϱf: arrow-of-spans C ‹ϱ.map f›
      using f arr-char ϱ.preserves-reflects-arr by fastforce
    let ?ϱf′ = ⦇Chn = ⟨C.dom f.leg0 ⟦f.leg0, f.dsrc⟧ f.leg0⟩,
                Dom = Dom f, Cod = Dom (f ⋆ src f)⦈
    have 2: C.inverse-arrows (Chn (ϱ.map f)) (Chn ?ϱf′)
      using 1 C.pullback-arr-cod(1) [of f.dom.leg0] by simp
    interpret ϱf′: arrow-of-spans C ?ϱf′
    proof
      show Dom: C.span (Leg0 (Dom ?ϱf′)) (Leg1 (Dom ?ϱf′))
        using f 1 by auto
      interpret Dom: span-in-category C ‹Dom ?ϱf′›
        using Dom by (unfold-locales, auto)
      show Cod: C.span (Leg0 (Cod ?ϱf′)) (Leg1 (Cod ?ϱf′))
        using f 1 ϱf.dom.is-span by auto
      interpret Cod: span-in-category C ‹Cod ?ϱf′›
        using Cod by (unfold-locales, auto)
      show «Chn ?ϱf′ : Dom.apex →C Cod.apex»
        using f src-def trg-def hcomp-def ide-mkObj Cod.apex-def Dom.apex-def
              C.comp-arr-dom C.comp-cod-arr
        by auto
      show Cod.leg0 · Chn ?ϱf′ = Dom.leg0
        using 2 ϱf.leg0-commutes C.invert-side-of-triangle
        by (metis (no-types, lifting) 1 C.inverse-unique C.isoI ϱf.dom.is-span
            arrow-of-spans-data.select-convs(2) arrow-of-spans-data.select-convs(3))
      show Cod.leg1 · Chn ?ϱf′ = Dom.leg1
        using 2 ϱf.leg1-commutes C.invert-side-of-triangle
        by (metis (no-types, lifting) 1 C.inverse-unique C.isoI ϱf.dom.is-span
            arrow-of-spans-data.select-convs(2) arrow-of-spans-data.select-convs(3))
    qed
    have inverse-arrows (ϱ.map f) ?ϱf′
    proof
      show ide (?ϱf′ ⋅ ϱ.map f)
        using 2 C.comp-inv-arr ϱf′.arrow-of-spans-axioms ϱf.arrow-of-spans-axioms
              ϱf.chine-simps(2) dom-char f ideD(2) vcomp-def
        by force
      show ide (ϱ.map f ⋅ ?ϱf′)
      proof −
        have ϱ.map f ⋅ ?ϱf′ = dom ?ϱf′
        proof −
          have ϱ.map f ⋅ ?ϱf′ = ⦇Chn = Chn f, Dom = Dom ?ϱf′, Cod = Dom ?ϱf′⦈
```

```
          using f 1 2 f.arrow-of-spans-axioms
                ϱf.arrow-of-spans-axioms ϱf'.arrow-of-spans-axioms
                vcomp-def ide-char arr-char
          by fastforce
        also have ... = dom ?ϱf'
          using 1 ϱf'.arrow-of-spans-axioms arr-char dom-char by simp
        finally show ?thesis by blast
      qed
      thus ?thesis
        using ϱf'.arrow-of-spans-axioms arr-char by simp
    qed
  qed
  thus ?thesis by auto
 qed
qed

lemma ϱ-is-natural-isomorphism:
shows natural-isomorphism vcomp vcomp R map ϱ.map
 ..

sublocale R: equivalence-functor vcomp vcomp R
 using R.isomorphic-to-identity-is-equivalence ϱ.natural-isomorphism-axioms by simp

lemma equivalence-functor-R:
shows equivalence-functor vcomp vcomp R
 ..

definition unit (‹i[-]›)
where i[a] ≡ ⦇Chn = p0[Chn a, Chn a], Dom = Dom (a ⋆ a), Cod = Cod a⦈

lemma unit-in-hom [intro]:
assumes obj a
shows in-hhom i[a] a a
and «i[a] : a ⋆ a ⇒ a»
proof −
 show «i[a] : a ⋆ a ⇒ a»
 proof (intro in-homI)
  interpret a: identity-arrow-of-spans C a
   using assms obj-char ide-char' by auto
  have 0: src a = trg a
   using assms arr-char obj-char src-def trg-def by (elim objE, auto)
  interpret aa: arrow-of-spans C ‹a ⋆ a›
   using assms 0 a.arrow-of-spans-axioms arrow-of-spans-hcomp by auto
  interpret aa: identity-arrow-of-spans C ‹a ⋆ a›
  proof
   have ide (a ⋆ a)
    using assms 0 obj-char H.preserves-ide by simp
   thus C.ide aa.chine using ide-char by auto
  qed
```

```
    have 1: «p0[a.chine, a.chine] : a.chine ↓↓ a.chine →C a.chine» ∧
            «p1[a.chine, a.chine] : a.chine ↓↓ a.chine →C a.chine»
      by auto
    have 2: a.dom.leg0 = a.chine ∧ a.dom.leg1 = a.chine ∧
            a.cod.leg0 = a.chine ∧ a.cod.leg1 = a.chine
      using assms obj-char by (cases a, simp-all)
    have 3: a ⋆ a = ⦇Chn = a.chine ↓↓ a.chine,
                    Dom = ⦇Leg0 = p0[a.chine, a.chine], Leg1 = p1[a.chine, a.chine]⦈,
                    Cod = ⦇Leg0 = p0[a.chine, a.chine], Leg1 = p1[a.chine, a.chine]⦈⦈
      using assms 0 1 2 chine-hcomp-ide-ide hcomp-def C.comp-cod-arr
            a.identity-arrow-of-spans-axioms ide-char'
      by auto
    have aa.apex = a.chine ↓↓ a.chine
      using 3 aa.chine-eq-apex by auto
    interpret ia: arrow-of-spans C ‹i[a]›
    proof
      have 4: Dom i[a] = Dom (a ⋆ a)
        using assms hcomp-def unit-def by simp
      have 5: Cod i[a] = Cod a
        using assms unit-def by simp
      show Dom: C.span (Leg0 (Dom i[a])) (Leg1 (Dom i[a]))
        using 4 by simp
      interpret Dom: span-in-category C ‹Dom i[a]›
        using Dom by (unfold-locales, auto)
      show Cod: C.span (Leg0 (Cod i[a])) (Leg1 (Cod i[a]))
        using 5 by simp
      interpret Cod: span-in-category C ‹Cod i[a]›
        using Cod by (unfold-locales, auto)
      show «Chn i[a] : Dom.apex →C Cod.apex»
        by (simp add: ‹aa.apex = a.chine ↓↓ a.chine› unit-def)
      show Cod.leg0 · Chn i[a] = Dom.leg0
        unfolding unit-def using 1 2 3 C.comp-cod-arr by auto
      show Cod.leg1 · Chn i[a] = Dom.leg1
        unfolding unit-def using 1 2 3 C.comp-cod-arr C.pullback-ide-self by auto
    qed
    show arr i[a]
      using ia.arrow-of-spans-axioms arr-char by simp
    show dom i[a] = hcomp a a
      using 3 unit-def ia.arrow-of-spans-axioms arr-char dom-char ia.dom.apex-def
      by auto
    show cod i[a] = a
      using assms 3 obj-char arr-char dom-char cod-char unit-def
            ia.arrow-of-spans-axioms
      by auto
  qed
  thus in-hhom i[a] a a
    using assms
    by (metis arrI in-hhom-def objE vconn-implies-hpar(1) vconn-implies-hpar(2−4))
qed
```

lemma *unit-simps* [*simp*]:
assumes *obj a*
shows *src* i[*a*] = *a* **and** *trg* i[*a*] = *a*
and *dom* i[*a*] = *hcomp a a* **and** *cod* i[*a*] = *a*
using *assms unit-in-hom* **by** *auto*

lemma *iso-unit*:
assumes *obj a*
shows *iso* i[*a*]
proof −
have *Chn* i[*a*] = p_0[*Chn a*, *Chn a*]
unfolding *unit-def* **by** *simp*
moreover have *C.iso* p_0[*Chn a*, *Chn a*]
using *assms C.ide-is-iso C.iso-is-arr C.iso-pullback-ide ide-char* **by** *blast*
ultimately show *?thesis*
using *assms unit-in-hom iso-char* **by** *auto*
qed

end

locale *two-composable-arrows-of-spans* =
span-bicategory +
μ: *arrow-of-spans C* μ +
ν: *arrow-of-spans C* ν
for μ (**structure**)
and ν (**structure**) +
assumes *composable*: *src* μ = *trg* ν
begin

lemma *are-arrows* [*simp*]:
shows *arr* μ **and** *arr* ν
using *arr-char* μ*.arrow-of-spans-axioms* ν*.arrow-of-spans-axioms* **by** *auto*

lemma *legs-form-cospan*:
shows *C.cospan* μ*.dom.leg0* ν*.dom.leg1* **and** *C.cospan* μ*.cod.leg0* ν*.cod.leg1*
using *composable src-def trg-def* **by** *auto*

interpretation $\mu\nu$: *arrow-of-spans C* ‹$\mu \star \nu$›
using *arrow-of-spans-hcomp composable* **by** *auto*

lemma *composite-is-arrow* [*simp*]:
shows *arr* ($\mu \star \nu$)
using $\mu\nu$*.arrow-of-spans-axioms arr-char* **by** *auto*

lemma *composite-in-hom* [*intro*]:
shows «$\mu \star \nu$: *dom* $\mu \star$ *dom* $\nu \Rightarrow$ *cod* $\mu \star$ *cod* ν»
using *composable* **by** *auto*

lemma *composite-simps* [*simp*]:
shows *src* $(\mu \star \nu)$ = *src* ν **and** *trg* $(\mu \star \nu)$ = *trg* μ
and *dom* $(\mu \star \nu)$ = *dom* $\mu \star$ *dom* ν **and** *cod* $(\mu \star \nu)$ = *cod* $\mu \star$ *cod* ν
by (*simp-all add*: *composable*)

lemma *chine-composite*:
shows *Chn* $(\mu \star \nu) = \langle \mu.chine \cdot \mathrm{p}_1[\mu.dom.leg0,\ \nu.dom.leg1]$
$[\![\mu.cod.leg0,\ \nu.cod.leg1]\!]$
$\nu.chine \cdot \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1]\rangle$
unfolding *hcomp-def chine-hcomp-def* **using** *composable* **by** *simp*

lemma *chine-composite-in-hom* [*intro*]:
shows «*Chn* $(\mu \star \nu)$: $\mu.dom.leg0 \downarrow\downarrow \nu.dom.leg1 \rightarrow_C \mu.cod.leg0 \downarrow\downarrow \nu.cod.leg1$»
using *hcomp-def chine-hcomp-props*(*1*) *composable* **by** *auto*

end

sublocale *two-composable-arrows-of-spans* $\subseteq$ *arrow-of-spans C* ‹$\mu \star \nu$›
proof −
interpret *Domμ-Domν*: *composite-span C prj0 prj1* ‹*Dom* μ› ‹*Dom* ν›
using *legs-form-cospan*(*1*) **by** (*unfold-locales*, *auto*)
interpret *Codμ-Codν*: *composite-span C prj0 prj1* ‹*Cod* μ› ‹*Cod* ν›
using *legs-form-cospan*(*1*) **by** (*unfold-locales*, *auto*)
interpret *Dom-μν*: *span-in-category C* ‹*Dom* $(\mu \star \nu)$›
apply *unfold-locales* **apply** (*unfold hcomp-def*)
using *Domμ-Domν.apex-def Domμ-Domν.leg-simps*(*1*) *are-arrows*(*1*) *composable* **by** *auto*
interpret *Cod-μν*: *span-in-category C* ‹*Cod* $(\mu \star \nu)$›
apply *unfold-locales* **apply** (*unfold hcomp-def*)
using *Codμ-Codν.apex-def Codμ-Codν.leg-simps*(*1*) *are-arrows*(*1*) *composable* **by** *auto*
show *arrow-of-spans C* $(\mu \star \nu)$
proof
show «*Chn* (*hcomp* μ ν) : *Dom-μν.apex* $\rightarrow_C$ *Cod-μν.apex*»
unfolding *hcomp-def*
using *are-arrows*(*1*) *are-arrows*(*2*) *arrow-of-spans-hcomp composable hcomp-def*
arrow-of-spans.chine-in-hom Codμ-Codν.leg-simps(*4*) *Domμ-Domν.leg-simps*(*3*)
Domμ-Domν.leg-simps(*4*) *chine-composite-in-hom*
by *auto*
show *Cod-μν.leg0* · *Chn* (*hcomp* μ ν) = *Dom-μν.leg0*
proof (*unfold hcomp-def*)
have *arrow-of-spans C*
$(\!|$*Chn* = *chine-hcomp* μ ν, *Dom* = *Domμ-Domν.this*, *Cod* = *Codμ-Codν.this*$|\!)$
using *arrow-of-spans-hcomp composable hcomp-def* **by** *auto*
thus *Leg0* (*Cod* (*if arr* $\nu \wedge$ *arr* $\mu \wedge$ *src* μ = *trg* ν *then*
$(\!|$*Chn* = *chine-hcomp* μ ν,
Dom = *Domμ-Domν.this*, *Cod* = *Codμ-Codν.this*$|\!)$
else null)) ·
Chn (*if arr* $\nu \wedge$ *arr* $\mu \wedge$ *src* μ = *trg* ν *then*
$(\!|$*Chn* = *chine-hcomp* μ ν,
Dom = *Domμ-Domν.this*, *Cod* = *Codμ-Codν.this*$|\!)$

```
                    else null) =
          Leg0 (Dom (if arr ν ∧ arr μ ∧ src μ = trg ν then
                    ⦇Chn = chine-hcomp μ ν,
                     Dom = Domμ-Domν.this, Cod = Codμ-Codν.this⦈
                  else null))
        using arrow-of-spans.leg0-commutes composable by fastforce
    qed
    show Cod-μν.leg1 · Chn (hcomp μ ν) = Dom-μν.leg1
    proof (unfold hcomp-def)
      have arrow-of-spans C
            ⦇Chn = chine-hcomp μ ν, Dom = Domμ-Domν.this, Cod = Codμ-Codν.this⦈
        using arrow-of-spans-hcomp composable hcomp-def by force
      thus Leg1 (Cod (if arr ν ∧ arr μ ∧ src μ = trg ν then
                    ⦇Chn = chine-hcomp μ ν,
                     Dom = Domμ-Domν.this, Cod = Codμ-Codν.this⦈
                  else null)) ·
               Chn (if arr ν ∧ arr μ ∧ src μ = trg ν then
                    ⦇Chn = chine-hcomp μ ν,
                     Dom = Domμ-Domν.this, Cod = Codμ-Codν.this⦈
                  else null) =
          Leg1 (Dom (if arr ν ∧ arr μ ∧ src μ = trg ν then
                    ⦇Chn = chine-hcomp μ ν,
                     Dom = Domμ-Domν.this, Cod = Codμ-Codν.this⦈
                  else null))
        using arrow-of-spans.leg1-commutes composable by force
      qed
  qed
qed

locale two-composable-identity-arrows-of-spans =
   two-composable-arrows-of-spans +
μ: identity-arrow-of-spans C μ +
ν: identity-arrow-of-spans C ν
begin

  lemma are-identities [simp]:
  shows ide μ and ide ν
    using ide-char μ.arrow-of-spans-axioms ν.arrow-of-spans-axioms by auto

  interpretation H: functor VV.comp vcomp ‹λνμ. fst νμ ⋆ snd νμ›
    using hcomp-is-functor by auto

  interpretation μν: identity-arrow-of-spans C ‹μ ⋆ ν›
    using are-identities(1−2) composable ide-char′ by blast

  lemma ide-composite [simp]:
  shows ide (μ ⋆ ν)
    using μν.identity-arrow-of-spans-axioms arrow-of-spans-axioms ide-char by auto
```

lemma *apex-composite*:
shows $\mu\nu.apex = \mu.leg0 \downarrow\downarrow \nu.leg1$
using *dom.apex-def hcomp-def chine-hcomp-ide-ide composable legs-form-cospan*
by *simp*

lemma *leg0-composite*:
shows $\mu\nu.leg0 = \nu.leg0 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1]$
using *dom.apex-def hcomp-def composable legs-form-cospan* **by** *simp*

lemma *leg1-composite*:
shows $\mu\nu.leg1 = \mu.leg1 \cdot \mathrm{p}_1[\mu.leg0, \nu.leg1]$
using *dom.apex-def hcomp-def composable legs-form-cospan* **by** *simp*

lemma *chine-composite*:
shows $Chn\ (\mu \star \nu) = \mu.leg0 \downarrow\downarrow \nu.leg1$
unfolding *hcomp-def* **using** *chine-hcomp-ide-ide composable* **by** *simp*

abbreviation prj_0
where $prj_0 \equiv \mathrm{p}_0[\mu.leg0, \nu.leg1]$

abbreviation prj_1
where $prj_1 \equiv \mathrm{p}_1[\mu.leg0, \nu.leg1]$

lemma *prj-in-hom* [*intro*]:
shows «$prj_1 : \mu.leg0 \downarrow\downarrow \nu.leg1 \rightarrow_C \mu.apex$»
and «$prj_0 : \mu.leg0 \downarrow\downarrow \nu.leg1 \rightarrow_C \nu.apex$»
using *legs-form-cospan* **by** *auto*

lemma *prj-simps* [*simp*]:
shows $C.arr\ prj_1$ **and** $C.dom\ prj_1 = \mu.leg0 \downarrow\downarrow \nu.leg1$ **and** $C.cod\ prj_1 = \mu.apex$
and $C.arr\ prj_0$ **and** $C.dom\ prj_0 = \mu.leg0 \downarrow\downarrow \nu.leg1$ **and** $C.cod\ prj_0 = \nu.apex$
using *prj-in-hom* **by** *auto*

sublocale *identity-arrow-of-spans* C ‹$\mu \star \nu$›
using *apex-composite dom.ide-apex chine-composite* **by** (*unfold-locales*, *auto*)

end

locale *three-composable-arrows-of-spans* =
span-bicategory +
μ: *arrow-of-spans* C μ +
ν: *arrow-of-spans* C ν +
π: *arrow-of-spans* C π +
$\mu\nu$: *two-composable-arrows-of-spans* C *prj0 prj1* μ ν +
$\nu\pi$: *two-composable-arrows-of-spans* C *prj0 prj1* ν π
for μ (**structure**)
and ν (**structure**)
and π (**structure**)
begin

interpretation $\mu\nu\pi$: *arrow-of-spans C* ‹$\mu \star \nu \star \pi$›
  using *μ.arrow-of-spans-axioms $\nu\pi$.arrow-of-spans-axioms*
    arrow-of-spans-hcomp arr-char $\mu\nu$.composable $\nu\pi$.composable
  by *force*

interpretation $\mu\nu$-π: *arrow-of-spans C* ‹$(\mu \star \nu) \star \pi$›
  using *$\mu\nu$.arrow-of-spans-axioms π.arrow-of-spans-axioms*
    arrow-of-spans-hcomp arr-char $\mu\nu$.composable $\nu\pi$.composable
  by *force*

lemma *composites-are-arrows* [*simp*]:
shows *arr* $(\mu \star \nu \star \pi)$ **and** *arr* $((\mu \star \nu) \star \pi)$
  using *$\mu\nu\pi$.arrow-of-spans-axioms $\mu\nu$-π.arrow-of-spans-axioms arr-char* **by** *auto*

lemma *composite-in-hom* [*intro*]:
shows «$\mu \star \nu \star \pi$: *dom* $\mu \star$ *dom* $\nu \star$ *dom* $\pi \Rightarrow$ *cod* $\mu \star$ *cod* $\nu \star$ *cod* π»
and «$(\mu \star \nu) \star \pi$: (*dom* $\mu \star$ *dom* ν) $\star$ *dom* $\pi \Rightarrow$ (*cod* $\mu \star$ *cod* ν) $\star$ *cod* π»
  using *$\mu\nu$.composable $\nu\pi$.composable* **by** *auto*

lemma *composite-simps* [*simp*]:
shows *src* $(\mu \star \nu \star \pi) =$ *src* π
and *src* $((\mu \star \nu) \star \pi) =$ *src* π
and *trg* $(\mu \star \nu \star \pi) =$ *trg* μ
and *trg* $((\mu \star \nu) \star \pi) =$ *trg* μ
and *dom* $(\mu \star \nu \star \pi) =$ *dom* $\mu \star$ *dom* $\nu \star$ *dom* π
and *dom* $((\mu \star \nu) \star \pi) =$ (*dom* $\mu \star$ *dom* ν) $\star$ *dom* π
and *cod* $(\mu \star \nu \star \pi) =$ *cod* $\mu \star$ *cod* $\nu \star$ *cod* π
and *cod* $((\mu \star \nu) \star \pi) =$ (*cod* $\mu \star$ *cod* ν) $\star$ *cod* π
  by (*auto simp add*: *$\mu\nu$.composable $\nu\pi$.composable*)

lemma *chine-composite*:
shows *$\mu\nu\pi$.chine* =
    $\langle$*μ.chine* $\cdot$ p_1[*μ.dom.leg0*, *ν.dom.leg1* $\cdot$ p_1[*ν.dom.leg0*, *π.dom.leg1*]]
      $[\![$*μ.cod.leg0*, *ν.cod.leg1* $\cdot$ p_1[*ν.cod.leg0*, *π.cod.leg1*]$]\!]$
     $\langle$*ν.chine* $\cdot$ p_1[*ν.dom.leg0*, *π.dom.leg1*]
      $[\![$*ν.cod.leg0*, *π.cod.leg1*$]\!]$
     π.chine $\cdot$ p_0[*ν.dom.leg0*, *π.dom.leg1*]$\rangle$ $\cdot$
      p_0[*μ.dom.leg0*, *ν.dom.leg1* $\cdot$ p_1[*ν.dom.leg0*, *π.dom.leg1*]]$\rangle$
and *$\mu\nu$-π.chine* =
    $\langle\langle$*μ.chine* $\cdot$ p_1[*μ.dom.leg0*, *ν.dom.leg1*]
      $[\![$*μ.cod.leg0*, *ν.cod.leg1*$]\!]$
     ν.chine $\cdot$ p_0[*μ.dom.leg0*, *ν.dom.leg1*]$\rangle$ $\cdot$
      p_1[*ν.dom.leg0* $\cdot$ p_0[*μ.dom.leg0*, *ν.dom.leg1*], *π.dom.leg1*]
      $[\![$*ν.cod.leg0* $\cdot$ p_0[*μ.cod.leg0*, *ν.cod.leg1*], *π.cod.leg1*$]\!]$
     π.chine $\cdot$ p_0[*ν.dom.leg0* $\cdot$ p_0[*μ.dom.leg0*, *ν.dom.leg1*], *π.dom.leg1*]$\rangle$
proof −
  show *$\mu\nu\pi$.chine* =
    $\langle$*μ.chine* $\cdot$ p_1[*μ.dom.leg0*, *ν.dom.leg1* $\cdot$ p_1[*ν.dom.leg0*, *π.dom.leg1*]]

$[\![\mu.cod.leg0,\ \nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0,\ \pi.cod.leg1]]\!]$
$\langle\nu.chine \cdot \mathrm{p}_1[\nu.dom.leg0,\ \pi.dom.leg1]$
$[\![\nu.cod.leg0,\ \pi.cod.leg1]\!]$
$\pi.chine \cdot \mathrm{p}_0[\nu.dom.leg0,\ \pi.dom.leg1]\rangle \cdot$
$\mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1 \cdot \mathrm{p}_1[\nu.dom.leg0,\ \pi.dom.leg1]]\rangle$
unfolding *hcomp-def chine-hcomp-def* $\mu\nu$*.composable* $\nu\pi$*.composable*
using *trg-def* $\nu\pi$*.are-arrows*(*1*−*2*) $\nu\pi$*.composable* $\nu\pi$*.composite-is-arrow*
$\nu\pi$*.composite-simps*(*2*) *hcomp-def*
by (*simp add*: *chine-hcomp-def*)
show $\mu\nu$-π*.chine* =
$\langle\langle\mu.chine \cdot \mathrm{p}_1[\mu.dom.leg0,\ \nu.dom.leg1]$
$[\![\mu.cod.leg0,\ \nu.cod.leg1]\!]$
$\nu.chine \cdot \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1]\rangle \cdot$
$\mathrm{p}_1[\nu.dom.leg0 \cdot \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1],\ \pi.dom.leg1]$
$[\![\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1],\ \pi.cod.leg1]\!]$
$\pi.chine \cdot \mathrm{p}_0[\nu.dom.leg0 \cdot \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1],\ \pi.dom.leg1]\rangle$
unfolding *hcomp-def chine-hcomp-def* $\mu\nu$*.composable* $\nu\pi$*.composable*
using *src-def* $\mu\nu$*.are-arrows*(*1*−*2*) $\mu\nu$*.composable* $\mu\nu$*.composite-is-arrow*
$\mu\nu$*.composite-simps*(*1*) *hcomp-def* $\nu\pi$*.composable*
by (*simp add*: *chine-hcomp-def*)
qed

end

locale *three-composable-identity-arrows-of-spans* =
three-composable-arrows-of-spans +
μ: *identity-arrow-of-spans C* μ +
ν: *identity-arrow-of-spans C* ν +
π: *identity-arrow-of-spans C* π +
$\mu\nu$: *two-composable-identity-arrows-of-spans C prj0 prj1* μ ν +
$\nu\pi$: *two-composable-identity-arrows-of-spans C prj0 prj1* ν π
begin

lemma *composites-are-identities* [*simp*]:
shows *ide* $(\mu \star \nu \star \pi)$ **and** *ide* $((\mu \star \nu) \star \pi)$
by (*auto simp add*: $\mu\nu$*.composable* $\nu\pi$*.composable*)

interpretation $\mu\nu\pi$: *identity-arrow-of-spans C* ‹$\mu \star \nu \star \pi$›
using *composites-are-identities ide-char′* **by** *auto*
interpretation $\mu\nu$-π: *identity-arrow-of-spans C* ‹$(\mu \star \nu) \star \pi$›
using *composites-are-identities ide-char′* **by** *auto*

abbreviation Prj_{11}
where $Prj_{11} \equiv \mathrm{p}_1[\mu.leg0,\ \nu.leg1] \cdot \mathrm{p}_1[\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1],\ \pi.leg1]$
abbreviation Prj_{01}
where $Prj_{01} \equiv \mathrm{p}_0[\mu.leg0,\ \nu.leg1] \cdot \mathrm{p}_1[\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1],\ \pi.leg1]$
abbreviation Prj_0
where $Prj_0 \equiv \mathrm{p}_0[\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1],\ \pi.leg1]$

abbreviation Prj_1
where $Prj_1 \equiv \mathrm{p}_1[\mu.leg0,\ \nu.leg1 \cdot \mathrm{p}_1[\nu.leg0,\ \pi.leg1]]$
abbreviation Prj_{10}
where $Prj_{10} \equiv \mathrm{p}_1[\nu.leg0,\ \pi.leg1] \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1 \cdot \mathrm{p}_1[\nu.leg0,\ \pi.leg1]]$
abbreviation Prj_{00}
where $Prj_{00} \equiv \mathrm{p}_0[\nu.leg0,\ \pi.leg1] \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1 \cdot \mathrm{p}_1[\nu.leg0,\ \pi.leg1]]$

lemma *leg0-composite*:
shows $\mu\nu\pi.leg0 = \pi.leg0 \cdot Prj_{00}$
and $\mu\nu\text{-}\pi.leg0 = \pi.leg0 \cdot Prj_0$
proof −
  show $\mu\nu\pi.leg0 = \pi.leg0 \cdot Prj_{00}$
    using *hcomp-def* $\mu\nu$*.composable* $\nu\pi$*.composite-is-arrow* $\nu\pi$*.composite-simps*(*2*)
      C.comp-assoc
    by *auto*
  show $\mu\nu\text{-}\pi.leg0 = \pi.leg0 \cdot Prj_0$
    using *hcomp-def* $\mu\nu$*.composite-is-arrow* $\mu\nu$*.composite-simps*(*1*) $\nu\pi$*.composable* **by** *auto*
qed

lemma *leg1-composite*:
shows $\mu\nu\pi.leg1 = \mu.leg1 \cdot Prj_1$
and $\mu\nu\text{-}\pi.leg1 = \mu.leg1 \cdot Prj_{11}$
proof −
  show $\mu\nu\pi.leg1 = \mu.leg1 \cdot Prj_1$
    using *hcomp-def* $\mu\nu$*.composable* $\nu\pi$*.composite-is-arrow* $\nu\pi$*.composite-simps*(*2*) **by** *auto*
  show $\mu\nu\text{-}\pi.leg1 = \mu.leg1 \cdot Prj_{11}$
    using *hcomp-def* $\mu\nu$*.composite-is-arrow* $\mu\nu$*.composite-simps*(*1*) $\nu\pi$*.composable*
      C.comp-assoc
    by *auto*
qed

definition *chine-assoc*
where *chine-assoc* $\equiv$
  $\langle Prj_{11}\ [\![\mu.leg0,\ \nu.leg1 \cdot \mathrm{p}_1[\nu.leg0,\ \pi.leg1]]\!]\ \langle Prj_{01}\ [\![\nu.leg0,\ \pi.leg1]\!]\ Prj_0\rangle\rangle$

definition *chine-assoc′*
where *chine-assoc′* $\equiv$
  $\langle\langle Prj_1\ [\![\mu.leg0,\ \nu.leg1]\!]\ Prj_{10}\rangle\ [\![\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1],\ \pi.leg1]\!]\ Prj_{00}\rangle$

lemma *chine-composite*:
shows $\mu\nu\text{-}\pi.chine = \nu.leg0 \cdot \mu\nu.prj_0 \downarrow\downarrow \pi.leg1$
and $\mu\nu\pi.chine = \mu.leg0 \downarrow\downarrow \nu.leg1 \cdot \nu\pi.prj_1$
proof −
  show $\mu\nu\text{-}\pi.chine = \nu.leg0 \cdot \mu\nu.prj_0 \downarrow\downarrow \pi.leg1$
    using *hcomp-def chine-hcomp-arr-ide* [*of hcomp* μ ν π] *chine-hcomp-ide-ide*
      src-def trg-def $\mu\nu$*.composable* $\nu\pi$*.composable* $\mu\nu$*.ide-composite*
      $\mu\nu$*.are-identities* $\nu\pi$*.are-identities*(*2*)
    by *simp*

show $\mu\nu\pi.chine = \mu.leg0 \downarrow\downarrow \nu.leg1 \cdot \nu\pi.prj_1$
using *hcomp-def chine-hcomp-ide-arr* [*of* μ *hcomp* ν π] *chine-hcomp-ide-ide*
src-def trg-def $\mu\nu$.*composable* $\nu\pi$.*composable* $\nu\pi$.*ide-composite*
$\mu\nu$.*are-identities* $\nu\pi$.*are-identities*(*2*)
by *simp*
qed

lemma *prj-in-hom* [*intro*]:
shows «Prj_{11} : $\mu\nu$-π.*chine* $\rightarrow_C$ μ.*apex*»
and «Prj_{01} : $\mu\nu$-π.*chine* $\rightarrow_C$ ν.*apex*»
and «Prj_0 : $\mu\nu$-π.*chine* $\rightarrow_C$ π.*apex*»
and «Prj_1 : $\mu\nu\pi$.*chine* $\rightarrow_C$ μ.*apex*»
and «Prj_{10} : $\mu\nu\pi$.*chine* $\rightarrow_C$ ν.*apex*»
and «Prj_{00} : $\mu\nu\pi$.*chine* $\rightarrow_C$ π.*apex*»
using $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def chine-composite* **by** *auto*

lemma *prj-simps* [*simp*]:
shows C.*arr* Prj_{11}
and C.*arr* Prj_{01}
and C.*arr* Prj_0
and C.*dom* Prj_{11} = $\mu\nu$-π.*chine*
and C.*dom* Prj_{01} = $\mu\nu$-π.*chine*
and C.*dom* Prj_0 = $\mu\nu$-π.*chine*
and C.*cod* Prj_{11} = μ.*apex*
and C.*cod* Prj_{01} = ν.*apex*
and C.*cod* Prj_0 = π.*apex*
and C.*arr* Prj_1
and C.*arr* Prj_{10}
and C.*arr* Prj_{00}
and C.*dom* Prj_1 = $\mu\nu\pi$.*chine*
and C.*dom* Prj_{10} = $\mu\nu\pi$.*chine*
and C.*dom* Prj_{00} = $\mu\nu\pi$.*chine*
and C.*cod* Prj_1 = μ.*apex*
and C.*cod* Prj_{10} = ν.*apex*
and C.*cod* Prj_{00} = π.*apex*
using *prj-in-hom* **by** *auto*

lemma *chine-assoc-props*:
shows «*chine-assoc* : $\mu\nu$-π.*chine* $\rightarrow_C$ $\mu\nu\pi$.*chine*»
and $Prj_1 \cdot$ *chine-assoc* = Prj_{11}
and $Prj_{10} \cdot$ *chine-assoc* = Prj_{01}
and $Prj_{00} \cdot$ *chine-assoc* = Prj_0
proof −
have *1*: $\nu.leg0 \cdot Prj_{01} = \pi.leg1 \cdot Prj_0$
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def*
C.*pullback-commutes* [*of* $\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1]\ \pi.leg1$] C.*comp-assoc*
by *auto*
have *2*: $\mu.leg0 \cdot Prj_{11} = \nu.leg1 \cdot Prj_{01}$
by (*metis* C.*comp-assoc* C.*pullback-commutes'* $\mu\nu$.*legs-form-cospan*(*1*))

show «*chine-assoc* : $\mu\nu$-π.*chine* $\rightarrow_C$ $\mu\nu\pi$.*chine*»
unfolding *chine-assoc-def*
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable* *1 2*
src-def trg-def chine-composite C.comp-assoc **by** *auto*
show Prj_1 · *chine-assoc* = Prj_{11}
unfolding *chine-assoc-def*
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable* *1 2*
src-def trg-def C.comp-assoc
by *auto*
show Prj_{10} · *chine-assoc* = Prj_{01}
unfolding *chine-assoc-def*
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable* *1 2*
src-def trg-def C.comp-assoc
by *auto*
show Prj_{00} · *chine-assoc* = Prj_0
unfolding *chine-assoc-def*
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable* *1 2*
src-def trg-def C.comp-assoc
by *auto*
qed

lemma *chine-assoc-in-hom* [*intro*]:
shows «*chine-assoc* : $\mu\nu$-π.*chine* $\rightarrow_C$ $\mu\nu\pi$.*chine*»
using *chine-assoc-props*(*1*) **by** *simp*

lemma *prj-chine-assoc* [*simp*]:
shows Prj_1 · *chine-assoc* = Prj_{11}
and Prj_{10} · *chine-assoc* = Prj_{01}
and Prj_{00} · *chine-assoc* = Prj_0
using *chine-assoc-props*(*2*−*4*) **by** *auto*

lemma *chine-assoc′-props*:
shows «*chine-assoc′* : $\mu\nu\pi$.*chine* $\rightarrow_C$ $\mu\nu$-π.*chine*»
and Prj_{11} · *chine-assoc′* = Prj_1
and Prj_{01} · *chine-assoc′* = Prj_{10}
and Prj_0 · *chine-assoc′* = Prj_{00}
proof −
have *1*: μ.*leg0* · Prj_1 = ν.*leg1* · Prj_{10}
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable*
src-def trg-def C.pullback-commutes [*of* μ.*leg0* ν.*leg1* · p_1[ν.*leg0*, π.*leg1*]]
C.comp-assoc
by *auto*
have *2*: ν.*leg0* · Prj_{10} = π.*leg1* · Prj_{00}
by (*metis C.comp-assoc C.pullback-commutes′* $\nu\pi$.*legs-form-cospan*(*1*))
show «*chine-assoc′* : $\mu\nu\pi$.*chine* $\rightarrow_C$ $\mu\nu$-π.*chine*»
unfolding *chine-assoc′-def*
using $\mu\nu$.*are-identities* $\nu\pi$.*are-identities* $\mu\nu$.*composable* $\nu\pi$.*composable* *1 2*
src-def trg-def chine-composite C.comp-assoc
by *auto*

show $Prj_{11} \cdot$ *chine-assoc'* $= Prj_1$
unfolding *chine-assoc'-def*
using $\mu\nu$*.are-identities* $\nu\pi$*.are-identities* $\mu\nu$*.composable* $\nu\pi$*.composable 1 2*
src-def trg-def C.comp-assoc
by *auto*
show $Prj_{01} \cdot$ *chine-assoc'* $= Prj_{10}$
unfolding *chine-assoc'-def*
using $\mu\nu$*.are-identities* $\nu\pi$*.are-identities* $\mu\nu$*.composable* $\nu\pi$*.composable 1 2*
src-def trg-def C.comp-assoc
by *auto*
show $Prj_0 \cdot$ *chine-assoc'* $= Prj_{00}$
unfolding *chine-assoc'-def*
using $\mu\nu$*.are-identities* $\nu\pi$*.are-identities* $\mu\nu$*.composable* $\nu\pi$*.composable 1 2*
src-def trg-def C.comp-assoc
by *auto*
qed

lemma *chine-assoc'-in-hom* [*intro*]:
shows «*chine-assoc'* : $\mu\nu\pi$*.chine* $\rightarrow_C$ $\mu\nu$*-*π*.chine*»
using *chine-assoc'-props*(*1*) **by** *simp*

lemma *prj-chine-assoc'* [*simp*]:
shows $Prj_{11} \cdot$ *chine-assoc'* $= Prj_1$
and $Prj_{01} \cdot$ *chine-assoc'* $= Prj_{10}$
and $Prj_0 \cdot$ *chine-assoc'* $= Prj_{00}$
using *chine-assoc'-props*(*2*–*4*) **by** *auto*

lemma *prj-joint-monic*:
assumes «h: $a \rightarrow_C$ $\mu\nu$*-*π*.chine*» **and** «h': $a \rightarrow_C$ $\mu\nu$*-*π*.chine*»
and $Prj_{11} \cdot h = Prj_{11} \cdot h'$ **and** $Prj_{01} \cdot h = Prj_{01} \cdot h'$ **and** $Prj_0 \cdot h = Prj_0 \cdot h'$
shows $h = h'$
proof –
have $\mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h = \mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h'$
proof –
have $\mu\nu.prj_0 \cdot \mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h = \mu\nu.prj_0 \cdot \mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h'$
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def chine-composite*(*1*)
C.comp-assoc
by *force*
moreover
have $\mu\nu.prj_1 \cdot \mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h = \mu\nu.prj_1 \cdot \mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h'$
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def chine-composite*(*1*)
C.comp-assoc
by *force*
ultimately show *?thesis*
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def*
chine-composite(*1*) *cod-char*
C.prj-joint-monic
[*of* μ*.leg0* ν*.leg1* $\mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h$
$\mathrm{p}_1[\nu.leg0 \cdot \mu\nu.prj_0, \pi.leg1] \cdot h'$]

by *auto*
qed
moreover have $Prj_0 \cdot h = Prj_0 \cdot h'$
using *assms cod-char* **by** *simp*
ultimately show *?thesis*
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def*
chine-composite(*1*) *cod-char*
C.prj-joint-monic [*of* ν*.leg0* $\cdot$ $\mu\nu.prj_0$ π*.leg1 h h'*]
by *auto*
qed

lemma *prj'-joint-monic*:
assumes «h: $a \rightarrow_C \mu\nu\pi.chine$» **and** «$h'$: $a \rightarrow_C \mu\nu\pi.chine$»
and $Prj_1 \cdot h = Prj_1 \cdot h'$ **and** $Prj_{10} \cdot h = Prj_{10} \cdot h'$ **and** $Prj_{00} \cdot h = Prj_{00} \cdot h'$
shows $h = h'$
proof −
have $\mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h = \mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h'$
proof −
have $\nu\pi.prj_0 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h = \nu\pi.prj_0 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h'$
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def chine-composite*(*2*)
C.comp-assoc
by *force*
moreover
have $\nu\pi.prj_1 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h = \nu\pi.prj_1 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h'$
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def chine-composite*(*2*)
C.comp-assoc
by *force*
ultimately show *?thesis*
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def*
chine-composite(*2*) *cod-char*
C.prj-joint-monic
[*of* ν*.leg0* π*.leg1* $\mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h$
$\mathrm{p}_0[\mu.leg0, \nu.leg1 \cdot \nu\pi.prj_1] \cdot h'$]
by *auto*
qed
moreover have $Prj_1 \cdot h = Prj_1 \cdot h'$
using *assms cod-char* **by** *simp*
ultimately show *?thesis*
using *assms* $\mu\nu$*.composable* $\nu\pi$*.composable src-def trg-def chine-composite*(*2*)
C.prj-joint-monic [*of* μ*.leg0* ν*.leg1* $\cdot$ $\nu\pi.prj_1$ *h h'*]
by *auto*
qed

lemma *chine-assoc-inverse*:
shows *C.inverse-arrows chine-assoc chine-assoc'*
proof
show *C.ide* (*chine-assoc'* $\cdot$ *chine-assoc*)
proof −
have *1*: *C.ide* $\mu\nu$-π*.chine*

using *chine-assoc-props*(*1*) *C.ide-dom* **by** *fastforce*
have *chine-assoc′* · *chine-assoc* = *μν-π.chine*
proof −
have *2*: *C.seq chine-assoc′ chine-assoc*
using *chine-assoc-props*(*1*) *chine-assoc′-props*(*1*) **by** *auto*
have *3*: *C.seq* Prj_{11} *chine-assoc′* ∧ *C.seq* Prj_{01} *chine-assoc′* ∧ *C.seq* Prj_0 *chine-assoc′*
using *prj-in-hom chine-assoc′-props*(*1*) **by** *auto*
have Prj_{11} · *chine-assoc′* · *chine-assoc* = Prj_{11} · *μν-π.chine*
proof −
have Prj_{11} · *chine-assoc′* · *chine-assoc* = (Prj_{11} · *chine-assoc′*) · *chine-assoc*
using *C.comp-assoc* **by** *metis*
thus *?thesis* **using** *1 C.comp-arr-dom* **by** *simp*
qed
moreover have Prj_{01} · *chine-assoc′* · *chine-assoc* = Prj_{01} · *μν-π.chine*
proof −
have Prj_{01} · *chine-assoc′* · *chine-assoc* = (Prj_{01} · *chine-assoc′*) · *chine-assoc*
using *C.comp-assoc* **by** *metis*
thus *?thesis* **using** *1 C.comp-arr-dom* **by** *simp*
qed
moreover have Prj_0 · *chine-assoc′* · *chine-assoc* = Prj_0 · *μν-π.chine*
proof −
have Prj_0 · *chine-assoc′* · *chine-assoc* = (Prj_0 · *chine-assoc′*) · *chine-assoc*
using *C.comp-assoc* **by** *metis*
thus *?thesis* **using** *1 C.comp-arr-dom C.comp-arr-ide prj-in-hom*(*3*) **by** *auto*
qed
moreover have «*μν-π.chine* : *μν-π.chine* $\rightarrow_C$ *μν-π.chine*»
using *chine-assoc-props*(*1*) *C.ide-dom* [*of chine-assoc*]
by (*elim C.in-homE*, *auto*)
ultimately show *?thesis*
using *chine-assoc-props*(*1*) *chine-assoc′-props*(*1*)
prj-joint-monic [*of chine-assoc′* · *chine-assoc μν-π.chine μν-π.chine*]
by *auto*
qed
thus *?thesis*
using *1* **by** *simp*
qed
show *C.ide* (*chine-assoc* · *chine-assoc′*)
proof −
have *1*: *C.ide μνπ.chine*
using *chine-assoc-props*(*1*) *C.ide-cod* **by** *fastforce*
have *chine-assoc* · *chine-assoc′* = *μνπ.chine*
proof −
have *2*: *C.seq chine-assoc chine-assoc′*
using *chine-assoc-props*(*1*) *chine-assoc′-props*(*1*) **by** *auto*
have *3*: *C.seq* Prj_1 *chine-assoc* ∧ *C.seq* Prj_{10} *chine-assoc* ∧ *C.seq* Prj_{00} *chine-assoc*
using *prj-in-hom chine-assoc-props*(*1*) **by** *auto*
have Prj_1 · *chine-assoc* · *chine-assoc′* = Prj_1 · *μνπ.chine*
proof −
have Prj_1 · *chine-assoc* · *chine-assoc′* = (Prj_1 · *chine-assoc*) · *chine-assoc′*

```
        using C.comp-assoc by metis
      thus ?thesis using 1 C.comp-arr-dom prj-in-hom(4) by auto
    qed
    moreover have Prj10 · chine-assoc · chine-assoc′ = Prj10 · μνπ.chine
    proof −
      have Prj10 · chine-assoc · chine-assoc′ = (Prj10 · chine-assoc) · chine-assoc′
        using C.comp-assoc by metis
      thus ?thesis using 1 C.comp-arr-dom by simp
    qed
    moreover have Prj00 · chine-assoc · chine-assoc′ = Prj00 · μνπ.chine
    proof −
      have Prj00 · chine-assoc · chine-assoc′ = (Prj00 · chine-assoc) · chine-assoc′
        using C.comp-assoc by metis
      thus ?thesis using 1 C.comp-arr-dom by simp
    qed
    moreover have «μνπ.chine : μνπ.chine →C μνπ.chine»
      using chine-assoc′-props(1) C.ide-dom [of chine-assoc′]
      by (elim C.in-homE, auto)
    ultimately show ?thesis
      using chine-assoc-props(1) chine-assoc′-props(1)
            prj′-joint-monic [of chine-assoc · chine-assoc′ μνπ.chine μνπ.chine]
      by auto
  qed
  thus ?thesis
    using 1 by simp
 qed
qed

end

context three-composable-arrows-of-spans
begin

  interpretation V: category vcomp
    using is-category by auto
  interpretation H: horizontal-homs vcomp src trg
    using has-horizontal-homs by auto

  interpretation dom-μ: arrow-of-spans C ‹dom μ›
    using μ.arrow-of-spans-axioms arr-char [of dom μ] by auto
  interpretation dom-ν: arrow-of-spans C ‹dom ν›
    using ν.arrow-of-spans-axioms arr-char [of dom ν] by auto
  interpretation dom-π: arrow-of-spans C ‹dom π›
    using π.arrow-of-spans-axioms arr-char [of dom π] by auto
  interpretation doms: three-composable-identity-arrows-of-spans C prj0 prj1
                    ‹dom μ› ‹dom ν› ‹dom π›
    using μν.composable νπ.composable ide-char [of dom μ] ide-char [of dom ν]
          ide-char [of dom π]
    by (unfold-locales, auto)
```

interpretation *cod-μ*: *arrow-of-spans C* ‹*cod μ*›
  using *μ.arrow-of-spans-axioms arr-char* [*of cod μ*] **by** *auto*
interpretation *cod-ν*: *arrow-of-spans C* ‹*cod ν*›
  using *ν.arrow-of-spans-axioms arr-char* [*of cod ν*] **by** *auto*
interpretation *cod-π*: *arrow-of-spans C* ‹*cod π*›
  using *π.arrow-of-spans-axioms arr-char* [*of cod π*] **by** *auto*
interpretation *cods*: *three-composable-identity-arrows-of-spans C prj0 prj1*
    ‹*cod μ*› ‹*cod ν*› ‹*cod π*›
  using *μν.composable νπ.composable ide-char* [*of cod μ*] *ide-char* [*of cod ν*]
    ide-char [*of cod π*]
  by (*unfold-locales*, *auto*)

interpretation *μνπ*: *arrow-of-spans C* ‹*μ* ⋆ *ν* ⋆ *π*›
  using *μ.arrow-of-spans-axioms νπ.arrow-of-spans-axioms*
    arrow-of-spans-hcomp arr-char μν.composable νπ.composable
  by *force*

interpretation *μν-π*: *arrow-of-spans C* ‹(*μ* ⋆ *ν*) ⋆ *π*›
  using *μν.arrow-of-spans-axioms π.arrow-of-spans-axioms*
    arrow-of-spans-hcomp arr-char μν.composable νπ.composable
  by *force*

lemma *chine-composite′*:
shows $\mu\nu\pi.chine = \langle \mu.chine \cdot doms.Prj_1$
    $[\![\mu.cod.leg0,\ \nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0,\ \pi.cod.leg1]]\!]$
    $\langle \nu.chine \cdot doms.Prj_{10}\ [\![\nu.cod.leg0,\ \pi.cod.leg1]\!]\ \pi.chine \cdot doms.Prj_{00}\rangle\rangle$
and $\mu\nu\text{-}\pi.chine = \langle\langle \mu.chine \cdot doms.Prj_{11}\ [\![\mu.cod.leg0,\ \nu.cod.leg1]\!]\ \nu.chine \cdot doms.Prj_{01}\rangle$
    $[\![\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1],\ \pi.cod.leg1]\!]$
    $\pi.chine \cdot doms.Prj_0\rangle$
proof −
  show $\mu\nu\text{-}\pi.chine = \langle\langle \mu.chine \cdot doms.Prj_{11}\ [\![\mu.cod.leg0,\ \nu.cod.leg1]\!]\ \nu.chine \cdot doms.Prj_{01}\rangle$
    $[\![\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1],\ \pi.cod.leg1]\!]$
    $\pi.chine \cdot doms.Prj_0\rangle$
  proof −
    have *arr* (*μ* ⋆ *ν*) **by** *simp*
    thus *?thesis*
      unfolding *hcomp-def chine-hcomp-def dom-char*
      using *μν.composable νπ.composable src-def trg-def dom-char chine-hcomp-props*
        C.comp-tuple-arr C.pullback-commutes C.comp-assoc
      by *auto*
  qed
  show $\mu\nu\pi.chine = \langle \mu.chine \cdot doms.Prj_1$
    $[\![\mu.cod.leg0,\ \nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0,\ \pi.cod.leg1]]\!]$
    $\langle \nu.chine \cdot doms.Prj_{10}\ [\![\nu.cod.leg0,\ \pi.cod.leg1]\!]\ \pi.chine \cdot doms.Prj_{00}\rangle\rangle$
  proof −
    have *arr* (*ν* ⋆ *π*) **by** *simp*
    thus *?thesis*
      unfolding *hcomp-def chine-hcomp-def dom-char*

using $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def dom-char chine-hcomp-props*
C.comp-tuple-arr C.pullback-commutes C.comp-assoc
by *auto*
qed
qed

lemma *chine-composite-in-hom* [*intro*]:
shows «$\mu\nu$-π.*chine* : *Chn* ((*dom* μ $\star$ *dom* ν) $\star$ *dom* π) $\rightarrow_C$ *Chn* ((*cod* μ $\star$ *cod* ν) $\star$ *cod* π)»
and «$\mu\nu\pi$.*chine* : *Chn* (*dom* μ $\star$ *dom* ν $\star$ *dom* π) $\rightarrow_C$ *Chn* (*cod* μ $\star$ *cod* ν $\star$ *cod* π)»
using *Chn-in-hom* **by** *auto*

lemma *cospan-*$\mu\nu$:
shows *C.cospan* μ.*dom.leg0* ν.*dom.leg1*
using $\mu\nu$.*legs-form-cospan* **by** *simp*

lemma *cospan-*$\nu\pi$:
shows *C.cospan* ν.*dom.leg0* π.*dom.leg1*
using $\nu\pi$.*legs-form-cospan* **by** *simp*

lemma *commutativities*:
shows μ.*cod.leg0* $\cdot$ μ.*chine* $\cdot$ *doms.Prj*$_{11}$ = ν.*cod.leg1* $\cdot$ ν.*chine* $\cdot$ *doms.Prj*$_{01}$
and π.*cod.leg1* $\cdot$ π.*chine* $\cdot$ *doms.Prj*$_0$ =
(ν.*cod.leg0* $\cdot$ p_0[μ.*cod.leg0*, ν.*cod.leg1*]) $\cdot$
$\langle$$\mu$.*chine* $\cdot$ *doms.Prj*$_{11}$ $[\![$$\mu$.*cod.leg0*, ν.*cod.leg1*$]\!]$ ν.*chine* $\cdot$ *doms.Prj*$_{01}$$\rangle$
proof −
have *AB*: μ.*dom.leg0* $\cdot$ *doms.Prj*$_{11}$ = ν.*dom.leg1* $\cdot$ *doms.Prj*$_{01}$
proof −
have μ.*dom.leg0* $\cdot$ *doms.Prj*$_{11}$ =
(μ.*dom.leg0* $\cdot$ p_1[μ.*dom.leg0*, ν.*dom.leg1*]) $\cdot$
p_1[ν.*dom.leg0* $\cdot$ p_0[μ.*dom.leg0*, ν.*dom.leg1*], π.*dom.leg1*]
using $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def dom-char C.comp-assoc*
by *simp*
also have ... = (ν.*dom.leg1* $\cdot$ p_0[μ.*dom.leg0*, ν.*dom.leg1*]) $\cdot$
p_1[ν.*dom.leg0* $\cdot$ p_0[μ.*dom.leg0*, ν.*dom.leg1*], π.*dom.leg1*]
using *C.pullback-commutes′* $\mu\nu$.*legs-form-cospan* **by** *auto*
also have ... = ν.*dom.leg1* $\cdot$ *doms.Prj*$_{01}$
using $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def dom-char C.comp-assoc*
by *simp*
finally show *?thesis* **by** *auto*
qed
have *BC*: ν.*dom.leg0* $\cdot$ *doms.Prj*$_{01}$ = π.*dom.leg1* $\cdot$ *doms.Prj*$_0$
proof −
have ν.*dom.leg0* $\cdot$ *doms.Prj*$_{01}$ =
(ν.*dom.leg0* $\cdot$ p_0[μ.*dom.leg0*, ν.*dom.leg1*]) $\cdot$
p_1[ν.*dom.leg0* $\cdot$ p_0[μ.*dom.leg0*, ν.*dom.leg1*], π.*dom.leg1*]
using $\mu\nu$.*composable* $\nu\pi$.*composable src-def trg-def dom-char C.comp-assoc*
by *simp*
also have ... = π.*dom.leg1* $\cdot$ *doms.Prj*$_0$
using *C.pullback-commutes′ dom-char cod-char* $\mu\nu$.*legs-form-cospan* $\nu\pi$.*legs-form-cospan*

by *auto*
finally show *?thesis* **by** *simp*
qed
show *1*: $\mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_{11} = \nu.cod.leg1 \cdot \nu.chine \cdot doms.Prj_{01}$
using *AB C.comp-assoc* [*of* $\mu.cod.leg0$ $\mu.chine$]
C.comp-assoc [*of* $\nu.cod.leg1$ $\nu.chine$ $doms.Prj_{01}$]
by *simp*
show $\pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_0 =$
$(\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0, \nu.cod.leg1]) \cdot$
$\langle \mu.chine \cdot doms.Prj_{11} \llbracket \mu.cod.leg0, \nu.cod.leg1 \rrbracket \nu.chine \cdot doms.Prj_{01} \rangle$
proof −
have $(\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0, \nu.cod.leg1]) \cdot$
$\langle \mu.chine \cdot doms.Prj_{11} \llbracket \mu.cod.leg0, \nu.cod.leg1 \rrbracket \nu.chine \cdot doms.Prj_{01} \rangle =$
$\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0, \nu.cod.leg1] \cdot$
$\langle \mu.chine \cdot doms.Prj_{11} \llbracket \mu.cod.leg0, \nu.cod.leg1 \rrbracket \nu.chine \cdot doms.Prj_{01} \rangle$
using *1* $\mu\nu.composable$ $\nu\pi.composable$ *src-def trg-def dom-char C.comp-assoc* **by** *simp*
also have ... $= \nu.cod.leg0 \cdot \nu.chine \cdot doms.Prj_{01}$
using *1 dom-char* $\mu\nu.composable$ $\nu\pi.composable$ *src-def trg-def* **by** *simp*
also have ... $= (\nu.cod.leg0 \cdot \nu.chine) \cdot doms.Prj_{01}$
using *C.comp-assoc* [*of* $\nu.cod.leg0$ $\nu.chine$ $doms.Prj_{01}$] **by** *simp*
also have ... $= (\pi.cod.leg1 \cdot \pi.chine) \cdot doms.Prj_0$
using *BC* **by** *simp*
also have ... $= \pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_0$
using *C.comp-assoc* **by** *blast*
finally show *?thesis* **by** *simp*
qed
qed

lemma *prj-chine-composite*:
shows $cods.Prj_{11} \cdot \mu\nu\text{-}\pi.chine = \mu.chine \cdot doms.Prj_{11}$
and $cods.Prj_{01} \cdot \mu\nu\text{-}\pi.chine = \nu.chine \cdot doms.Prj_{01}$
and $cods.Prj_0 \cdot \mu\nu\text{-}\pi.chine = \pi.chine \cdot doms.Prj_0$
using $\mu\nu.composable$ $\nu\pi.composable$ *src-def trg-def dom-char cod-char commutativities chine-composite′ C.comp-assoc*
by *auto*

lemma *commutativities′*:
shows $\nu.cod.leg0 \cdot \nu.chine \cdot doms.Prj_{10} = \pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_{00}$
and $\mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_1 =$
$(\nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]) \cdot$
$\langle \nu.chine \cdot doms.Prj_{10} \llbracket \nu.cod.leg0, \pi.cod.leg1 \rrbracket \pi.chine \cdot doms.Prj_{00} \rangle$
proof −
have *AB*: $\mu.dom.leg0 \cdot doms.Prj_1 = \nu.dom.leg1 \cdot doms.Prj_{10}$
using *C.pullback-commutes′ dom-char cod-char* $\mu\nu.legs\text{-}form\text{-}cospan$ $\nu\pi.legs\text{-}form\text{-}cospan$ *C.comp-assoc*
by *auto*
have *BC*: $\nu.dom.leg0 \cdot doms.Prj_{10} = \pi.dom.leg1 \cdot doms.Prj_{00}$
proof −
have $\nu.dom.leg0 \cdot doms.Prj_{10} =$

$(\nu.dom.leg0 \cdot \mathrm{p}_1[\nu.dom.leg0, \pi.dom.leg1]) \cdot$
$\mathrm{p}_0[\mu.dom.leg0, \nu.dom.leg1 \cdot \mathrm{p}_1[\nu.dom.leg0, \pi.dom.leg1]]$
using *dom-char* $\mu\nu$.*legs-form-cospan* $\nu\pi$.*legs-form-cospan* C.*comp-assoc* **by** *simp*
also have ... $= \pi.dom.leg1 \cdot doms.Prj_{00}$
using C.*pullback-commutes'* *dom-char* $\mu\nu$.*legs-form-cospan* $\nu\pi$.*legs-form-cospan* C.*comp-assoc*
by *simp*
finally show *?thesis* **by** *auto*
qed
show *1*: $\nu.cod.leg0 \cdot \nu.chine \cdot doms.Prj_{10} = \pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_{00}$
using *BC* C.*comp-assoc* [*of* $\nu.cod.leg0$ $\nu.chine$ $doms.Prj_{10}$]
C.*comp-assoc* [*of* $\pi.cod.leg1$ $\pi.chine$ $doms.Prj_{00}$]
doms.prj-in-hom(*5*−*6*) *dom-char*
by *force*
show $\mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_1 =$
$(\nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]) \cdot$
$\langle \nu.chine \cdot doms.Prj_{10} \llbracket \nu.cod.leg0, \pi.cod.leg1 \rrbracket \pi.chine \cdot doms.Prj_{00} \rangle$
proof −
have $(\nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]) \cdot$
$\langle \nu.chine \cdot doms.Prj_{10} \llbracket \nu.cod.leg0, \pi.cod.leg1 \rrbracket \pi.chine \cdot doms.Prj_{00} \rangle =$
$\nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1] \cdot$
$\langle \nu.chine \cdot doms.Prj_{10} \llbracket \nu.cod.leg0, \pi.cod.leg1 \rrbracket \pi.chine \cdot doms.Prj_{00} \rangle$
using *1* $\mu\nu$.*composable* $\nu\pi$.*composable* *src-def* *trg-def* *dom-char* C.*comp-assoc* **by** *simp*
also have ... $= \nu.cod.leg1 \cdot \nu.chine \cdot doms.Prj_{10}$
using *1* *dom-char* $\mu\nu$.*composable* $\nu\pi$.*composable* *src-def* *trg-def*
by *simp*
also have ... $= (\nu.cod.leg1 \cdot \nu.chine) \cdot doms.Prj_{10}$
using C.*comp-assoc* [*of* $\nu.cod.leg1$ $\nu.chine$ $doms.Prj_{10}$] **by** *auto*
also have ... $= (\mu.cod.leg0 \cdot \mu.chine) \cdot doms.Prj_1$
using *AB* **by** *simp*
also have ... $= \mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_1$
using C.*comp-assoc* [*of* $\mu.cod.leg0$ $\mu.chine$ $doms.Prj_1$] *doms.prj-in-hom*(*4*) *dom-char*
by *auto*
finally show *?thesis* **by** *simp*
qed
qed

lemma *prj'-chine-composite*:
shows $cods.Prj_1 \cdot \mu\nu\pi.chine = \mu.chine \cdot doms.Prj_1$
and $cods.Prj_{10} \cdot \mu\nu\pi.chine = \nu.chine \cdot doms.Prj_{10}$
and $cods.Prj_{00} \cdot \mu\nu\pi.chine = \pi.chine \cdot doms.Prj_{00}$
using $\mu\nu$.*composable* $\nu\pi$.*composable* *src-def* *trg-def* *dom-char* *cod-char* *commutativities'*
chine-composite' C.*comp-assoc*
by *auto*

lemma *chine-assoc-naturality*:
shows $cods.chine\text{-}assoc \cdot \mu\nu\text{-}\pi.chine = \mu\nu\pi.chine \cdot doms.chine\text{-}assoc$
proof −
have «$cods.chine\text{-}assoc \cdot \mu\nu\text{-}\pi.chine$:
$Chn\ ((dom\ \mu \star dom\ \nu) \star dom\ \pi) \rightarrow_C Chn\ (cod\ \mu \star cod\ \nu \star cod\ \pi)$»

using *cods.chine-assoc-props*(*1*) *chine-composite-in-hom*(*1*) **by** *blast*
moreover have «$\mu\nu\pi$.*chine* · *doms.chine-assoc* :
Chn ((*dom* μ ⋆ *dom* ν) ⋆ *dom* π) $\rightarrow_C$ *Chn* (*cod* μ ⋆ *cod* ν ⋆ *cod* π)»
using *doms.chine-assoc-props*(*1*) *chine-composite-in-hom*(*2*) **by** *blast*
moreover
have *cods.Prj*$_1$ · *cods.chine-assoc* · $\mu\nu$-π.*chine* =
cods.Prj$_1$ · $\mu\nu\pi$.*chine* · *doms.chine-assoc*
using *C.comp-assoc doms.chine-assoc-props*(*2*) *cods.chine-assoc-props*(*2*)
prj-chine-composite prj′-chine-composite
by *metis*
moreover have *cods.Prj*$_{10}$ · *cods.chine-assoc* · $\mu\nu$-π.*chine* =
cods.Prj$_{10}$ · $\mu\nu\pi$.*chine* · *doms.chine-assoc*
using *C.comp-assoc doms.chine-assoc-props*(*3*) *cods.chine-assoc-props*(*3*)
prj-chine-composite prj′-chine-composite
by *metis*
moreover have *cods.Prj*$_{00}$ · *cods.chine-assoc* · $\mu\nu$-π.*chine* =
cods.Prj$_{00}$ · $\mu\nu\pi$.*chine* · *doms.chine-assoc*
using *C.comp-assoc doms.chine-assoc-props*(*4*) *cods.chine-assoc-props*(*4*)
prj-chine-composite prj′-chine-composite
by *metis*
ultimately show *?thesis*
using *cods.prj′-joint-monic* **by** *auto*
qed

end

context *span-bicategory*
begin

abbreviation (*input*) *assoc*$_{SB}$
where *assoc*$_{SB}$ *f g h* ≡ ⦇*Chn* = *three-composable-identity-arrows-of-spans.chine-assoc*
C prj0 prj1 f g h,
Dom = *Dom* ((*f* ⋆ *g*) ⋆ *h*), *Cod* = *Cod* (*f* ⋆ *g* ⋆ *h*)⦈

abbreviation (*input*) *assoc′*$_{SB}$
where *assoc′*$_{SB}$ *f g h* ≡ ⦇*Chn* = *three-composable-identity-arrows-of-spans.chine-assoc′*
C prj0 prj1 f g h,
Dom = *Cod* (*f* ⋆ *g* ⋆ *h*), *Cod* = *Dom* ((*f* ⋆ *g*) ⋆ *h*)⦈

lemma *assoc-props*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *src* (*assoc*$_{SB}$ *f g h*) = *src h* **and** *trg* (*assoc*$_{SB}$ *f g h*) = *trg f*
and «*assoc*$_{SB}$ *f g h* : (*f* ⋆ *g*) ⋆ *h* ⇒ *f* ⋆ *g* ⋆ *h*»
and *src* (*assoc′*$_{SB}$ *f g h*) = *src h* **and** *trg* (*assoc′*$_{SB}$ *f g h*) = *trg f*
and «*assoc′*$_{SB}$ *f g h* : *f* ⋆ *g* ⋆ *h* ⇒ (*f* ⋆ *g*) ⋆ *h*»
proof −
have *fgh*: *VVV.ide* (*f*, *g*, *h*)
using *assms VVV.ide-char*$_{SbC}$ *VV.ide-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *simp*

interpret *f*: *arrow-of-spans C f*
  using *assms arr-char* **by** *fastforce*
interpret *g*: *arrow-of-spans C g*
  using *assms arr-char* **by** *fastforce*
interpret *h*: *arrow-of-spans C h*
  using *assms arr-char* **by** *fastforce*
interpret *fgh*: *three-composable-arrows-of-spans C prj0 prj1 f g h*
  using *assms arr-char* **by** (*unfold-locales*, *auto*)
interpret *fgh*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g h*
  using *assms ide-char* **by** (*unfold-locales*, *auto*)
interpret *HHfgh*: *arrow-of-spans C* ‹$(f \star g) \star h$›
  using *assms fgh.composites-are-arrows arrow-of-spans-hcomp* **by** *simp*
interpret *HfHgh*: *arrow-of-spans C* ‹$f \star g \star h$›
  using *assms fgh.composites-are-arrows arrow-of-spans-hcomp* **by** *simp*
interpret *assoc-fgh*: *arrow-of-spans C* ‹$assoc_{SB}\ f\ g\ h$›
  apply *unfold-locales*
      apply *simp-all*
    apply (*metis C.ideD*(*2*) *C.ideD*(*3*) *HHfgh.chine-simps*(*2*) *HfHgh.chine-simps*(*3*)
      fgh.composites-are-identities(*1*) *fgh.composites-are-identities*(*2*)
      fgh.chine-assoc-in-hom ide-char)
  apply (*metis C.comp-assoc fgh.composites-are-identities*(*1*) *fgh.leg0-composite*(*1*−*2*)
            fgh.prj-chine-assoc(*3*) *ide-char′ identity-arrow-of-spans.cod-simps*(*2*))
  by (*metis C.comp-assoc fgh.composites-are-identities*(*1*) *fgh.leg1-composite*(*1*−*2*)
      fgh.prj-chine-assoc(*1*) *ide-char′ identity-arrow-of-spans.cod-simps*(*3*))
interpret *assoc′-fgh*: *arrow-of-spans C* ‹$assoc'_{SB}\ f\ g\ h$›
  apply *unfold-locales*
      apply *simp-all*
    apply (*metis C.ideD*(*2*) *C.ideD*(*3*) *HHfgh.chine-simps*(*2*) *HfHgh.chine-simps*(*3*)
      fgh.composites-are-identities(*1*) *fgh.composites-are-identities*(*2*)
      fgh.chine-assoc′-in-hom ide-char)
  using *C.comp-assoc fgh.composites-are-identities*(*1*) *fgh.leg0-composite*(*1*−*2*)
        ide-char′ identity-arrow-of-spans.cod-simps(*2*)
   apply *force*
  using *C.comp-assoc fgh.composites-are-identities*(*1*) *fgh.leg1-composite*(*1*−*2*)
        fgh.prj-chine-assoc′(*1*) *ide-char′ identity-arrow-of-spans.cod-simps*(*3*)
  by *force*
show *1*: «$assoc_{SB}\ f\ g\ h : (f \star g) \star h \Rightarrow f \star g \star h$»
proof
  show *1*: *arr* ($assoc_{SB}\ f\ g\ h$)
    using *assoc-fgh.arrow-of-spans-axioms arr-char* **by** *blast*
  show *dom* ($assoc_{SB}\ f\ g\ h$) = $(f \star g) \star h$
    using *fgh 1 dom-char fgh.μν.composable fgh.νπ.composable ideD*(*2*)
    by *auto*
  show *cod* ($assoc_{SB}\ f\ g\ h$) = $f \star g \star h$
    using *fgh 1 HoVH-def cod-char fgh.μν.composable fgh.νπ.composable ideD*(*3*)
    by *auto*
qed
show *2*: «$assoc'_{SB}\ f\ g\ h : f \star g \star h \Rightarrow (f \star g) \star h$»
proof

```
    show 1: arr (assoc'_SB f g h)
      using assoc'-fgh.arrow-of-spans-axioms arr-char by blast
    show dom (assoc'_SB f g h) = f ⋆ g ⋆ h
      using fgh 1 dom-char cod-char ideD(3) by auto
    show cod (assoc'_SB f g h) = (f ⋆ g) ⋆ h
      using fgh 1 cod-char dom-char ideD(2) by auto
  qed
  show 3: src (assoc_SB f g h) = src h
  proof −
    have 4: src (assoc_SB f g h) =
              ⦇Chn = assoc-fgh.dsrc, Dom = ⦇Leg0 = assoc-fgh.dsrc, Leg1 = assoc-fgh.dsrc⦈,
               Cod = ⦇Leg0 = assoc-fgh.dsrc, Leg1 = assoc-fgh.dsrc⦈⦈
      unfolding src-def using 1 by auto
    also have ... = src h
      using fgh.composite-simps(2) src-def by auto
    finally show ?thesis by blast
  qed
  show src (assoc'_SB f g h) = src h
  proof −
    have src (assoc'_SB f g h) =
            ⦇Chn = assoc'-fgh.dsrc, Dom = ⦇Leg0 = assoc'-fgh.dsrc, Leg1 = assoc'-fgh.dsrc⦈,
             Cod = ⦇Leg0 = assoc'-fgh.dsrc, Leg1 = assoc'-fgh.dsrc⦈⦈
      unfolding src-def using 2 by auto
    also have ... = src h
      using 1 3 assoc-fgh.cod-src-eq-dom-src arrI src-def by auto
    finally show ?thesis by blast
  qed
  show 4: trg (assoc_SB f g h) = trg f
  proof −
    have 5: trg (assoc_SB f g h) =
            ⦇Chn = assoc-fgh.dtrg, Dom = ⦇Leg0 = assoc-fgh.dtrg, Leg1 = assoc-fgh.dtrg⦈,
             Cod = ⦇Leg0 = assoc-fgh.dtrg, Leg1 = assoc-fgh.dtrg⦈⦈
      unfolding trg-def using 1 by auto
    also have ... = trg f
      using fgh.composite-simps(4) trg-def by auto
    finally show ?thesis by blast
  qed
  show trg (assoc'_SB f g h) = trg f
  proof −
    have 5: trg (assoc'_SB f g h) =
            ⦇Chn = assoc'-fgh.dtrg, Dom = ⦇Leg0 = assoc'-fgh.dtrg, Leg1 = assoc'-fgh.dtrg⦈,
             Cod = ⦇Leg0 = assoc'-fgh.dtrg, Leg1 = assoc'-fgh.dtrg⦈⦈
      unfolding trg-def using 2 by auto
    also have ... = trg f
      using 1 4 assoc-fgh.cod-trg-eq-dom-trg arrI trg-def by auto
    finally show ?thesis by blast
  qed
qed
```

lemma *assoc-in-hom* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows «$assoc_{SB}$ *f g h* : (*f* ⋆ *g*) ⋆ *h* ⇒ *f* ⋆ *g* ⋆ *h*»
  using *assms assoc-props* **by** *auto*

lemma *assoc′-in-hom* [*intro*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows «$assoc'_{SB}$ *f g h* : *f* ⋆ *g* ⋆ *h* ⇒ (*f* ⋆ *g*) ⋆ *h*»
  using *assms assoc-props* **by** *auto*

lemma *assoc-simps* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *arr* ($assoc_{SB}$ *f g h*) **and** *dom* ($assoc_{SB}$ *f g h*) = (*f* ⋆ *g*) ⋆ *h*
and *cod* ($assoc_{SB}$ *f g h*) = *f* ⋆ *g* ⋆ *h*
and *src* ($assoc_{SB}$ *f g h*) = *src h* **and** *trg* ($assoc_{SB}$ *f g h*) = *trg f*
  using *assms assoc-props*(*1*−*3*) **by** (*fast*, *fast*, *fast*, *auto*)

lemma *assoc′-simps* [*simp*]:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *arr* ($assoc'_{SB}$ *f g h*) **and** *dom* ($assoc'_{SB}$ *f g h*) = *f* ⋆ *g* ⋆ *h*
and *cod* ($assoc'_{SB}$ *f g h*) = (*f* ⋆ *g*) ⋆ *h*
and *src* ($assoc'_{SB}$ *f g h*) = *src h* **and** *trg* ($assoc'_{SB}$ *f g h*) = *trg f*
  using *assms assoc-props*(*4*−*6*) **by** (*fast*, *fast*, *fast*, *auto*)

lemma *inverse-assoc-assoc′*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows *inverse-arrows* ($assoc_{SB}$ *f g h*) ($assoc'_{SB}$ *f g h*)
proof −
  interpret *f*: *arrow-of-spans C f* **using** *assms arr-char ideD*(*1*) **by** *simp*
  interpret *g*: *arrow-of-spans C g* **using** *assms arr-char ideD*(*1*) **by** *simp*
  interpret *h*: *arrow-of-spans C h* **using** *assms arr-char ideD*(*1*) **by** *simp*
  interpret *fgh*: *three-composable-arrows-of-spans C prj0 prj1 f g h*
    using *assms arr-char* **by** (*unfold-locales*, *auto*)
  interpret *fgh*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g h*
    using *assms ide-char*
    by *unfold-locales blast*+
  interpret *afgh*: *arrow-of-spans C* ‹$assoc_{SB}$ *f g h*›
    using *assms assoc-props*(*3*) *arr-char* **by** *blast*
  interpret *a′fgh*: *arrow-of-spans C* ‹$assoc'_{SB}$ *f g h*›
    using *assms assoc-props*(*6*) *arr-char* **by** *blast*
  show *?thesis*
  proof −
    have *inverse-arrows* ($assoc_{SB}$ *f g h*)
                (|*Chn* = *C.inv* (*Chn* ($assoc_{SB}$ *f g h*)),
                 Dom = *Cod* ($assoc_{SB}$ *f g h*), *Cod* = *Dom* ($assoc_{SB}$ *f g h*)|)
      using *inverse-arrows* [*of* $assoc_{SB}$ *f g h*] *afgh.arrow-of-spans-axioms*
          arr-char fgh.chine-assoc-inverse
      by *auto*
    moreover have *C.inv* (*Chn* ($assoc_{SB}$ *f g h*)) = *fgh.chine-assoc′*

using *fgh.chine-assoc-inverse C.inv-is-inverse C.inverse-arrow-unique* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
qed

interpretation α: *transformation-by-components VVV.comp vcomp HoHV HoVH*
‹λ*fgh. assoc*$_{SB}$ *(fst fgh) (fst (snd fgh)) (snd (snd fgh))*›
proof
show $*$: $\bigwedge$*fgh. VVV.ide fgh* $\Longrightarrow$ «*assoc*$_{SB}$ *(fst fgh) (fst (snd fgh)) (snd (snd fgh)) :*
HoHV fgh $\Rightarrow$ *HoVH fgh*»
proof −
fix *fgh*
assume *fgh*: *VVV.ide fgh*
show «*assoc*$_{SB}$ *(fst fgh) (fst (snd fgh)) (snd (snd fgh)) : HoHV fgh* $\Rightarrow$ *HoVH fgh*»
unfolding *HoHV-def HoVH-def*
using *fgh assoc-in-hom* [*of fst fgh fst (snd fgh) snd (snd fgh)*]
VVV.arr-char$_{SbC}$ *VVV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by *simp*
qed
show $\bigwedge\mu\nu\pi$. *VVV.arr* $\mu\nu\pi$ $\Longrightarrow$
assoc$_{SB}$ *(fst (VVV.cod* $\mu\nu\pi$*)) (fst (snd (VVV.cod* $\mu\nu\pi$*)))*
(snd (snd (VVV.cod $\mu\nu\pi$*))) ·*
HoHV $\mu\nu\pi$ =
HoVH $\mu\nu\pi$ · *assoc*$_{SB}$ *(fst (VVV.dom* $\mu\nu\pi$*)) (fst (snd (VVV.dom* $\mu\nu\pi$*)))*
(snd (snd (VVV.dom $\mu\nu\pi$*)))*
proof −
fix $\mu\nu\pi$
assume $\mu\nu\pi$: *VVV.arr* $\mu\nu\pi$
interpret μ: *arrow-of-spans C* ‹*fst* $\mu\nu\pi$›
using $\mu\nu\pi$ *VVV.ide-char VVV.arr-char*$_{SbC}$ *arr-char* **by** *auto*
interpret ν: *arrow-of-spans C* ‹*fst (snd* $\mu\nu\pi$*)*›
using $\mu\nu\pi$ *VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char* **by** *auto*
interpret π: *arrow-of-spans C* ‹*snd (snd* $\mu\nu\pi$*)*›
using $\mu\nu\pi$ *VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char* **by** *auto*
interpret $\mu\nu\pi$: *three-composable-arrows-of-spans C prj0 prj1*
‹*fst* $\mu\nu\pi$› ‹*fst (snd* $\mu\nu\pi$*)*› ‹*snd (snd* $\mu\nu\pi$*)*›
using $\mu\nu\pi$ *VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char*
by (*unfold-locales*, *auto*)

interpret *HoHV-*$\mu\nu\pi$: *arrow-of-spans C* ‹(*fst* $\mu\nu\pi$ $\star$ *fst (snd* $\mu\nu\pi$*))* $\star$ *snd (snd* $\mu\nu\pi$*)*›
proof −
have *arr (HoHV* $\mu\nu\pi$*)*
using $\mu\nu\pi$ **by** *simp*
thus *arrow-of-spans C ((fst* $\mu\nu\pi$ $\star$ *fst (snd* $\mu\nu\pi$*))* $\star$ *snd (snd* $\mu\nu\pi$*))*
using $\mu\nu\pi$ *HoHV-def arr-char* **by** *auto*
qed
interpret *HoVH-*$\mu\nu\pi$: *arrow-of-spans C* ‹*fst* $\mu\nu\pi$ $\star$ *fst (snd* $\mu\nu\pi$*)* $\star$ *snd (snd* $\mu\nu\pi$*)*›
proof −
have *arr (HoVH* $\mu\nu\pi$*)*

using $\mu\nu\pi$ **by** *simp*
thus *arrow-of-spans C* (*fst* $\mu\nu\pi$ $\star$ *fst* (*snd* $\mu\nu\pi$) $\star$ *snd* (*snd* $\mu\nu\pi$))
using $\mu\nu\pi$ *HoVH-def arr-char* **by** *auto*
qed

have *dom-*$\mu\nu\pi$: *VVV.ide* (*VVV.dom* $\mu\nu\pi$)
using $\mu\nu\pi$ *VVV.ide-dom* **by** *blast*
interpret *dom-*μ: *identity-arrow-of-spans C* ‹*fst* (*VVV.dom* $\mu\nu\pi$)›
using *dom-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *dom-*ν: *identity-arrow-of-spans C* ‹*fst* (*snd* (*VVV.dom* $\mu\nu\pi$))›
using *dom-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *dom-*π: *identity-arrow-of-spans C* ‹*snd* (*snd* (*VVV.dom* $\mu\nu\pi$))›
using *dom-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *dom-*$\mu\nu\pi$: *three-composable-identity-arrows-of-spans C prj0 prj1*
‹*fst* (*VVV.dom* $\mu\nu\pi$)› ‹*fst* (*snd* (*VVV.dom* $\mu\nu\pi$))›
‹*snd* (*snd* (*VVV.dom* $\mu\nu\pi$))›
using *dom-*$\mu\nu\pi$ *VVV.ide-char* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
by (*unfold-locales*, *auto*)
interpret *assoc-dom-*$\mu\nu\pi$: *arrow-of-spans C*
‹$assoc_{SB}$ (*fst* (*VVV.dom* $\mu\nu\pi$)) (*fst* (*snd* (*VVV.dom* $\mu\nu\pi$)))
(*snd* (*snd* (*VVV.dom* $\mu\nu\pi$)))›
using $\mu\nu\pi$ *VVV.ide-dom* $*$ *arr-char* **by** *fast*

have *cod-*$\mu\nu\pi$: *VVV.ide* (*VVV.cod* $\mu\nu\pi$)
using $\mu\nu\pi$ *VVV.ide-cod* **by** *blast*
interpret *cod-*μ: *identity-arrow-of-spans C* ‹*fst* (*VVV.cod* $\mu\nu\pi$)›
using *cod-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *cod-*ν: *identity-arrow-of-spans C* ‹*fst* (*snd* (*VVV.cod* $\mu\nu\pi$))›
using *cod-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *cod-*π: *identity-arrow-of-spans C* ‹*snd* (*snd* (*VVV.cod* $\mu\nu\pi$))›
using *cod-*$\mu\nu\pi$ $VVV.ide\text{-}char_{SbC}$ *ide-char'* **by** *blast*
interpret *cod-*$\mu\nu\pi$: *three-composable-identity-arrows-of-spans C prj0 prj1*
‹*fst* (*VVV.cod* $\mu\nu\pi$)› ‹*fst* (*snd* (*VVV.cod* $\mu\nu\pi$))›
‹*snd* (*snd* (*VVV.cod* $\mu\nu\pi$))›
using *cod-*$\mu\nu\pi$ *VVV.ide-char* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
by (*unfold-locales*, *auto*)
interpret *assoc-cod-*$\mu\nu\pi$: *arrow-of-spans C*
‹$assoc_{SB}$ (*fst* (*VVV.cod* $\mu\nu\pi$)) (*fst* (*snd* (*VVV.cod* $\mu\nu\pi$)))
(*snd* (*snd* (*VVV.cod* $\mu\nu\pi$)))›
using $\mu\nu\pi$ *VVV.ide-cod* $*$ *arr-char* **by** *fast*

have *dom-legs*:
dom-μ*.leg0* $=$ μ*.dom.leg0* $\wedge$ *dom-*ν*.leg0* $=$ ν*.dom.leg0* $\wedge$ *dom-*π*.leg0* $=$ π*.dom.leg0* $\wedge$
dom-μ*.leg1* $=$ μ*.dom.leg1* $\wedge$ *dom-*ν*.leg1* $=$ ν*.dom.leg1* $\wedge$ *dom-*π*.leg1* $=$ π*.dom.leg1*
using $VVV.arr\text{-}char_{SbC}$ $VVV.dom\text{-}char_{SbC}$ *dom-char* $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$
by *auto*
have *cod-legs*:
cod-μ*.leg0* $=$ μ*.cod.leg0* $\wedge$ *cod-*ν*.leg0* $=$ ν*.cod.leg0* $\wedge$ *cod-*π*.leg0* $=$ π*.cod.leg0* $\wedge$
cod-μ*.leg1* $=$ μ*.cod.leg1* $\wedge$ *cod-*ν*.leg1* $=$ ν*.cod.leg1* $\wedge$ *cod-*π*.leg1* $=$ π*.cod.leg1*

using $\mu\nu\pi$ $VVV.cod\text{-}char_{SbC}$ *cod-char* **by** *auto*

have Prj_{11}*-dom*: *dom-*$\mu\nu\pi.Prj_{11}$ =
$\mathrm{p}_1[\mu.dom.leg0,\ \nu.dom.leg1]\ \cdot$
$\mathrm{p}_1[\nu.dom.leg0\ \cdot\ \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1],\ \pi.dom.leg1]$
using *dom-legs* **by** *argo*
have Prj_{11}*-cod*: *cod-*$\mu\nu\pi.Prj_{11}$ =
$\mathrm{p}_1[\mu.cod.leg0,\ \nu.cod.leg1]\ \cdot$
$\mathrm{p}_1[\nu.cod.leg0\ \cdot\ \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1],\ \pi.cod.leg1]$
using *cod-legs* **by** *argo*
have Prj_0*-dom*: *dom-*$\mu\nu\pi.Prj_0 = \mathrm{p}_0[\nu.dom.leg0\ \cdot\ \mathrm{p}_0[\mu.dom.leg0,\ \nu.dom.leg1],\ \pi.dom.leg1]$
using *dom-legs* **by** *argo*
have Prj_0*-cod*: *cod-*$\mu\nu\pi.Prj_0 = \mathrm{p}_0[\nu.cod.leg0\ \cdot\ \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1],\ \pi.cod.leg1]$
using *cod-legs* **by** *argo*

have *Dom*: *Dom* ((*fst* ($VVV.dom\ \mu\nu\pi$) $\star$ *fst* (*snd* ($VVV.dom\ \mu\nu\pi$))) $\star$
snd (*snd* ($VVV.dom\ \mu\nu\pi$))) =
$(\!|Leg0 = \pi.dom.leg0\ \cdot\ dom\text{-}\mu\nu\pi.Prj_0,\ Leg1 = \mu.dom.leg1\ \cdot\ dom\text{-}\mu\nu\pi.Prj_{11}|\!)$
using *dom-*$\mu\nu\pi$*.leg0-composite*(*2*) *dom-*$\mu\nu\pi$*.leg1-composite*(*2*) *dom-legs* **by** *auto*
have *Cod*: *Cod* (*fst* ($VVV.dom\ \mu\nu\pi$) $\star$ *fst* (*snd* ($VVV.dom\ \mu\nu\pi$)) $\star$
snd (*snd* ($VVV.dom\ \mu\nu\pi$))) =
$(\!|Leg0 = \pi.dom.leg0\ \cdot\ dom\text{-}\mu\nu\pi.Prj_{00},\ Leg1 = \mu.dom.leg1\ \cdot\ dom\text{-}\mu\nu\pi.Prj_1|\!)$
proof −
have *arr* (*dom* (*fst* (*snd* $\mu\nu\pi$)) $\star$ *dom* (*snd* (*snd* $\mu\nu\pi$)))
using $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$ **by** *simp*
thus *?thesis*
using $\mu\nu\pi$ *hcomp-def dom-legs ide-dom dom-char*
apply *simp*
using $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$ *src-def trg-def dom-char* $C.comp\text{-}assoc$
$VVV.dom\text{-}simp$ $VV.dom\text{-}simp$
by *auto*
qed
have *Dom′*: *Dom* ((*fst* ($VVV.cod\ \mu\nu\pi$) $\star$ *fst* (*snd* ($VVV.cod\ \mu\nu\pi$))) $\star$
snd (*snd* ($VVV.cod\ \mu\nu\pi$))) =
$(\!|Leg0 = \pi.cod.leg0\ \cdot\ cod\text{-}\mu\nu\pi.Prj_0,\ Leg1 = \mu.cod.leg1\ \cdot\ cod\text{-}\mu\nu\pi.Prj_{11}|\!)$
by (*simp add*: *cod-*$\mu\nu\pi$*.leg0-composite*(*2*) *cod-*$\mu\nu\pi$*.leg1-composite*(*2*) *cod-legs*)
have *Cod′*: *Cod* (*fst* ($VVV.cod\ \mu\nu\pi$) $\star$ *fst* (*snd* ($VVV.cod\ \mu\nu\pi$)) $\star$
snd (*snd* ($VVV.cod\ \mu\nu\pi$))) =
$(\!|Leg0 = \pi.cod.leg0\ \cdot\ cod\text{-}\mu\nu\pi.Prj_{00},\ Leg1 = \mu.cod.leg1\ \cdot\ cod\text{-}\mu\nu\pi.Prj_1|\!)$
proof −
have *arr* (*cod* (*fst* (*snd* $\mu\nu\pi$)) $\star$ *cod* (*snd* (*snd* $\mu\nu\pi$)))
using $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$ **by** *simp*
moreover have $\mu.dsrc = \nu.dtrg$
using $\mu\nu\pi.\mu\nu.composable$ *src-def trg-def cod-char* **by** *simp*
ultimately show *?thesis*
using $\mu\nu\pi$ *hcomp-def cod-legs ide-cod cod-char* $VVV.cod\text{-}simp$ $VV.cod\text{-}simp$
apply *simp*
using $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$ *src-def trg-def cod-char* $C.comp\text{-}assoc$
by *auto*

qed

have *assoc-dom*:
$assoc_{SB}$ (*fst* (*VVV.dom* $\mu\nu\pi$)) (*fst* (*snd* (*VVV.dom* $\mu\nu\pi$)))
(*snd* (*snd* (*VVV.dom* $\mu\nu\pi$))) =
⦇*Chn* = *dom-*$\mu\nu\pi$*.chine-assoc*,
Dom = ⦇*Leg0* = π*.dom.leg0* · *dom-*$\mu\nu\pi$*.Prj*$_0$, *Leg1* = μ*.dom.leg1* · *dom-*$\mu\nu\pi$*.Prj*$_{11}$⦈,
Cod = ⦇*Leg0* = π*.dom.leg0* · *dom-*$\mu\nu\pi$*.Prj*$_{00}$, *Leg1* = μ*.dom.leg1* · *dom-*$\mu\nu\pi$*.Prj*$_1$⦈⦈
using *Dom Cod* **by** *simp*
have *assoc-cod*:
$assoc_{SB}$ (*fst* (*VVV.cod* $\mu\nu\pi$)) (*fst* (*snd* (*VVV.cod* $\mu\nu\pi$)))
(*snd* (*snd* (*VVV.cod* $\mu\nu\pi$))) =
⦇*Chn* = *cod-*$\mu\nu\pi$*.chine-assoc*,
Dom = ⦇*Leg0* = π*.cod.leg0* · *cod-*$\mu\nu\pi$*.Prj*$_0$, *Leg1* = μ*.cod.leg1* · *cod-*$\mu\nu\pi$*.Prj*$_{11}$⦈,
Cod = ⦇*Leg0* = π*.cod.leg0* · *cod-*$\mu\nu\pi$*.Prj*$_{00}$, *Leg1* = μ*.cod.leg1* · *cod-*$\mu\nu\pi$*.Prj*$_1$⦈⦈
using *Dom′ Cod′* **by** *simp*
have *HoHV-*$\mu\nu\pi$:
HoHV $\mu\nu\pi$ =
⦇*Chn* = *HoHV-*$\mu\nu\pi$*.chine*,
Dom = ⦇*Leg0* = π*.dom.leg0* · *dom-*$\mu\nu\pi$*.Prj*$_0$, *Leg1* = μ*.dom.leg1* · *dom-*$\mu\nu\pi$*.Prj*$_{11}$⦈,
Cod = ⦇*Leg0* = π*.cod.leg0* · *cod-*$\mu\nu\pi$*.Prj*$_0$, *Leg1* = μ*.cod.leg1* · *cod-*$\mu\nu\pi$*.Prj*$_{11}$⦈⦈
proof −
have *arr* ⦇*Chn* = ⟨μ*.chine* · p_1[μ*.dom.leg0*, ν*.dom.leg1*]
⟦μ*.cod.leg0*, ν*.cod.leg1*⟧
ν*.chine* · p_0[μ*.dom.leg0*, ν*.dom.leg1*]⟩,
Dom = ⦇*Leg0* = ν*.dom.leg0* · p_0[μ*.dom.leg0*, ν*.dom.leg1*],
Leg1 = μ*.dom.leg1* · p_1[μ*.dom.leg0*, ν*.dom.leg1*]⦈,
Cod = ⦇*Leg0* = ν*.cod.leg0* · p_0[μ*.cod.leg0*, ν*.cod.leg1*],
Leg1 = μ*.cod.leg1* · p_1[μ*.cod.leg0*, ν*.cod.leg1*]⦈⦈
unfolding *hcomp-def chine-hcomp-def*
using $\mu\nu\pi$ *hcomp-def dom-legs cod-legs ide-dom ide-cod dom-char cod-char*
$\mu\nu\pi$*.*$\mu\nu$*.composable*
by (*metis* (*no-types*, *lifting*) *hseq-char*(*1*) $\mu\nu\pi$*.*$\mu\nu$*.composite-is-arrow chine-hcomp-def*)
moreover have (μ*.cod.leg1* · p_1[μ*.cod.leg0*, ν*.cod.leg1*]) ·
p_1[ν*.cod.leg0* · p_0[μ*.cod.leg0*, ν*.cod.leg1*], π*.cod.leg1*] =
μ*.cod.leg1* · p_1[μ*.cod.leg0*, ν*.cod.leg1*] ·
p_1[ν*.cod.leg0* · p_0[μ*.cod.leg0*, ν*.cod.leg1*], π*.cod.leg1*]
using *C.comp-assoc* **by** *simp*
ultimately show *?thesis*
unfolding *HoHV-def hcomp-def chine-hcomp-def*
using $\mu\nu\pi$ $\mu\nu\pi$*.*$\mu\nu$*.composable* $\mu\nu\pi$*.*$\nu\pi$*.composable src-def trg-def dom-legs cod-legs*
C.comp-assoc
by *simp*
qed
have *HoVH-*$\mu\nu\pi$:
HoVH $\mu\nu\pi$ =
⦇*Chn* = *HoVH-*$\mu\nu\pi$*.chine*,
Dom = ⦇*Leg0* = π*.dom.leg0* · *dom-*$\mu\nu\pi$*.Prj*$_{00}$, *Leg1* = μ*.dom.leg1* · *dom-*$\mu\nu\pi$*.Prj*$_1$⦈,
Cod = ⦇*Leg0* = π*.cod.leg0* · *cod-*$\mu\nu\pi$*.Prj*$_{00}$, *Leg1* = μ*.cod.leg1* · *cod-*$\mu\nu\pi$*.Prj*$_1$⦈⦈

proof −
have arr ⦇Chn = ⟨$\nu.chine$ · $\mathrm{p}_1[\nu.dom.leg0, \pi.dom.leg1]$
⟦$\nu.cod.leg0, \pi.cod.leg1$⟧
$\pi.chine$ · $\mathrm{p}_0[\nu.dom.leg0, \pi.dom.leg1]$⟩,
Dom = ⦇$Leg0$ = $\pi.dom.leg0$ · $\mathrm{p}_0[\nu.dom.leg0, \pi.dom.leg1]$,
$Leg1$ = $\nu.dom.leg1$ · $\mathrm{p}_1[\nu.dom.leg0, \pi.dom.leg1]$⦈,
Cod = ⦇$Leg0$ = $\pi.cod.leg0$ · $\mathrm{p}_0[\nu.cod.leg0, \pi.cod.leg1]$,
$Leg1$ = $\nu.cod.leg1$ · $\mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]$⦈⦈
unfolding *hcomp-def chine-hcomp-def*
using $\mu\nu\pi$ *hcomp-def dom-legs cod-legs ide-dom ide-cod dom-char cod-char*
$\mu\nu\pi.\nu\pi.composable$
by (*metis* (*no-types*, *lifting*) *hseq-char* $\mu\nu\pi.\nu\pi.composite\text{-}is\text{-}arrow$ *chine-hcomp-def*)
moreover have ($\pi.cod.leg0$ · $\mathrm{p}_0[\nu.cod.leg0, \pi.cod.leg1]$) · $\mathrm{p}_0[\mu.cod.leg0, \nu.cod.leg1$ ·
$\mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]]$ =
$\pi.cod.leg0$ · $\mathrm{p}_0[\nu.cod.leg0, \pi.cod.leg1]$ · $\mathrm{p}_0[\mu.cod.leg0, \nu.cod.leg1$ ·
$\mathrm{p}_1[\nu.cod.leg0, \pi.cod.leg1]]$
using $C.comp\text{-}assoc$ **by** *simp*
ultimately show *?thesis*
unfolding *HoVH-def hcomp-def chine-hcomp-def*
using $\mu\nu\pi$ $\mu\nu\pi.\mu\nu.composable$ $\mu\nu\pi.\nu\pi.composable$ *src-def trg-def dom-legs cod-legs*
$C.comp\text{-}assoc$
by *simp*
qed
have $assoc_{SB}$ (*fst* ($VVV.cod$ $\mu\nu\pi$)) (*fst* (*snd* ($VVV.cod$ $\mu\nu\pi$)))
(*snd* (*snd* ($VVV.cod$ $\mu\nu\pi$))) ·
$HoHV$ $\mu\nu\pi$ =
⦇Chn = $cod\text{-}\mu\nu\pi.chine\text{-}assoc$ · $HoHV\text{-}\mu\nu\pi.chine$,
Dom = ⦇$Leg0$ = $\pi.dom.leg0$ · $dom\text{-}\mu\nu\pi.Prj_0$, $Leg1$ = $\mu.dom.leg1$ · $dom\text{-}\mu\nu\pi.Prj_{11}$⦈,
Cod = ⦇$Leg0$ = $\pi.cod.leg0$ · $cod\text{-}\mu\nu\pi.Prj_{00}$, $Leg1$ = $\mu.cod.leg1$ · $cod\text{-}\mu\nu\pi.Prj_1$⦈⦈
proof −
have arr ($HoHV$ $\mu\nu\pi$)
using $\mu\nu\pi$ **by** *simp*
thus *?thesis*
using *vcomp-def* $HoHV\text{-}\mu\nu\pi$ $HoHV\text{-}\mu\nu\pi.arrow\text{-}of\text{-}spans\text{-}axioms$
$assoc\text{-}cod\text{-}\mu\nu\pi.arrow\text{-}of\text{-}spans\text{-}axioms$ *assoc-cod dom-legs cod-legs*
arr-char
by *simp*
qed
moreover
have $HoVH$ $\mu\nu\pi$ ·
$assoc_{SB}$ (*fst* ($VVV.dom$ $\mu\nu\pi$)) (*fst* (*snd* ($VVV.dom$ $\mu\nu\pi$)))
(*snd* (*snd* ($VVV.dom$ $\mu\nu\pi$))) =
⦇Chn = $HoVH\text{-}\mu\nu\pi.chine$ · $dom\text{-}\mu\nu\pi.chine\text{-}assoc$,
Dom = ⦇$Leg0$ = $\pi.dom.leg0$ · $dom\text{-}\mu\nu\pi.Prj_0$, $Leg1$ = $\mu.dom.leg1$ · $dom\text{-}\mu\nu\pi.Prj_{11}$⦈,
Cod = ⦇$Leg0$ = $\pi.cod.leg0$ · $cod\text{-}\mu\nu\pi.Prj_{00}$, $Leg1$ = $\mu.cod.leg1$ · $cod\text{-}\mu\nu\pi.Prj_1$⦈⦈
proof −
have arr ($HoVH$ $\mu\nu\pi$)
using $\mu\nu\pi$ **by** *simp*
thus *?thesis*

using *vcomp-def HoVH-μνπ HoVH-μνπ.arrow-of-spans-axioms*
assoc-dom-μνπ.arrow-of-spans-axioms assoc-dom dom-legs cod-legs
arr-char
by *simp*
qed
moreover
have *cod-μνπ.chine-assoc* · *HoHV-μνπ.chine* = *HoVH-μνπ.chine* · *dom-μνπ.chine-assoc*
using *μνπ HoHV-def HoVH-def μνπ.chine-assoc-naturality*
VVV.dom-simp VV.dom-simp VVV.cod-simp VV.cod-simp **by** *simp*
ultimately show $assoc_{SB}$ (*fst* (*VVV.cod μνπ*)) (*fst* (*snd* (*VVV.cod μνπ*)))
(*snd* (*snd* (*VVV.cod μνπ*))) ·
HoHV μνπ =
HoVH μνπ ·
$assoc_{SB}$ (*fst* (*VVV.dom μνπ*)) (*fst* (*snd* (*VVV.dom μνπ*)))
(*snd* (*snd* (*VVV.dom μνπ*)))
by *argo*
qed
qed

definition *assoc* (‹a[-, -, -]›)
where *assoc* ≡ λ*μ ν π*. *α.map* (*μ*, *ν*, *π*)

abbreviation (*input*) α_{SB}
where α_{SB} ≡ λ*μνπ*. *assoc* (*fst μνπ*) (*fst* (*snd μνπ*)) (*snd* (*snd μνπ*))

lemma *α-ide*:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows α_{SB} (*f*, *g*, *h*) = $assoc_{SB}$ *f g h*
using *assms assoc-def α.map-simp-ide* $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
by *simp*

lemma *natural-transformation-α*:
shows *natural-transformation VVV.comp vcomp HoHV HoVH* α_{SB}
using *assoc-def α.natural-transformation-axioms* **by** *auto*

interpretation *α*: *natural-transformation VVV.comp vcomp HoHV HoVH* α_{SB}
using *natural-transformation-α* **by** *simp*

sublocale *α*: *natural-isomorphism VVV.comp vcomp HoHV HoVH* α_{SB}
proof
show ⋀*fgh*. *VVV.ide fgh* ⟹ *iso* (α_{SB} *fgh*)
proof −
fix *fgh*
assume *fgh*: *VVV.ide fgh*
interpret *f*: *arrow-of-spans C* ‹*fst fgh*›
using *fgh VVV.ide-char* $VVV.arr\text{-}char_{SbC}$ *arr-char* **by** *auto*
interpret *g*: *arrow-of-spans C* ‹*fst* (*snd fgh*)›

using *fgh VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char* **by** *auto*
interpret *h*: *arrow-of-spans C* ‹*snd* (*snd fgh*)›
using *fgh VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char* **by** *auto*
interpret *fgh*: *three-composable-arrows-of-spans C prj0 prj1*
‹*fst fgh*› ‹*fst* (*snd fgh*)› ‹*snd* (*snd fgh*)›
using *fgh VVV.ide-char VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *arr-char*
by (*unfold-locales*, *auto*)
interpret *fgh*: *three-composable-identity-arrows-of-spans C prj0 prj1*
‹*fst fgh*› ‹*fst* (*snd fgh*)› ‹*snd* (*snd fgh*)›
using *fgh VVV.ide-char*$_{SbC}$ *ide-char*
by *unfold-locales blast+*
have *1*: *arr* (α_{SB} *fgh*)
using *fgh* α*.preserves-reflects-arr VVV.ideD*(*1*) **by** *blast*
have *2*: α_{SB} *fgh* = *assoc*$_{SB}$ (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*))
using *fgh assoc-def* α*-ide* [*of fst fgh fst* (*snd fgh*) *snd* (*snd fgh*)]
VVV.ide-char$_{SbC}$ *VV.ide-char*$_{SbC}$ *VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$
by *simp*
moreover have *iso* ...
using *1 2 iso-char* [*of assoc*$_{SB}$ (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*))]
fgh.chine-assoc-inverse **by** *auto*
ultimately show *iso* (α_{SB} *fgh*) **by** *argo*
qed
qed

lemma *natural-isomorphism-*α:
shows *natural-isomorphism VVV.comp vcomp HoHV HoVH* α_{SB}
..

end

locale *four-composable-arrows-of-spans* =
span-bicategory +
μ: *arrow-of-spans C* μ +
ν: *arrow-of-spans C* ν +
π: *arrow-of-spans C* π +
ϱ: *arrow-of-spans C* ϱ +
$\mu\nu$: *two-composable-arrows-of-spans C prj0 prj1* μ ν +
$\nu\pi$: *two-composable-arrows-of-spans C prj0 prj1* ν π +
$\pi\varrho$: *two-composable-arrows-of-spans C prj0 prj1* π ϱ +
$\mu\nu\pi$: *three-composable-arrows-of-spans C prj0 prj1* μ ν π +
$\nu\pi\varrho$: *three-composable-arrows-of-spans C prj0 prj1* ν π ϱ
for μ (**structure**)
and ν (**structure**)
and π (**structure**)
and ϱ (**structure**)

locale *four-composable-identity-arrows-of-spans* =
four-composable-arrows-of-spans +
μ: *identity-arrow-of-spans C* μ +

ν: *identity-arrow-of-spans C* ν +
π: *identity-arrow-of-spans C* π +
ϱ: *identity-arrow-of-spans C* ϱ +
$\mu\nu$: *two-composable-identity-arrows-of-spans C prj0 prj1* μ ν +
$\nu\pi$: *two-composable-identity-arrows-of-spans C prj0 prj1* ν π +
$\pi\varrho$: *two-composable-identity-arrows-of-spans C prj0 prj1* π ϱ +
$\mu\nu\pi$: *three-composable-identity-arrows-of-spans C prj0 prj1* μ ν π +
$\nu\pi\varrho$: *three-composable-identity-arrows-of-spans C prj0 prj1* ν π ϱ
begin

interpretation *H*: *horizontal-composition vcomp hcomp src trg*
using *has-horizontal-composition* **by** *auto*

The following interpretations provide us with some systematic names for a lot of things.

interpretation $H\mu H\nu\pi$: *identity-arrow-of-spans C* ‹$\mu \star \nu \star \pi$›
using $\mu\nu\pi$*.composites-are-identities ide-char′* **by** *auto*
interpretation $HH\mu\nu\pi$: *identity-arrow-of-spans C* ‹$(\mu \star \nu) \star \pi$›
using $\mu\nu\pi$*.composites-are-identities ide-char′* **by** *auto*
interpretation $H\nu H\pi\varrho$: *identity-arrow-of-spans C* ‹$\nu \star \pi \star \varrho$›
using $\nu\pi\varrho$*.composites-are-identities ide-char′* **by** *auto*
interpretation $HH\nu\pi\varrho$: *identity-arrow-of-spans C* ‹$(\nu \star \pi) \star \varrho$›
using $\nu\pi\varrho$*.composites-are-identities ide-char′* **by** *auto*

interpretation $H\mu H\nu H\pi\varrho$: *arrow-of-spans C* ‹$\mu \star \nu \star \pi \star \varrho$›
using *arrow-of-spans-hcomp* $\mu\nu$*.composable* $\nu\pi\varrho$*.composites-are-arrows*(*1*) **by** *auto*
interpretation $H\mu HH\nu\pi\varrho$: *arrow-of-spans C* ‹$\mu \star (\nu \star \pi) \star \varrho$›
using *arrow-of-spans-hcomp* $\mu\nu$*.composable* $\nu\pi\varrho$*.composites-are-arrows*(*1*) **by** *auto*
interpretation $HH\mu\nu H\pi\varrho$: *arrow-of-spans C* ‹$(\mu \star \nu) \star \pi \star \varrho$›
using *hseq-char′ match-4* $\mu\nu\pi$*.composites-are-arrows*(*2*) $\pi\varrho$*.composite-is-arrow arr-char*
by *auto*
interpretation $HH\mu H\nu\pi\varrho$: *arrow-of-spans C* ‹$(\mu \star \nu \star \pi) \star \varrho$›
using *arrow-of-spans-hcomp* $\pi\varrho$*.composable* $\mu\nu\pi$*.composites-are-arrows*(*1*) **by** *auto*
interpretation $HHH\mu\nu\pi\varrho$: *arrow-of-spans C* ‹$((\mu \star \nu) \star \pi) \star \varrho$›
using *arrow-of-spans-hcomp* $\pi\varrho$*.composable* $\mu\nu\pi$*.composites-are-arrows*(*2*) **by** *auto*

interpretation $assoc\mu\nu\pi$: *arrow-of-spans C* ‹$assoc_{SB}$ μ ν π›
using *arr-char* $\mu\nu$*.composable* $\nu\pi$*.composable assoc-simps*(*1*) **by** *auto*
interpretation $assoc\nu\pi\varrho$: *arrow-of-spans C* ‹$assoc_{SB}$ ν π ϱ›
using *arr-char* $\nu\pi$*.composable* $\pi\varrho$*.composable assoc-simps*(*1*) **by** *auto*

interpretation μ-$\nu\pi$: *two-composable-identity-arrows-of-spans C prj0 prj1* μ ‹$\nu \star \pi$›
using $\mu\nu$*.composable* $\nu\pi$*.composable* **by** (*unfold-locales*, *auto*)
interpretation $\mu\nu$-π: *two-composable-identity-arrows-of-spans C prj0 prj1* ‹$\mu \star \nu$› π
using $\mu\nu$*.composable* $\nu\pi$*.composable* **by** (*unfold-locales*, *auto*)
interpretation ν-$\pi\varrho$: *two-composable-identity-arrows-of-spans C prj0 prj1* ν ‹$\pi \star \varrho$›
using $\nu\pi$*.composable* $\pi\varrho$*.composable* **by** (*unfold-locales*, *auto*)
interpretation $\nu\pi$-ϱ: *two-composable-identity-arrows-of-spans C prj0 prj1* ‹$\nu \star \pi$› ϱ
using $\nu\pi$*.composable* $\pi\varrho$*.composable* **by** (*unfold-locales*, *auto*)

interpretation μ-$\nu\pi$-ϱ: *three-composable-identity-arrows-of-spans C prj0 prj1* μ ‹$\nu \star \pi$› ϱ **..**
interpretation μ-ν-$\pi\varrho$: *three-composable-identity-arrows-of-spans C prj0 prj1* μ ν ‹$\pi \star \varrho$› **..**
interpretation $\mu\nu$-π-ϱ: *three-composable-identity-arrows-of-spans C prj0 prj1* ‹$\mu \star \nu$› π ϱ **..**

lemma *chines-eq*:
shows $H\mu HH\nu\pi\varrho$.*chine* = μ.*leg0* $\downarrow\downarrow$ $HH\nu\pi\varrho$.*leg1*
and $HH\mu H\nu\pi\varrho$.*chine* = *assoc*$\mu\nu\pi$.*cod.leg0* $\downarrow\downarrow$ ϱ.*leg1*
and $H\mu H\nu H\pi\varrho$.*chine* = μ.*leg0* $\downarrow\downarrow$ $H\nu H\pi\varrho$.*leg1*
proof −
  show $H\mu HH\nu\pi\varrho$.*chine* = μ.*leg0* $\downarrow\downarrow$ $HH\nu\pi\varrho$.*leg1*
    using *hcomp-def* [*of* μ *hcomp* (*hcomp* ν π) ϱ] *chine-hcomp-ide-ide* $\mu\nu$.*composable*
    by *simp*
  show $HH\mu H\nu\pi\varrho$.*chine* = *assoc*$\mu\nu\pi$.*cod.leg0* $\downarrow\downarrow$ ϱ.*leg1*
    by (*simp add*: μ-$\nu\pi$.*leg0-composite* μ-$\nu\pi$-ϱ.*chine-composite*(*1*))
  show $H\mu H\nu H\pi\varrho$.*chine* = μ.*leg0* $\downarrow\downarrow$ $H\nu H\pi\varrho$.*leg1*
    using *hcomp-def* [*of* μ *hcomp* ν (*hcomp* π ϱ)] *chine-hcomp-ide-ide* $\mu\nu$.*composable*
    by *simp*
qed

lemma *cospan-*μ*0-*$H\nu H\pi\varrho$*1*:
shows *C.cospan* μ.*leg0* $H\nu H\pi\varrho$.*leg1*
  by (*metis C.cod-comp* $H\nu H\pi\varrho$.*dom.leg-simps*(*3*) μ-ν-$\pi\varrho$.*cospan-*$\mu\nu$ $\nu\pi\varrho$.*leg1-composite*(*1*))

lemma *assoc-in-homs*:
shows «$\mu \star (assoc_{SB}\ \nu\ \pi\ \varrho) : \mu \star (\nu \star \pi) \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
and «$assoc_{SB}\ \mu\ (\nu \star \pi)\ \varrho : (\mu \star \nu \star \pi) \star \varrho \Rightarrow \mu \star (\nu \star \pi) \star \varrho$»
and «$assoc_{SB}\ \mu\ \nu\ \pi \star \varrho : ((\mu \star \nu) \star \pi) \star \varrho \Rightarrow (\mu \star \nu \star \pi) \star \varrho$»
and «$assoc_{SB}\ \mu\ \nu\ (\pi \star \varrho) : (\mu \star \nu) \star \pi \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
and «$assoc_{SB}\ (\mu \star \nu)\ \pi\ \varrho : ((\mu \star \nu) \star \pi) \star \varrho \Rightarrow (\mu \star \nu) \star \pi \star \varrho$»
proof −
  show «$\mu \star (assoc_{SB}\ \nu\ \pi\ \varrho) : \mu \star (\nu \star \pi) \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
    using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *auto*
  show «$assoc_{SB}\ \mu\ (\nu \star \pi)\ \varrho : (\mu \star \nu \star \pi) \star \varrho \Rightarrow \mu \star (\nu \star \pi) \star \varrho$»
    using *assoc-in-hom* $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *simp*
  show «$assoc_{SB}\ \mu\ \nu\ \pi \star \varrho : ((\mu \star \nu) \star \pi) \star \varrho \Rightarrow (\mu \star \nu \star \pi) \star \varrho$»
    using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *auto*
  show «$assoc_{SB}\ \mu\ \nu\ (\pi \star \varrho) : (\mu \star \nu) \star \pi \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
    using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable*
    by *auto*
  show «$assoc_{SB}\ (\mu \star \nu)\ \pi\ \varrho : ((\mu \star \nu) \star \pi) \star \varrho \Rightarrow (\mu \star \nu) \star \pi \star \varrho$»
    using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *auto*
qed

lemma *chine-composites*:
shows *Chn* ($\mu \star assoc_{SB}\ \nu\ \pi\ \varrho$) = *chine-hcomp* μ ($assoc_{SB}\ \nu\ \pi\ \varrho$)
and *Chn* ($assoc_{SB}\ \mu\ (\nu \star \pi)\ \varrho$) = μ-$\nu\pi$-ϱ.*chine-assoc*

and *Chn* ($assoc_{SB}$ μ ν π $\star$ ϱ) = *chine-hcomp* ($assoc_{SB}$ μ ν π) ϱ
and *Chn* ($assoc_{SB}$ μ ν (π $\star$ ϱ)) = μ-ν-$\pi\varrho$.*chine-assoc*
and *Chn* ($assoc_{SB}$ (μ $\star$ ν) π ϱ) = $\mu\nu$-π-ϱ.*chine-assoc*
proof −
  show *Chn* (μ $\star$ $assoc_{SB}$ ν π ϱ) = *chine-hcomp* μ ($assoc_{SB}$ ν π ϱ)
    using *hcomp-def* [*of* μ $assoc_{SB}$ ν π ϱ] *chine-hcomp-def* [*of* μ $assoc_{SB}$ ν π ϱ]
      $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable*
    by *auto*
  show *Chn* ($assoc_{SB}$ μ ν π $\star$ ϱ) = *chine-hcomp* ($assoc_{SB}$ μ ν π) ϱ
    using *assoc-in-homs*(*2*−*3*) *hcomp-def*
    by (*metis arrI arrow-of-spans-data.select-convs*(*1*) *hseqE*)
  show *Chn* ($assoc_{SB}$ μ (ν $\star$ π) ϱ) = μ-$\nu\pi$-ϱ.*chine-assoc*
    using *hcomp-def*
    by (*meson arrow-of-spans-data.select-convs*(*1*))
  show *Chn* ($assoc_{SB}$ μ ν (π $\star$ ϱ)) = μ-ν-$\pi\varrho$.*chine-assoc*
    using *hcomp-def*
    by (*meson arrow-of-spans-data.select-convs*(*1*))
  show *Chn* ($assoc_{SB}$ (μ $\star$ ν) π ϱ) = $\mu\nu$-π-ϱ.*chine-assoc*
    using *hcomp-def*
    by (*meson arrow-of-spans-data.select-convs*(*1*))
qed

lemma *prj-in-homs* [*intro*, *simp*]:
shows «p_0[μ.*leg0*, $HH\nu\pi\varrho$.*leg1*] : $H\mu HH\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\nu\pi\varrho$.*chine*»
and «p_1[μ.*leg0*, $H\nu H\pi\varrho$.*leg1*] : $H\mu H\nu H\pi\varrho$.*chine* $\rightarrow_C$ μ.*apex*»
and «p_1[$assoc\mu\nu\pi$.*cod.leg0*, ϱ.*cod.leg1*] : $HH\mu H\nu\pi\varrho$.*chine* $\rightarrow_C$ $H\mu H\nu\pi$.*chine*»
and «p_0[$HH\mu\nu\pi$.*leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ ϱ.*chine*»
and «p_1[$HH\mu\nu\pi$.*leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\mu\nu\pi$.*chine*»
and «p_1[$\nu\pi$.*leg0* · μ-$\nu\pi$.prj_0, ϱ.*leg1*] : $HH\mu H\nu\pi\varrho$.*chine* $\rightarrow_C$ $H\mu H\nu\pi$.*chine*»
and «p_1[$assoc\mu\nu\pi$.*dom.leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\mu\nu\pi$.*chine*»
and «μ-$\nu\pi$.prj_0 : $H\mu H\nu\pi$.*chine* $\rightarrow_C$ $\nu\pi$.*apex*»
proof −
  show «p_0[μ.*leg0*, $HH\nu\pi\varrho$.*leg1*] : $H\mu HH\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\nu\pi\varrho$.*chine*»
    by (*metis C.dom-comp C.prj0-in-hom′ C.prj1-simps-arr* $HH\nu\pi\varrho$.*chine-eq-apex*
      $HH\nu\pi\varrho$.*cod.apex-def* $HH\nu\pi\varrho$.*cod.is-span* $HH\nu\pi\varrho$.*cod-simps*(*1*) $HH\nu\pi\varrho$.*cod-simps*(*3*)
      μ-$\nu\pi$-ϱ.*prj-simps*(*10*) μ-$\nu\pi$-ϱ.*prj-simps*(*11*) μ-$\nu\pi$-ϱ.*prj-simps*(*14*) $\nu\pi$-ϱ.*leg1-composite*)
  show «p_1[$assoc\mu\nu\pi$.*cod.leg0*, ϱ.*cod.leg1*] : $HH\mu H\nu\pi\varrho$.*chine* $\rightarrow_C$ $H\mu H\nu\pi$.*chine*»
    using $\mu\nu\pi$.*leg0-composite*(*1*) $\mu\nu$-π-ϱ.*prj-simps*(*3*) μ-$\nu\pi$.*leg0-composite*
      μ-$\nu\pi$-ϱ.*prj-simps*(*6*)
    by *auto*
  show «p_0[$HH\mu\nu\pi$.*leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ ϱ.*chine*»
    by (*simp add*: $\mu\nu$-π.*leg0-composite* $\mu\nu$-π-ϱ.*prj-in-hom*(*3*))
  show «p_1[$HH\mu\nu\pi$.*leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\mu\nu\pi$.*chine*»
    using $\mu\nu$.*leg0-composite* $\mu\nu$-π.*leg0-composite* $\mu\nu$-π-ϱ.*prj-in-hom*(*2*) **by** *fastforce*
  show «p_1[$\nu\pi$.*leg0* · μ-$\nu\pi$.prj_0, ϱ.*leg1*] : $HH\mu H\nu\pi\varrho$.*chine* $\rightarrow_C$ $H\mu H\nu\pi$.*chine*»
    using $\mu\nu\pi$.*leg0-composite*(*1*) μ-$\nu\pi$.*leg0-composite* μ-$\nu\pi$-ϱ.*prj-simps*(*3*)
      μ-$\nu\pi$-ϱ.*prj-simps*(*6*)
    by *force*
  show «p_1[$assoc\mu\nu\pi$.*dom.leg0*, ϱ.*leg1*] : $HHH\mu\nu\pi\varrho$.*chine* $\rightarrow_C$ $HH\mu\nu\pi$.*chine*»

using ‹«$\mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$: $HHH\mu\nu\pi\varrho.chine \to_C HH\mu\nu\pi.chine$»› **by** *fastforce*
show «$\mathrm{p}_1[\mu.leg0,\ H\nu H\pi\varrho.leg1]$: $H\mu H\nu H\pi\varrho.chine \to_C \mu.apex$»
by (*simp add*: μ-ν-$\pi\varrho$.*prj-in-hom*(*4*) ν-$\pi\varrho$.*leg1-composite*)
show «μ-$\nu\pi.prj_0$: $H\mu H\nu\pi.chine \to_C \nu\pi.apex$»
using μ-$\nu\pi$.*chine-composite* μ-$\nu\pi$.*prj-in-hom*(*2*) **by** *presburger*
qed

lemma *chine-in-homs* [*intro*, *simp*]:
shows «$assoc\mu\nu\pi.chine$: $HH\mu\nu\pi.chine \to_C H\mu H\nu\pi.chine$»
and «$assoc\nu\pi\varrho.chine$: $HH\nu\pi\varrho.chine \to_C H\nu H\pi\varrho.chine$»
and «*chine-hcomp* μ ($assoc_{SB}$ ν π ϱ) : $H\mu HH\nu\pi\varrho.chine \to_C H\mu H\nu H\pi\varrho.chine$»
and «*chine-hcomp* ($assoc_{SB}$ μ ν π) ϱ : $HHH\mu\nu\pi\varrho.chine \to_C HH\mu H\nu\pi\varrho.chine$»
proof −
show *1*: «$assoc\mu\nu\pi.chine$: $HH\mu\nu\pi.chine \to_C H\mu H\nu\pi.chine$»
using $\mu\nu\pi$.*chine-assoc-in-hom* **by** *simp*
show «$assoc\nu\pi\varrho.chine$: $HH\nu\pi\varrho.chine \to_C H\nu H\pi\varrho.chine$»
using $\nu\pi\varrho$.*chine-assoc-in-hom* **by** *simp*
show «*chine-hcomp* μ ($assoc_{SB}$ ν π ϱ) : $H\mu HH\nu\pi\varrho.chine \to_C H\mu H\nu H\pi\varrho.chine$»
using *Chn-in-hom* **by** (*metis assoc-in-homs*(*1*) *chine-composites*(*1*))
show «*chine-hcomp* ($assoc_{SB}$ μ ν π) ϱ : $HHH\mu\nu\pi\varrho.chine \to_C HH\mu H\nu\pi\varrho.chine$»
using *Chn-in-hom* **by** (*metis assoc-in-homs*(*3*) *chine-composites*(*3*))
qed

lemma *commutative-squares* [*intro*, *simp*]:
shows $C.commutative\text{-}square$ $\nu\pi.leg0$ $\varrho.leg1$ μ-$\nu\pi$-$\varrho.Prj_{01}$ μ-$\nu\pi$-$\varrho.Prj_0$
and $C.commutative\text{-}square$ $\nu.leg0$ $\pi\varrho.leg1$ μ-ν-$\pi\varrho.Prj_{01}$ μ-ν-$\pi\varrho.Prj_0$
and $C.commutative\text{-}square$ $\mathrm{p}_0[\mu.cod.leg0,\ assoc\nu\pi\varrho.cod.leg1]$ $assoc\nu\pi\varrho.chine$
($chine\text{-}hcomp$ μ ($assoc_{SB}$ ν π ϱ)) $\mathrm{p}_0[\mu.leg0,\ assoc\nu\pi\varrho.dom.leg1]$
and $C.commutative\text{-}square$ $\mathrm{p}_1[\mu.cod.leg0,\ assoc\nu\pi\varrho.cod.leg1]$ $\mu.chine$
($chine\text{-}hcomp$ μ ($assoc_{SB}$ ν π ϱ)) $\mathrm{p}_1[\mu.leg0,\ assoc\nu\pi\varrho.dom.leg1]$
and $C.commutative\text{-}square$ $assoc\mu\nu\pi.cod.leg0$ $\varrho.cod.leg1$
($assoc\mu\nu\pi.chine \cdot \mathrm{p}_1[assoc\mu\nu\pi.dom.leg0,\ \varrho.leg1]$)
($\varrho.chine \cdot \mathrm{p}_0[assoc\mu\nu\pi.dom.leg0,\ \varrho.leg1]$)
and $C.commutative\text{-}square$ $\mu.leg0$ ($\nu\pi.leg1 \cdot \nu\pi$-$\varrho.prj_1$) μ-$\nu\pi$-$\varrho.Prj_{11}$
$\langle\mu$-$\nu\pi$-$\varrho.Prj_{01}$ $[\![\nu\pi.leg0,\ \varrho.leg1]\!]$ μ-$\nu\pi$-$\varrho.Prj_0\rangle$
and $C.commutative\text{-}square$ $\mu.leg0$ ($\nu.leg1 \cdot \nu$-$\pi\varrho.prj_1$) μ-ν-$\pi\varrho.Prj_{11}$
$\langle\mu$-ν-$\pi\varrho.Prj_{01}$ $[\![\nu.leg0,\ \pi\varrho.leg1]\!]$ μ-ν-$\pi\varrho.Prj_0\rangle$
proof −
show *1*: $C.commutative\text{-}square$ $\nu\pi.leg0$ $\varrho.leg1$ μ-$\nu\pi$-$\varrho.Prj_{01}$ μ-$\nu\pi$-$\varrho.Prj_0$
proof −
have *1*: $C.arr$ μ-$\nu\pi$-$\varrho.Prj_0 \wedge C.dom$ μ-$\nu\pi$-$\varrho.Prj_0 = HH\mu H\nu\pi\varrho.chine \wedge$
$C.cod$ μ-$\nu\pi$-$\varrho.Prj_0 = \varrho.apex$
by (*meson* $C.in\text{-}homE$ μ-$\nu\pi$-ϱ.*prj-in-hom*(*3*))
hence ($\nu\pi.leg0 \cdot \mu$-$\nu\pi.prj_0$) $\cdot \mathrm{p}_1[\nu\pi.leg0 \cdot \mu$-$\nu\pi.prj_0,\ \varrho.leg1] = \varrho.leg1 \cdot \mu$-$\nu\pi$-$\varrho.Prj_0$
by (*meson* $C.prj0\text{-}simps\text{-}arr$ $C.pullback\text{-}commutes'$)
thus *?thesis*
using *1* $C.comp\text{-}assoc$ $\nu\pi$-ϱ.*legs-form-cospan*(*1*) **by** *simp*
qed
show *2*: $C.commutative\text{-}square$ $\nu.leg0$ $\pi\varrho.leg1$ μ-ν-$\pi\varrho.Prj_{01}$ μ-ν-$\pi\varrho.Prj_0$

proof −
  have ν.*leg0* · $\mu\nu$.*prj*$_0$ · p_1[ν.*leg0* · $\mu\nu$.*prj*$_0$, $\pi\varrho$.*leg1*] =
      $\pi\varrho$.*leg1* · p_0[ν.*leg0* · $\mu\nu$.*prj*$_0$, $\pi\varrho$.*leg1*]
    by (*metis* (*no-types*) *C.category-axioms C.prj0-simps-arr C.pullback-commutes′*
      category.comp-reduce μ-ν-$\pi\varrho$.*prj-simps*(*2*) μ-ν-$\pi\varrho$.*prj-simps*(*3*))
  thus *?thesis*
    using *C.commutative-square-def* μ-ν-$\pi\varrho$.*cospan*-$\nu\pi$
      μ-ν-$\pi\varrho$.*prj-simps*(*2*) μ-ν-$\pi\varrho$.*prj-simps*(*3*) μ-ν-$\pi\varrho$.*prj-simps*(*5*) μ-ν-$\pi\varrho$.*prj-simps*(*6*)
      μ-ν-$\pi\varrho$.*prj-simps*(*8*) ν.*dom.apex-def*
    by *presburger*
qed
show *C.commutative-square* p_0[μ.*cod.leg0*, *assoc*$\nu\pi\varrho$.*cod.leg1*] *assoc*$\nu\pi\varrho$.*chine*
      (*chine-hcomp* μ (*assoc*$_{SB}$ ν π ϱ)) p_0[μ.*leg0*, *assoc*$\nu\pi\varrho$.*dom.leg1*]
  using *assoc-in-homs*(*1*) *chine-hcomp-props*(*4*) [*of assoc*$_{SB}$ ν π ϱ μ] *hseq-char* **by** *blast*
show *C.commutative-square* p_1[μ.*cod.leg0*, *assoc*$\nu\pi\varrho$.*cod.leg1*] μ.*chine*
      (*chine-hcomp* μ (*assoc*$_{SB}$ ν π ϱ)) p_1[μ.*leg0*, *assoc*$\nu\pi\varrho$.*dom.leg1*]
  using *chine-hcomp-props*(*3*) [*of assoc*$_{SB}$ ν π ϱ μ] *hseq-char*
      $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable*
  by *auto*
show *C.commutative-square assoc*$\mu\nu\pi$.*cod.leg0* ϱ.*cod.leg1*
      (*assoc*$\mu\nu\pi$.*chine* · p_1[*assoc*$\mu\nu\pi$.*dom.leg0*, ϱ.*leg1*])
      (ϱ.*chine* · p_0[*assoc*$\mu\nu\pi$.*dom.leg0*, ϱ.*leg1*])
  using *assoc-in-homs*(*3*) *hseq-char chine-hcomp-props*(*2*) **by** *blast*
show *C.commutative-square* μ.*leg0* ($\nu\pi$.*leg1* · $\nu\pi$-ϱ.*prj*$_1$) μ-$\nu\pi$-ϱ.*Prj*$_{11}$
      ⟨μ-$\nu\pi$-ϱ.*Prj*$_{01}$ ⟦$\nu\pi$.*leg0*, ϱ.*leg1*⟧ μ-$\nu\pi$-ϱ.*Prj*$_0$⟩
proof
  show *C.cospan* μ.*leg0* ($\nu\pi$.*leg1* · $\nu\pi$-ϱ.*prj*$_1$)
    using *HH*$\nu\pi\varrho$.*dom.leg-simps*(*1*) μ-$\nu\pi$-ϱ.*cospan*-$\mu\nu$ *C.arrI* **by** *fastforce*
  show *C.span* μ-$\nu\pi$-ϱ.*Prj*$_{11}$ ⟨μ-$\nu\pi$-ϱ.*Prj*$_{01}$ ⟦$\nu\pi$.*leg0*, ϱ.*leg1*⟧ μ-$\nu\pi$-ϱ.*Prj*$_0$⟩
    using *1* μ-$\nu\pi$-ϱ.*prj-in-hom*(*1*) **by** *auto*
  show *C.dom* μ.*leg0* = *C.cod* μ-$\nu\pi$-ϱ.*Prj*$_{11}$
    by *simp*
  show μ.*leg0* · μ-$\nu\pi$-ϱ.*Prj*$_{11}$ =
      ($\nu\pi$.*leg1* · $\nu\pi$-ϱ.*prj*$_1$) · ⟨μ-$\nu\pi$-ϱ.*Prj*$_{01}$ ⟦$\nu\pi$.*leg0*, ϱ.*leg1*⟧ μ-$\nu\pi$-ϱ.*Prj*$_0$⟩
    by (*metis* (*no-types*, *lifting*) *1 C.comp-assoc C.prj-tuple*(*2*) *C.pullback-commutes′*
      μ-$\nu\pi$-ϱ.*cospan*-$\mu\nu$)
qed
show *C.commutative-square* μ.*leg0* (ν.*leg1* · ν-$\pi\varrho$.*prj*$_1$) μ-ν-$\pi\varrho$.*Prj*$_{11}$
      ⟨μ-ν-$\pi\varrho$.*Prj*$_{01}$ ⟦ν.*leg0*, $\pi\varrho$.*leg1*⟧ μ-ν-$\pi\varrho$.*Prj*$_0$⟩
proof
  show *C.cospan* μ.*leg0* (ν.*leg1* · ν-$\pi\varrho$.*prj*$_1$)
    using *C.arrI* μ-ν-$\pi\varrho$.*prj-in-hom*(*4*) **by** *auto*
  show *C.span* μ-ν-$\pi\varrho$.*Prj*$_{11}$ ⟨μ-ν-$\pi\varrho$.*Prj*$_{01}$ ⟦ν.*leg0*, $\pi\varrho$.*leg1*⟧ μ-ν-$\pi\varrho$.*Prj*$_0$⟩
    using *2* **by** *fastforce*
  thus *C.dom* μ.*leg0* = *C.cod* μ-ν-$\pi\varrho$.*Prj*$_{11}$
    using μ-ν-$\pi\varrho$.*cospan*-$\mu\nu$ **by** *simp*
  show μ.*leg0* · μ-ν-$\pi\varrho$.*Prj*$_{11}$ =
      (ν.*leg1* · ν-$\pi\varrho$.*prj*$_1$) · ⟨μ-ν-$\pi\varrho$.*Prj*$_{01}$ ⟦ν.*leg0*, $\pi\varrho$.*leg1*⟧ μ-ν-$\pi\varrho$.*Prj*$_0$⟩
    by (*metis* (*no-types*, *lifting*) *2 C.comp-assoc C.prj-tuple*(*2*) *C.pullback-commutes′*

μ-ν-$\pi\varrho$.*cospan*-$\mu\nu$)
qed
qed

lemma *chine-pentagon*:
shows *Chn* $((\mu \star \mathit{assoc}_{SB}\ \nu\ \pi\ \varrho) \cdot \mathit{assoc}_{SB}\ \mu\ (\nu \star \pi)\ \varrho \cdot (\mathit{assoc}_{SB}\ \mu\ \nu\ \pi \star \varrho)) =$
$\mathit{Chn}\ (\mathit{assoc}_{SB}\ \mu\ \nu\ (\pi \star \varrho) \cdot \mathit{assoc}_{SB}\ (\mu \star \nu)\ \pi\ \varrho)$
proof −
let *?LHS* $= (\mu \star \mathit{assoc}_{SB}\ \nu\ \pi\ \varrho) \cdot \mathit{assoc}_{SB}\ \mu\ (\nu \star \pi)\ \varrho \cdot (\mathit{assoc}_{SB}\ \mu\ \nu\ \pi \star \varrho)$
let *?RHS* $= \mathit{assoc}_{SB}\ \mu\ \nu\ (\pi \star \varrho) \cdot \mathit{assoc}_{SB}\ (\mu \star \nu)\ \pi\ \varrho$

have *LHS-in-hom*: «*?LHS* : $((\mu \star \nu) \star \pi) \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *auto*
have *RHS-in-hom*: «*?RHS* : $((\mu \star \nu) \star \pi) \star \varrho \Rightarrow \mu \star \nu \star \pi \star \varrho$»
using $\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable* **by** *auto*

have *1*: *arrow-of-spans* $(\cdot)$ *?LHS*
using *arr-char assoc-in-homs*(*1*−*3*) **by** *blast*

have *L*: *Chn ?LHS* = *chine-hcomp* μ $(\mathit{assoc}_{SB}\ \nu\ \pi\ \varrho) \cdot \mu$-$\nu\pi$-$\varrho$.*chine-assoc* ·
chine-hcomp $(\mathit{assoc}_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using *Chn-vcomp 1 arr-char chine-composites*(*1*) *chine-composites*(*3*) *seq-char*
by *fastforce*
have *R*: *Chn ?RHS* = μ-ν-$\pi\varrho$.*chine-assoc* · $\mu\nu$-π-ϱ.*chine-assoc*
using *Chn-vcomp assoc-in-homs*(*4*) *assoc-in-homs*(*5*) *seqI′* **by** *auto*

The outline of the proof is to show that the compositions of *?LHS* and *?RHS* with the two projections $\mathrm{p}_0[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}]$ and $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}]$ are equal, and then apply $\nu\pi\varrho$.*prj′*-*joint-monic*.

The case for projection $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}]$ does not have subcases, so we'll dispatch that one first.

have $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{Chn}\ \mathit{?LHS} = \mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{Chn}\ \mathit{?RHS}$
proof −
have $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{Chn}\ \mathit{?LHS} = \mu\nu.\mathit{prj}_1 \cdot \mu\nu$-$\pi$-$\varrho.\mathit{Prj}_{11}$
proof −
have $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{Chn}\ \mathit{?LHS} =$
$\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{chine\text{-}hcomp}\ \mu\ (\mathit{assoc}_{SB}\ \nu\ \pi\ \varrho) \cdot \mu$-$\nu\pi$-$\varrho$.*chine-assoc* ·
chine-hcomp $(\mathit{assoc}_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using *L* **by** *simp*
also have ... $= \mu.\mathit{chine} \cdot \mathrm{p}_1[\mu.\mathit{leg0},\ HH\nu\pi\varrho.\mathit{leg1}] \cdot \mu$-$\nu\pi$-$\varrho$.*chine-assoc* ·
chine-hcomp $(\mathit{assoc}_{SB}\ \mu\ \nu\ \pi)\ \varrho$
proof −
have $\mathrm{p}_1[\mu.\mathit{leg0},\ H\nu H\pi\varrho.\mathit{leg1}] \cdot \mathit{chine\text{-}hcomp}\ \mu\ (\mathit{assoc}_{SB}\ \nu\ \pi\ \varrho) =$
$\mu.\mathit{chine} \cdot \mathrm{p}_1[\mu.\mathit{leg0},\ HH\nu\pi\varrho.\mathit{leg1}]$
proof −
have *C*.*commutative-square* $\mathrm{p}_1[\mu.\mathit{cod.leg0},\ \mathit{assoc}\nu\pi\varrho.\mathit{cod.leg1}]$ $\mu.\mathit{chine}$
$(\mathit{chine\text{-}hcomp}\ \mu\ (\mathit{assoc}_{SB}\ \nu\ \pi\ \varrho))$ $\mathrm{p}_1[\mu.\mathit{leg0},\ \mathit{assoc}\nu\pi\varrho.\mathit{dom.leg1}]$
by *blast*
thus *?thesis* **by** *auto*

```
      qed
      thus ?thesis
      using C.comp-permute [of p1[μ.leg0, HνHπϱ.leg1] chine-hcomp μ (assocSB ν π ϱ)
                                μ.chine p1[μ.leg0, HHνπϱ.leg1]
                                μ-νπ-ϱ.chine-assoc · chine-hcomp (assocSB μ ν π) ϱ]
        by blast
    qed
    also have ... = μ.chine · μ-νπ-ϱ.Prj11 · chine-hcomp (assocSB μ ν π) ϱ
      using C.comp-reduce [of p1[μ.leg0, HHνπϱ.leg1] μ-νπ-ϱ.chine-assoc]
          νπ-ϱ.leg1-composite
      by fastforce
    also have ... = μ.chine · μν.prj1 · μν-π-ϱ.Prj11
    proof −
      have μ.chine · μ-νπ-ϱ.Prj11 · chine-hcomp (assocSB μ ν π) ϱ =
          μ.chine · μ-νπ.prj1 · p1[νπ.leg0 · μ-νπ.prj0, ϱ.leg1] ·
            chine-hcomp (assocSB μ ν π) ϱ
        using C.comp-assoc by simp
      also have ... = μ.chine · (μ-νπ.prj1 · assocμνπ.chine) · p1[assocμνπ.dom.leg0, ϱ.leg1]
      proof −
        have p1[νπ.leg0 · μ-νπ.prj0, ϱ.leg1] · chine-hcomp (assocSB μ ν π) ϱ =
            assocμνπ.chine · p1[assocμνπ.dom.leg0, ϱ.leg1]
          using chine-hcomp-props(6) [of ϱ assocSB μ ν π] hcomp-def [of μ hcomp ν π]
              μν.composable νπ.composable πϱ.composable
              hseq-char assoc-in-homs(3)
          by auto
        thus ?thesis
          using C.comp-assoc by auto
      qed
      also have ... = μ.chine · μνπ.Prj11 · p1[assocμνπ.dom.leg0, ϱ.leg1]
        using μνπ.prj-chine-assoc(1) hcomp-def νπ.composable by auto
      also have ... = μ.chine · μν.prj1 · μν-π-ϱ.Prj11
      proof −
        have μνπ.Prj11 · p1[assocμνπ.dom.leg0, ϱ.leg1] = μν.prj1 · μν-π-ϱ.Prj11
          by (simp add: C.comp-assoc μν.leg0-composite μν-π.leg0-composite)
        thus ?thesis by simp
      qed
      finally show ?thesis by blast
    qed
    also have ... = μν.prj1 · μν-π-ϱ.Prj11
      using μν-π-ϱ.prj-in-hom(1) hcomp-def [of μ ν] chine-hcomp-ide-ide μν.cod.apex-def
          μν.composable μ-ν-πϱ.cospan-μν C.comp-ide-arr
      by auto
    finally show ?thesis by blast
  qed
  also have ... = p1[μ.leg0, HνHπϱ.leg1] · Chn ?RHS
    by (metis C.comp-assoc R μν.leg0-composite μν-π-ϱ.prj-chine-assoc(1)
            μ-ν-πϱ.prj-chine-assoc(1) ν-πϱ.leg1-composite πϱ.leg1-composite)
  finally show ?thesis by blast
qed
```

Now for the case of $\mathrm{p}_0[\mu.leg0,\ H\nu H\pi\varrho.leg1]$. We have to consider three sub-cases, involving the compositions with the projections $\nu\pi\varrho.Prj_1$, $\nu\pi\varrho.Prj_{10}$, and $\nu\pi\varrho.Prj_{00}$.

moreover have $\mathrm{p}_0[\mu.leg0,\ H\nu H\pi\varrho.leg1] \cdot Chn\ ?LHS =$
$\mathrm{p}_0[\mu.leg0,\ H\nu H\pi\varrho.leg1] \cdot Chn\ ?RHS$
proof −

have A: $\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc =$
$\langle \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01}\ [\![\nu\pi.leg0,\ \varrho.leg1]\!]\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_0\rangle$
using $\mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc\text{-}def\ \nu\pi\text{-}\varrho.leg1\text{-}composite$ **by** *auto*
have B: $\mu\text{-}\nu\pi\text{-}\varrho.Prj_{01} \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\mu\text{-}\nu\pi.prj_0 \cdot \mu\nu\pi.chine\text{-}assoc \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
proof −
have $\mu\text{-}\nu\pi\text{-}\varrho.Prj_{01} \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$(\mu\text{-}\nu\pi.prj_0 \cdot \mathrm{p}_1[assoc\mu\nu\pi.cod.leg0,\ \varrho.cod.leg1]) \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using $\mu\text{-}\nu\pi.composable\ \nu\pi.composite\text{-}is\text{-}arrow\ hcomp\text{-}def$ **by** *auto*
also have ... $= \mu\text{-}\nu\pi.prj_0 \cdot \mathrm{p}_1[assoc\mu\nu\pi.cod.leg0,\ \varrho.cod.leg1] \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using $C.comp\text{-}assoc$ **by** *simp*
also have ... $= \mu\text{-}\nu\pi.prj_0 \cdot \mu\nu\pi.chine\text{-}assoc \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
proof −
have $HH\mu\nu\pi.leg0 = assoc\mu\nu\pi.dom.leg0$
using $hcomp\text{-}def$ [*of hcomp* $\mu\ \nu\ \pi$] **by** *simp*
moreover have $C.commutative\text{-}square\ assoc\mu\nu\pi.cod.leg0\ \varrho.cod.leg1$
$(assoc\mu\nu\pi.chine \cdot \mathrm{p}_1[assoc\mu\nu\pi.dom.leg0,\ \varrho.leg1])$
$(\varrho.chine \cdot \mathrm{p}_0[assoc\mu\nu\pi.dom.leg0,\ \varrho.leg1])$
by *blast*
ultimately show *?thesis*
using $chine\text{-}hcomp\text{-}def$ [*of* $assoc_{SB}\ \mu\ \nu\ \pi\ \varrho$] **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
have $*$: $assoc\nu\pi\varrho.chine \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\langle \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ [\![\nu.leg0,\ \pi\varrho.leg1]\!]\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0\rangle \cdot \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
proof −

Subcase $\nu\pi\varrho.Prj_1$:

have $\nu\pi\varrho.Prj_1 \cdot assoc\nu\pi\varrho.chine \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\nu\pi\varrho.Prj_1 \cdot \langle \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ [\![\nu.leg0,\ \pi\varrho.leg1]\!]\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0\rangle \cdot \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
proof −
have $\nu\pi\varrho.Prj_1 \cdot assoc\nu\pi\varrho.chine \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\nu\pi\varrho.Prj_{11} \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using $\nu\pi\varrho.chine\text{-}assoc\text{-}props(1)\ C.prj0\text{-}in\text{-}hom$ [*of* $\mu.leg0\ HH\nu\pi\varrho.leg1$] $cospan\text{-}\mu0\text{-}H\nu H\pi\varrho1$
$C.comp\text{-}reduce$ [*of* $\nu\pi\varrho.Prj_1\ assoc\nu\pi\varrho.chine\ \nu\pi\varrho.Prj_{11}$
$\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$]

by *auto*
also have ... = $\nu\pi\varrho.Prj_{11} \cdot \langle \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01}\ [\![\nu\pi.leg0,\ \varrho.leg1]\!]\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_0\rangle \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using *A C.comp-reduce* [*of* $\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1]\ \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc$]
by *fastforce*
also have ... = $\nu\pi.prj_1 \cdot \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01} \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
by (*metis A C.comp-assoc* $\mu\text{-}\nu\pi\text{-}\varrho.prj\text{-}chine\text{-}assoc(2)$ $\nu\pi.leg0\text{-}composite$
$\nu\pi\text{-}\varrho.leg1\text{-}composite$)
also have ... = $\nu\pi.prj_1 \cdot \mu\text{-}\nu\pi.prj_0 \cdot \mu\nu\pi.chine\text{-}assoc \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
using *B* **by** *simp*
also have ... = $\mu\nu\pi.Prj_{01} \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
using *hcomp-def* [*of* $\nu\ \pi$] $\nu\pi.composable$ *C.comp-assoc*
C.comp-reduce [*of* $\mu\nu\pi.Prj_{10}$ $\mu\nu\pi.chine\text{-}assoc$ $\mu\nu\pi.Prj_{01}$ $\mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$]
$\mu\nu\pi.prj\text{-}in\text{-}hom(5)$ $\mu\nu\pi.prj\text{-}chine\text{-}assoc(2)$
by *auto*
also have ... = $\nu\pi\varrho.Prj_1 \cdot$
$\langle \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ [\![\nu.leg0,\ \pi\varrho.leg1]\!]\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0\rangle \cdot \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
by (*metis* (*no-types*, *lifting*) *C.comp-assoc C.prj-tuple*(2) $\mu\nu.leg0\text{-}composite$
$\mu\nu\pi.leg0\text{-}composite(2)$ $\mu\nu\text{-}\pi\text{-}\varrho.prj\text{-}chine\text{-}assoc(1)$ $\pi\varrho.leg1\text{-}composite$
commutative-squares(2))
finally show *?thesis* **by** *simp*
qed
moreover

Subcase $\nu\pi\varrho.Prj_{10}$:

have $\nu\pi\varrho.Prj_{10} \cdot assoc\nu\pi\varrho.chine \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\nu\pi\varrho.Prj_{10} \cdot \langle \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ [\![\nu.leg0,\ \pi\varrho.leg1]\!]\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0\rangle \cdot \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
proof −
have $\nu\pi\varrho.Prj_{10} \cdot assoc\nu\pi\varrho.chine \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho =$
$\nu\pi\varrho.Prj_{01} \cdot \mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using *C.comp-reduce* [*of* $\nu\pi\varrho.Prj_{10}$ $assoc\nu\pi\varrho.chine$ $\nu\pi\varrho.Prj_{01}$
$\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] \cdot \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$]
by *auto*
also have ... = $\nu\pi\varrho.Prj_{01} \cdot \langle \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01}\ [\![\nu\pi.leg0,\ \varrho.leg1]\!]\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_0\rangle \cdot$
$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
using *A C.comp-reduce* [*of* $\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1]\ \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc$]
by *fastforce*
also have ... = $\nu\pi.prj_0 \cdot \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01} \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
by (*metis A C.comp-assoc* $\mu\text{-}\nu\pi\text{-}\varrho.prj\text{-}chine\text{-}assoc(2)$ $\nu\pi.leg0\text{-}composite$
$\nu\pi\text{-}\varrho.leg1\text{-}composite$)
also have ... = $\nu\pi.prj_0 \cdot \mu\text{-}\nu\pi.prj_0 \cdot \mu\nu\pi.chine\text{-}assoc \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
using *B* **by** *simp*
also have ... = $\mu\nu\pi.Prj_0 \cdot \mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1]$
using *hcomp-def* [*of* $\nu\ \pi$] $\nu\pi.composable$ $\mu\nu\pi.prj\text{-}in\text{-}hom(6)$
C.comp-reduce [*of* $\nu\pi.prj_0$ $\mu\text{-}\nu\pi.prj_0$ $\mu\nu\pi.Prj_{00}$

$\mu\nu\pi$.*chine-assoc* · p_1[*HH*$\mu\nu\pi$.*leg0*, ϱ.*leg1*]]
C.comp-reduce [*of* $\mu\nu\pi$.*Prj*$_{00}$ $\mu\nu\pi$.*chine-assoc* $\mu\nu\pi$.*Prj*$_0$
p_1[*HH*$\mu\nu\pi$.*leg0*, ϱ.*leg1*]]
by *fastforce*
also have ... = $\nu\pi\varrho$.*Prj*$_{10}$ ·
$\langle\mu$-ν-$\pi\varrho$.*Prj*$_{01}$ $[\![\nu$.*leg0*, $\pi\varrho$.*leg1*$]\!]$ μ-ν-$\pi\varrho$.*Prj*$_0\rangle$ · $\mu\nu$-π-ϱ.*chine-assoc*
by (*metis C.comp-assoc C.tuple-prj* $\mu\nu$.*leg0-composite* $\mu\nu$-π.*leg0-composite*
$\mu\nu$-π-ϱ.*prj-chine-assoc*(*2*) μ-ν-$\pi\varrho$.*cospan-*$\nu\pi$ μ-ν-$\pi\varrho$.*prj-chine-assoc*(*2*)
μ-ν-$\pi\varrho$.*prj-chine-assoc*(*3*) μ-ν-$\pi\varrho$.*prj-simps*(*2*) $\pi\varrho$.*leg1-composite*)
finally show *?thesis* **by** *blast*
qed
moreover

Subcase $\nu\pi\varrho$.*Prj*$_{00}$:

have $\nu\pi\varrho$.*Prj*$_{00}$ · *assoc*$\nu\pi\varrho$.*chine* · p_0[μ.*leg0*, *HH*$\nu\pi\varrho$.*leg1*] · μ-$\nu\pi$-ϱ.*chine-assoc* ·
chine-hcomp (*assoc*$_{SB}$ μ ν π) ϱ =
$\nu\pi\varrho$.*Prj*$_{00}$ · $\langle\mu$-ν-$\pi\varrho$.*Prj*$_{01}$ $[\![\nu$.*leg0*, $\pi\varrho$.*leg1*$]\!]$ μ-ν-$\pi\varrho$.*Prj*$_0\rangle$ · $\mu\nu$-π-ϱ.*chine-assoc*
proof −
have $\nu\pi\varrho$.*Prj*$_{00}$ · *assoc*$\nu\pi\varrho$.*chine* · p_0[μ.*leg0*, *HH*$\nu\pi\varrho$.*leg1*] · μ-$\nu\pi$-ϱ.*chine-assoc* ·
chine-hcomp (*assoc*$_{SB}$ μ ν π) ϱ =
$\nu\pi\varrho$.*Prj*$_0$ · p_0[μ.*leg0*, *HH*$\nu\pi\varrho$.*leg1*] · μ-$\nu\pi$-ϱ.*chine-assoc* ·
chine-hcomp (*assoc*$_{SB}$ μ ν π) ϱ
using *C.comp-reduce* [*of* $\nu\pi\varrho$.*Prj*$_{00}$ *assoc*$\nu\pi\varrho$.*chine* $\nu\pi\varrho$.*Prj*$_0$
p_0[μ.*leg0*, *HH*$\nu\pi\varrho$.*leg1*] · μ-$\nu\pi$-ϱ.*chine-assoc* ·
chine-hcomp (*assoc*$_{SB}$ μ ν π) ϱ]
by *fastforce*
also have ... = $\nu\pi\varrho$.*Prj*$_0$ · $\langle\mu$-$\nu\pi$-ϱ.*Prj*$_{01}$ $[\![\nu\pi$.*leg0*, ϱ.*leg1*$]\!]$ μ-$\nu\pi$-ϱ.*Prj*$_0\rangle$ ·
chine-hcomp (*assoc*$_{SB}$ μ ν π) ϱ
using *A C.comp-reduce* [*of* p_0[μ.*leg0*, *HH*$\nu\pi\varrho$.*leg1*] μ-$\nu\pi$-ϱ.*chine-assoc*]
by *fastforce*
also have ... = μ-$\nu\pi$-ϱ.*Prj*$_0$ · *chine-hcomp* (*assoc*$_{SB}$ μ ν π) ϱ
by (*metis A C.comp-assoc* μ-$\nu\pi$-ϱ.*prj-chine-assoc*(*3*) $\nu\pi$.*leg0-composite*
$\nu\pi$-ϱ.*leg1-composite*)
also have ... = ϱ.*chine* · p_0[*assoc*$\mu\nu\pi$.*dom.leg0*, ϱ.*leg1*]
proof −
have μ-$\nu\pi$-ϱ.*Prj*$_0$ = p_0[*assoc*$\mu\nu\pi$.*cod.leg0*, ϱ.*cod.leg1*]
using μ-$\nu\pi$.*composable* $\nu\pi$.*composite-is-arrow hcomp-def* **by** *auto*
thus *?thesis*
using *chine-hcomp-props*(*5*) [*of* ϱ *assoc*$_{SB}$ μ ν π]
$\mu\nu$.*composable* $\nu\pi$.*composable* $\pi\varrho$.*composable*
by *simp*
qed
also have ... = p_0[*HH*$\mu\nu\pi$.*leg0*, ϱ.*leg1*]
by (*metis C.comp-cod-arr C.in-homE arrow-of-spans-data.select-convs*(*2*)
prj-in-homs(*4*))
also have ... = $\nu\pi\varrho$.*Prj*$_{00}$ ·
$\langle\mu$-ν-$\pi\varrho$.*Prj*$_{01}$ $[\![\nu$.*leg0*, $\pi\varrho$.*leg1*$]\!]$ μ-ν-$\pi\varrho$.*Prj*$_0\rangle$ · $\mu\nu$-π-ϱ.*chine-assoc*
by (*metis C.comp-assoc C.tuple-prj* $\mu\nu$.*leg0-composite* $\mu\nu$-π.*leg0-composite*
$\mu\nu$-π-ϱ.*prj-chine-assoc*(*3*) μ-ν-$\pi\varrho$.*cospan-*$\nu\pi$ μ-ν-$\pi\varrho$.*prj-chine-assoc*(*2*)

```
            μ-ν-πϱ.prj-chine-assoc(3) μ-ν-πϱ.prj-simps(2) πϱ.leg1-composite)
    finally show ?thesis by blast
  qed
  moreover have «assocνπϱ.chine · p0[μ.leg0, HHνπϱ.leg1] · μ-νπ-ϱ.chine-assoc ·
                chine-hcomp (assocSB μ ν π) ϱ : HHHμνπϱ.chine →C HνHπϱ.chine»
    using νπϱ.chine-assoc-props(1) by fast
  moreover have «⟨μ-ν-πϱ.Prj01 ⟦ν.leg0, πϱ.leg1⟧ μ-ν-πϱ.Prj0⟩ · μν-π-ϱ.chine-assoc :
                HHHμνπϱ.chine →C HνHπϱ.chine»
  proof −
    have «μν-π-ϱ.chine-assoc : HHHμνπϱ.chine →C HHμνHπϱ.chine»
      using μν-π-ϱ.chine-assoc-props(1) by blast
    moreover have «⟨μ-ν-πϱ.Prj01 ⟦ν.leg0, πϱ.leg1⟧ μ-ν-πϱ.Prj0⟩ :
                  HHμνHπϱ.chine →C HνHπϱ.chine»
      using chine-hcomp-ide-ide hcomp-def [of ν hcomp π ϱ] νπ.composable by auto
    ultimately show ?thesis by blast
  qed
  ultimately show ?thesis
    using νπϱ.prj′-joint-monic
          [of assocνπϱ.chine · p0[μ.leg0, HHνπϱ.leg1] · μ-νπ-ϱ.chine-assoc ·
                chine-hcomp (assocSB μ ν π) ϱ
              HHHμνπϱ.chine
              ⟨μ-ν-πϱ.Prj01 ⟦ν.leg0, πϱ.leg1⟧ μ-ν-πϱ.Prj0⟩ · μν-π-ϱ.chine-assoc]
    by simp
qed
```

Now use fact ∗ to finish off the $\mathrm{p}_0[\mu.leg0,\ H\nu H\pi\varrho.leg1]$ case.

```
have p0[μ.leg0, HνHπϱ.leg1] · Chn ?LHS =
      assocνπϱ.chine · p0[μ.leg0, HHνπϱ.leg1] · μ-νπ-ϱ.chine-assoc ·
        chine-hcomp (assocSB μ ν π) ϱ
proof −
  have p0[μ.leg0, HνHπϱ.leg1] · Chn ?LHS =
        p0[μ.leg0, HνHπϱ.leg1] · chine-hcomp μ (assocSB ν π ϱ) · μ-νπ-ϱ.chine-assoc ·
          chine-hcomp (assocSB μ ν π) ϱ
    using L by simp
  also have ... = assocνπϱ.chine · p0[μ.leg0, HHνπϱ.leg1] · μ-νπ-ϱ.chine-assoc ·
                  chine-hcomp (assocSB μ ν π) ϱ
  proof −
    have p0[μ.leg0, HνHπϱ.leg1] · chine-hcomp μ (assocSB ν π ϱ) =
          assocνπϱ.chine · p0[μ.leg0, HHνπϱ.leg1]
    proof −
      have C.commutative-square p0[μ.cod.leg0, assocνπϱ.cod.leg1] assocνπϱ.chine
            (chine-hcomp μ (assocSB ν π ϱ)) p0[μ.leg0, assocνπϱ.dom.leg1]
        by blast
      thus ?thesis
        using νπϱ.chine-assoc-props(1) by auto
    qed
    moreover have C.seq p0[μ.leg0, HνHπϱ.leg1] (chine-hcomp μ (assocSB ν π ϱ))
      using cospan-μ0-HνHπϱ1 prj-in-homs(2) by fastforce
    moreover have C.seq (chine-hcomp μ (assocSB ν π ϱ))
```

$(\mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc \cdot chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho)$
by *blast*
ultimately show *?thesis*
using *chine-hcomp-props*(*4*) *C.comp-permute* **by** *auto*
qed
finally show *?thesis* **by** *blast*
qed
also have ... $= \langle \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ [\![\nu.leg0,\ \pi\varrho.leg1]\!]\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0\rangle \cdot \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
using $*$ **by** *simp*
also have ... $= \mathrm{p}_0[\mu.leg0,\ H\nu H\pi\varrho.leg1] \cdot Chn\ ?RHS$
by (*metis C.comp-assoc C.tuple-prj R* $\mu\text{-}\nu\text{-}\pi\varrho.cospan\text{-}\nu\pi$ $\mu\text{-}\nu\text{-}\pi\varrho.prj\text{-}chine\text{-}assoc(2)$
$\mu\text{-}\nu\text{-}\pi\varrho.prj\text{-}chine\text{-}assoc(3)$ $\mu\text{-}\nu\text{-}\pi\varrho.prj\text{-}simps(2)$ $\nu\text{-}\pi\varrho.leg1\text{-}composite$)
finally show *?thesis* **by** *blast*
qed
moreover have $C.seq\ \mathrm{p}_1[\mu.leg0,\ H\nu H\pi\varrho.leg1]\ (Chn\ ?LHS)$
using *LHS-in-hom Chn-in-hom* **by** *blast*
moreover have $C.seq\ \mathrm{p}_1[\mu.leg0,\ H\nu H\pi\varrho.leg1]\ (Chn\ ?RHS)$
using *RHS-in-hom Chn-in-hom* **by** *blast*
ultimately show $Chn\ ?LHS = Chn\ ?RHS$
using $cospan\text{-}\mu0\text{-}H\nu H\pi\varrho1$ *C.prj-joint-monic* **by** *blast*
qed

end

context *span-bicategory*
begin

lemma *pentagon*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *ide k*
and $src\ f = trg\ g$ **and** $src\ g = trg\ h$ **and** $src\ h = trg\ k$
shows $(f \star \alpha_{SB}\ (g,\ h,\ k)) \cdot \alpha_{SB}\ (f,\ g \star h,\ k) \cdot (\alpha_{SB}\ (f,\ g,\ h) \star k) =$
$\alpha_{SB}\ (f,\ g,\ h \star k) \cdot \alpha_{SB}\ (f \star g,\ h,\ k)$
proof −
interpret *f*: *identity-arrow-of-spans C f*
using *assms ide-char′* **by** *auto*
interpret *g*: *identity-arrow-of-spans C g*
using *assms ide-char′* **by** *auto*
interpret *h*: *identity-arrow-of-spans C h*
using *assms ide-char′* **by** *auto*
interpret *k*: *identity-arrow-of-spans C k*
using *assms ide-char′* **by** *auto*

interpret *fghk*: *four-composable-identity-arrows-of-spans C prj0 prj1 f g h k*
using *assms* **by** (*unfold-locales*, *auto*)

let $?LHS = (f \star assoc_{SB}\ g\ h\ k) \cdot (assoc_{SB}\ f\ (g \star h)\ k) \cdot (assoc_{SB}\ f\ g\ h \star k)$
let $?RHS = assoc_{SB}\ f\ g\ (h \star k) \cdot assoc_{SB}\ (f \star g)\ h\ k$

have $(f \star \alpha_{SB}\ (g,\ h,\ k)) \cdot \alpha_{SB}\ (f,\ g \star h,\ k) \cdot (\alpha_{SB}\ (f,\ g,\ h) \star k) = ?LHS$

using *assms* α*-ide ide-hcomp src-hcomp trg-hcomp* **by** *simp*
also have ... = *?RHS*
using *fghk*.$\mu\nu$.*composable fghk*.$\nu\pi$.*composable fghk*.$\pi\varrho$.*composable fghk.chine-pentagon*
by (*intro arr-eqI*, *auto*)
also have ... = α_{SB} (f, g, $h \star k$) $\cdot$ α_{SB} ($f \star g$, h, k)
using *assms* α*-ide ide-hcomp src-hcomp trg-hcomp* **by** *simp*
finally show *?thesis* **by** *blast*
qed

lemma *extends-to-bicategory*:
shows *bicategory vcomp hcomp assoc unit src trg*
using *unit-in-hom obj-char iso-unit assoc-def pentagon*
apply *unfold-locales* **by** *auto*

sublocale *bicategory vcomp hcomp assoc unit src trg*
using *extends-to-bicategory* **by** *auto*

2.1.6 Miscellaneous Formulas

no-notation *in-hom* (‹«- : - → -»›)
notation *in-hom* (‹«- : - ⇒ -»›)

notation *lunit* (‹l[-]›)
notation *runit* (‹r[-]›)
notation *lunit′* (‹l^{-1}[-]›)
notation *runit′* (‹r^{-1}[-]›)
notation *assoc* (‹a[-, -, -]›)
notation a′ (‹a^{-1}[-, -, -]›)

lemma α'*-ide*:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows α' (f, g, h) = $assoc'_{SB}$ f g h
proof −
have *fgh*: *VVV.ide* (f, g, h)
using *assms* $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *simp*
interpret *f*: *arrow-of-spans C f*
using *assms arr-char* [*of f*] **by** *auto*
interpret *g*: *arrow-of-spans C g*
using *assms arr-char* [*of g*] **by** *auto*
interpret *h*: *arrow-of-spans C h*
using *assms arr-char* [*of h*] **by** *auto*
interpret *fgh*: *three-composable-arrows-of-spans C prj0 prj1 f g h*
using *assms* **by** (*unfold-locales*, *auto*)
interpret *fgh*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g h*
using *assms ide-char* **by** (*unfold-locales*, *auto*)
have α' (f, g, h) = *inv* (α (f, g, h))
using *fgh* α'*.inverts-components*
by (*simp add*: α*-def*)

```
  moreover have inv (α (f, g, h)) = ⦇Chn = C.inv (Chn (α (f, g, h))),
                                     Dom = Cod (α (f, g, h)),
                                     Cod = Dom (α (f, g, h))⦈
    using fgh α.components-are-iso inv-eq
    by (simp add: α-def fgh.μν.composable fgh.νπ.composable)
  moreover have ... = assoc'_SB f g h
    using assms fgh α-ide [of f g h] fgh.chine-assoc-inverse C.inverse-unique
    by (simp add: α-def)
  ultimately show ?thesis by simp
qed
```

The following give explicit expressions for the unitors, derived from their characterizing properties and the definition of the associators.

```
lemma runit-ide-eq:
assumes ide f
shows r[f] = ⦇Chn = p1[Leg0 (Dom f), C.cod (Leg0 (Dom f))],
             Dom = ⦇Leg0 = p0[Leg0 (Dom f), C.cod (Leg0 (Dom f))],
                   Leg1 = Leg1 (Dom f) · p1[Leg0 (Dom f), C.cod (Leg0 (Dom f))]⦈,
             Cod = Cod f⦈
proof −
  interpret f: identity-arrow-of-spans C f
    using assms ide-char' by auto
  interpret src: identity-arrow-of-spans C ‹src f›
    using assms ide-char' ide-src by auto
  interpret f-src: two-composable-identity-arrows-of-spans C prj0 prj1 f ‹src f›
    using assms by (unfold-locales, simp)
  interpret src-src: two-composable-identity-arrows-of-spans C prj0 prj1 ‹src f› ‹src f›
    by (unfold-locales, simp)
  interpret f-src-src: three-composable-identity-arrows-of-spans C prj0 prj1 f ‹src f› ‹src f›
    ..

  let ?rf = ⦇Chn = p1[f.leg0, f.dsrc],
            Dom = ⦇Leg0 = p0[f.leg0, f.dsrc], Leg1 = f.leg1 · p1[f.leg0, f.dsrc]⦈,
            Cod = Cod f⦈
  have ?rf = r[f]
  proof (intro runit-eqI)
    show ide f by fact
    interpret rf: arrow-of-spans C ?rf
    proof −
      interpret dom-rf: span-in-category C
                    ‹⦇Leg0 = p0[f.leg0, f.dsrc], Leg1 = f.leg1 · p1[f.leg0, f.dsrc]⦈›
        by (unfold-locales, simp-all)
      show arrow-of-spans C ?rf
        using dom-rf.apex-def C.comp-cod-arr C.pullback-commutes [of f.leg0 f.dsrc]
        apply unfold-locales by auto
    qed
    show rf-in-hom: «?rf : f ⋆ src f ⇒ f»
    proof
      show arr ?rf
```

```
    using rf.arrow-of-spans-axioms arr-char by simp
  show cod ?rf = f
    using cod-char rf.arrow-of-spans-axioms arr-char by simp
  show dom ?rf = f ⋆ src f
  using dom-char rf.arrow-of-spans-axioms src.arrow-of-spans-axioms arr-char hcomp-def
        f.arrow-of-spans-axioms f-src.composable chine-hcomp-ide-ide src-def ide-char
        C.comp-cod-arr rf.dom.apex-def
    by simp
qed
show ?rf ⋆ src f = (f ⋆ i[src f]) ⋅ a[f, src f, src f]
proof (intro arr-eqI)
  show par: par (?rf ⋆ src f) ((f ⋆ i[src f]) ⋅ a[f, src f, src f])
  proof −
    have «?rf ⋆ src f : (f ⋆ src f) ⋆ src f ⇒ f ⋆ src f»
      using f-src-src.composites-are-arrows(2) rf-in-hom src-src.are-identities(1)
      by blast
    thus ?thesis by auto
  qed
  show Chn (?rf ⋆ src f) = Chn ((f ⋆ i[src f]) ⋅ a[f, src f, src f])
  proof −
    have Chn (?rf ⋆ src f) = ⟨f-src-src.Prj11 ⟦f.leg0, src.leg1⟧ f-src-src.Prj01⟩
    proof −
      have Chn (?rf ⋆ src f) =
            ⟨f-src-src.Prj11 ⟦f.leg0, src.leg1⟧ p0[f-src.prj0, src.leg1]⟩
        using assms src-def trg-def hcomp-def arr-char ide-char
              rf.arrow-of-spans-axioms src.identity-arrow-of-spans-axioms
              chine-hcomp-arr-ide C.comp-cod-arr
        by (simp add: f.arrow-of-spans-axioms identity-arrow-of-spans-def)
      moreover have p0[f-src.prj0, src.leg1] = f-src-src.Prj01
      proof −
        have src f = ⦇Chn = f.dsrc,
                      Dom = ⦇Leg0 = f.dsrc, Leg1 = f.dsrc⦈,
                      Cod = ⦇Leg0 = f.dsrc, Leg1 = f.dsrc⦈⦈
          using assms src-def by simp
        thus ?thesis
          by (simp add: C.comp-cod-arr C.pullback-commutes′)
      qed
      ultimately show ?thesis by auto
    qed
    also have ... = Chn ((f ⋆ i[src f]) ⋅ a[f, src f, src f])
    proof −
      have Chn ((f ⋆ i[src f]) ⋅ a[f, src f, src f]) =
            ⟨f-src-src.Prj1 ⟦f.leg0, src.leg1⟧ f-src-src.Prj10⟩ ⋅ f-src-src.chine-assoc
      proof −
        have Chn ((f ⋆ i[src f]) ⋅ a[f, src f, src f]) =
              Chn (f ⋆ i[src f]) ⋅ Chn a[f, src f, src f]
          using par vcomp-eq [of f ⋆ i[src f] a[f, src f, src f]]
          by simp
        moreover have Chn (f ⋆ i[src f]) =
```

$\langle$*f-src-src.Prj*$_1$ $[\![$*f.leg0*, *src.leg1*$]\!]$ *f-src-src.Prj*$_{10}\rangle$
proof −
have i[*src f*] = (|*Chn* = p$_1$[*f.dsrc*, *f.dsrc*],
Dom = (|*Leg0* = p$_1$[*f.dsrc*, *f.dsrc*], *Leg1* = p$_1$[*f.dsrc*, *f.dsrc*]|),
Cod = (|*Leg0* = *f.dsrc*, *Leg1* = *f.dsrc*|)|)
using *unit-def src-def trg-def hcomp-def src.arrow-of-spans-axioms arr-char f.arrow-of-spans-axioms C.comp-cod-arr*
by *simp*
moreover have *arrow-of-spans C* i[*src f*]
using *assms arr-char* [*of* i[*src f*]] **by** *simp*
ultimately show *?thesis*
using *assms unit-def hcomp-def chine-hcomp-ide-arr rf.arrow-of-spans-axioms src.arrow-of-spans-axioms f.arrow-of-spans-axioms arr-char C.comp-cod-arr src-def trg-def*
by *simp*
qed
ultimately show *?thesis*
using *α-ide* **by** *simp*
qed
also have ... = $\langle$*f-src-src.Prj*$_1$ · *f-src-src.chine-assoc* $[\![$*f.leg0*, *src.leg1*$]\!]$ *f-src-src.Prj*$_{10}$ · *f-src-src.chine-assoc*$\rangle$
using *C.comp-assoc C.comp-tuple-arr C.pullback-commutes′ f-src-src.cospan-μν f-src-src.cospan-νπ*
by *simp*
also have ... = $\langle$*f-src-src.Prj*$_{11}$ $[\![$*f.leg0*, *src.leg1*$]\!]$ *f-src-src.Prj*$_{01}\rangle$
by *simp*
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
qed
qed
thus *?thesis* **by** *simp*
qed

lemma *lunit-ide-eq*:
assumes *ide f*
shows l[*f*] = (|*Chn* = p$_0$[*C.cod* (*Leg1* (*Dom f*)), *Leg1* (*Dom f*)],
Dom = (|*Leg0* = *Leg0* (*Dom f*) · p$_0$[*C.cod* (*Leg1* (*Dom f*)), *Leg1* (*Dom f*)],
Leg1 = p$_1$[*C.cod* (*Leg1* (*Dom f*)), *Leg1* (*Dom f*)]|),
Cod = *Cod f*|)
proof −
interpret *f*: *identity-arrow-of-spans C f*
using *assms ide-char′* **by** *auto*
interpret *trg*: *identity-arrow-of-spans C* ‹*trg f*›
using *assms ide-char′ ide-trg* **by** *auto*
interpret *trg-f*: *two-composable-identity-arrows-of-spans C prj0 prj1* ‹*trg f*› *f*

```
  using assms by (unfold-locales, simp)
interpret trg-trg: two-composable-identity-arrows-of-spans C prj0 prj1 ‹trg f› ‹trg f›
  by (unfold-locales, simp)
interpret trg-trg-f: three-composable-identity-arrows-of-spans C prj0 prj1 ‹trg f› ‹trg f› f
  ..

let ?lf = ⦇Chn = p0[f.dtrg, f.leg1],
          Dom = ⦇Leg0 = f.leg0 · p0[f.dtrg, f.leg1], Leg1 = p1[f.dtrg, f.leg1]⦈,
          Cod = Cod f⦈
have ?lf = l[f]
proof (intro lunit-eqI)
  show ide f by fact
  interpret lf: arrow-of-spans C ?lf
  proof −
    interpret dom-lf: span-in-category C
                      ‹⦇Leg0 = f.leg0 · p0[f.dtrg, f.leg1], Leg1 = p1[f.dtrg, f.leg1]⦈›
      by (unfold-locales, simp-all)
    show arrow-of-spans C ?lf
      using dom-lf.apex-def C.comp-cod-arr C.pullback-commutes [of f.dtrg f.leg1]
      apply unfold-locales by auto
  qed
  show lf-in-hom: «?lf : trg f ⋆ f ⇒ f»
  proof
    show arr ?lf
      using lf.arrow-of-spans-axioms arr-char by simp
    show cod ?lf = f
      using cod-char lf.arrow-of-spans-axioms arr-char by simp
    show dom ?lf = trg f ⋆ f
     using dom-char lf.arrow-of-spans-axioms trg.arrow-of-spans-axioms arr-char hcomp-def
           f.arrow-of-spans-axioms trg-f.composable chine-hcomp-ide-ide trg-def ide-char
           C.comp-cod-arr lf.dom.apex-def
      by simp
  qed
  show trg f ⋆ ?lf = (i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f]
  proof (intro arr-eqI)
    show par: par (trg f ⋆ ?lf) ((i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f])
    proof −
      have «trg f ⋆ ?lf : trg f ⋆ (trg f ⋆ f) ⇒ trg f ⋆ f»
      proof −
        have trg f ⋆ ?lf = L ?lf
          using assms lf-in-hom src-def trg-def arr-char lf.arrow-of-spans-axioms
                f.arrow-of-spans-axioms
          by simp
        moreover have «L ?lf : trg f ⋆ (trg f ⋆ f) ⇒ trg f ⋆ f»
          using lf-in-hom L.preserves-hom [of ?lf trg f ⋆ f f] by simp
        ultimately show ?thesis by auto
      qed
      thus ?thesis by auto
    qed
```

```
show Chn (trg f ⋆ ?lf) = Chn ((i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f])
proof −
  have Chn (trg f ⋆ ?lf) = ⟨trg-trg-f.Prj₁₀ ⟦trg.leg0, f.leg1⟧ trg-trg-f.Prj₀₀⟩
  proof −
    have Chn (trg f ⋆ ?lf) =
          ⟨p₁[trg.leg0, trg-f.prj₁] ⟦trg.leg0, f.leg1⟧ trg-trg-f.Prj₀₀⟩
      using assms src-def trg-def hcomp-def arr-char ide-char
            lf.arrow-of-spans-axioms trg.identity-arrow-of-spans-axioms
            chine-hcomp-ide-arr C.comp-cod-arr
      by (simp add: f.arrow-of-spans-axioms identity-arrow-of-spans-def)
    moreover have p₁[trg.leg0, trg-f.prj₁] = trg-trg-f.Prj₁₀
    proof −
      have trg f = ⦇Chn = f.dtrg,
                    Dom = ⦇Leg0 = f.dtrg, Leg1 = f.dtrg⦈,
                    Cod = ⦇Leg0 = f.dtrg, Leg1 = f.dtrg⦈⦈
        using assms trg-def by simp
      thus ?thesis
        apply (simp add: C.comp-cod-arr C.pullback-commutes′)
        by (metis C.comp-cod-arr C.pullback-commutes′ select-convs(1) select-convs(2)
            select-convs(3) f.cod-simps(3) lf.cod-trg-eq-dom-trg lf.dom.leg-simps(3)
            span-data.select-convs(1) span-data.select-convs(2) trg.chine-eq-apex
            trg-trg-f.cospan-νπ trg-trg-f.prj-simps(10) trg-trg-f.prj-simps(16))
    qed
    ultimately show ?thesis by auto
  qed
  also have ... = Chn ((i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f])
  proof −
    have Chn ((i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f]) =
          ⟨trg-trg-f.Prj₀₁ ⟦trg.leg0, f.leg1⟧ trg-trg-f.Prj₀⟩ · trg-trg-f.chine-assoc′
    proof −
      have Chn ((i[trg f] ⋆ f) · a⁻¹[trg f, trg f, f]) =
            Chn (i[trg f] ⋆ f) · Chn a⁻¹[trg f, trg f, f]
        using par vcomp-eq [of i[trg f] ⋆ f a⁻¹[trg f, trg f, f]]
        by simp
      moreover have Chn (i[trg f] ⋆ f) =
                    ⟨trg-trg-f.Prj₀₁ ⟦trg.leg0, f.leg1⟧ trg-trg-f.Prj₀⟩
      proof −
        have i[trg f] = ⦇Chn = p₁[f.dtrg, f.dtrg],
                         Dom = ⦇Leg0 = p₁[f.dtrg, f.dtrg], Leg1 = p₁[f.dtrg, f.dtrg]⦈,
                         Cod = ⦇Leg0 = f.dtrg, Leg1 = f.dtrg⦈⦈
          using unit-def src-def trg-def hcomp-def trg.arrow-of-spans-axioms arr-char
                f.arrow-of-spans-axioms C.comp-cod-arr
          by simp
        moreover have arrow-of-spans C i[trg f]
          using assms arr-char [of i[trg f]] by simp
        ultimately show ?thesis
          using assms unit-def hcomp-def chine-hcomp-arr-ide
                lf.arrow-of-spans-axioms trg.arrow-of-spans-axioms
                f.arrow-of-spans-axioms arr-char C.comp-cod-arr
```

```
            src-def trg-def
          by simp
        qed
        moreover have Chn a⁻¹[trg f, trg f, f] = trg-trg-f.chine-assoc′
        proof −
          have iso (α (trg f, trg f, f))
            by (simp add: α-def)
          moreover have C.inv trg-trg-f.chine-assoc = trg-trg-f.chine-assoc′
            using trg-trg-f.chine-assoc-inverse C.inv-is-inverse C.inverse-arrow-unique
            by auto
          ultimately show ?thesis
            using assms by (simp add: a′-def α′-ide)
        qed
        ultimately show ?thesis
          by simp
      qed
      also have ... = ⟨trg-trg-f.Prj01 · trg-trg-f.chine-assoc′
                        ⟦trg.leg0, f.leg1⟧
                      trg-trg-f.Prj0 · trg-trg-f.chine-assoc′⟩
      proof −
        have C.commutative-square trg.leg0 f.leg1 trg-trg-f.Prj01 trg-trg-f.Prj0
        proof
          show C.cospan trg.leg0 f.leg1
            using trg-f.legs-form-cospan(1) by auto
          show C.span trg-trg-f.Prj01 trg-trg-f.Prj0
            using trg-trg-f.prj-in-hom by auto
          show C.dom trg.leg0 = C.cod trg-trg-f.Prj01
            by simp
          show trg.leg0 · trg-trg-f.Prj01 = f.leg1 · trg-trg-f.Prj0
            by (metis C.comp-assoc C.prj0-simps-arr C.pullback-commutes′
                ‹C.span trg-trg-f.Prj01 trg-trg-f.Prj0›)
        qed
        moreover have C.seq trg-trg-f.Prj01 trg-trg-f.chine-assoc′
          by blast
        ultimately show ?thesis
          using C.comp-tuple-arr [of trg.leg0 f.leg1 trg-trg-f.Prj01 trg-trg-f.Prj0
                                    trg-trg-f.chine-assoc′]
          by auto
      qed
      also have ... = ⟨trg-trg-f.Prj10 ⟦trg.leg0, f.leg1⟧ trg-trg-f.Prj00⟩
        by simp
      finally show ?thesis by simp
    qed
    finally show ?thesis by blast
  qed
 qed
 qed
 thus ?thesis by simp
qed
```

lemma *runit′-ide-eq*:
assumes *ide f*
shows $\mathrm{r}^{-1}[f] = (\!|\mathit{Chn} = \langle \mathit{Chn}\ f\ [\![\mathit{Leg0}\ (\mathit{Dom}\ f),\ C.\mathit{cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))]\!]\ \mathit{Leg0}\ (\mathit{Dom}\ f)\rangle,$
$\mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = (\!|\mathit{Leg0} = \mathrm{p}_0[\mathit{Leg0}\ (\mathit{Dom}\ f),\ C.\mathit{cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))],$
$\mathit{Leg1} = \mathit{Leg1}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_1[\mathit{Leg0}\ (\mathit{Dom}\ f),\ C.\mathit{cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))]|\!)|\!)$
proof −
  interpret *f*: *identity-arrow-of-spans C f*
    using *assms ide-char′* **by** *auto*
  show $\mathrm{r}^{-1}[f] = (\!|\mathit{Chn} = \langle f.\mathit{chine}\ [\![f.\mathit{leg0},\ f.\mathit{dsrc}]\!]\ f.\mathit{leg0}\rangle,\ \mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = (\!|\mathit{Leg0} = \mathrm{p}_0[f.\mathit{leg0},\ f.\mathit{dsrc}],\ \mathit{Leg1} = f.\mathit{leg1} \cdot \mathrm{p}_1[f.\mathit{leg0},\ f.\mathit{dsrc}]|\!)|\!)$
  proof −
    have $C.\mathit{inverse\text{-}arrows}\ \mathrm{p}_1[f.\mathit{leg0},\ f.\mathit{dsrc}]\ \langle f.\mathit{chine}\ [\![f.\mathit{leg0},\ f.\mathit{dsrc}]\!]\ f.\mathit{leg0}\rangle$
      using *C.pullback-arr-cod*(*1*) *f.chine-eq-apex f.dom.apex-def f.dom.leg-simps*(*1*)
      by *presburger*
    hence $C.\mathit{inv}\ \mathrm{p}_1[f.\mathit{leg0},\ f.\mathit{dsrc}] = \langle f.\mathit{chine}\ [\![f.\mathit{leg0},\ f.\mathit{dsrc}]\!]\ f.\mathit{leg0}\rangle$
      using *C.inv-is-inverse C.inverse-arrow-unique* **by** *auto*
    hence $\mathrm{r}^{-1}[f] = (\!|\mathit{Chn} = \langle f.\mathit{chine}\ [\![f.\mathit{leg0},\ f.\mathit{dsrc}]\!]\ f.\mathit{leg0}\rangle,$
$\mathit{Dom} = \mathit{Cod}\ \mathrm{r}[f],\ \mathit{Cod} = \mathit{Dom}\ \mathrm{r}[f]|\!)$
      using *assms runit-ide-eq inv-eq* [*of* $\mathrm{r}[f]$] *iso-runit* **by** *simp*
    thus *?thesis*
      using *assms runit-ide-eq* **by** *simp*
  qed
qed

lemma *lunit′-ide-eq*:
assumes *ide f*
shows $\mathrm{l}^{-1}[f] = (\!|\mathit{Chn} = \langle \mathit{Leg1}\ (\mathit{Dom}\ f)\ [\![C.\mathit{cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)]\!]\ \mathit{Chn}\ f\rangle,$
$\mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = (\!|\mathit{Leg0} = \mathit{Leg0}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_0[C.\mathit{cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)],$
$\mathit{Leg1} = \mathrm{p}_1[C.\mathit{cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)]|\!)|\!)$
proof −
  interpret *f*: *identity-arrow-of-spans C f*
    using *assms ide-char′* **by** *auto*
  show $\mathrm{l}^{-1}[f] = (\!|\mathit{Chn} = \langle f.\mathit{leg1}\ [\![f.\mathit{dtrg},\ f.\mathit{leg1}]\!]\ f.\mathit{chine}\rangle,\ \mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = (\!|\mathit{Leg0} = f.\mathit{leg0} \cdot \mathrm{p}_0[f.\mathit{dtrg},\ f.\mathit{leg1}],\ \mathit{Leg1} = \mathrm{p}_1[f.\mathit{dtrg},\ f.\mathit{leg1}]|\!)|\!)$
  proof −
    have $C.\mathit{inverse\text{-}arrows}\ \mathrm{p}_0[f.\mathit{dtrg},\ f.\mathit{leg1}]\ \langle f.\mathit{leg1}\ [\![f.\mathit{dtrg},\ f.\mathit{leg1}]\!]\ f.\mathit{chine}\rangle$
      using *C.pullback-arr-cod*(*2*) *f.chine-eq-apex f.dom.apex-def f.dom.is-span*
      by *presburger*
    hence $C.\mathit{inv}\ \mathrm{p}_0[f.\mathit{dtrg},\ f.\mathit{leg1}] = \langle f.\mathit{leg1}\ [\![f.\mathit{dtrg},\ f.\mathit{leg1}]\!]\ f.\mathit{chine}\rangle$
      using *C.inv-is-inverse C.inverse-arrow-unique* **by** *auto*
    hence $\mathrm{l}^{-1}[f] = (\!|\mathit{Chn} = \langle f.\mathit{leg1}\ [\![f.\mathit{dtrg},\ f.\mathit{leg1}]\!]\ f.\mathit{chine}\rangle,$
$\mathit{Dom} = \mathit{Cod}\ \mathrm{l}[f],\ \mathit{Cod} = \mathit{Dom}\ \mathrm{l}[f]|\!)$
      using *assms lunit-ide-eq inv-eq* [*of* $\mathrm{l}[f]$] *iso-lunit* **by** *simp*
    thus *?thesis*
      using *assms lunit-ide-eq* **by** *simp*
  qed

qed

end

locale *adjunction-data-in-span-bicategory* =
  span-bicategory C prj0 prj1 +
  adjunction-data-in-bicategory vcomp hcomp assoc unit src trg f g η ε
for C :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹·› *55*)
and *prj0* :: $'a \Rightarrow 'a \Rightarrow 'a$ (‹p_0[-, -]›)
and *prj1* :: $'a \Rightarrow 'a \Rightarrow 'a$ (‹p_1[-, -]›)
and f :: $'a$ *arrow-of-spans-data*
and g :: $'a$ *arrow-of-spans-data*
and η :: $'a$ *arrow-of-spans-data*
and ε :: $'a$ *arrow-of-spans-data*
begin

  interpretation f: *identity-arrow-of-spans C f*
    using *ide-char′* [*of f*] **by** *auto*
  interpretation g: *identity-arrow-of-spans C g*
    using *ide-char′* [*of g*] **by** *auto*

  interpretation *gf*: *two-composable-identity-arrows-of-spans C prj0 prj1 g f*
    using *antipar* **by** (*unfold-locales*, *auto*)
  interpretation *fg*: *two-composable-identity-arrows-of-spans C prj0 prj1 f g*
    using *antipar* **by** (*unfold-locales*, *auto*)

  interpretation *fgf*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g f* **..**
  interpretation *gfg*: *three-composable-identity-arrows-of-spans C prj0 prj1 g f g* **..**

  interpretation η: *arrow-of-spans C η*
    using *arr-char unit-in-hom* **by** *auto*
  interpretation ε: *arrow-of-spans C ε*
    using *arr-char counit-in-hom* **by** *auto*

  lemma *chine-unit-in-hom*:
  shows «η.*chine* : f.*dsrc* $\rightarrow_C$ g.*leg0* $\downarrow\downarrow$ f.*leg1*»
  proof −
    have «η.*chine* : η.*dom.apex* $\rightarrow_C$ η.*cod.apex*»
      using η.*chine-in-hom* **by** *simp*
    moreover have η.*dom.apex* = f.*dsrc*
      using η.*dom.apex-def dom-char unit-simps src-def* **by** *auto*
    moreover have η.*cod.apex* = g.*leg0* $\downarrow\downarrow$ f.*leg1*
      by (*metis arrow-of-spans-data.select-convs*(*1*) *cod-char gf.chine-composite*
          unit-simps(*1*,*3*))
    ultimately show *?thesis* **by** *simp*
  qed

  lemma *chine-counit-in-hom*:
  shows «ε.*chine* : f.*leg0* $\downarrow\downarrow$ g.*leg1* $\rightarrow_C$ f.*dtrg*»

proof −
  have «ε.*chine* : ε.*dom.apex* $\rightarrow_C$ ε.*cod.apex*»
    using ε.*chine-in-hom* **by** *simp*
  moreover have ε.*cod.apex* = *f.dtrg*
    using ε.*cod.apex-def cod-char counit-simps trg-def gf.composable* **by** *auto*
  moreover have ε.*dom.apex* = *f.leg0* $\downarrow\downarrow$ *g.leg1*
    by (*metis Chn-in-hom* ε.*chine-simps*(*2*) *category.in-homE counit-in-hom*(*2*)
        fg.chine-composite span-vertical-category-axioms span-vertical-category-def)
  ultimately show *?thesis* **by** *simp*
qed

lemma η-*leg-simps*:
shows η.*dom.leg0* = *f.dsrc* **and** η.*dom.leg1* = *f.dsrc*
and η.*cod.leg0* = *gf.leg0* **and** η.*cod.leg1* = *gf.leg1*
proof −
  show η.*dom.leg0* = *f.dsrc*
    using *dom-char unit-simps*(*2*) *src-def* **by** *auto*
  show η.*dom.leg1* = *f.dsrc*
    using *dom-char unit-simps*(*2*) *src-def* **by** *auto*
  show η.*cod.leg0* = *gf.leg0*
    using *cod-char unit-simps*(*1*,*3*)
    by (*metis* (*no-types*, *lifting*) *arrow-of-spans-data.select-convs*(*2*))
  show η.*cod.leg1* = *gf.leg1*
    using *cod-char unit-simps*(*1*,*3*)
    by (*metis* (*no-types*, *lifting*) *arrow-of-spans-data.select-convs*(*2*))
qed

lemma ε-*leg-simps*:
shows ε.*cod.leg0* = *f.dtrg* **and** ε.*cod.leg1* = *f.dtrg*
and ε.*dom.leg0* = *fg.leg0* **and** ε.*dom.leg1* = *fg.leg1*
proof −
  show ε.*cod.leg0* = *f.dtrg*
    using *cod-char counit-simps*(*3*) *trg-def gf.composable* **by** *auto*
  show ε.*cod.leg1* = *f.dtrg*
    using *cod-char counit-simps*(*3*) *trg-def gf.composable* **by** *auto*
  show ε.*dom.leg0* = *fg.leg0*
    using *dom-char counit-simps hcomp-def fg.composable* **by** *simp*
  show ε.*dom.leg1* = *fg.leg1*
    using *dom-char counit-simps hcomp-def fg.composable* **by** *simp*
qed

lemma *Chn-triangle-eq*:
shows *Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f]) = gf.prj_0 \cdot \eta.chine \cdot f.leg0$
and *Chn* $(\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g]) = gf.prj_1 \cdot \eta.chine \cdot g.leg1$
proof −
  have *1*: *Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f]) =$
      $\mathrm{p}_0[f.dtrg, f.leg1] \cdot$ *chine-hcomp* ε *f* $\cdot$ *fgf.chine-assoc'* $\cdot$ *chine-hcomp f* η $\cdot$
        $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
  proof −

have *Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f]) =$
Chn $\mathrm{l}[f]$ · *Chn* $(\varepsilon \star f)$ · *Chn* $\mathrm{a}^{-1}[f, g, f]$ · *Chn* $(f \star \eta)$ · *Chn* $\mathrm{r}^{-1}[f]$
using *antipar Chn-vcomp* **by** *auto*
also have ... $= \mathrm{p}_0[f.dtrg, f.leg1]$ · *chine-hcomp* ε f · *fgf.chine-assoc′* ·
chine-hcomp f η · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
using *α-ide fg.composable gf.composable fgf.chine-assoc-inverse*
C.inverse-unique inv-eq iso-assoc lunit-ide-eq hcomp-def [*of* ε f]
gf.composable hcomp-def [*of* f η] *fg.composable runit′-ide-eq*
by *simp*
finally show *?thesis* **by** *blast*
qed
moreover have *C.arr* (*Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f])$)
by (*meson arrI arrow-of-spans.chine-simps*(*1*) *arr-char triangle-in-hom*(*3*))
ultimately have *2*: *C.arr* ($\mathrm{p}_0[f.dtrg, f.leg1]$ · *chine-hcomp* ε f · *fgf.chine-assoc′* ·
chine-hcomp f η · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$)
by *simp*

have *Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f]) =$
$\mathrm{p}_0[f.dtrg, f.leg1]$ · *chine-hcomp* ε f · *fgf.chine-assoc′* · *chine-hcomp* f η ·
$\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
using *1* **by** *simp*
also have
3: ... $=$ *fgf.Prj*$_0$ · *fgf.chine-assoc′* · *chine-hcomp* f η · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
by (*metis* (*no-types*, *lifting*) *C.comp-assoc C.comp-cod-arr ε-leg-simps*(*1*,*3*)
chine-hcomp-props(*5*) *counit-simps*(*1*,*4*) *f.chine-eq-apex f.cod-simps*(*3*)
fg.composite-is-arrow fg.leg0-composite fgf.prj-simps(*3*,*9*) *hseqE*)
also have *4*: ... $=$ *fgf.Prj*$_{00}$ · *chine-hcomp* f η · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
using *C.comp-reduce* [*of fgf.Prj*$_0$ *fgf.chine-assoc′ fgf.Prj*$_{00}$
chine-hcomp f η · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$]
2 fgf.prj-chine-assoc′(*3*)
by *blast*
also have ... $=$ (*gf.prj*$_0$ · η.*chine* · $\mathrm{p}_0[f.leg0, f.dsrc]$) · $\langle f.chine\ [\![f.leg0, f.dsrc]\!]\ f.leg0\rangle$
proof −
have *fgf.Prj*$_{00}$ · *chine-hcomp* f η $=$ *gf.prj*$_0$ · η.*chine* · $\mathrm{p}_0[f.leg0, f.dsrc]$
proof −
have *fgf.Prj*$_{00}$ · *chine-hcomp* f η $=$
(*gf.prj*$_0$ · $\mathrm{p}_0[f.leg0, gf.leg1]$) ·
$\langle \mathrm{p}_1[f.leg0, f.dsrc]\ [\![f.leg0, gf.leg1]\!]\ \eta.chine \cdot \mathrm{p}_0[f.leg0, f.dsrc]\rangle$
using *hcomp-def fg.composable gf.composable chine-hcomp-ide-arr η-leg-simps* **by** *auto*
also have ... $=$ *gf.prj*$_0$ · $\mathrm{p}_0[f.leg0, gf.leg1]$ ·
$\langle \mathrm{p}_1[f.leg0, f.dsrc]\ [\![f.leg0, gf.leg1]\!]\ \eta.chine \cdot \mathrm{p}_0[f.leg0, f.dsrc]\rangle$
using *C.comp-assoc* **by** *simp*
also have ... $=$ *gf.prj*$_0$ · η.*chine* · $\mathrm{p}_0[f.leg0, f.dsrc]$
proof −
have *C.commutative-square f.leg0 gf.leg1* $\mathrm{p}_1[f.leg0, f.dsrc]$ (η.*chine* · $\mathrm{p}_0[f.leg0, f.dsrc]$)
proof
show *C.cospan f.leg0 gf.leg1*
using *hcomp-def gf.composable fgf.prj-in-hom*(*5*) **by** *auto*
show *C.span* $\mathrm{p}_1[f.leg0, f.dsrc]$ (η.*chine* · $\mathrm{p}_0[f.leg0, f.dsrc]$)

```
        using chine-unit-in-hom by auto
      show C.dom f.leg0 = C.cod p₁[f.leg0, f.dsrc] by simp
      show f.leg0 · p₁[f.leg0, f.dsrc] = gf.leg1 · η.chine · p₀[f.leg0, f.dsrc]
        by (metis C.comp-assoc C.pullback-commutes' η.cod-trg-eq-dom-trg
            η.dom.leg-simps(1) η.leg1-commutes η-leg-simps(1−2,4)
            ‹C.cospan f.leg0 gf.leg1›)
    qed
    thus ?thesis by simp
  qed
  finally show ?thesis by simp
qed
moreover have C.seq fgf.Prj₀₀ (chine-hcomp f η)
  using chine-hcomp-props(1) by (metis 2 3 4 C.match-2 C.seqE)
moreover have C.seq (chine-hcomp f η) ⟨f.chine ⟦f.leg0, f.dsrc⟧ f.leg0⟩
  using chine-hcomp-props(1) by (metis 2 C.seqE)
ultimately show ?thesis
  using C.comp-reduce by simp
qed
also have ... = gf.prj₀ · η.chine · p₀[f.leg0, f.dsrc] · ⟨f.chine ⟦f.leg0, f.dsrc⟧ f.leg0⟩
  using C.comp-assoc by simp
also have ... = gf.prj₀ · η.chine · f.leg0
  using C.comp-cod-arr f.leg0-commutes by simp
finally show Chn (l[f] · (ε ⋆ f) · a⁻¹[f, g, f] · (f ⋆ η) · r⁻¹[f]) = gf.prj₀ · η.chine · f.leg0
  by simp

have 1: Chn (r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g]) =
        p₁[g.leg0, g.dsrc] · chine-hcomp g ε · gfg.chine-assoc · chine-hcomp η g ·
          ⟨g.leg1 ⟦g.dtrg, g.leg1⟧ g.chine⟩
proof −
  have Chn (r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g]) =
        Chn r[g] · Chn (g ⋆ ε) · Chn a[g, f, g] · Chn (η ⋆ g) · Chn l⁻¹[g]
    using antipar Chn-vcomp by auto
  also have ... = p₁[g.leg0, g.dsrc] · chine-hcomp g ε · gfg.chine-assoc · chine-hcomp η g ·
                  ⟨g.leg1 ⟦g.dtrg, g.leg1⟧ g.chine⟩
    using α-ide gf.composable fg.composable runit-ide-eq hcomp-def [of g ε]
          fg.composable hcomp-def [of η g] gf.composable lunit'-ide-eq
    by simp
  finally show ?thesis by blast
qed
hence 2: C.arr (p₁[g.leg0, g.dsrc] · chine-hcomp g ε · gfg.chine-assoc ·
                chine-hcomp η g · ⟨g.leg1 ⟦g.dtrg, g.leg1⟧ g.chine⟩)
  by (metis arrI arr-char arrow-of-spans.chine-simps(1) triangle-in-hom(4))

have Chn (r[g] · (g ⋆ ε) · a[g, f, g] · (η ⋆ g) · l⁻¹[g]) =
      p₁[g.leg0, g.dsrc] · chine-hcomp g ε · gfg.chine-assoc · chine-hcomp η g ·
        ⟨g.leg1 ⟦g.dtrg, g.leg1⟧ g.chine⟩
  using 1 by simp
also have
  ... = gfg.Prj₁ · gfg.chine-assoc · chine-hcomp η g · ⟨g.leg1 ⟦g.dtrg, g.leg1⟧ g.chine⟩
```

proof −
have $\mathrm{p}_1[g.leg0,\ g.dsrc] \cdot chine\text{-}hcomp\ g\ \varepsilon = \mathrm{p}_1[g.leg0,\ \varepsilon.dom.leg1]$
by (*metis C.comp-cod-arr ε-leg-simps(2) ε-leg-simps(4) arrI chine-hcomp-props(6)*
fg.leg1-composite g.cod-simps(2) g.identity-arrow-of-spans-axioms
gfg.cospan-μν gfg.prj-simps(10) gfg.prj-simps(16) hseqE
identity-arrow-of-spans.chine-eq-apex seqE triangle-in-hom(4))
also have ... $= gfg.Prj_1$
using *dom-char counit-simps hcomp-def fg.composable* **by** *simp*
finally have $\mathrm{p}_1[g.leg0,\ g.dsrc] \cdot chine\text{-}hcomp\ g\ \varepsilon = gfg.Prj_1$
by *simp*
moreover have $C.seq\ \mathrm{p}_1[g.leg0,\ g.dsrc]\ (chine\text{-}hcomp\ g\ \varepsilon)$
using *chine-hcomp-props(1) [of g ε] gf.composable calculation gfg.prj-in-hom(4)*
by *auto*
moreover have $C.seq\ (chine\text{-}hcomp\ g\ \varepsilon)$
$(gfg.chine\text{-}assoc \cdot chine\text{-}hcomp\ \eta\ g \cdot \langle g.leg1\ [\![g.dtrg,\ g.leg1]\!]\ g.chine\rangle)$
using *chine-hcomp-props(1) gf.composable 2* **by** (*metis C.seqE*)
ultimately show *?thesis*
using *C.comp-reduce* **by** *simp*
qed
also have ... $= gfg.Prj_{11} \cdot chine\text{-}hcomp\ \eta\ g \cdot \langle g.leg1\ [\![g.dtrg,\ g.leg1]\!]\ g.chine\rangle$
using *C.comp-reduce* [*of* $gfg.Prj_1$ *gfg.chine-assoc* $gfg.Prj_{11}$
$chine\text{-}hcomp\ \eta\ g \cdot \langle g.leg1\ [\![g.dtrg,\ g.leg1]\!]\ g.chine\rangle$]
2 gfg.prj-chine-assoc(1)
by *blast*
also have ... $= (gf.prj_1 \cdot \eta.chine \cdot \mathrm{p}_1[g.dtrg,\ g.leg1]) \cdot \langle g.leg1\ [\![g.dtrg,\ g.leg1]\!]\ g.chine\rangle$
by (*metis C.comp-assoc η-leg-simps(1) η-leg-simps(3) arrI chine-hcomp-props(6)*
g.cod-simps(3) gf.leg0-composite gfg.cospan-νπ hseqE seqE triangle-in-hom(4))
also have ... $= gf.prj_1 \cdot \eta.chine \cdot \mathrm{p}_1[g.dtrg,\ g.leg1] \cdot \langle g.leg1\ [\![g.dtrg,\ g.leg1]\!]\ g.chine\rangle$
using *C.comp-assoc* **by** *simp*
also have ... $= gf.prj_1 \cdot \eta.chine \cdot g.leg1$
using *C.comp-cod-arr g.leg1-commutes* **by** *simp*
finally show $Chn\ (\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g]) = gf.prj_1 \cdot \eta.chine \cdot g.leg1$
by *simp*
qed

end

2.1.7 Maps in Span(C)

In this section, we chararacterize the maps (*i.e* the left adjoints) in a span bicategory. This is Proposition 2 of [4].

context *span-bicategory*
begin

abbreviation *adjoint-of-map*
where *adjoint-of-map f* ≡ ⦇*Chn* = *Chn f*,
Dom = ⦇*Leg0* = *Leg1* (*Dom f*), *Leg1* = *Leg0* (*Dom f*)⦈,
Cod = ⦇*Leg0* = *Leg1* (*Dom f*), *Leg1* = *Leg0* (*Dom f*)⦈⦈

abbreviation *unit-for-map*
where *unit-for-map f* ≡ ⦇*Chn* = ⟨*C.inv* (*Leg0* (*Dom f*))
⟦*Leg1* (*Dom f*), *Leg1* (*Dom f*)⟧
C.inv (*Leg0* (*Dom f*))⟩,
Dom = *Dom* (*src f*),
Cod = *Dom* (*hcomp* (*adjoint-of-map f*) *f*)⦈

abbreviation *counit-for-map*
where *counit-for-map f* ≡ ⦇*Chn* = *Leg1* (*Dom f*) · p_0[*Leg0* (*Dom f*), *Leg0* (*Dom f*)],
Dom = *Dom* (*hcomp f* (*adjoint-of-map f*)),
Cod = *Dom* (*trg f*)⦈

lemma *is-left-adjoint-char*:
shows *is-left-adjoint f* ⟷ *ide f* ∧ *C.iso* (*Leg0* (*Dom f*))
and *is-left-adjoint f* ⟹
adjunction-in-bicategory vcomp hcomp assoc unit src trg f
(*adjoint-of-map f*) (*unit-for-map f*) (*counit-for-map f*)
proof
show *1*: *is-left-adjoint f* ⟹ *ide f* ∧ *C.iso* (*Leg0* (*Dom f*))
proof
assume *f*: *is-left-adjoint f*
obtain *g η ε* **where** *adj*: *adjunction-in-bicategory vcomp hcomp assoc unit src trg f g η ε*
using *f adjoint-pair-def* **by** *blast*
interpret *adjunction-in-bicategory vcomp hcomp assoc unit src trg f g η ε*
using *adj* **by** *auto*
show *ide f* **by** *simp*

interpret *f*: *identity-arrow-of-spans C f*
using *ide-char′* [*of f*] **by** *auto*
interpret *g*: *identity-arrow-of-spans C g*
using *ide-char′* [*of g*] **by** *auto*

interpret *gf*: *two-composable-identity-arrows-of-spans C prj0 prj1 g f*
using *antipar* **by** (*unfold-locales*, *auto*)
interpret *fg*: *two-composable-identity-arrows-of-spans C prj0 prj1 f g*
using *antipar* **by** (*unfold-locales*, *auto*)

interpret *fgf*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g f* **..**

interpret *src-f*: *arrow-of-spans C* ‹*src f*›
using *arr-char gf.are-arrows*(*2*) **by** *blast*
interpret *src-f*: *identity-arrow-of-spans C* ‹*src f*›
using *ide-char ide-src src-def* **by** (*unfold-locales*, *simp*)

interpret *η*: *arrow-of-spans C η*
using *arr-char unit-in-hom* **by** *auto*
interpret *ε*: *arrow-of-spans C ε*
using *arr-char counit-in-hom* **by** *auto*

```
    interpret adjunction-data-in-span-bicategory C prj0 prj1 f g η ε
      ..
    show C.iso f.leg0
    proof −
      have C.section f.leg0
      proof −
        have f.chine = (gf.prj₀ · η.chine) · f.leg0
          using triangle-left′ Chn-triangle-eq(1) C.comp-assoc by simp
        thus ?thesis
          using f.chine-is-identity by auto
      qed
      moreover have C.retraction f.leg0
        using C.retractionI [of f.leg0 gf.prj₀ · η.chine] hcomp-def C.comp-assoc
              η.leg0-commutes gf.leg0-composite η-leg-simps
        by auto
      ultimately show ?thesis
        by (simp add: C.iso-iff-section-and-retraction)
    qed
  qed
  have 2: ide f ∧ C.iso (Leg0 (Dom f)) ⟹
            adjunction-in-bicategory vcomp hcomp assoc unit src trg f
              (adjoint-of-map f) (unit-for-map f) (counit-for-map f)
```

The right adjoint g is obtained by exchanging the legs of f. The unit is obtained by tupling *C.inv f.leg0* with itself, via the pullback of *f.leg1* with itself. The counit is given by the legs of $f \star g$, which are equal, because the two legs of a pullback of the isomorphism *f.leg0* with itself must be equal. It then remains to verify the triangle identities.

```
  proof −
    assume f: ide f ∧ C.iso (Leg0 (Dom f))
    interpret f: identity-arrow-of-spans C f
      using f ide-char′ by auto
    interpret Dom-src: span-in-category C ‹⦇Leg0 = f.dsrc, Leg1 = f.dsrc⦈›
      using f by (unfold-locales, auto)
    interpret Dom-trg: span-in-category C ‹⦇Leg0 = f.dtrg, Leg1 = f.dtrg⦈›
      using f by (unfold-locales, auto)

    define g where g = adjoint-of-map f
    interpret Dom-g: span-in-category C ‹⦇Leg0 = f.leg1, Leg1 = f.leg0⦈›
      by (unfold-locales, simp)
    interpret g: arrow-of-spans C g
      unfolding g-def
      using Dom-g.apex-def f.leg0-commutes f.leg1-commutes
      by (unfold-locales, auto)
    interpret g: identity-arrow-of-spans C g
      using g-def
      by (unfold-locales, auto)
    have ide-g: ide g
      using f ide-char g.arrow-of-spans-axioms by simp
```

interpret *fg*: *two-composable-arrows-of-spans C prj0 prj1 f g*
  apply *unfold-locales*
  using *g-def src-def trg-def arr-char f.arrow-of-spans-axioms g.arrow-of-spans-axioms*
  by *auto*
interpret *fg*: *two-composable-identity-arrows-of-spans C prj0 prj1 f g*
  ..
interpret *gf*: *two-composable-arrows-of-spans C prj0 prj1 g f*
  apply *unfold-locales*
  using *g-def src-def trg-def arr-char f.arrow-of-spans-axioms g.arrow-of-spans-axioms*
  by *auto*
interpret *gf*: *two-composable-identity-arrows-of-spans C prj0 prj1 g f*
  ..
have *hcomp-fg-eq*: *hcomp f g* = ⦇*Chn* = *f.leg0* ↓↓ *f.leg0*,
  Dom = ⦇*Leg0* = *f.leg1* · p_1[*f.leg0*, *f.leg0*],
    Leg1 = *f.leg1* · p_1[*f.leg0*, *f.leg0*]⦈,
  Cod = ⦇*Leg0* = *f.leg1* · p_1[*f.leg0*, *f.leg0*],
    Leg1 = *f.leg1* · p_1[*f.leg0*, *f.leg0*]⦈⦈
  using *f g-def hcomp-def fg.composable src-def trg-def arr-char f.arrow-of-spans-axioms g.arrow-of-spans-axioms chine-hcomp-def gf.are-identities*(*1*) *chine-hcomp-ide-ide C.pullback-iso-self*
  by *auto*
have *hcomp-gf-eq*: *hcomp g f* = ⦇*Chn* = *f.leg1* ↓↓ *f.leg1*,
  Dom = ⦇*Leg0* = *f.leg0* · p_0[*f.leg1*, *f.leg1*],
    Leg1 = *f.leg0* · p_1[*f.leg1*, *f.leg1*]⦈,
  Cod = ⦇*Leg0* = *f.leg0* · p_0[*f.leg1*, *f.leg1*],
    Leg1 = *f.leg0* · p_1[*f.leg1*, *f.leg1*]⦈⦈
  using *g-def hcomp-def gf.composable src-def trg-def chine-hcomp-ide-ide arr-char f.arrow-of-spans-axioms g.arrow-of-spans-axioms ide-char*
  by *simp*

define *η* **where** *η* = *unit-for-map f*
interpret *Dom-gf*: *span-in-category C* ‹⦇*Leg0* = *f.leg0* · p_0[*f.leg1*, *f.leg1*],
  Leg1 = *f.leg0* · p_1[*f.leg1*, *f.leg1*]⦈›
  by (*unfold-locales*, *simp-all*)
interpret *η*: *arrow-of-spans C η*
  using *f g-def η-def hcomp-def src-def trg-def f.arrow-of-spans-axioms g.arrow-of-spans-axioms arr-char C.comp-arr-inv′ C.tuple-in-hom* [*of f.leg1 f.leg1 C.inv f.leg0 C.inv f.leg0*] *Dom-src.apex-def Dom-gf.apex-def*
  apply *unfold-locales* **by** (*simp-all add*: *C.comp-assoc*)
have *unit-in-hom*: «*η* : *src f* ⇒ *hcomp g f*»
proof
  show *1*: *arr η*
    using *arr-char η.arrow-of-spans-axioms* **by** *simp*
  show *dom η* = *src f*
    using *1 η-def dom-char src-def Dom-src.apex-def* **by** *simp*
  show *cod η* = *hcomp g f*
    using *1 η-def g-def cod-char hcomp-gf-eq Dom-gf.apex-def* **by** *simp*
qed

define ε **where** ε = *counit-for-map f*
interpret *Dom-fg*: *span-in-category C* ‹⦇*Leg0* = *f.leg1* · p_1[*f.leg0*, *f.leg0*],
Leg1 = *f.leg1* · p_1[*f.leg0*, *f.leg0*]⦈›
by (*unfold-locales*, *simp-all*)
interpret ε: *arrow-of-spans C* ε
using *f g-def* ε*-def hcomp-def src-def trg-def f.arrow-of-spans-axioms*
g.arrow-of-spans-axioms arr-char C.comp-cod-arr C.pullback-iso-self
Dom-trg.apex-def Dom-fg.apex-def
apply *unfold-locales* **by** *auto*
have *counit-in-hom*: «ε : *hcomp f g* ⇒ *trg f*»
proof
show *1*: *arr* ε
using *arr-char* ε*.arrow-of-spans-axioms* **by** *simp*
show *cod* ε = *trg f*
using *1* ε*-def cod-char trg-def Dom-trg.apex-def* **by** *simp*
show *dom* ε = *hcomp f g*
using *1 g-def* ε*-def dom-char hcomp-fg-eq Dom-fg.apex-def* **by** *simp*
qed
interpret *adj*: *adjunction-data-in-bicategory vcomp hcomp assoc unit src trg f g* η ε
using *f ide-g unit-in-hom counit-in-hom gf.composable*
by (*unfold-locales*, *simp-all*)
interpret *adjunction-data-in-span-bicategory C prj0 prj1 f g* η ε
..
have *triangle-left*: $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
proof −
have $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
proof (*intro arr-eqI*)
show *par* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f])$ *f*
using *f ide-in-hom* [*of f*] *adj.triangle-in-hom*(*3*)
by (*metis* (*no-types*, *lifting*) *in-homE*)
show *Chn* $(\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, g, f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f])$ = *f.chine*
using *f g-def* η*-def Chn-triangle-eq*(*1*) *C.comp-tuple-arr C.comp-inv-arr′* **by** *simp*
qed
thus *?thesis*
using *adj.triangle-equiv-form* **by** *simp*
qed
have *triangle-right*: $(g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
proof −
have $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] = g$
proof (*intro arr-eqI*)
show *par* $(\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g])$ *g*
using *adj.ide-right ide-in-hom* [*of g*] *adj.triangle-in-hom*(*4*)
by (*metis* (*no-types*, *lifting*) *in-homE*)
show *Chn* $(\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g])$ = *g.chine*
using *f g-def* η*-def Chn-triangle-eq*(*2*) *C.comp-tuple-arr C.comp-inv-arr′* **by** *simp*
qed
thus *?thesis*
using *adj.triangle-equiv-form* **by** *simp*

```
      qed
      interpret adj: adjunction-in-bicategory vcomp hcomp assoc unit src trg f g η ε
        using triangle-left triangle-right by (unfold-locales, simp-all)
      show adjunction-in-bicategory vcomp hcomp assoc unit src trg f g η ε ..
    qed
    show ide f ∧ C.iso (Leg0 (Dom f)) ⟹ is-left-adjoint f
      using 2 adjoint-pair-def by blast
    show is-left-adjoint f ⟹ adjunction-in-bicategory vcomp hcomp assoc unit src trg f
           (adjoint-of-map f) (unit-for-map f) (counit-for-map f)
      using 1 2 by blast
  qed

 end

end
```

2.2 Tabulations

theory *Tabulation*
imports *CanonicalIsos InternalAdjunction*
begin

A "tabulation" is a kind of bicategorical limit that associates with a 1-cell r a triple (f, ϱ, g), where f and g are 1-cells having a common source, and ϱ is a 2-cell from g to $r \cdot f$, such that a certain biuniversal property is satisfied. The notion was introduced in a study of bicategories of spans and relations by Carboni, Kasangian, and Street [4] (hereinafter, "CKS"), who named it after a related, but different notion previously used by Freyd in his study of the algebra of relations. One can find motivation for the concept of tabulation by considering the problem of trying to find some kind of universal way of factoring a 1-cell r, up to isomorphism, as the composition $g \cdot f^*$ of a map g and the right adjoint f^* of a map f. In order to be able to express this as a bicategorical limit, CKS consider, instead of an isomorphism «$\varphi : g \star f^* \Rightarrow r$», its transpose $\varrho : g \Rightarrow r \star f$ under the adjunction $f \dashv f^*$.

2.2.1 Definition of Tabulation

The following locale sets forth the "signature" of the data involved in a tabulation, and establishes some basic facts.

$$\begin{tikzcd}
 & \mathrm{src}\ g = \mathrm{src}\ f \arrow[dl, "g"'] \arrow[dr, "f"] \arrow[d, phantom, "\overset{\varrho}{\Rightarrow}"] & \\
\mathrm{trg}\ r & & \mathrm{src}\ r \arrow[ll, "r"]
\end{tikzcd}$$

locale *tabulation-data* =
  bicategory +
fixes r :: $'a$
  and ϱ :: $'a$
  and f :: $'a$
  and g :: $'a$
assumes *ide-base*: *ide r*
and *ide-leg0*: *ide f*
and *tab-in-vhom'*: «ϱ : $g \Rightarrow r \star f$»
begin

  lemma *base-in-hom* [*intro*]:
  shows «r : *src r* $\rightarrow$ *trg r*» **and** «r : $r \Rightarrow r$»
    using *ide-base* **by** *auto*

  lemma *base-simps* [*simp*]:
  shows *ide r* **and** *arr r*
  and *dom r* = *r* **and** *cod r* = *r*
    using *ide-base* **by** *auto*

  lemma *tab-in-hom* [*intro*]:
  shows «ϱ : *src f* $\rightarrow$ *trg r*» **and** «ϱ : $g \Rightarrow r \star f$»
    using *tab-in-vhom' src-dom* [*of* ϱ] *trg-dom* [*of* ϱ] *base-in-hom* **apply** *auto*
    by (*metis arrI hcomp-simps*(*1*) *hcomp-simps*(*2*) *in-hhomI not-arr-null*
      src.extensionality src.preserves-hom vconn-implies-hpar(*1*)
      vconn-implies-hpar(*2*) *vconn-implies-hpar*(*3*) *vconn-implies-hpar*(*4*))

  lemma *ide-leg1*:
  shows *ide g*
    using *tab-in-hom* **by** *auto*

  lemma *leg1-in-hom* [*intro*]:
  shows «g : *src f* $\rightarrow$ *trg r*» **and** «g : $g \Rightarrow g$»
    using *ide-leg1* **apply** *auto*
    using *tab-in-hom ide-dom* [*of* ϱ]
    apply (*elim conjE in-homE*) **by** *auto*

  lemma *leg1-simps* [*simp*]:
  shows *ide g* **and** *arr g*
  and *src g* = *src f* **and** *trg g* = *trg r*
  and *dom g* = *g***and** *cod g* = *g*
    using *ide-leg1 leg1-in-hom* **by** *auto*

  lemma *tab-simps* [*simp*]:
  shows *arr* ϱ **and** *src* ϱ = *src f* **and** *trg* ϱ = *trg r*
  and *dom* ϱ = *g* **and** *cod* ϱ = $r \star f$
    using *tab-in-hom* **by** *auto*

  lemma *leg0-in-hom* [*intro*]:

shows «f : *src* f → *src* r» **and** «f : f ⇒ f»
  using *ide-leg0* **apply** *auto*
  using *tab-in-hom ide-cod* [*of* ϱ] *hseq-char* [*of* r f]
  apply (*elim conjE in-homE*) **by** *auto*

lemma *leg0-simps* [*simp*]:
shows *ide* f **and** *arr* f
and *trg* f = *src* r
and *dom* f = f **and** *cod* f = f
  using *ide-leg0 leg0-in-hom* **by** *auto*

The following function, which composes ϱ with a 2-cell «ϑ : $f \star w \Rightarrow u$» to obtain a 2-cell «$(r \star \vartheta) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) : g \star w \Rightarrow r \star u$», occurs frequently in the sequel.

abbreviation (*input*) *composite-cell*
where *composite-cell* w $\vartheta \equiv (r \star \vartheta) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w)$

lemma *composite-cell-in-hom*:
assumes *ide* w **and** «w : *src* u → *src* f» **and** «ϑ : $f \star w \Rightarrow u$»
shows «*composite-cell* w ϑ : $g \star w \Rightarrow r \star u$»
proof (*intro comp-in-homI*)
  show «$\varrho \star w : g \star w \Rightarrow (r \star f) \star w$»
    using *assms tab-in-hom*
    apply (*elim conjE in-hhomE in-homE*)
    by (*intro hcomp-in-vhom*, *auto*)
  show «$\mathrm{a}[r, f, w] : (r \star f) \star w \Rightarrow r \star f \star w$»
    using *assms ide-base ide-leg0 tab-in-hom* **by** *fastforce*
  show «$r \star \vartheta : r \star f \star w \Rightarrow r \star u$»
    using *assms ide-base ide-leg0 tab-in-hom* **by** *fastforce*
qed

We define some abbreviations for various combinations of conditions that occur in the hypotheses and conclusions of the tabulation axioms.

abbreviation (*input*) $uw\vartheta\omega$
where $uw\vartheta\omega$ u w ϑ $\omega \equiv$ *ide* $w \wedge$ «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «ω : *dom* $\omega \Rightarrow r \star u$»

abbreviation (*input*) $uw\vartheta\omega\nu$
where $uw\vartheta\omega\nu$ u w ϑ ω $\nu \equiv$
  ide $w \wedge$ «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «ν : *dom* $\omega \Rightarrow g \star w$» $\wedge$ *iso* $\nu \wedge$
  $(r \star \vartheta) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu = \omega$

abbreviation (*input*) $uw\vartheta w'\vartheta'\beta$
where $uw\vartheta w'\vartheta'\beta$ u w ϑ w' ϑ' $\beta \equiv$
  ide $u \wedge$ *ide* $w \wedge$ *ide* $w' \wedge$
  «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «$\vartheta' : f \star w' \Rightarrow u$» $\wedge$ «$\beta : g \star w \Rightarrow g \star w'$» $\wedge$
  $(r \star \vartheta) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) = (r \star \vartheta') \cdot \mathrm{a}[r, f, w'] \cdot (\varrho \star w') \cdot \beta$

end

CKS define two notions of tabulation. The first, which they call simply "tabulation", is restricted to triples (f, ϱ, g) where the "input leg" f is a map, and assumes only a

weak form of the biuniversal property that only applies to $(u,\ \omega,\ v)$ for which u is a map. The second notion, which they call "wide tabulation", concerns arbitrary $(f,\ \varrho,\ g)$, and assumes a strong form of the biuniversal property that applies to all $(u,\ \omega,\ v)$. On its face, neither notion implies the other: "tabulation" has the stronger assumption that f is a map, but requires a weaker biuniversal property, and "wide tabulation" omits the assumption on f, but requires a stronger biuniversal property. CKS Proposition 1(c) states that if $(f,\ \varrho,\ g)$ is a wide tabulation, then f is automatically a map. This is in fact true, but it took me a long time to reconstruct the details of the proof.

CKS' definition of "bicategory of spans" uses their notion "tabulation", presumably because it is only applied in situations where maps are involved and it is more desirable to have axioms that involve a weaker biuniversal property rather than a stronger one. However I am more interested in "wide tabulation", as it is in some sense the nicer notion, and since I have had to establish various kinds of preservation results that I don't want to repeat for both tabulation and wide tabulation, I am using wide tabulation everywhere, calling it simply "tabulation". The fact that the "input leg" of a tabulation must be a map is an essential ingredient throughout.

I have attempted to follow CKS variable naming conventions as much as possible in this development to avoid confusion when comparing with their paper, even though these are sometimes at odds with what I have been using elsewhere in this document.

locale *tabulation* =
  tabulation-data +
assumes *T1*: $\bigwedge u\ \omega.$
  $[\![\ \mathit{ide}\ u;\ «\omega : \mathit{dom}\ \omega \Rightarrow r \star u»\ ]\!] \Longrightarrow$
  $\exists\, w\ \vartheta\ \nu.\ \mathit{ide}\ w \wedge «\vartheta : f \star w \Rightarrow u» \wedge «\nu : \mathit{dom}\ \omega \Rightarrow g \star w» \wedge \mathit{iso}\ \nu \wedge$
    $\mathit{composite\text{-}cell}\ w\ \vartheta \cdot \nu = \omega$
  and *T2*: $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.$
  $[\![\ \mathit{ide}\ w;\ \mathit{ide}\ w';\ «\vartheta : f \star w \Rightarrow u»;\ «\vartheta' : f \star w' \Rightarrow u»;\ «\beta : g \star w \Rightarrow g \star w'»;$
    $\mathit{composite\text{-}cell}\ w\ \vartheta = \mathit{composite\text{-}cell}\ w'\ \vartheta' \cdot \beta\ ]\!] \Longrightarrow$
  $\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$

T1:

The following definition includes the additional axiom *T0*, which states that the

"input leg" f is a map.

locale *tabulation-data-with-T0* =
  tabulation-data +
  T0: *map-in-bicategory V H* a i *src trg f*
begin

  abbreviation η **where** $\eta \equiv T0.\eta$
  abbreviation ε **where** $\varepsilon \equiv T0.\varepsilon$

If «$\varrho : g \Rightarrow r \star f$» is a 2-cell and f is a map, then «$T0.trnr_\varepsilon\ r\ \varrho : g \star f^* \Rightarrow r$», where $T0.trnr_\varepsilon\ r\ \varrho$ is the adjoint transpose of ϱ. We will show (CKS Proposition 1(d)) that if ϱ is a tabulation, then $\psi = T0.trnr_\varepsilon\ r\ \varrho$ is an isomorphism. However, regardless of whether ϱ is a tabulation, the mapping $\varrho \mapsto \psi$ is injective, and we can recover ϱ by the formula: $\varrho = (\psi \star f) \cdot T0.trnr_\eta\ g\ (g \star f^*)$. The proof requires only *T0* and the "syntactic" properties of the tabulation data, and in particular does not require the tabulation conditions *T1* and *T2*. In case ϱ is in fact a tabulation, then this formula can be interpreted as expressing that ϱ is obtained by transposing the identity «$g \star f^* : g \star f^* \Rightarrow g \star f^*$» to obtain a 2-cell «$T0.trnr_\eta\ g\ (g \star f^*) : g \Rightarrow (g \star f^*) \star f$» (which may be regarded as the canonical tabulation of $g \star f^*$), and then composing with the isomorphism «$\psi \star f : (g \star f^*) \star f \Rightarrow r \star f$» to obtain a tabulation of r. This fact will end up being very important in establishing the characterization of bicategories of spans. Strangely, CKS doesn't make any explicit mention of it.

lemma *rep-in-hom* [*intro*]:
shows «$T0.trnr_\varepsilon\ r\ \varrho : g \star f^* \Rightarrow r$»
proof (*unfold* $T0.trnr_\varepsilon$*-def*, *intro comp-in-homI*)
  show «$\varrho \star f^* : g \star f^* \Rightarrow (r \star f) \star f^*$»
    using *tab-in-hom T0.antipar*(*1*) **by** *auto*
  show «$\mathrm{a}[r, f, f^*] : (r \star f) \star f^* \Rightarrow r \star f \star f^*$»
    using *T0.antipar*(*1*−*2*) **by** *auto*
  show «$r \star \varepsilon : r \star f \star f^* \Rightarrow r \star src\ r$»
    using *T0.antipar* **by** *auto*
  show «$\mathrm{r}[r] : r \star src\ r \Rightarrow r$»
    by *auto*
qed

lemma ϱ*-in-terms-of-rep*:
shows $\varrho = (T0.trnr_\varepsilon\ r\ \varrho \star f) \cdot T0.trnr_\eta\ g\ (g \star f^*)$
proof −
  have $(T0.trnr_\varepsilon\ r\ \varrho \star f) \cdot T0.trnr_\eta\ g\ (g \star f^*) =$
      $(\mathrm{r}[r] \cdot composite\text{-}cell\ f^*\ \varepsilon \star f) \cdot ((g \star f^*) \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] \cdot (g \star \eta) \cdot \mathrm{r}^{-1}[g]$
    unfolding $T0.trnr_\varepsilon$*-def* $T0.trnr_\eta$*-def* **by** *simp*

also have ... $= (\mathrm{r}[r] \cdot \textit{composite-cell}\ f^*\ \varepsilon \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] \cdot (g \star \eta) \cdot \mathrm{r}^{-1}[g]$
proof −
  have $((g \star f^*) \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] = \mathrm{a}^{-1}[g, f^*, f]$
    using *comp-cod-arr T0.antipar* **by** *simp*
  thus *?thesis*
    using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot (\textit{composite-cell}\ f^*\ \varepsilon \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] \cdot (g \star \eta) \cdot \mathrm{r}^{-1}[g]$
  using *comp-assoc T0.antipar whisker-right* [*of f* r[*r*] *composite-cell* f^* ε]
  by *fastforce*
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \star f) \cdot ((\varrho \star f^*) \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] \cdot$
$(g \star \eta) \cdot \mathrm{r}^{-1}[g]$
  using *T0.antipar whisker-right* [*of f* $(r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*]$ $\varrho \star f^*$] *comp-assoc*
  by *fastforce*
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot (\mathrm{a}[r, f, f^*] \star f) \cdot$
$((\varrho \star f^*) \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] \cdot (g \star \eta) \cdot \mathrm{r}^{-1}[g]$
  using *T0.antipar whisker-right* [*of f r* $\star$ ε a[*r*, *f*, f^*]] *comp-assoc* **by** *fastforce*
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot (\mathrm{a}[r, f, f^*] \star f) \cdot$
$\mathrm{a}^{-1}[r \star f, f^*, f] \cdot (\varrho \star f^* \star f) \cdot (g \star \eta) \cdot \mathrm{r}^{-1}[g]$
proof −
  have $((\varrho \star f^*) \star f) \cdot \mathrm{a}^{-1}[g, f^*, f] = \mathrm{a}^{-1}[r \star f, f^*, f] \cdot (\varrho \star f^* \star f)$
    using *assoc′-naturality* [*of* ϱ f^* *f*] *T0.antipar* **by** *simp*
  thus *?thesis*
    using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot$
$(\mathrm{a}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r \star f, f^*, f] \cdot$

$((r \star f) \star \eta) \cdot (\varrho \star src\ (f)) \cdot \mathrm{r}^{-1}[g]$
proof −
have $(\varrho \star f^* \star f) \cdot (g \star \eta) = ((r \star f) \star \eta) \cdot (\varrho \star src\ (f))$
using *comp-arr-dom comp-cod-arr T0.antipar interchange* [*of* $\varrho\ g\ f^* \star f\ \eta$]
interchange [*of* $r \star f\ \varrho\ \eta\ src\ (f)$]
by *auto*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot (\mathrm{a}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r \star f, f^*, f] \cdot$
$((r \star f) \star \eta) \cdot \mathrm{r}^{-1}[r \star f] \cdot \varrho$
using *runit′-naturality* [*of* ϱ] **by** *simp*
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot$
$\mathrm{a}^{-1}[r, f \star f^*, f] \cdot (r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot \mathrm{a}[r, f, f^* \star f] \cdot$
$((r \star f) \star \eta) \cdot \mathrm{r}^{-1}[r \star f] \cdot \varrho$
proof −
have $(\mathrm{a}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r \star f, f^*, f] =$
$\mathrm{a}^{-1}[r, f \star f^*, f] \cdot (r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot \mathrm{a}[r, f, f^* \star f]$
proof −
have $\mathrm{a}^{-1}[r \star f, f^*, f] =$
$(\mathrm{a}^{-1}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r, f \star f^*, f] \cdot (r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot \mathrm{a}[r, f, f^* \star f]$
using *pentagon′* [*of* $r\ f\ f^*\ f$] *T0.antipar iso-assoc comp-assoc*
invert-side-of-triangle(*2*)
[*of* $((\mathrm{a}^{-1}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r, f \star f^*, f]) \cdot (r \star \mathrm{a}^{-1}[f, f^*, f])$
$\mathrm{a}^{-1}[r \star f, f^*, f]\ \mathrm{a}^{-1}[r, f, f^* \star f]$]
by *fastforce*
hence $(\mathrm{a}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r \star f, f^*, f] =$
$((\mathrm{a}[r, f, f^*] \star f) \cdot (\mathrm{a}^{-1}[r, f, f^*] \star f)) \cdot$
$\mathrm{a}^{-1}[r, f \star f^*, f] \cdot (r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot \mathrm{a}[r, f, f^* \star f]$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}^{-1}[r, f \star f^*, f] \cdot (r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot \mathrm{a}[r, f, f^* \star f]$
proof −
have $(\mathrm{a}[r, f, f^*] \star f) \cdot (\mathrm{a}^{-1}[r, f, f^*] \star f) \cdot \mathrm{a}^{-1}[r, f \star f^*, f] =$
$((r \star f \star f^*) \star f) \cdot \mathrm{a}^{-1}[r, f \star f^*, f]$
using *comp-cod-arr comp-assoc iso-assoc comp-arr-inv T0.antipar*
whisker-right [*of* $f\ \mathrm{a}[r, f, f^*]\ \mathrm{a}^{-1}[r, f, f^*]$] *comp-assoc-assoc′*
by *simp*
also have ... $= \mathrm{a}^{-1}[r, f \star f^*, f]$
using *comp-cod-arr T0.antipar* **by** *auto*
finally show *?thesis*
using *comp-assoc* **by** *metis*
qed
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot \mathrm{a}^{-1}[r, src\ r, f] \cdot (r \star \varepsilon \star f) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f]) \cdot (r \star f \star \eta) \cdot \mathrm{a}[r, f, src\ (f)] \cdot \mathrm{r}^{-1}[r \star f] \cdot \varrho$

```
    proof −
      have ((r ⋆ ε) ⋆ f) · a⁻¹[r, f ⋆ f*, f] = a⁻¹[r, src r, f] · (r ⋆ ε ⋆ f)
        using assoc′-naturality [of r ε f] by auto
      moreover have a[r, f, f* ⋆ f] · ((r ⋆ f) ⋆ η) = (r ⋆ f ⋆ η) · a[r, f, src (f)]
        using assoc-naturality [of r f η] T0.antipar by auto
      ultimately show ?thesis
        using comp-assoc by metis
    qed
    also have ... = (r[r] ⋆ f) · a⁻¹[r, src r, f] · (r ⋆ (ε ⋆ f) ·
                    a⁻¹[f, f*, f] · (f ⋆ η)) · a[r, f, src (f)] · r⁻¹[r ⋆ f] · ϱ
    proof −
      have seq a⁻¹[f, f*, f] (f ⋆ η)
        using T0.antipar by force
      moreover have seq (ε ⋆ f) (a⁻¹[f, f*, f] · (f ⋆ η))
        using T0.antipar by fastforce
      ultimately have (r ⋆ ε ⋆ f) · (r ⋆ a⁻¹[f, f*, f]) · (r ⋆ f ⋆ η) =
                      r ⋆ (ε ⋆ f) · a⁻¹[f, f*, f] · (f ⋆ η)
        using T0.antipar whisker-left [of r a⁻¹[f, f*, f] f ⋆ η]
              whisker-left [of r ε ⋆ f a⁻¹[f, f*, f] · (f ⋆ η)]
        by auto
      thus ?thesis
        using comp-assoc by metis
    qed
    also have ... = (r[r] ⋆ f) · a⁻¹[r, src r, f] · (r ⋆ l⁻¹[f] · r[f]) ·
                    a[r, f, src (f)] · r⁻¹[r ⋆ f] · ϱ
      using T0.triangle-left by simp
    also have ... = ((r[r] ⋆ f) · a⁻¹[r, src r, f] · (r ⋆ l⁻¹[f])) ·
                    ((r ⋆ r[f]) · a[r, f, src (f)] · r⁻¹[r ⋆ f]) · ϱ
      using whisker-left [of r l⁻¹[f] r[f]] comp-assoc by simp
    also have ... = ((r ⋆ l[f]) · (r ⋆ l⁻¹[f])) · (r[r ⋆ f] · r⁻¹[r ⋆ f]) · ϱ
      using triangle′ [of r f] runit-hcomp [of r f] comp-assoc by simp
    also have ... = ϱ
    proof −
      have (r ⋆ l[f]) · (r ⋆ l⁻¹[f]) = r ⋆ f
        using iso-lunit comp-arr-inv′ whisker-left [of r l[f] l⁻¹[f]] by simp
      moreover have (r[r ⋆ f] · r⁻¹[r ⋆ f]) = r ⋆ f
        using iso-runit inv-is-inverse comp-arr-inv′ by auto
      ultimately show ?thesis
        using comp-cod-arr by simp
    qed
    finally show ?thesis by simp
  qed

end
```

The following corresponds to what CKS call "tabulation"; it supposes axiom *T0*, but involves weaker versions of *T1* and *T2*. I am calling it "narrow tabulation".

```
locale narrow-tabulation =
  tabulation-data-with-T0 +
```

assumes *T1*: $\bigwedge u\ \omega.$
$[\![$ *is-left-adjoint* u; «ω : *dom* $\omega \Rightarrow r \star u$» $]\!] \Longrightarrow$
$\exists\, w\ \vartheta\ \nu.$ *ide* $w \wedge$ «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «ν : *dom* $\omega \Rightarrow g \star w$» $\wedge$ *iso* $\nu \wedge$
composite-cell $w\ \vartheta \cdot \nu = \omega$
and *T2*: $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.$
$[\![$ *is-left-adjoint* u; *ide* w; *ide* w';
«$\vartheta : f \star w \Rightarrow u$»; «$\vartheta' : f \star w' \Rightarrow u$»; «$\beta : g \star w \Rightarrow g \star w'$»;
composite-cell $w\ \vartheta =$ *composite-cell* $w'\ \vartheta' \cdot \beta\ ]\!] \Longrightarrow$
$\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$

The next few locales are used to bundle up some routine consequences of the situations described by the hypotheses and conclusions of the tabulation axioms, so we don't have to keep deriving them over and over again in each context, and also so as to keep the simplification rules oriented consistently with each other.

locale *uwϑ* =
tabulation-data +
fixes u :: $'a$
and w :: $'a$
and ϑ :: $'a$
assumes *uwϑ*: *ide* $w \wedge$ «$\vartheta : f \star w \Rightarrow u$»
begin

lemma *ide-u*:
shows *ide* u
using *uwϑ* **by** *force*

lemma *u-in-hom* [*intro*]:
shows «u : *src* $u \rightarrow$ *src* r»
using *uwϑ ide-u ide-cod* [*of* ϑ] *hseq-char* [*of f w*]
apply (*intro in-hhomI*, *simp-all*)
by (*metis arr-dom in-homE leg0-simps*(*3*) *trg-hcomp vconn-implies-hpar*(*4*))

lemma *u-simps* [*simp*]:
shows *ide* u **and** *arr* u
and *trg* $u =$ *src* r
and *dom* $u = u$ **and** *cod* $u = u$
using *ide-u u-in-hom* **by** *auto*

lemma *ide-w*:
shows *ide* w
using *uwϑ* **by** *auto*

lemma *w-in-hom* [*intro*]:
shows «w : *src* $u \rightarrow$ *src* f» **and** «$w : w \Rightarrow w$»
proof −
show «$w : w \Rightarrow w$»
using *ide-w* **by** *auto*
show «w : *src* $u \rightarrow$ *src* f»
proof

```
      show arr w using ide-w by simp
      show src w = src u
        using uwϑ ide-dom [of ϑ] hseq-char [of f w]
        by (metis arr-dom in-homE src-cod src-dom hcomp-simps(1))
      show trg w = src f
        using uwϑ ide-dom [of ϑ] hseq-char [of f w]
        by (metis arr-dom in-homE)
    qed
  qed

  lemma w-simps [simp]:
  shows ide w and arr w
  and src w = src u and trg w = src f
  and dom w = w and cod w = w
    using ide-w w-in-hom by auto

  lemma ϑ-in-hom [intro]:
  shows «ϑ : src u → src r» and «ϑ : f ⋆ w ⇒ u»
  proof −
    show «ϑ : f ⋆ w ⇒ u»
      using uwϑ by simp
    show «ϑ : src u → src r»
      using uwϑ hcomp-simps(1−2)
      by (metis arrI in-hhomI u-simps(3) vconn-implies-hpar(1−4))
  qed

  lemma ϑ-simps [simp]:
  shows arr ϑ and src ϑ = src u and trg ϑ = src r
  and dom ϑ = f ⋆ w and cod ϑ = u
    using ϑ-in-hom by auto

end

locale uwϑω =
  uwϑ +
fixes ω :: 'a
assumes uwϑω: uwϑω u w ϑ ω
begin

  lemma ω-in-hom [intro]:
  shows «ω : src w → trg r» and «ω : dom ω ⇒ r ⋆ u»
  proof −
    show «ω : src w → trg r»
      using uwϑω src-cod [of ω] trg-cod [of ω]
      apply (elim conjE in-homE)
      by simp
    show «ω : dom ω ⇒ r ⋆ u»
      using uwϑω by auto
  qed
```

```
  lemma ω-simps [simp]:
  shows arr ω and src ω = src w and trg ω = trg r
  and cod ω = r ⋆ u
    using ω-in-hom by auto

end

locale uwϑων =
  uwϑ +
fixes ω :: 'a
and ν :: 'a
assumes uwϑων: uwϑων u w ϑ ω ν
begin

  lemma ν-in-hom [intro]:
  shows «ν : src u → trg r» and «ν : dom ω ⇒ g ⋆ w»
  proof −
    show «ν : dom ω ⇒ g ⋆ w»
      using uwϑων by auto
    show «ν : src u → trg r»
    proof
      show 1: arr ν
        using uwϑων by auto
      show src ν = src u
      proof −
        have src (cod ν) = src u
          using uwϑων
          by (metis arr-cod hcomp-simps(1) in-homE w-simps(3))
        thus ?thesis by simp
      qed
      show trg ν = trg r
      proof −
        have trg (cod ν) = trg r
          using uwϑων
          by (metis arr-cod hcomp-simps(2) in-homE leg1-simps(4))
        thus ?thesis by simp
      qed
    qed
  qed

  lemma ν-simps [simp]:
  shows iso ν and arr ν and src ν = src u and trg ν = trg r
  and cod ν = g ⋆ w
    using uwϑων ν-in-hom by auto

  sublocale uwϑω
  proof (unfold-locales, intro conjI)
    show ide w
```

```
      using uwϑων by simp
    show «ϑ : f ⋆ w ⇒ u»
      using uwϑων by simp
    have «(r ⋆ ϑ) · a[r, f, w] · (ϱ ⋆ w) · ν : dom ν ⇒ r ⋆ u»
      using ide-base ide-leg0 ide-w by fastforce
    thus «ω : dom ω ⇒ r ⋆ u»
      using uwϑων by auto
  qed

end


locale uwϑw′ϑ′ =
  tabulation-data V H a ι src trg r ϱ f g +
  uwϑ: uwϑ V H a ι src trg r ϱ f g u w ϑ +
  uw′ϑ′: uwϑ V H a ι src trg r ϱ f g u w′ ϑ′
for V :: ′a comp                (infixr ‹·› 55)
and H :: ′a ⇒ ′a ⇒ ′a           (infixr ‹⋆› 53)
and a :: ′a ⇒ ′a ⇒ ′a ⇒ ′a      (‹a[-, -, -]›)
and ι :: ′a ⇒ ′a                (‹i[-]›)
and src :: ′a ⇒ ′a
and trg :: ′a ⇒ ′a
and r :: ′a
and ϱ :: ′a
and f :: ′a
and g :: ′a
and u :: ′a
and w :: ′a
and ϑ :: ′a
and w′ :: ′a
and ϑ′ :: ′a

locale uwϑw′ϑ′γ =
  uwϑw′ϑ′ +
fixes γ :: ′a
assumes γ-in-vhom: «γ : w ⇒ w′»
and ϑ = ϑ′ · (f ⋆ γ)
begin

  lemma γ-in-hom [intro]:
  shows «γ : src u → src f» and «γ : w ⇒ w′»
  proof −
    show «γ : w ⇒ w′»
      using γ-in-vhom by simp
    show «γ : src u → src f»
    proof
      show arr γ
        using γ-in-vhom by auto
      show src γ = src u
```

```
      using γ-in-vhom src-dom [of γ]
      apply (elim in-homE) by simp
    show trg γ = src f
      using γ-in-vhom trg-dom [of γ]
      apply (elim in-homE) by simp
  qed
qed

lemma γ-simps [simp]:
shows arr γ
and src γ = src u and trg γ = src f
and dom γ = w and cod γ = w'
  using γ-in-hom by auto

end

locale uwϑw'ϑ'β =
  uwϑw'ϑ' +
fixes β :: 'a
assumes uwϑw'ϑ'β: uwϑw'ϑ'β u w ϑ w' ϑ' β
begin

lemma β-in-hom [intro]:
shows «β : src u → trg r» and «β : g ⋆ w ⇒ g ⋆ w'»
proof −
  show «β : g ⋆ w ⇒ g ⋆ w'»
    using uwϑw'ϑ'β by auto
  show «β : src u → trg r»
    using uwϑw'ϑ'β src-dom [of β] trg-dom [of β] hseq-char [of g w]
    apply (elim conjE in-homE) by auto
qed

lemma β-simps [simp]:
shows arr β and src β = src u and trg β = trg r
and dom β = g ⋆ w and cod β = g ⋆ w'
  using β-in-hom by auto

end
```

2.2.2 Tabulations yield Factorizations

If (f, ϱ, g) is a (wide) tabulation, then f is automatically a map; this is CKS Proposition 1(c). The proof sketch provided by CKS is only three lines long, and for a long time I was only able to prove one of the two triangle identities. Finally, after gaining a lot of experience with the definitions I saw how to prove the other. CKS say nothing about the extra step that seems to be required.

```
context tabulation
begin
```

The following is used in order to allow us to apply the coherence theorem to shortcut proofs of equations between canonical arrows.

interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
notation *E.eval* (‹⦃-⦄›)

lemma *satisfies-T0*:
shows *is-left-adjoint f*
proof −

The difficulty is filling in details left out by CKS, and accounting for the fact that they have suppressed unitors and associators everywhere. In addition, their typography generally uses only parentheses, with no explicit operation symbols to distinguish between horizontal and vertical composition. In some cases, for example the statement of T2 in the definition of tabulation, this makes it difficult for someone not very experienced with the definitions to reconstruct the correct formulas.

CKS say to first apply *T1* with $u = src\ r$, $v = r$, and $\varrho' = r$. However, «$r : r \Rightarrow r$», not «$r : r \Rightarrow r \star src\ r$», so we have to take $\varrho' = \mathrm{r}^{-1}[r]$.

obtain $f_a\ \varepsilon\ \nu$
where f_a: $ide\ f_a \wedge$ «$\varepsilon : f \star f_a \Rightarrow src\ r$» $\wedge$ «$\nu : r \Rightarrow g \star f_a$» $\wedge\ iso\ \nu \wedge$
$composite\text{-}cell\ f_a\ \varepsilon \cdot \nu = \mathrm{r}^{-1}[r]$
using *T1* [*of src r* $\mathrm{r}^{-1}[r]$] *runit′-in-hom* [*of r*] *ide-base comp-assoc* **by** *auto*
have $f_a{}'$: $composite\text{-}cell\ f_a\ \varepsilon \cdot \nu = \mathrm{r}^{-1}[r]$
using f_a **by** *simp*
have f_a: $ide\ f_a \wedge$ «$\varepsilon : f \star f_a \Rightarrow src\ r$» $\wedge$ «$\nu : r \Rightarrow g \star f_a$» $\wedge\ iso\ \nu$
using f_a **by** *simp*
have *1*: $src\ f_a = trg\ f$
using $f_a\ f_a{}'$ *comp-assoc*
by (*metis ide-base leg0-simps*(*3*) *runit′-simps*(*1*) *seqE src-hcomp vconn-implies-hpar*(*1*) *vseq-implies-hpar*(*1*))
have *2*: $trg\ f_a = src\ g$
using f_a **by** *force*
have ε: «$\varepsilon : f \star f_a \Rightarrow trg\ f$» $\wedge$ «$\varepsilon : trg\ f \rightarrow trg\ f$» $\wedge$
$arr\ \varepsilon \wedge src\ \varepsilon = trg\ f \wedge trg\ \varepsilon = trg\ f \wedge dom\ \varepsilon = f \star f_a \wedge cod\ \varepsilon = trg\ f$
using f_a *1 2*
by (*metis in-hhomI in-homE leg0-simps*(*3*) *src-src trg-src vconn-implies-hpar*(*1−4*))
have ν: «$\nu : r \Rightarrow g \star f_a$» $\wedge$ «$\nu : trg\ f \rightarrow trg\ g$» $\wedge$
$arr\ \nu \wedge src\ \nu = trg\ f \wedge trg\ \nu = trg\ g \wedge dom\ \nu = r \wedge cod\ \nu = g \star f_a$
using f_a **by** *force*

Next, CKS say to apply *T2* with $w = trg\ f_a = src\ f$, $w' = f_a \star f$, $u = f$, to obtain the unit and the adjunction conditions, but they don't say explicitly what to use for ϑ, ϑ', and β. We need «$\vartheta : f \star w \Rightarrow u$» and «$\vartheta' : f \star w' \Rightarrow u$»; *i.e.* «$\vartheta : f \star trg\ f_a \Rightarrow f$» and «$\vartheta' : f \star f_a \star f \Rightarrow f$». Evidently, we may take $\vartheta = \varrho[f]$ and $\vartheta' = \mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f]$.

What should be taken for β? Reconstructing this is a little bit more difficult. *T2* requires «$\beta : g \star w \Rightarrow g \star w'$», hence «$\beta : g \star trg\ f_a \Rightarrow g \star f_a \star f$». We have the isomorphism «$\nu : r \Rightarrow g \star f_a$» from *T1*. Also «$\varrho : g \Rightarrow r \star f$». So «$\mathrm{a}[g, f_a, f] \cdot (\nu \star f)$

$\cdot\ \varrho \cdot \mathrm{r}[g] : g \star \mathit{trg}\ f_a \Rightarrow g \star f_a \star f$», suggesting that we take $\beta = \mathrm{a}[g, f_a, f] \cdot (\nu \star f) \cdot \varrho \cdot \mathrm{r}[g]$. Now, to apply *T2* we need to satisfy the equation:

$$(r \star \vartheta) \cdot \mathrm{a}[r, f, \mathit{trg}\ f_a] \cdot (\varrho \star \mathit{trg}\ f_a\ ) = (r \star \vartheta') \cdot \mathrm{a}[r, f, f_a \star f] \cdot (\varrho \star f_a \star f) \cdot \beta;$$

that is, with our choice of ϑ, ϑ', and β:

$(r \star \mathrm{r}[f]) \cdot \mathrm{a}[r, f, \mathit{trg}\ f_a] \cdot (\varrho \star \mathit{trg}\ f_a\ ) = (r \star \mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f]) \cdot \mathrm{a}[r, f, f_a \star f] \cdot (\varrho \cdot (f_a \star f)) \cdot \mathrm{a}[g, f_a, f] \cdot (\nu \star f) \cdot \varrho \cdot \mathrm{r}[g]$.

It is not too difficult to get the idea of showing that the left-hand side is equal to $\varrho \cdot \mathrm{r}[g]$ (note that $\mathit{trg}\ f_a = \mathit{src}\ f = \mathit{src}\ g$] and $\mathit{trg}\ f = \mathit{src}\ r$), so we should also try to prove that the right-hand side is equal to this as well. What we have to work with is the equation:

$$\mathrm{r}^{-1}[r] = (r \star \varepsilon) \cdot \mathrm{a}[r, f, f_a] \cdot (\varrho \star f_a\ ) \cdot \nu.$$

After some pondering, I realized that to apply this to the right-hand side of the equation to be shown requires that we re-associate everything to the left, so that f stands alone on the right.

```
let ?β = a[g, f_a, f] · (ν ⋆ f) · ϱ · r[g]
let ?ϑ = r[f]
let ?ϑ' = l[f] · (ε ⋆ f) · a⁻¹[f, f_a, f]
have β: «?β : g ⋆ src g ⇒ g ⋆ f_a ⋆ f» ∧ «?β : src f → trg g» ∧
        src ?β = src g ∧ trg ?β = trg g ∧ dom ?β = g ⋆ src g ∧ cod ?β = g ⋆ f_a ⋆ f
proof −
  have 3: «?β : g ⋆ src g ⇒ g ⋆ f_a ⋆ f»
    using f_a 1 2 by fastforce
  moreover have «?β : src f → trg g»
    using 1 2 3 f_a by auto
  ultimately show ?thesis
    by (auto simp add: in-hhom-def)
qed
have ϑ': «?ϑ' : f ⋆ f_a ⋆ f ⇒ f»
  using f_a 1 2 ε by fastforce
have A: composite-cell (trg f_a) r[f] = composite-cell (f_a ⋆ f) ?ϑ' · ?β
proof −
  have composite-cell (trg f_a) r[f] = ϱ · r[g]
    using 2 runit-hcomp runit-naturality [of ϱ] comp-assoc by simp
  also have ... = composite-cell (f_a ⋆ f) ?ϑ' · ?β
  proof −
    have composite-cell (f_a ⋆ f) ?ϑ' · ?β =
          (composite-cell (f_a ⋆ f) ?ϑ' · a[g, f_a, f]) · (ν ⋆ f) · ϱ · r[g]
      using comp-assoc by simp
    also have ... = ϱ · r[g]
    proof −
      have (composite-cell (f_a ⋆ f) ?ϑ' · a[g, f_a, f]) · (ν ⋆ f) = r ⋆ f
      proof −
        have (composite-cell (f_a ⋆ f) ?ϑ' · a[g, f_a, f]) · (ν ⋆ f) =
              r[r] · (r ⋆ ε) · a[r, f, f_a] · (ϱ ⋆ f_a) · ν ⋆ f
        proof −
```

have (*composite-cell* $(f_a \star f)$ *?ϑ′* $\cdot$ $\mathrm{a}[g, f_a, f]) \cdot (\nu \star f) =$
$(r \star \mathrm{l}[f]) \cdot (r \star \varepsilon \star f) \cdot$
composite-cell $(f_a \star f)$ $\mathrm{a}^{-1}[f, f_a, f] \cdot (\mathrm{a}[g, f_a, f] \cdot (\nu \star f))$
using f_a *1 2* ε *whisker-left comp-assoc* **by** *auto*
also have ... $= (\mathrm{r}[r] \star f) \cdot \mathrm{a}^{-1}[r,$ *src* $r, f] \cdot (r \star \varepsilon \star f) \cdot$
composite-cell $(f_a \star f)$ $\mathrm{a}^{-1}[f, f_a, f] \cdot (\mathrm{a}[g, f_a, f] \cdot (\nu \star f))$
using f_a *1 2 comp-assoc* **by** (*simp add*: *triangle′*)
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot \mathrm{a}^{-1}[r, f \star f_a, f] \cdot$
composite-cell $(f_a \star f)$ $\mathrm{a}^{-1}[f, f_a, f] \cdot (\mathrm{a}[g, f_a, f] \cdot (\nu \star f))$
proof −
have $\mathrm{a}^{-1}[r,$ *src* $r, f] \cdot (r \star \varepsilon \star f) = ((r \star \varepsilon) \star f) \cdot \mathrm{a}^{-1}[r, f \star f_a, f]$
using f_a ε *assoc′-naturality* [*of* r ε f] **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot$
$(\mathrm{a}[r, f, f_a] \star f) \cdot \mathrm{a}^{-1}[r \star f, f_a, f] \cdot (\varrho \star f_a \star f) \cdot$
$\mathrm{a}[g, f_a, f] \cdot (\nu \star f)$
proof −
have $(\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot \mathrm{a}^{-1}[r, f \star f_a, f] \cdot$
composite-cell $(f_a \star f)$ $\mathrm{a}^{-1}[f, f_a, f] \cdot (\mathrm{a}[g, f_a, f] \cdot (\nu \star f)) =$
$(\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot$
$(\mathrm{a}^{-1}[r, f \star f_a, f] \cdot (r \star \mathrm{a}^{-1}[f, f_a, f]) \cdot \mathrm{a}[r, f, f_a \star f]) \cdot$
$(\varrho \star f_a \star f) \cdot \mathrm{a}[g, f_a, f] \cdot (\nu \star f)$
by (*simp add*: *comp-assoc*)
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot$
$((\mathrm{a}[r, f, f_a] \star f) \cdot \mathrm{a}^{-1}[r \star f, f_a, f]) \cdot$
$(\varrho \star f_a \star f) \cdot \mathrm{a}[g, f_a, f] \cdot (\nu \star f)$
proof −
have $\mathrm{a}^{-1}[r, f \star f_a, f] \cdot (r \star \mathrm{a}^{-1}[f, f_a, f]) \cdot \mathrm{a}[r, f, f_a \star f] =$
$(\mathrm{a}[r, f, f_a] \star f) \cdot \mathrm{a}^{-1}[r \star f, f_a, f]$
proof −

have $\mathrm{a}^{-1}[r, f \star f_a, f] \cdot (r \star \mathrm{a}^{-1}[f, f_a, f]) \cdot \mathrm{a}[r, f, f_a \star f] =$
$\{\!|\mathbf{a}^{-1}[\langle r\rangle, \langle f\rangle \star \langle f_a\rangle, \langle f\rangle] \cdot (\langle r\rangle \star \mathbf{a}^{-1}[\langle f\rangle, \langle f_a\rangle, \langle f\rangle]) \cdot$
$\mathbf{a}[\langle r\rangle, \langle f\rangle, \langle f_a\rangle \star \langle f\rangle]|\!\}$
using f_a *1 2* a'*-def* α*-def assoc′-eq-inv-assoc* **by** *auto*
also have ... $= \{\!|(\mathbf{a}[\langle r\rangle, \langle f\rangle, \langle f_a\rangle] \star \langle f\rangle) \cdot \mathbf{a}^{-1}[\langle r\rangle \star \langle f\rangle, \langle f_a\rangle, \langle f\rangle]|\!\}$
using f_a *1 2* **by** (*intro E.eval-eqI*, *auto*)
also have ... $= (\mathrm{a}[r, f, f_a] \star f) \cdot \mathrm{a}^{-1}[r \star f, f_a, f]$
using f_a *1 2* a'*-def* α*-def assoc′-eq-inv-assoc* **by** *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (\mathrm{r}[r] \star f) \cdot ((r \star \varepsilon) \star f) \cdot (\mathrm{a}[r, f, f_a] \star f) \cdot$
$\mathrm{a}^{-1}[r \star f, f_a, f] \cdot (\varrho \star f_a \star f) \cdot \mathrm{a}[g, f_a, f] \cdot (\nu \star f)$
by (*simp add*: *comp-assoc*)
finally show *?thesis* **by** *blast*

```
        qed
        also have ... = (r[r] ⋆ f) · ((r ⋆ ε) ⋆ f) ·
                        (a[r, f, f_a] ⋆ f) · ((ϱ ⋆ f_a) ⋆ f) · a⁻¹[g, f_a, f] ·
                          a[g, f_a, f] · (ν ⋆ f)
        proof −
          have a⁻¹[r ⋆ f, f_a, f] · (ϱ ⋆ f_a ⋆ f) = ((ϱ ⋆ f_a) ⋆ f) · a⁻¹[g, f_a, f]
            using f_a 1 2 assoc′-naturality [of ϱ f_a f] by auto
          thus ?thesis
            by (metis comp-assoc)
        qed
        also have ... = (r[r] ⋆ f) · ((r ⋆ ε) ⋆ f) · (a[r, f, f_a] ⋆ f) ·
                        ((ϱ ⋆ f_a) ⋆ f) · (ν ⋆ f)
        proof −
          have a⁻¹[g, f_a, f] · a[g, f_a, f] = (g ⋆ f_a) ⋆ f
            using f_a 1 2 comp-assoc-assoc′ by auto
          moreover have ((g ⋆ f_a) ⋆ f) · (ν ⋆ f) = ν ⋆ f
            by (simp add: ν comp-cod-arr)
          ultimately show ?thesis
            using comp-assoc by metis
        qed
        also have ... = (r[r] · (r ⋆ ε) · a[r, f, f_a] · (ϱ ⋆ f_a) · ν) ⋆ f
        proof −
          have arr (r[r] · (r ⋆ ε) · a[r, f, f_a] · (ϱ ⋆ f_a) · ν)
            using f_a′ comp-assoc by auto
          thus ?thesis
            using whisker-right by fastforce
        qed
        finally show ?thesis by blast
      qed
      also have ... = (r[r] · r⁻¹[r]) ⋆ f
        using f_a′ comp-assoc by simp
      also have ... = r ⋆ f
        using ide-base by (simp add: comp-arr-inv′)
      finally show ?thesis by blast
    qed
    thus ?thesis
      using ide-leg0 ide-leg1 tab-in-hom comp-cod-arr comp-assoc tab-simps(5) arrI
      by metis
  qed
  finally show ?thesis by argo
qed
obtain η where η: «η : trg f_a ⇒ f_a ⋆ f» ∧ ?β = g ⋆ η ∧
                (l[f] · (ε ⋆ f) · a⁻¹[f, f_a, f]) · (f ⋆ η) = r[f]
  using β ϑ′ A 1 2 f_a runit-in-hom ide-leg0 ide-hcomp src.preserves-ide
      T2 [of trg f_a f_a ⋆ f r[f] f l[f] · (ε ⋆ f) · a⁻¹[f, f_a, f] ?β] comp-assoc
      leg1-simps(3)
  by metis
```

have η': $?\beta = g \star \eta \wedge (\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f]) \cdot (f \star \eta) = \mathrm{r}[f]$
  using η **by** *simp*
have η: «η : $trg\ f_a \Rightarrow f_a \star f$» $\wedge$ «η : $src\ f \rightarrow src\ f$» $\wedge$
    $arr\ \eta \wedge src\ \eta = src\ f \wedge trg\ \eta = src\ f \wedge dom\ \eta = trg\ f_a \wedge cod\ \eta = f_a \star f$
  using η β *2* **by** *force*

have *adjunction-in-bicategory V H* a i *src trg f* f_a η ε
proof
  show *ide f* **using** *ide-leg0* **by** *simp*
  show *ide* f_a **using** f_a **by** *blast*
  show *η-in-hom*: «η : $src\ f \Rightarrow f_a \star f$»
    using η *2* **by** *simp*
  show *ε-in-hom*: «ε : $f \star f_a \Rightarrow src\ f_a$»
    using f_a *1* **by** *simp*
  show $*$: $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f]$
    using *ide-leg0 iso-lunit invert-side-of-triangle(1)* η' *comp-assoc* **by** *auto*

We have proved one of the triangle identities; now we have to show the other. This part, not mentioned by CKS, took me a while to discover. Apply *T2* again, this time with the following:

$$
\begin{aligned}
&w = src\ f \star f_a,\\
&\vartheta = (\varepsilon \star \varepsilon) \cdot \mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot (f \star \eta \star f_a),\\
&w' = f_a \star trg,\\
&\vartheta' = \varepsilon \star trg\ f,\\
&\beta = g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]
\end{aligned}
$$

Then the conditions for γ are satisfied by both $\mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$ and $(f_a \star \varepsilon) \cdot \mathrm{a}[f_a, f, f_a] \cdot (\eta \star f_a)$ so they are equal, as required.

show $(f_a \star \varepsilon) \cdot \mathrm{a}[f_a, f, f_a] \cdot (\eta \star f_a) = \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$
proof –
  let $?u = trg\ f \star trg\ f$
  let $?w = src\ f \star f_a$
  let $?w' = f_a \star trg\ f$
  let $?\vartheta = (\varepsilon \star \varepsilon) \cdot \mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot (f \star \eta \star f_a)$
  let $?\vartheta' = (\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f, f_a, trg\ f]$
  let $?\beta = g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$
  let $?\gamma = \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$
  let $?\gamma' = (f_a \star \varepsilon) \cdot \mathrm{a}[f_a, f, f_a] \cdot (\eta \star f_a)$
  have *ϑ-eq'*: $?\vartheta = (trg\ f \star \varepsilon) \cdot \mathrm{a}[trg\ f, f, f_a] \cdot (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star f_a) \cdot \mathrm{a}^{-1}[f, src\ f, f_a]$
  proof –
    have $?\vartheta = (trg\ f \star \varepsilon) \cdot (\varepsilon \star f \star f_a) \cdot$
      $(\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a])) \cdot (f \star \eta \star f_a)$
      using *interchange* [*of trg f* ε ε $f \star f_a$] *comp-arr-dom comp-cod-arr comp-assoc*
      by (*simp add*: ε)
    also have ... $= (trg\ f \star \varepsilon) \cdot (\varepsilon \star f \star f_a) \cdot$
      $(\mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a]) \cdot$
      $(f \star \eta \star f_a)$
    proof –

have $\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) =$
$\mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a]$
proof −
have $(\mathrm{a}[f \star f_a, f, f_a] \cdot ((\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a])) \cdot$
$(f \star \mathrm{a}^{-1}[f_a, f, f_a]) =$
$\mathrm{a}^{-1}[f, f_a, f \star f_a]$
using *1 2* ‹*ide* f_a› *ide-leg0 iso-assoc*
invert-side-of-triangle(*1*)
[*of* $((\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a]) \cdot (f \star \mathrm{a}^{-1}[f_a, f, f_a])$
$\mathrm{a}^{-1}[f \star f_a, f, f_a]$ $\mathrm{a}^{-1}[f, f_a, f \star f_a]$]
pentagon′ comp-assoc **by** *auto*
hence $(\mathrm{a}[f \star f_a, f, f_a] \cdot ((\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a])) =$
$\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a])$
using *1 2* ‹*ide* f_a›
invert-side-of-triangle(*2*)
[*of* $\mathrm{a}^{-1}[f, f_a, f \star f_a]$ $\mathrm{a}[f \star f_a, f, f_a] \cdot ((\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot$
$\mathrm{a}^{-1}[f, f_a \star f, f_a])$
$f \star \mathrm{a}^{-1}[f_a, f, f_a]$]
by *auto*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (\mathit{trg}\, f \star \varepsilon) \cdot ((\varepsilon \star f \star f_a) \cdot \mathrm{a}[f \star f_a, f, f_a]) \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot$
$\mathrm{a}^{-1}[f, f_a \star f, f_a] \cdot (f \star \eta \star f_a)$
using *comp-assoc* **by** *simp*
also have ... $= (\mathit{trg}\, f \star \varepsilon) \cdot \mathrm{a}[\mathit{trg}\, f, f, f_a] \cdot$
$((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f] \cdot (f \star \eta) \star f_a) \cdot$
$\mathrm{a}^{-1}[f, \mathit{src}\, f, f_a]$
proof −
have $((\varepsilon \star f \star f_a) \cdot \mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot$
$\mathrm{a}^{-1}[f, f_a \star f, f_a]) \cdot (f \star \eta \star f_a) =$
$(\mathrm{a}[\mathit{trg}\, f, f, f_a] \cdot ((\varepsilon \star f) \star f_a)) \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot$
$((f \star \eta) \star f_a) \cdot \mathrm{a}^{-1}[f, \mathit{src}\, f, f_a]$
using *assoc-naturality* [*of* ε f f_a] *assoc′-naturality* [*of* f η f_a]
by (*simp add*: *2* ε η ‹*ide* f_a› *comp-assoc*)
also have ... $= \mathrm{a}[\mathit{trg}\, f, f, f_a] \cdot$
$(((\varepsilon \star f) \star f_a) \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot ((f \star \eta) \star f_a)) \cdot$
$\mathrm{a}^{-1}[f, \mathit{src}\, f, f_a]$
using *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}[\mathit{trg}\, f, f, f_a] \cdot$
$((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f, f_a, f] \cdot (f \star \eta) \star f_a) \cdot$
$\mathrm{a}^{-1}[f, \mathit{src}\, f, f_a]$
using η' *comp-assoc whisker-right* ‹*ide* f_a› *null-is-zero*(*2*) *ide-leg0 ext*
runit-simps(*1*)
by *metis*
finally show *?thesis*
using *comp-assoc* **by** *simp*

qed
also have ... $= (trg\ f \star \varepsilon) \cdot \mathrm{a}[trg\ f,\ f,\ f_a] \cdot (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star f_a) \cdot \mathrm{a}^{-1}[f,\ src\ f,\ f_a]$
using $*$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
have ϑ*-eq*: $?\vartheta = (\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f,\ f_a,\ src\ f_a] \cdot (f \star ?\gamma)$
proof −
have $?\vartheta = (trg\ f \star \varepsilon) \cdot \mathrm{a}[trg\ f,\ f,\ f_a] \cdot (\mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star f_a) \cdot \mathrm{a}^{-1}[f,\ src\ f,\ f_a]$
using ϑ*-eq′* **by** *simp*
also have ... =
$(trg\ f \star \varepsilon) \cdot \mathrm{a}[trg\ f,\ f,\ f_a] \cdot (\mathrm{l}^{-1}[f] \star f_a) \cdot (\mathrm{r}[f] \star f_a) \cdot \mathrm{a}^{-1}[f,\ src\ f,\ f_a]$
using ‹*ide* f_a› *whisker-right comp-assoc* **by** *auto*
also have ... $= (trg\ f \star \varepsilon) \cdot ((\mathrm{a}[trg\ f,\ f,\ f_a] \cdot (\mathrm{a}^{-1}[trg\ f,\ f,\ f_a]) \cdot \mathrm{l}^{-1}[f \star f_a])) \cdot$
$(f \star \mathrm{l}[f_a])$
using *2* ‹*ide* f_a› *lunit-hcomp* [*of f* f_a] *invert-side-of-triangle*(*2*) *triangle′*
comp-assoc
by *auto*
also have ... $= (trg\ f \star \varepsilon) \cdot \mathrm{l}^{-1}[f \star f_a] \cdot (f \star \mathrm{l}[f_a])$
using f_a *2 comp-cod-arr iso-assoc comp-arr-inv lunit-hcomp*(*2*) *lunit-hcomp*(*4*)
ide-leg0 leg1-simps(*3*)
by *metis*
also have ... $= \mathrm{l}^{-1}[trg\ f] \cdot \varepsilon \cdot (f \star \mathrm{l}[f_a])$
using ε *lunit′-naturality comp-assoc* **by** *metis*
also have ... $= \mathrm{r}^{-1}[trg\ f] \cdot \varepsilon \cdot (f \star \mathrm{l}[f_a])$
using *unitor-coincidence* **by** *simp*
also have ... $= (\varepsilon \star trg\ f) \cdot \mathrm{r}^{-1}[f \star f_a] \cdot (f \star \mathrm{l}[f_a])$
using ε *runit′-naturality comp-assoc* **by** *metis*
also have ... $= (\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f,\ f_a,\ src\ f_a] \cdot (f \star \mathrm{r}^{-1}[f_a]) \cdot (f \star \mathrm{l}[f_a])$
using *2* ‹*ide* f_a› *runit-hcomp*(*2*) *comp-assoc* **by** *auto*
also have ... $= (\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f,\ f_a,\ src\ f_a] \cdot (f \star ?\gamma)$
using *2* ‹*ide* f_a› *whisker-left* **by** *simp*
finally show *?thesis* **by** *simp*
qed
have ϑ: «$?\vartheta : f \star ?w \Rightarrow ?u$»
using *1 2* ‹*ide* f_a› η*-in-hom* ε **by** *fastforce*
have ϑ': «$?\vartheta' : f \star ?w' \Rightarrow ?u$»
using f_a *1 2* ε **by** *auto*
have *ww′*: *ide* $?w \wedge$ *ide* $?w'$
by (*simp add*: *1 2* ‹*ide* f_a›)
have $\exists!\gamma.$ «$\gamma : ?w \Rightarrow ?w'$» $\wedge\ ?\beta = g \star \gamma \wedge\ ?\vartheta = ?\vartheta' \cdot (f \star \gamma)$
proof −
have «$?\beta : g \star ?w \Rightarrow g \star ?w'$»
using ‹*ide* f_a› *1 2* **by** *auto*
moreover have *composite-cell* $?w\ ?\vartheta =$ *composite-cell* $?w'\ ?\vartheta' \cdot ?\beta$
proof −
have *composite-cell* $?w'\ ?\vartheta' \cdot ?\beta =$
composite-cell $?w\ ((\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f,\ f_a,\ src\ f_a] \cdot (f \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]))$
proof −
have $\mathrm{a}[r,\ f,\ f_a \star trg\ f] \cdot (\varrho \star f_a \star trg\ f) \cdot (g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]) =$

composite-cell ?w $(f \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a])$
proof −
have $\mathrm{a}[r, f, f_a \star trg\ f] \cdot (\varrho \star f_a \star trg\ f) \cdot (g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]) =$
$(\mathrm{a}[r, f, f_a \star trg\ f] \cdot ((r \star f) \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a])) \cdot (\varrho \star src\ f \star f_a)$
proof −
have $(\varrho \star f_a \star trg\ f) \cdot (g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]) = \varrho \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$
using *interchange* [*of* ϱ g $f_a \star trg\ f$ $\mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$]
comp-arr-dom comp-cod-arr 1 2 ‹*ide* f_a›
by *simp*
also have ... $= ((r \star f) \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]) \cdot (\varrho \star src\ f \star f_a)$
proof −
have *seq* $(f_a \star trg\ f)$ $(\mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a])$
using f_a *1 2 ww′* **by** *auto*
thus *?thesis*
using *interchange comp-arr-dom comp-cod-arr 1 2* ‹*ide* f_a›
by (*metis ww′ comp-ide-arr dom-comp leg1-simps*(*3*)
lunit-simps(*4*) *tab-simps*(*1*) *tab-simps*(*5*))
qed
finally show *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... = *composite-cell ?w* $(f \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a])$
using *assoc-naturality* [*of* r f $\mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$] *1 2* ‹*ide* f_a› *comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
hence *composite-cell ?w′ ?ϑ′* · *?β* =
$((r \star (\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f, f_a, trg\ f]) \cdot (r \star f \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a])) \cdot$
$\mathrm{a}[r, f, src\ f \star f_a] \cdot (\varrho \star src\ f \star f_a)$
using *comp-assoc* **by** *simp*
also have
... = *composite-cell ?w* $(((\varepsilon \star trg\ f) \cdot \mathrm{a}^{-1}[f, f_a, trg\ f]) \cdot (f \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]))$
using *whisker-left 1 2* ‹*ide* f_a› *ide-base*
by (*metis* ‹«$(\varepsilon \star \varepsilon) \cdot \mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot (f \star \eta \star f_a)$:
$f \star src\ f \star f_a \Rightarrow trg\ f \star trg\ f$»›
ϑ-eq arrI comp-assoc)
finally show *?thesis*
using *comp-assoc* **by** (*simp add*: *1*)
qed
also have ... = *composite-cell ?w ?ϑ*
using *ϑ-eq* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
using *ww′ ϑ ϑ′ T2* [*of ?w ?w′ ?ϑ ?u ?ϑ′ ?β*] *comp-assoc* **by** *metis*
qed
moreover have «*?γ* : *?w* ⇒ *?w′*» ∧ *?β* = $g \star$ *?γ* ∧ *?ϑ* = *?ϑ′* · ($f \star$ *?γ*)
using *1 2* ‹*ide* f_a› *ϑ-eq comp-assoc* **by** *auto*
moreover have «*?γ′* : *?w* ⇒ *?w′*» ∧ *?β* = $g \star$ *?γ′* ∧ *?ϑ* = *?ϑ′* · ($f \star$ *?γ′*)
proof (*intro conjI*)

show «$?\gamma' : ?w \Rightarrow ?w'$»
  using *1 2* f_a *η-in-hom ε-in-hom* **by** *fastforce*
show $?\beta = g \star ?\gamma'$

This equation is not immediate. To show it, we have to recall the properties from the construction of ε and η. Use the property of η to replace $g \star \eta \star f_a$ by a 2-cell involving ε, ϱ, and ν. Use the property $(r \star \varepsilon) \cdot (\varrho \star f_a) \cdot \nu = \mathrm{r}[r]$ from the construction of ε to eliminate ε and ϱ in favor of inv ν and canonical isomorphisms. Cancelling ν and inv ν leaves the canonical 2-cell $g \star \mathrm{r}^{-1}[f_a] \cdot \mathrm{l}[f_a]$.

proof −
  have $g \star ?\gamma' = (g \star f_a \star \varepsilon) \cdot (g \star \mathrm{a}[f_a, f, f_a]) \cdot (g \star \eta \star f_a)$
    using *1 2* ‹*ide* f_a› *ε η whisker-left*
    by (*metis* ‹«$?\gamma' : ?w \Rightarrow ?w'$»› *arrI ide-leg1 seqE*)
  also have ... $= (g \star f_a \star \varepsilon) \cdot (g \star \mathrm{a}[f_a, f, f_a]) \cdot (g \star \eta \star f_a) \cdot$
      $\mathrm{a}[g, src\ f, f_a] \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *1 2* ‹*ide* f_a› *η comp-arr-dom hseq-char comp-assoc-assoc′*
    by *simp*
  also have ... $= (g \star f_a \star \varepsilon) \cdot (g \star \mathrm{a}[f_a, f, f_a]) \cdot ((g \star \eta \star f_a) \cdot$
      $\mathrm{a}[g, src\ f, f_a]) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *comp-assoc* **by** *simp*
  also have ... $= (g \star f_a \star \varepsilon) \cdot (g \star \mathrm{a}[f_a, f, f_a]) \cdot$
      $(\mathrm{a}[g, f_a \star f, f_a] \cdot ((g \star \eta) \star f_a)) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *1 2* ‹*ide* f_a› *ε η assoc-naturality* [*of g η* f_a] **by** *simp*
  also have ... $= (g \star f_a \star \varepsilon) \cdot (g \star \mathrm{a}[f_a, f, f_a]) \cdot \mathrm{a}[g, f_a \star f, f_a] \cdot$
      $(\mathrm{a}[g, f_a, f] \cdot (\nu \star f) \cdot \varrho \cdot \mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using η' *comp-assoc* **by** *simp*
  also have ... $= (g \star f_a \star \varepsilon) \cdot$
      $((g \star \mathrm{a}[f_a, f, f_a]) \cdot \mathrm{a}[g, f_a \star f, f_a] \cdot (\mathrm{a}[g, f_a, f] \star f_a)) \cdot$
      $((\nu \star f) \star f_a) \cdot (\varrho \star f_a) \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
  proof −
    have $\mathrm{a}[g, f_a, f] \cdot (\nu \star f) \cdot \varrho \cdot \mathrm{r}[g] \star f_a =$
        $(\mathrm{a}[g, f_a, f] \star f_a) \cdot ((\nu \star f) \star f_a) \cdot (\varrho \star f_a) \cdot (\mathrm{r}[g] \star f_a)$
      using *1 2* ‹*ide* f_a› *β ε η whisker-right* **by** (*metis arrI seqE*)
    thus *?thesis*
      using *comp-assoc* **by** *simp*
  qed
  also have ... $= ((g \star f_a \star \varepsilon) \cdot$
      $\mathrm{a}[g, f_a, f \star f_a]) \cdot (\mathrm{a}[g \star f_a, f, f_a] \cdot$
      $((\nu \star f) \star f_a)) \cdot (\varrho \star f_a) \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *1 2* ‹*ide* f_a› *pentagon comp-assoc* **by** *simp*
  also have ... $= (\mathrm{a}[g, f_a, trg\ f] \cdot ((g \star f_a) \star \varepsilon)) \cdot$
      $((\nu \star f \star f_a) \cdot \mathrm{a}[r, f, f_a]) \cdot$
      $(\varrho \star f_a) \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *1 2* ‹*ide* f_a› *assoc-naturality* [*of g* f_a *ε*] *assoc-naturality* [*of ν f* f_a]
    by (*simp add*: *ε ν*)
  also have ... $= \mathrm{a}[g, f_a, trg\ f] \cdot (((g \star f_a) \star \varepsilon) \cdot (\nu \star f \star f_a)) \cdot \mathrm{a}[r, f, f_a] \cdot$
      $(\varrho \star f_a) \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, src\ f, f_a]$
    using *1 2* ‹*ide* f_a› *assoc-naturality* [*of g* f_a *ε*] *assoc-naturality* [*of ν f* f_a]
      comp-assoc

by *simp*
also have ... $= \mathrm{a}[g, f_a, \mathit{trg}\ f] \cdot (\nu \star \mathit{trg}\ f) \cdot$
$\mathit{composite\text{-}cell}\ f_a\ \varepsilon \cdot$
$(\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, \mathit{src}\ f, f_a]$
proof −
have $((g \star f_a) \star \varepsilon) \cdot (\nu \star f \star f_a) = \nu \star \varepsilon$
using *1 2* ‹*ide* f_a› ν ε *interchange* [*of* $g \star f_a$ ν ε $f \star f_a$]
comp-arr-dom comp-cod-arr
by *simp*
also have ... $= (\nu \star \mathit{trg}\ f) \cdot (r \star \varepsilon)$
using ‹*ide* f_a› ν ε *interchange* [*of* ν r *trg* f ε] *comp-arr-dom comp-cod-arr*
by *simp*
finally show *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}[g, f_a, \mathit{trg}\ f] \cdot ((((\nu \star \mathit{trg}\ f) \cdot \mathrm{r}^{-1}[r]) \cdot \mathit{inv}\ \nu) \cdot (\mathrm{r}[g] \star f_a)) \cdot$
$\mathrm{a}^{-1}[g, \mathit{src}\ f, f_a]$
using *ide-base* f_a′ *comp-assoc* f_a *runit′-simps*(*1*) *invert-side-of-triangle*(*2*)
comp-assoc
by *presburger*
also have ... $= \mathrm{a}[g, f_a, \mathit{trg}\ f] \cdot \mathrm{r}^{-1}[g \star f_a] \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, \mathit{src}\ f, f_a]$
proof −
have $((\nu \star \mathit{trg}\ f) \cdot \mathrm{r}^{-1}[r]) \cdot \mathit{inv}\ \nu = \mathrm{r}^{-1}[g \star f_a]$
using *1 2* ‹*ide* f_a› ν *ide-base runit′-naturality* [*of* ν] *comp-arr-dom*
by (*metis* f_a *ide-compE inv-is-inverse inverse-arrowsE comp-assoc*
runit′-simps(*1*) *runit′-simps*(*4*))
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((\mathrm{a}[g, f_a, \mathit{trg}\ f] \cdot \mathrm{a}^{-1}[g, f_a, \mathit{src}\ f_a]) \cdot$
$(g \star \mathrm{r}^{-1}[f_a])) \cdot (\mathrm{r}[g] \star f_a) \cdot \mathrm{a}^{-1}[g, \mathit{src}\ f, f_a]$
using f_a *2 runit-hcomp* ‹*ide* f_a› *comp-assoc* **by** *simp*
also have ... $= (g \star \mathrm{r}^{-1}[f_a]) \cdot (g \star \mathrm{l}[f_a])$
using *1 2 comp-cod-arr* ‹*ide* f_a› *comp-assoc-assoc′ triangle′* **by** *simp*
also have ... $= ?\beta$
using *2* ‹*ide* f_a› *whisker-left* **by** *simp*
finally show *?thesis* **by** *simp*
qed
show $?\vartheta = ?\vartheta' \cdot (f \star ?\gamma')$
proof −
have $((\varepsilon \star \mathit{trg}\ f) \cdot \mathrm{a}^{-1}[f, f_a, \mathit{trg}\ f]) \cdot (f \star (f_a \star \varepsilon) \cdot \mathrm{a}[f_a, f, f_a] \cdot (\eta \star f_a)) =$
$((\varepsilon \star \mathit{trg}\ f) \cdot \mathrm{a}^{-1}[f, f_a, \mathit{trg}\ f]) \cdot (f \star f_a \star \varepsilon) \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot (f \star \eta \star f_a)$
using *1 2* ‹*ide* f_a› ε η *whisker-left*
by (*metis* ‹«$(f_a \star \varepsilon) \cdot \mathrm{a}[f_a, f, f_a] \cdot (\eta \star f_a)$: *src* $f \star f_a \Rightarrow f_a \star \mathit{trg}\ f$»›
arrI ide-leg0 seqE)
also have
... $= (\varepsilon \star \mathit{trg}\ f) \cdot (\mathrm{a}^{-1}[f, f_a, \mathit{trg}\ f] \cdot (f \star f_a \star \varepsilon)) \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot (f \star \eta \star f_a)$
using *comp-assoc* **by** *simp*
also have ... $= ((\varepsilon \star \mathit{trg}\ f) \cdot ((f \star f_a) \star \varepsilon)) \cdot$

$\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) \cdot$
$(f \star \eta \star f_a)$
using *1 2* ‹*ide* f_a› ε *assoc′-naturality* [*of f* f_a ε] *comp-assoc* **by** *simp*
also have ... $= (\mathit{trg}\ f \star \varepsilon) \cdot (\varepsilon \star f \star f_a) \cdot$
$(\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a])) \cdot$
$(f \star \eta \star f_a)$
using *1 2* ‹*ide* f_a› ε *interchange* [*of* ε $f \star f_a$ *trg f* ε]
interchange [*of trg f* ε ε $f \star f_a$] *comp-arr-dom comp-cod-arr comp-assoc*
by *simp*
also have ... $= (\mathit{trg}\ f \star \varepsilon) \cdot ((\varepsilon \star f \star f_a) \cdot$
$(\mathrm{a}[f \star f_a, f, f_a]) \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot (\mathrm{a}^{-1}[f, f_a \star f, f_a]) \cdot$
$(f \star \eta \star f_a))$
proof −
have $\mathrm{a}^{-1}[f, f_a, f \star f_a] \cdot (f \star \mathrm{a}[f_a, f, f_a]) =$
$\mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a]$
proof −
have *A*: $(\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a] \cdot (f \star \mathrm{a}^{-1}[f_a, f, f_a]) =$
$\mathrm{a}^{-1}[f \star f_a, f, f_a] \cdot \mathrm{a}^{-1}[f, f_a, f \star f_a]$
using *1 2* ‹*ide* f_a› *pentagon′ comp-assoc* **by** *fastforce*
hence *B*: $\mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a] \cdot$
$(f \star \mathrm{a}^{-1}[f_a, f, f_a]) =$
$\mathrm{a}^{-1}[f, f_a, f \star f_a]$
using *A 1 2* ‹*ide* f_a›
invert-side-of-triangle(*1*)
[*of* $(\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a] \cdot (f \star \mathrm{a}^{-1}[f_a, f, f_a])$
$\mathrm{a}^{-1}[f \star f_a, f, f_a]$ $\mathrm{a}^{-1}[f, f_a, f \star f_a]$]
by *auto*
show *?thesis*
proof −
have *C*: *iso* $(f \star \mathrm{a}^{-1}[f_a, f, f_a])$
using *1 2* ‹*ide* f_a› **by** *simp*
moreover have *inv* $(f \star \mathrm{a}^{-1}[f_a, f, f_a]) = f \star \mathrm{a}[f_a, f, f_a]$
using *C 1 2* ‹*ide* f_a› **by** *fastforce*
ultimately show *?thesis*
using *B 1 2* ‹*ide* f_a› *comp-assoc*
invert-side-of-triangle(*2*)
[*of* $\mathrm{a}^{-1}[f, f_a, f \star f_a]$
$\mathrm{a}[f \star f_a, f, f_a] \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot \mathrm{a}^{-1}[f, f_a \star f, f_a]$
$f \star \mathrm{a}^{-1}[f_a, f, f_a]$]
by *simp*
qed
qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (\mathit{trg}\ f \star \varepsilon) \cdot (\mathrm{a}[\mathit{trg}\ f, f, f_a] \cdot$
$((\varepsilon \star f) \star f_a)) \cdot (\mathrm{a}^{-1}[f, f_a, f] \star f_a) \cdot ((f \star \eta) \star f_a) \cdot$
$\mathrm{a}^{-1}[f, \mathit{src}\ f, f_a]$
using *1 2* ‹*ide* f_a› ‹*ide f*› η ε *assoc-naturality* [*of* ε f f_a]

```
            assoc′-naturality [of f η f_a] comp-assoc
          by simp
        also have ... = (trg f ⋆ ε) · a[trg f, f, f_a] ·
                        (((ε ⋆ f) ⋆ f_a) · (a⁻¹[f, f_a, f] ⋆ f_a) · ((f ⋆ η) ⋆ f_a)) ·
                        a⁻¹[f, src f, f_a]
          using comp-assoc by simp
        also have ... = (trg f ⋆ ε) · a[trg f, f, f_a] ·
                        ((ε ⋆ f) · a⁻¹[f, f_a, f] · (f ⋆ η) ⋆ f_a) ·
                        a⁻¹[f, src f, f_a]
          using 1 2 ‹ide f_a› ‹ide f› η ε whisker-right
          by (metis (full-types) ∗ ϑ ϑ-eq′ arrI hseqE seqE)
        also have ... = (trg f ⋆ ε) · a[trg f, f, f_a] · (l⁻¹[f] · r[f] ⋆ f_a) · a⁻¹[f, src f, f_a]
          using ∗ by simp
        also have ... = ?ϑ
          using ϑ-eq′ by simp
        finally show ?thesis by simp
      qed
    qed
    ultimately show ?γ′ = ?γ by blast
  qed
qed
thus ?thesis
  using adjoint-pair-def by auto
qed

sublocale tabulation-data-with-T0
  using satisfies-T0 by (unfold-locales, simp)
sublocale narrow-tabulation
  using adjoint-pair-antipar(1) T1 T2
  by (unfold-locales, auto)

end
```

A tabulation (f, ϱ, g) of r yields an isomorphism «$\psi : g \star f^* \Rightarrow r$» via adjoint transpose. The proof requires *T0*, in order to obtain ψ as the transpose of «$\varrho : g \Rightarrow r \star f$». However, it uses only the weaker versions of *T1* and *T2*.

```
context narrow-tabulation
begin

  interpretation E: self-evaluation-map V H a i src trg ..
  notation E.eval (‹⦃-⦄›)
```

The following is CKS Proposition 1(d), with the statement refined to incorporate the canonical isomorphisms that they omit. Note that we can easily show using *T1* that there is some 1-cell f_a and isomorphism ψ such that «$\psi : f \star f_a \Rightarrow r$» (this was already part of the proof that a tabulation satisfies *T0*). The more difficult content in the present result is that we may actually take f_a to be the left adjoint f^* of f.

```
  lemma yields-isomorphic-representation:
  shows «T0.trnr_ε r ϱ : g ⋆ f* ⇒ r» and iso (T0.trnr_ε r ϱ)
```

proof –

As stated in CKS, the first step of the proof is:

> "Apply *T1* with $X = A$, $u = 1_A$, $v = r$, $\omega = 1_R$, to obtain f', ϑ': $ff' \Rightarrow 1_A$, $\nu : r \simeq g\, f'$ with $1_R = (r\vartheta')(\varrho f')\nu$."

In our nomenclature: $X = trg\, f$, $u = trg\, f$, $v = r$, but $\omega = src\, f$ does not make any sense, since we need «$\omega : v \Rightarrow r \star u$». We have to take $\omega = \mathrm{r}^{-1}[r]$. It is not clear whether this is a typo, or whether it is a consequence of CKS having suppressed all canonical isomorphisms (unitors, in this case). The resulting equation obtained via T1 is:

$$\mathrm{r}^{-1}[r] = (r \star \vartheta') \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu,$$

which has $\mathrm{r}^{-1}[r]$ on the left-hand side, rather than 1_R, as in CKS. Also, we have inserted the omitted associativity.

```
obtain w ϑ' ν where wϑ'ν: ide w ∧ «ϑ' : f ⋆ w ⇒ src r» ∧ «ν : r ⇒ g ⋆ w» ∧ iso ν ∧
                        composite-cell w ϑ' · ν = r⁻¹[r]
  using ide-base obj-is-self-adjoint T1 [of src r r⁻¹[r]] comp-assoc by auto

interpret uwϑων V H a i src trg r ϱ f g ‹src r› w ϑ' ‹r⁻¹[r]› ν
  using ide-base tab-in-hom wϑ'ν comp-assoc by (unfold-locales, auto)
```

CKS now say:

> "Apply *T2* with $u = 1_A$, $w = f^*$, $w' = f'$, $\vartheta = \varepsilon$: $ff^* \Rightarrow 1$, ϑ': $ff' \Rightarrow 1$, $\beta = \nu(r\varepsilon)(\varrho f^*)$ to obtain $\gamma : f^* \Rightarrow f'$ with $g\gamma = \nu(r\varepsilon)(\varrho f^*)\varepsilon = \vartheta'(f\gamma)$."

The last equation is mysterious, but upon consideration one eventually realizes that it is definitely a typo, and what is meant is "$g\gamma = \nu(r\varepsilon)(\varrho f^*)$, $\varepsilon = \vartheta'(f\gamma)$".

So, we take $u = trg\, f$, $w = f^*$, $w' = w$, ϑ' as obtained from *T1*, $\vartheta = \varepsilon$, and $\beta = \nu \cdot \mathrm{r}[r] \cdot (r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot (\varrho \star f^*)$. (CKS mention neither the unitor term $\mathrm{r}[r]$ nor the associativity $\mathrm{a}[r, f, f^*]$ which are required for the expression for β to make sense.)

```
let ?ψ = r[r] · composite-cell f* ε
show ψ-in-hom: « T0.trnr_ε r ϱ : g ⋆ f* ⇒ r»
  using ide-base T0.trnr_ε-def rep-in-hom by simp
have A: «ν · ?ψ : g ⋆ f* ⇒ g ⋆ w»
  using ide-base T0.antipar hseq-char T0.trnr_ε-def rep-in-hom wϑ'ν
  apply (intro comp-in-homI') by auto
have B: composite-cell f* ε = composite-cell w ϑ' · ν · ?ψ
    using ide-base T0.antipar wϑ'ν comp-assoc
    by (metis A arrI invert-side-of-triangle(1) iso-runit)

obtain γ where γ: «γ : f* ⇒ w» ∧ ν · ?ψ = g ⋆ γ ∧ ε = ϑ' · (f ⋆ γ)
  using A B T0.counit-in-hom obj-is-self-adjoint T0.antipar comp-assoc
        T2 [of trg f f* w ε ϑ' ν · r[r] · composite-cell f* ε]
  by auto
have trg-γ-eq: trg γ = trg w
  using γ by fastforce
```

CKS say:

"The last equation implies $\gamma\colon f^* \Rightarrow f'$ is a split monic (coretraction), while the calculation:

$$\begin{aligned}
(g\gamma)(gf^*\vartheta')(g\eta f') &= \nu(r\varepsilon)(\varrho f^*)(gf^*\vartheta')(g\eta f') \\
&= \nu(r\varepsilon)(rff^*\vartheta')(\varrho f^* ff')(g\eta f') \\
&= \nu(r\vartheta')(r\varepsilon ff')(rf\eta f')(\varrho f') \\
&= \nu(r\vartheta')(\varrho f') = 1_{gf'},
\end{aligned}$$

shows that $g\gamma$ is a split epic. So $g\gamma = \nu(r\varepsilon)(\varrho f^*)\colon gf^* \Rightarrow gf'$ is invertible. So $(r\varepsilon)(\varrho f^*) = \nu^{-1}(g\gamma)$ is invertible."

We carry out the indicated calculations, inserting where required the canonical isomorphisms omitted by CKS. It is perhaps amusing to compare the four-line sketch given by CKS with the formalization below, but note that we have carried out the proof in full, with no hand waving about units or associativities.

```
have section (g ⋆ γ)
proof
  have (g ⋆ r[f*] · (f* ⋆ ϑ') · a[f*, f, w] · (η ⋆ w) · l⁻¹[w]) · (g ⋆ γ) = g ⋆ f*
  proof −
    have (r[f*] · (f* ⋆ ϑ') · a[f*, f, w] · (η ⋆ w) · l⁻¹[w]) · γ = f*
    proof −
      have (r[f*] · (f* ⋆ ϑ') · a[f*, f, w] · (η ⋆ w) · l⁻¹[w]) · γ =
            (r[f*] · (f* ⋆ ϑ') · a[f*, f, w] · (η ⋆ w)) · l⁻¹[w] · γ
        using comp-assoc by auto
      also have ... = (r[f*] · (f* ⋆ ϑ') · a[f*, f, w]) · ((η ⋆ w) · (trg w ⋆ γ)) · l⁻¹[f*]
        using γ trg-γ-eq lunit'-naturality [of γ] comp-assoc by auto
      also have ... = r[f*] · (f* ⋆ ϑ') · (a[f*, f, w] · ((f* ⋆ f) ⋆ γ)) · (η ⋆ f*) · l⁻¹[f*]
      proof −
        have (η ⋆ w) · (trg w ⋆ γ) = η ⋆ γ
          using A γ interchange comp-arr-dom comp-cod-arr
          by (metis T0.unit-simps(1−2) comp-ide-arr seqI' uwϑ w-in-hom(2) w-simps(4))
        also have ... = ((f* ⋆ f) ⋆ γ) · (η ⋆ f*)
          using γ interchange comp-arr-dom comp-cod-arr T0.antipar T0.unit-simps(1,3)
                in-homE
          by metis
        finally show ?thesis
          using comp-assoc by simp
      qed
      also have ... = r[f*] · (f* ⋆ ϑ') · ((f* ⋆ f ⋆ γ) · a[f*, f, f*]) · (η ⋆ f*) · l⁻¹[f*]
        using γ assoc-naturality [of f* f γ] trg-γ-eq T0.antipar by auto
      also have ... = r[f*] · ((f* ⋆ ε) · a[f*, f, f*] · (η ⋆ f*)) · l⁻¹[f*]
        using γ whisker-left trg-γ-eq T0.antipar comp-assoc by auto
      also have ... =  r[f*] · (r⁻¹[f*] · l[f*]) · l⁻¹[f*]
        using T0.triangle-right by simp
      also have ... = f*
        using comp-assoc by (simp add: comp-arr-dom comp-arr-inv')
      finally show ?thesis by blast
    qed
```

thus *?thesis*
using γ *whisker-left* [*of* g $\mathrm{r}[f^*] \cdot (f^* \star \vartheta') \cdot \mathrm{a}[f^*, f, w] \cdot (\eta \star w) \cdot \mathrm{l}^{-1}[w]$ γ]
T0.antipar
by *simp*
qed
thus *ide* $((g \star \mathrm{r}[f^*] \cdot (f^* \star \vartheta') \cdot \mathrm{a}[f^*, f, w] \cdot (\eta \star w) \cdot \mathrm{l}^{-1}[w]) \cdot (g \star \gamma))$
using *T0.antipar* **by** *simp*
qed
moreover have *retraction* $(g \star \gamma)$
proof
have «$(g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w]) :$
$g \star w \Rightarrow g \star w$»
using γ *T0.antipar hseq-char* **by** *force*
hence $**$: *arr* $((g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w]))$
by *auto*
show *ide* $((g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w]))$
proof −
have $((g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w])) =$
$g \star w$
proof −
have $((g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w])) =$
$\nu \cdot \mathrm{r}[r] \cdot ((r \star \mathrm{r}[src\ f^*]) \cdot (r \star src\ f^* \star \vartheta') \cdot (r \star \mathrm{a}[src\ f^*, f, w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f, trg\ w, w]) \cdot \mathrm{a}[r, f, trg\ w \star w] \cdot$
$(\varrho \star trg\ w \star w)) \cdot (g \star \mathrm{l}^{-1}[w])$
proof −
have $(g \star \gamma) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w]) =$
$(\nu \cdot \mathrm{r}[r] \cdot (r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot (\varrho \star f^*)) \cdot$
$(g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot$
$(g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w])$
using γ **by** *auto*
also have ... =
$\nu \cdot \mathrm{r}[r] \cdot (r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot$
$((\varrho \star f^*) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta')) \cdot$
$(g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w])$
using *comp-assoc* **by** *simp*
also have ... $= \nu \cdot \mathrm{r}[r] \cdot (r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot$
$(((r \star f) \star \mathrm{r}[f^*]) \cdot ((r \star f) \star f^* \star \vartheta') \cdot (\varrho \star f^* \star f \star w)) \cdot$
$(g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w) \cdot (g \star \mathrm{l}^{-1}[w])$
proof −
have $(\varrho \star f^*) \cdot (g \star \mathrm{r}[f^*]) \cdot (g \star f^* \star \vartheta') =$
$((r \star f) \star \mathrm{r}[f^*]) \cdot (\varrho \star f^* \star src\ f^*) \cdot (g \star f^* \star \vartheta')$
proof −
have $(\varrho \star f^*) \cdot (g \star \mathrm{r}[f^*]) = ((r \star f) \star \mathrm{r}[f^*]) \cdot (\varrho \star f^* \star src\ f^*)$
using *tab-in-hom comp-arr-dom comp-cod-arr T0.antipar*(*1*) *interchange*

by (*metis T0.ide-right in-homE runit-simps*(*1*,*4*−*5*))
thus *?thesis*
by (*metis comp-assoc*)
qed
also have ... $= ((r \star f) \star \mathrm{r}[f^*]) \cdot (\varrho \star f^* \star \vartheta')$
using *comp-arr-dom comp-cod-arr T0.antipar*
interchange [*of* ϱ g f^* $\star$ *src* f^* f^* $\star$ ϑ']
by *simp*
also have ... $= ((r \star f) \star \mathrm{r}[f^*]) \cdot ((r \star f) \star f^* \star \vartheta') \cdot (\varrho \star f^* \star f \star w)$
using *comp-arr-dom comp-cod-arr T0.antipar*
interchange [*of* $r \star f$ ϱ f^* $\star$ ϑ' f^* $\star$ f $\star$ w]
by *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $=$
$\nu \cdot \mathrm{r}[r] \cdot$
$((r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot ((r \star f) \star \mathrm{r}[f^*]) \cdot ((r \star f) \star f^* \star \vartheta')) \cdot$
$((\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w)) \cdot$
$(g \star \mathrm{l}^{-1}[w])$
using *comp-assoc* **by** *simp*
also have ... $= \nu \cdot \mathrm{r}[r] \cdot$
$((r \star \mathrm{r}[\mathit{src}\ f^*]) \cdot (r \star \mathit{src}\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]) \cdot$
$(((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot ((r \star f) \star \eta \star w) \cdot (\varrho \star \mathit{trg}\ w \star w)) \cdot$
$(g \star \mathrm{l}^{-1}[w])$
proof −
have *1*: $(r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot ((r \star f) \star \mathrm{r}[f^*]) \cdot ((r \star f) \star f^* \star \vartheta') =$
$(r \star \mathrm{r}[\mathit{src}\ f^*]) \cdot (r \star \mathit{src}\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]$
proof −
have $(r \star \varepsilon) \cdot \mathrm{a}[r, f, f^*] \cdot ((r \star f) \star \mathrm{r}[f^*]) \cdot ((r \star f) \star f^* \star \vartheta') =$
$(r \star \varepsilon) \cdot (r \star f \star \mathrm{r}[f^*]) \cdot \mathrm{a}[r, f, f^* \star \mathit{src}\ f^*] \cdot ((r \star f) \star f^* \star \vartheta')$
proof −
have $\mathrm{a}[r, f, f^*] \cdot ((r \star f) \star \mathrm{r}[f^*]) = (r \star f \star \mathrm{r}[f^*]) \cdot \mathrm{a}[r, f, f^* \star \mathit{src}\ f^*]$
using *assoc-naturality* [*of* r f $\mathrm{r}[f^*]$] *T0.antipar* **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
also have ... $= (r \star \varepsilon) \cdot (r \star f \star \mathrm{r}[f^*]) \cdot (r \star f \star f^* \star \vartheta') \cdot$
$\mathrm{a}[r, f, f^* \star f \star w]$
using *assoc-naturality* [*of* r f f^* $\star$ ϑ'] *T0.antipar* **by** *fastforce*
also have ... $= (r \star \mathrm{r}[\mathit{src}\ f^*]) \cdot (r \star \varepsilon \star \mathit{src}\ f^*) \cdot (r \star \mathrm{a}^{-1}[f, f^*, \mathit{src}\ f^*]) \cdot$
$(r \star f \star f^* \star \vartheta') \cdot \mathrm{a}[r, f, f^* \star f \star w]$
proof −
have $(r \star \varepsilon) \cdot (r \star f \star \mathrm{r}[f^*]) =$
$(r \star \mathrm{r}[\mathit{src}\ f^*]) \cdot (r \star \varepsilon \star \mathit{src}\ f^*) \cdot (r \star \mathrm{a}^{-1}[f, f^*, \mathit{src}\ f^*])$
proof −
have $(r \star \varepsilon) \cdot (r \star f \star \mathrm{r}[f^*]) = r \star (\varepsilon \cdot (f \star \mathrm{r}[f^*]))$
using *whisker-left T0.antipar* **by** *simp*

also have ... =
$(r \star \mathrm{r}[src\ f^*]) \cdot (r \star \varepsilon \star src\ f^*) \cdot (r \star \mathrm{a}^{-1}[f, f^*, src\ f^*])$
proof −
have $\varepsilon \cdot (f \star \mathrm{r}[f^*]) = \mathrm{r}[src\ f^*] \cdot (\varepsilon \star src\ f^*) \cdot \mathrm{a}^{-1}[f, f^*, src\ f^*]$
using *ide-leg0 T0.antipar runit-hcomp invert-side-of-triangle*(*2*)
runit-naturality comp-assoc
by (*metis* (*no-types*, *lifting*) *T0.counit-simps*(*1*−*4*) *T0.ide-right*)
thus *?thesis*
using *whisker-left T0.antipar* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **using** *comp-assoc* **by** *metis*
qed
also have ... =
$(r \star \mathrm{r}[src\ f^*]) \cdot (r \star \varepsilon \star src\ f^*) \cdot$
$((r \star \mathrm{a}^{-1}[f, f^*, src\ f^*]) \cdot (r \star f \star f^* \star \vartheta')) \cdot$
$\mathrm{a}[r, f, f^* \star f \star w]$
using *comp-assoc* **by** *simp*
also have ... = $(r \star \mathrm{r}[src\ f^*]) \cdot (r \star \varepsilon \star src\ f^*) \cdot$
$((r \star (f \star f^*) \star \vartheta') \cdot (r \star \mathrm{a}^{-1}[f, f^*, f \star w])) \cdot$
$\mathrm{a}[r, f, f^* \star f \star w]$
proof −
have $(r \star \mathrm{a}^{-1}[f, f^*, src\ f^*]) \cdot (r \star f \star f^* \star \vartheta') =$
$(r \star (f \star f^*) \star \vartheta') \cdot (r \star \mathrm{a}^{-1}[f, f^*, f \star w])$
proof −
have $(r \star \mathrm{a}^{-1}[f, f^*, src\ f^*]) \cdot (r \star f \star f^* \star \vartheta') =$
$r \star \mathrm{a}^{-1}[f, f^*, src\ f^*] \cdot (f \star f^* \star \vartheta')$
using *whisker-left T0.antipar* **by** *simp*
also have ... = $r \star ((f \star f^*) \star \vartheta') \cdot \mathrm{a}^{-1}[f, f^*, f \star w]$
using *assoc′-naturality* [*of f f* ϑ′*] *T0.antipar* **by** *auto*
also have ... = $(r \star (f \star f^*) \star \vartheta') \cdot (r \star \mathrm{a}^{-1}[f, f^*, f \star w])$
using *whisker-left T0.antipar* **by** *auto*
finally show *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = $(r \star \mathrm{r}[src\ f^*]) \cdot (r \star \varepsilon \star src\ f^*) \cdot (r \star (f \star f^*) \star \vartheta') \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]$
using *comp-assoc* **by** *simp*
also have ... =
$(r \star \mathrm{r}[src\ f^*]) \cdot ((r \star \varepsilon \star src\ f^*) \cdot (r \star (f \star f^*) \star \vartheta')) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]$
using *comp-assoc* **by** *simp*
also have ... = $(r \star \mathrm{r}[src\ f^*]) \cdot ((r \star src\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w)) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]$
proof −
have $(r \star \varepsilon \star src\ f^*) \cdot (r \star (f \star f^*) \star \vartheta') =$
$(r \star src\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w)$

proof −
have $(r \star \varepsilon \star src\ f^*) \cdot (r \star (f \star f^*) \star \vartheta') =$
$r \star (\varepsilon \star src\ f^*) \cdot ((f \star f^*) \star \vartheta')$
using *whisker-left T0.antipar* **by** *simp*
also have ... $= r \star \varepsilon \star \vartheta'$
using *interchange* [*of* $\varepsilon\ f \star f^*\ src\ f^*\ \vartheta'$]
T0.antipar comp-arr-dom comp-cod-arr
by *auto*
also have ... $= r \star (src\ f^* \star \vartheta') \cdot (\varepsilon \star f \star w)$
using *interchange* [*of* $src\ f^*\ \varepsilon\ \vartheta'\ f \star w$]
T0.antipar comp-arr-dom comp-cod-arr
by *auto*
also have ... $= (r \star src\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w)$
using *whisker-left T0.antipar* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (r \star \mathrm{r}[src\ f^*]) \cdot (r \star src\ f^* \star \vartheta') \cdot (r \star \varepsilon \star f \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f, f^*, f \star w]) \cdot \mathrm{a}[r, f, f^* \star f \star w]$
using *comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
have *2*: $(\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w) =$
$((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot ((r \star f) \star \eta \star w) \cdot (\varrho \star trg\ w \star w)$
proof −
have $(\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w]) \cdot (g \star \eta \star w) =$
$((\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w])) \cdot (g \star \eta \star w)$
using *comp-assoc* **by** *simp*
also have ... $= (((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot (\varrho \star (f^* \star f) \star w)) \cdot (g \star \eta \star w)$
proof −
have $(\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w]) =$
$((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot (\varrho \star (f^* \star f) \star w)$
proof −
have $(\varrho \star f^* \star f \star w) \cdot (g \star \mathrm{a}[f^*, f, w]) =$
$\varrho \cdot g \star (f^* \star f \star w) \cdot \mathrm{a}[f^*, f, w]$
using *interchange T0.antipar* **by** *auto*
also have ... $= \varrho \star \mathrm{a}[f^*, f, w]$
using *comp-arr-dom comp-cod-arr T0.antipar* **by** *auto*
also have ... $= (r \star f) \cdot \varrho \star \mathrm{a}[f^*, f, w] \cdot ((f^* \star f) \star w)$
using *comp-arr-dom comp-cod-arr T0.antipar* **by** *auto*
also have ... $= ((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot (\varrho \star (f^* \star f) \star w)$
using *interchange T0.antipar* **by** *auto*
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= ((r \star f) \star \mathrm{a}[f^*, f, w]) \cdot (\varrho \star (f^* \star f) \star w) \cdot (g \star \eta \star w)$
using *comp-assoc* **by** *simp*

```
    also have ... = ((r ⋆ f) ⋆ a[f*, f, w]) · ((r ⋆ f) ⋆ η ⋆ w) · (ϱ ⋆ trg w ⋆ w)
    proof −
      have (ϱ ⋆ (f* ⋆ f) ⋆ w) · (g ⋆ η ⋆ w) = ((r ⋆ f) ⋆ η ⋆ w) · (ϱ ⋆ trg w ⋆ w)
      proof −
        have (ϱ ⋆ (f* ⋆ f) ⋆ w) · (g ⋆ η ⋆ w) = ϱ · g ⋆ (f* ⋆ f) · η ⋆ w · w
        proof −
          have «g ⋆ η ⋆ w : g ⋆ trg w ⋆ w ⇒ g ⋆ (f* ⋆ f) ⋆ w»
            by (intro hcomp-in-vhom, auto)
          thus ?thesis
            using interchange whisker-right T0.antipar by auto
        qed
        also have ... = (r ⋆ f) · ϱ ⋆ η · trg w ⋆ w · w
          using comp-arr-dom comp-cod-arr by auto
        also have ... = ((r ⋆ f) ⋆ η ⋆ w) · (ϱ ⋆ trg w ⋆ w)
          using interchange [of r ⋆ f ϱ η ⋆ w trg w ⋆ w]
                interchange [of η trg w w w]
                comp-arr-dom comp-cod-arr T0.unit-in-hom
          by auto
        finally show ?thesis by simp
      qed
      thus ?thesis by simp
    qed
    finally show ?thesis by simp
  qed
  show ?thesis
    using 1 2 by simp
qed
also have ... =
          ν · r[r] ·
            ((r ⋆ r[src r]) · (r ⋆ src r ⋆ ϑ′) ·
              ((r ⋆ a[src r, f, w]) · (r ⋆ (ε ⋆ f) ⋆ w) · (r ⋆ a⁻¹[f ⋆ f*, f, w])) ·
             (r ⋆ a⁻¹[f, f*, f ⋆ w]) · a[r, f, f* ⋆ f ⋆ w]) ·
            (((r ⋆ f) ⋆ a[f*, f, w]) ·
              (a⁻¹[r, f, (f* ⋆ f) ⋆ w] · (r ⋆ a[f, f* ⋆ f, w]) ·
                (r ⋆ (f ⋆ η) ⋆ w) ·
                (r ⋆ a⁻¹[f, trg w, w]) · a[r, f, trg w ⋆ w]) ·
             (ϱ ⋆ trg w ⋆ w)) · (g ⋆ l⁻¹[w])
proof −
  have 3: r ⋆ ε ⋆ f ⋆ w =
          (r ⋆ a[src r, f, w]) · (r ⋆ (ε ⋆ f) ⋆ w) · (r ⋆ a⁻¹[f ⋆ f*, f, w])
  proof −
    have r ⋆ ε ⋆ f ⋆ w =
          ((r ⋆ a[src r, f, w]) · (r ⋆ a⁻¹[src r, f, w])) · (r ⋆ ε ⋆ f ⋆ w)
      using T0.antipar whisker-left [of r a[src r, f, w] a⁻¹[src r, f, w]]
            comp-cod-arr comp-assoc-assoc′
      by simp
   also have ... = (r ⋆ a[src r, f, w]) · (r ⋆ (ε ⋆ f) ⋆ w) ·
                   (r ⋆ a⁻¹[f ⋆ f*, f, w])
      using assoc′-naturality [of ε f w]
```

whisker-left [*of r* a^{-1}[*src r*, *f*, *w*] $\varepsilon \star f \star w$]
whisker-left comp-assoc T0.antipar
by *simp*
finally show *?thesis*
using *T0.antipar* **by** *simp*
qed
have *4*: $(r \star f) \star \eta \star w =$
$\mathrm{a}^{-1}[r, f, (f^* \star f) \star w] \cdot (r \star \mathrm{a}[f, f^* \star f, w]) \cdot$
$(r \star (f \star \eta) \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f, \mathit{trg}\ w, w]) \cdot \mathrm{a}[r, f, \mathit{trg}\ w \star w]$
proof −
have $(r \star f) \star \eta \star w =$
$(\mathrm{a}^{-1}[r, f, (f^* \star f) \star w] \cdot$
$((r \star \mathrm{a}[f, f^* \star f, w]) \cdot (r \star \mathrm{a}^{-1}[f, f^* \star f, w])) \cdot$
$\mathrm{a}[r, f, (f^* \star f) \star w]) \cdot$
$((r \star f) \star \eta \star w)$
proof −
have *ide r* **by** *simp*
moreover have *seq* $\mathrm{a}[f, f^* \star f, w]$ $\mathrm{a}^{-1}[f, f^* \star f, w]$
using *T0.antipar comp-cod-arr ide-base* **by** *simp*
ultimately have $(r \star \mathrm{a}[f, f^* \star f, w]) \cdot (r \star \mathrm{a}^{-1}[f, f^* \star f, w]) =$
$r \star \mathrm{a}[f, f^* \star f, w] \cdot \mathrm{a}^{-1}[f, f^* \star f, w]$
using *whisker-left* **by** *metis*
thus *?thesis*
using *T0.antipar comp-cod-arr comp-assoc-assoc′* **by** *simp*
qed
also have ... =
$\mathrm{a}^{-1}[r, f, (f^* \star f) \star w] \cdot$
$(r \star \mathrm{a}[f, f^* \star f, w]) \cdot ((r \star \mathrm{a}^{-1}[f, f^* \star f, w]) \cdot$
$(r \star f \star \eta \star w)) \cdot$
$\mathrm{a}[r, f, \mathit{trg}\ w \star w]$
using *assoc-naturality* [*of r f* $\eta \star w$] *comp-assoc* **by** *fastforce*
also have ... =
$\mathrm{a}^{-1}[r, f, (f^* \star f) \star w] \cdot$
$(r \star \mathrm{a}[f, f^* \star f, w]) \cdot (r \star (f \star \eta) \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f, \mathit{trg}\ w, w]) \cdot$
$\mathrm{a}[r, f, \mathit{trg}\ w \star w]$
using *assoc′-naturality* [*of f* η *w*] *T0.antipar comp-assoc*
whisker-left [*of r* $\mathrm{a}^{-1}[f, f^* \star f, w]$ $f \star \eta \star w$]
whisker-left [*of r* $(f \star \eta) \star w$ $\mathrm{a}^{-1}[f, \mathit{trg}\ w, w]$]
by *simp*
finally show *?thesis* **by** *blast*
qed
show *?thesis*
using *3 4 T0.antipar* **by** *simp*
qed
also have ... $= \nu \cdot \mathrm{r}[r] \cdot ((r \star \mathrm{r}[\mathit{src}\ r]) \cdot (r \star \mathit{src}\ r \star \vartheta') \cdot$
$(r \star \mathrm{a}[\mathit{src}\ r, f, w]) \cdot$
$((r \star (\varepsilon \star f) \star w) \cdot$

$((r \star \mathrm{a}^{-1}[f \star f^*,\ f,\ w]) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f \star w]) \cdot$
$\mathrm{a}[r,\ f,\ f^* \star f \star w] \cdot ((r \star f) \star \mathrm{a}[f^*,\ f,\ w]) \cdot$
$\mathrm{a}^{-1}[r,\ f,\ (f^* \star f) \star w] \cdot (r \star \mathrm{a}[f,\ f^* \star f,\ w])) \cdot$
$(r \star (f \star \eta) \star w)) \cdot$
$(r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot \mathrm{a}[r,\ f,\ trg\ w \star w] \cdot$
$(\varrho \star trg\ w \star w)) \cdot (g \star \mathrm{l}^{-1}[w])$
using *comp-assoc T0.antipar* **by** *auto*
also have ... $= \nu \cdot \mathrm{r}[r] \cdot ((r \star \mathrm{r}[src\ r]) \cdot (r \star src\ r \star \vartheta') \cdot$
$(r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$((r \star (\varepsilon \star f) \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f] \star w) \cdot$
$(r \star (f \star \eta) \star w)) \cdot$
$(r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot \mathrm{a}[r,\ f,\ trg\ w \star w] \cdot$
$(\varrho \star trg\ w \star w)) \cdot (g \star \mathrm{l}^{-1}[w])$
proof −
have $(r \star \mathrm{a}^{-1}[f \star f^*,\ f,\ w]) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f \star w]) \cdot$
$\mathrm{a}[r,\ f,\ f^* \star f \star w] \cdot ((r \star f) \star \mathrm{a}[f^*,\ f,\ w]) \cdot$
$\mathrm{a}^{-1}[r,\ f,\ (f^* \star f) \star w] \cdot (r \star \mathrm{a}[f,\ f^* \star f,\ w]) =$
$r \star \mathrm{a}^{-1}[f,\ f^*,\ f] \star w$
proof −

We can compress the reasoning about the associativities using coherence.

have $(r \star \mathrm{a}^{-1}[f \star f^*,\ f,\ w]) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f \star w]) \cdot$
$\mathrm{a}[r,\ f,\ f^* \star f \star w] \cdot ((r \star f) \star \mathrm{a}[f^*,\ f,\ w]) \cdot$
$\mathrm{a}^{-1}[r,\ f,\ (f^* \star f) \star w] \cdot (r \star \mathrm{a}[f,\ f^* \star f,\ w]) =$
$\{\!|(\langle r\rangle \star \mathbf{a}^{-1}[\langle f\rangle \star \langle f^*\rangle,\ \langle f\rangle,\ \langle w\rangle]) \cdot (\langle r\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ \langle f^*\rangle,\ \langle f\rangle \star \langle w\rangle]) \cdot$
$\mathbf{a}[\langle r\rangle,\ \langle f\rangle,\ \langle f^*\rangle \star \langle f\rangle \star \langle w\rangle] \cdot ((\langle r\rangle \star \langle f\rangle) \star \mathbf{a}[\langle f^*\rangle,\ \langle f\rangle,\ \langle w\rangle]) \cdot$
$\mathbf{a}^{-1}[\langle r\rangle,\ \langle f\rangle,\ (\langle f^*\rangle \star \langle f\rangle) \star \langle w\rangle] \cdot (\langle r\rangle \star \mathbf{a}[\langle f\rangle,\ \langle f^*\rangle \star \langle f\rangle,\ \langle w\rangle])|\!\}$
using *T0.antipar* a'*-def* α*-def assoc′-eq-inv-assoc* **by** *auto*
also have ... $= \{\!|\langle r\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ \langle f^*\rangle,\ \langle f\rangle] \star \langle w\rangle|\!\}$
using *T0.antipar* **by** (*intro E.eval-eqI*, *auto*)
also have ... $= r \star \mathrm{a}^{-1}[f,\ f^*,\ f] \star w$
using *T0.antipar* a'*-def* α*-def assoc′-eq-inv-assoc* **by** *simp*
finally show *?thesis*
by *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= \nu \cdot \mathrm{r}[r] \cdot ((r \star \mathrm{r}[src\ r]) \cdot (r \star src\ r \star \vartheta') \cdot$
$(r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot$
$(r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot \mathrm{a}[r,\ f,\ trg\ w \star w] \cdot$
$(\varrho \star trg\ w \star w)) \cdot (g \star \mathrm{l}^{-1}[w])$
proof −
have $(r \star (\varepsilon \star f) \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f] \star w) \cdot (r \star (f \star \eta) \star w) =$
$r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w$
proof −
have $(r \star (\varepsilon \star f) \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ f^*,\ f] \star w) \cdot (r \star (f \star \eta) \star w) =$
$r \star (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ f^*,\ f] \cdot (f \star \eta) \star w$
using *whisker-left whisker-right T0.antipar* **by** *simp*

```
      also have ... = r ⋆ l⁻¹[f] · r[f] ⋆ w
        using T0.triangle-left by simp
      finally show ?thesis by blast
    qed
    thus ?thesis by simp
  qed
  also have ... = ν · r[r] · ((r ⋆ r[src f*]) · (r ⋆ src f* ⋆ ϑ′) · (r ⋆ a[src f*, f, w]) ·
                  (r ⋆ l⁻¹[f] · r[f] ⋆ w) ·
                    (r ⋆ a⁻¹[f, trg w, w]) · a[r, f, trg w ⋆ w] ·
                      (ϱ ⋆ trg w ⋆ w)) · (g ⋆ l⁻¹[w])
    using T0.antipar by simp
  finally show ?thesis by simp
qed
also have ... = ν · r[r] ·
                  ((r ⋆ r[src r]) · (r ⋆ src r ⋆ ϑ′)) ·
                    (r ⋆ a[src r, f, w]) · (r ⋆ l⁻¹[f] · r[f] ⋆ w) ·
                      (r ⋆ a⁻¹[f, trg w, w]) · a[r, f, trg w ⋆ w] ·
                        ((ϱ ⋆ trg w ⋆ w) · (g ⋆ l⁻¹[w]))
  using comp-assoc T0.antipar by simp
also have ... = ν · r[r] ·
                  ((r ⋆ ϑ′) · (r ⋆ l[f ⋆ w])) ·
                    (r ⋆ a[src r, f, w]) · (r ⋆ l⁻¹[f] · r[f] ⋆ w) ·
                      (r ⋆ a⁻¹[f, trg w, w]) · a[r, f, trg w ⋆ w] ·
                        (((r ⋆ f) ⋆ l⁻¹[w]) · (ϱ ⋆ w))
proof −
  have (r ⋆ r[src r]) · (r ⋆ src r ⋆ ϑ′) = (r ⋆ ϑ′) · (r ⋆ l[f ⋆ w])
  proof −
    have (r ⋆ r[src r]) · (r ⋆ src r ⋆ ϑ′) = r ⋆ r[src r] · (src r ⋆ ϑ′)
      using whisker-left by simp
    also have ... = r ⋆ ϑ′ · l[f ⋆ w]
      using lunit-naturality [of ϑ′] unitor-coincidence by simp
    also have ... = (r ⋆ ϑ′) · (r ⋆ l[f ⋆ w])
      using whisker-left by simp
    finally show ?thesis by simp
  qed
  moreover have (ϱ ⋆ trg w ⋆ w) · (g ⋆ l⁻¹[w]) = ((r ⋆ f) ⋆ l⁻¹[w]) · (ϱ ⋆ w)
  proof −
    have (ϱ ⋆ trg w ⋆ w) · (g ⋆ l⁻¹[w]) = ϱ · g ⋆ (trg w ⋆ w) · l⁻¹[w]
      using interchange by simp
    also have ... = ϱ ⋆ l⁻¹[w]
      using comp-arr-dom comp-cod-arr by simp
    also have ... = (r ⋆ f) · ϱ ⋆ l⁻¹[w] · w
      using comp-arr-dom comp-cod-arr by simp
    also have ... = ((r ⋆ f) ⋆ l⁻¹[w]) · (ϱ ⋆ w)
      using interchange by simp
    finally show ?thesis by simp
  qed
  ultimately show ?thesis by simp
qed
```

also have ... $= \nu \cdot \mathrm{r}[r] \cdot (r \star \vartheta') \cdot$
$((r \star \mathrm{l}[f \star w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$\mathrm{a}[r,\ f,\ trg\ w \star w] \cdot ((r \star f) \star \mathrm{l}^{-1}[w])) \cdot$
$(\varrho \star w)$
using *comp-assoc* **by** *simp*
also have ... $= \nu \cdot \mathrm{r}[r] \cdot (r \star \vartheta') \cdot \mathrm{a}[r,\ f,\ w] \cdot (\varrho \star w)$
proof −
have $((r \star \mathrm{l}[f \star w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$\mathrm{a}[r,\ f,\ trg\ w \star w] \cdot ((r \star f) \star \mathrm{l}^{-1}[w])) =$
$\mathrm{a}[r,\ f,\ w]$
proof −
have $((r \star \mathrm{l}[f \star w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$\mathrm{a}[r,\ f,\ trg\ w \star w] \cdot ((r \star f) \star \mathrm{l}^{-1}[w])) =$
$((r \star (\mathrm{l}[f] \star w) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$(r \star f \star \mathrm{l}^{-1}[w])) \cdot \mathrm{a}[r,\ f,\ w]$
using *comp-assoc assoc-naturality* [*of r f* $\mathrm{l}^{-1}[w]$] *lunit-hcomp* **by** *simp*
also have ... $= \mathrm{a}[r,\ f,\ w]$
proof −
have $(r \star (\mathrm{l}[f] \star w) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$(r \star f \star \mathrm{l}^{-1}[w]) =$
$r \star f \star w$
proof −

Again, get a little more mileage out of coherence.

have $(r \star (\mathrm{l}[f] \star w) \cdot \mathrm{a}^{-1}[trg\ f,\ f,\ w]) \cdot (r \star \mathrm{a}[src\ r,\ f,\ w]) \cdot$
$(r \star \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \star w) \cdot (r \star \mathrm{a}^{-1}[f,\ trg\ w,\ w]) \cdot$
$(r \star f \star \mathrm{l}^{-1}[w]) =$
$\{\!|(\langle r\rangle \star (\mathbf{l}[\langle f\rangle] \star \langle w\rangle) \cdot \mathbf{a}^{-1}[E.Trg\ \langle f\rangle,\ \langle f\rangle,\ \langle w\rangle]) \cdot$
$(\langle r\rangle \star \mathbf{a}[E.Src\ \langle r\rangle,\ \langle f\rangle,\ \langle w\rangle]) \cdot$
$(\langle r\rangle \star \mathbf{l}^{-1}[\langle f\rangle] \cdot \mathbf{r}[\langle f\rangle] \star \langle w\rangle) \cdot (\langle r\rangle \star \mathbf{a}^{-1}[\langle f\rangle,\ E.Trg\ \langle w\rangle,\ \langle w\rangle]) \cdot$
$(\langle r\rangle \star \langle f\rangle \star \mathbf{l}^{-1}[\langle w\rangle])|\!\}$
using $\mathfrak{l}$*-ide-simp* $\mathfrak{r}$*-ide-simp* a′*-def* α*-def assoc′-eq-inv-assoc* **by** *simp*
also have ... $= \{\!|\langle r\rangle \star \langle f\rangle \star \langle w\rangle|\!\}$
by (*intro E.eval-eqI*, *auto*)
also have ... $= r \star f \star w$
by *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-cod-arr*
by (*metis assoc-naturality1 base-simps*(*2*−*3*) *leg0-simps*(*2*−*4*)
w-simps(*2*) *w-simps*(*4*) *w-simps*(*5*))
qed
finally show *?thesis* **by** *blast*

```
        qed
        thus ?thesis by simp
      qed
      also have ... = ν · r[r] · r⁻¹[r] · inv ν
      proof −
        have r⁻¹[r] · inv ν = (r ⋆ ϑ′) · a[r, f, w] · (ϱ ⋆ w)
         using ** wϑ′ν ide-base ide-leg0 tab-in-hom invert-side-of-triangle(2) comp-arr-dom
               T0.antipar comp-assoc runit′-simps(1)
          by metis
        thus ?thesis by simp
      qed
      also have ... = g ⋆ w
        using ** wϑ′ν ide-base comp-arr-inv′
        by (metis calculation in-homE invert-side-of-triangle(1) iso-runit iso-runit′)
      finally show ?thesis by simp
    qed
    thus ?thesis by simp
  qed
qed
ultimately have 1: iso (g ⋆ γ)
  using iso-iff-section-and-retraction by simp
have iso (inv (ν · r[r]) · (g ⋆ γ))
proof −
  have iso (inv (ν · r[r]))
    using wϑ′ν γ iso-runit
    by (elim conjE in-homE, intro iso-inv-iso isos-compose, auto)
  thus ?thesis
    using 1 wϑ′ν γ trg-γ-eq isos-compose
    by (elim conjE in-homE, auto)
qed
moreover have inv (ν · r[r]) · (g ⋆ γ) = composite-cell f* ε
proof −
  have inv (ν · r[r]) · (g ⋆ γ) = inv (ν · r[r]) · ν · r[r] · composite-cell f* ε
    using γ by auto
  also have ... = ((inv (ν · r[r]) · (ν · r[r])) · (r ⋆ ε)) · a[r, f, f*] · (ϱ ⋆ f*)
      using wϑ′ν comp-assoc by auto
  also have ... = composite-cell f* ε
  proof −
    have dom ν = r
      using wϑ′ν by auto
    thus ?thesis
      using iso-runit wϑ′ν isos-compose comp-cod-arr whisker-left comp-inv-arr′
      by auto
  qed
  finally show ?thesis by blast
qed
ultimately have iso (composite-cell f* ε) by simp
thus iso (T0.trnr_ε r ϱ)
  using T0.trnr_ε-def ide-base runit-in-hom iso-runit isos-compose
```

by (*metis A arrI seqE*)
qed

It is convenient to have a simpler version of the previous result for when we do not care about the details of the isomorphism.

lemma *yields-isomorphic-representation′*:
obtains ψ **where** «$\psi : g \star f^* \Rightarrow r$» **and** *iso* ψ
using *yields-isomorphic-representation adjoint-pair-def* **by** *simp*

end

It is natural to ask whether if «$\psi : g \star f^* \Rightarrow r$» is an isomorphism then $\varrho = (\psi \star f) \cdot T0.trnr_\eta\ g\ (g \star f^*)$ is a tabulation of r. This is not true without additional conditions on f and g (*cf.* the comments following CKS Proposition 6). So only rather special isomorphisms «$\psi : g \star f^* \Rightarrow r$» result from tabulations of r.

2.2.3 Tabulation of a Right Adjoint

Here we obtain a tabulation of the right adjoint of a map. This is CKS Proposition 1(e). It was somewhat difficult to find the correct way to insert the unitors that CKS omit. At first I thought I could only prove this under the assumption that the bicategory is normal, but later I saw how to do it in the general case.

context *adjunction-in-bicategory*
begin

lemma *tabulation-of-right-adjoint*:
shows *tabulation V H* a i *src trg g* η *f* (*src f*)
proof −
interpret *T*: *tabulation-data V H* a i *src trg g* η *f* ‹*src f*›
using *unit-in-hom antipar* **by** (*unfold-locales*, *simp-all*)
show *?thesis*
proof
show *T1*: $\bigwedge u\ \omega.\ [\![\ ide\ u;\ «\omega : dom\ \omega \Rightarrow g \star u»\ ]\!] \Longrightarrow$
$\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : f \star w \Rightarrow u» \wedge «\nu : dom\ \omega \Rightarrow src\ f \star w» \wedge iso\ \nu \wedge$
$T.composite\text{-}cell\ w\ \vartheta \cdot \nu = \omega$
proof −
fix $u\ v\ \omega$
assume *u*: *ide u*
assume ω: «$\omega : v \Rightarrow g \star u$»
have *v*: *ide v*
using ω **by** *auto*
have *1*: *src g* = *trg u*
using ω **by** (*metis arr-cod in-homE not-arr-null seq-if-composable*)
have *2*: *src f* = *trg v*
using ω *1 u ide-right antipar*(*1*) *vconn-implies-hpar*(*4*) **by** *force*

It seems clear that we need to take $w = v$ and $\nu = \mathrm{l}^{-1}[v]$.

let *?w* = *v*

let $?\nu = \mathrm{l}^{-1}[v]$
have ν: «$?\nu : v \Rightarrow src\ f \star ?w$» $\wedge$ *iso* $?\nu$
using *v 2 iso-lunit′* **by** *auto*

We need ϑ, defined to satisfy «$\vartheta : f \star v \Rightarrow u$» and $\omega = (v \star \vartheta) \cdot \mathrm{a}[v, f, v] \cdot (\eta \star w) \cdot \mathrm{l}^{-1}[v]$. We have «$\omega : v \Rightarrow g \star u$», so we can get arrow «$\vartheta : f \star v \Rightarrow u$» by adjoint transpose. Note that this uses adjoint transpose on the *left*, rather than on the right.

let $?\vartheta = trnl_\varepsilon\ u\ \omega$
have ϑ: «$?\vartheta : f \star ?w \Rightarrow u$»
using *u v antipar 1 2* ω *adjoint-transpose-left(2)* [*of u v*] **by** *auto*

Now, $trnl_\eta\ v\ \vartheta \equiv (g \star \vartheta) \cdot \mathrm{a}[g, f, v] \cdot (\eta \star v) \cdot \mathrm{l}^{-1}[v]$, which suggests that we ought to have $\omega = trnl_\eta\ v\ \vartheta$ and $\nu = \mathrm{l}^{-1}[v]$;

have *T.composite-cell ?w ?ϑ · ?ν* $= \omega$
using *u v* ω *1 2 adjoint-transpose-left(4)* [*of u v* ω] $trnl_\eta$*-def comp-assoc* **by** *simp*
thus $\exists\, w\ \vartheta\ \nu.$ *ide w* $\wedge$ «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «$\nu : v \Rightarrow src\ f \star w$» $\wedge$ *iso* ν $\wedge$
T.composite-cell w ϑ · ν $= \omega$
using *v ϑ ν antipar comp-assoc* **by** *blast*
qed
show *T2*: $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.$
$[\![$ *ide w*; *ide w′*; «$\vartheta : f \star w \Rightarrow u$»; «$\vartheta' : f \star w' \Rightarrow u$»;
«$\beta : src\ f \star w \Rightarrow src\ f \star w'$»;
T.composite-cell w ϑ $=$ *T.composite-cell w′ ϑ′ · β* $]\!] \Longrightarrow$
$\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = src\ f \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
proof −
fix *u w w′ ϑ ϑ′ β*
assume *w*: *ide w*
assume *w′*: *ide w′*
assume ϑ: «$\vartheta : f \star w \Rightarrow u$»
assume ϑ': «$\vartheta' : f \star w' \Rightarrow u$»
assume β: «$\beta : src\ f \star w \Rightarrow src\ f \star w'$»
assume *E*: *T.composite-cell w ϑ* $=$ *T.composite-cell w′ ϑ′ · β*
interpret *T*: *uwϑw′ϑ′β V H* a i *src trg g η f* ‹*src f*› *u w ϑ w′ ϑ′ β*
using *w w′ ϑ ϑ′ β E comp-assoc* **by** (*unfold-locales*, *auto*)
have *2*: *src f* $=$ *trg β*
using *antipar* **by** *simp*
show $\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = src\ f \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
proof −

The requirement $\beta = src\ f \star \gamma$ means we have to essentially invert $\lambda\gamma.\ src\ f \star \gamma$ to obtain γ. CKS say only: "the strong form of *T2* is clear since $g = 1$" (here by "g" they are referring to *dom* η, the "output leg" of the span in the tabulation). This would mean that we would have to take $\gamma = \beta$, which doesn't work for a general bicategory (we don't necessarily have $src\ f \star \gamma = \gamma$). For a general bicategory, we have to take $\gamma = \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w]$.

let $?\gamma = \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w]$
have γ: «$?\gamma : w \Rightarrow w'$»
using β **by** *simp*

```
have 3: β = src f ⋆ ?γ
proof −
  have β = l⁻¹[w′] · ?γ · l[w]
    using β iso-lunit
    by (simp add: comp-arr-dom invert-side-of-triangle(1) comp-assoc)
  also have ... = l⁻¹[w′] · l[w′] · (src f ⋆ ?γ)
    using γ lunit-naturality
    by (metis T.uwϑ.w-simps(4) in-homE trg-dom)
  also have ... = (l⁻¹[w′] · l[w′]) · (src f ⋆ ?γ)
    using comp-assoc by simp
  also have ... = src f ⋆ ?γ
    using γ iso-lunit comp-inv-arr comp-cod-arr
   by (metis T.β-simps(1) calculation comp-ide-arr inv-is-inverse inverse-arrowsE w′)
  finally show ?thesis by simp
qed
have ϑ = ϑ′ · (f ⋆ ?γ)
proof −
  have ϑ = trnl_ε u (trnl_η w ϑ)
    using ϑ adjoint-transpose-left(3) [of u w ϑ] by simp
  also have ... = trnl_ε u (trnl_η w′ ϑ′ · l[w′] · β · l⁻¹[w])
  proof −
    have trnl_η w ϑ = trnl_η w′ ϑ′ · l[w′] · β · l⁻¹[w]
    proof −
      have trnl_η w ϑ · l[w] = (T.composite-cell w ϑ · l⁻¹[w]) · l[w]
        unfolding trnl_η-def using comp-assoc by simp
      also have ... = T.composite-cell w ϑ · (l⁻¹[w] · l[w])
        using comp-assoc by simp
      also have 4: ... = T.composite-cell w ϑ
        using comp-arr-dom by (simp add: comp-inv-arr′)
      also have ... = T.composite-cell w′ ϑ′ · β
        using E by simp
      also have ... = (T.composite-cell w′ ϑ′ · l⁻¹[w′]) · l[w′] · β
      proof −
        have (l⁻¹[w′] · l[w′]) · β = β
          using iso-lunit β comp-cod-arr comp-assoc comp-inv-arr′ by simp
        thus ?thesis
          using comp-assoc by simp
      qed
      also have ... = trnl_η w′ ϑ′ · l[w′] · β
        unfolding trnl_η-def using comp-assoc by simp
      finally have trnl_η w ϑ · l[w] = trnl_η w′ ϑ′ · l[w′] · β
        by simp
      thus ?thesis
        using β 4 invert-side-of-triangle(2) adjoint-transpose-left iso-lunit
              trnl_η-def comp-assoc
        by metis
    qed
    thus ?thesis by simp
  qed
```

also have ... $= \mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \mathit{trnl}_\eta\ w'\ \vartheta' \cdot \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w])$
using $\mathit{trnl}_\varepsilon$*-def* **by** *simp*
also have
... $= \mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \mathit{trnl}_\eta\ w'\ \vartheta') \cdot (f \star \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w])$
using *ide-left ide-right w w′ 2 β ϑ antipar* $\mathit{trnl}_\varepsilon$*-def adjoint-transpose-left whisker-left*
by (*metis T.uwϑ.ϑ-simps*(*1*) *calculation hseqE seqE*)
also have
... $= (\mathrm{l}[u] \cdot (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \mathit{trnl}_\eta\ w'\ \vartheta')) \cdot (f \star \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w])$
using *comp-assoc* **by** *simp*
also have ... $= \mathit{trnl}_\varepsilon\ u\ (\mathit{trnl}_\eta\ w'\ \vartheta') \cdot (f \star \mathrm{l}[w'] \cdot \beta \cdot \mathrm{l}^{-1}[w])$
unfolding $\mathit{trnl}_\varepsilon$*-def* **by** *simp*
also have ... $= \vartheta' \cdot (f \star ?\gamma)$
using *ϑ′ adjoint-transpose-left*(*3*) **by** *auto*
finally show *?thesis* **by** *simp*
qed
hence $\exists \gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = \mathit{src}\ f \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
using *γ 3 hcomp-obj-arr* **by** *blast*
moreover have $\bigwedge \gamma\ \gamma'.\ «\gamma : w \Rightarrow w'» \wedge \beta = \mathit{src}\ f \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma) \wedge$
$«\gamma' : w \Rightarrow w'» \wedge \beta = \mathit{src}\ f \star \gamma' \wedge \vartheta = \vartheta' \cdot (f \star \gamma') \Longrightarrow \gamma = \gamma'$
proof −
fix $\gamma\ \gamma'$
assume $\gamma\gamma'$: $«\gamma : w \Rightarrow w'» \wedge \beta = \mathit{src}\ f \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma) \wedge$
$«\gamma' : w \Rightarrow w'» \wedge \beta = \mathit{src}\ f \star \gamma' \wedge \vartheta = \vartheta' \cdot (f \star \gamma')$
show $\gamma = \gamma'$
using *γγ′ vconn-implies-hpar*(*2*) *L.is-faithful* [*of γ γ′*] **by** *force*
qed
ultimately show *?thesis* **by** *blast*
qed
qed
qed
qed

end

2.2.4 Preservation by Isomorphisms

Next, we show that tabulations are preserved under composition on all three sides by isomorphisms. This is something that we would expect to hold if "tabulation" is a properly bicategorical notion.

context *tabulation*
begin

Tabulations are preserved under composition of an isomorphism with the "input leg".

lemma *preserved-by-input-iso*:
assumes $«\varphi : f \Rightarrow f'»$ **and** *iso φ*
shows *tabulation V H* a i *src trg r* $((r \star \varphi) \cdot \varrho)\ f'\ g$
proof −
interpret T': *tabulation-data V H* a i *src trg r* $‹(r \star \varphi) \cdot \varrho›\ f'$

```
    using assms(1) tab-in-hom
    apply unfold-locales
      apply auto
    by force
  show ?thesis
  proof
    show ⋀u ω. ⟦ ide u; «ω : dom ω ⇒ r ⋆ u» ⟧ ⟹
          ∃ w ϑ ν. ide w ∧ «ϑ : f′ ⋆ w ⇒ u» ∧ «ν : dom ω ⇒ g ⋆ w» ∧
                iso ν ∧ T′.composite-cell w ϑ · ν = ω
    proof −
      fix u ω
      assume u: ide u and ω: «ω : dom ω ⇒ r ⋆ u»
      obtain w ϑ ν where wϑν: ide w ∧ «ϑ : f ⋆ w ⇒ u» ∧ «ν : dom ω ⇒ g ⋆ w» ∧
                          iso ν ∧ composite-cell w ϑ · ν = ω
        using u ω T1 by blast
      interpret T1: uwϑων V H a i src trg r ϱ f g u w ϑ ω ν
        using wϑν comp-assoc by (unfold-locales, auto)
      have 1: «inv φ ⋆ w : f′ ⋆ w ⇒ f ⋆ w»
        using assms by (intro hcomp-in-vhom, auto)
      have ide w ∧ «ϑ · (inv φ ⋆ w) : f′ ⋆ w ⇒ u» ∧ «ν : dom ω ⇒ g ⋆ w» ∧ iso ν ∧
           T′.composite-cell w (ϑ · (inv φ ⋆ w)) · ν = ω
        using wϑν 1
        apply (intro conjI)
          apply auto[4]
      proof −
        show T′.composite-cell w (ϑ · (inv φ ⋆ w)) · ν = ω
        proof −
          have T′.composite-cell w (ϑ · (inv φ ⋆ w)) · ν =
                (r ⋆ ϑ) · ((r ⋆ inv φ ⋆ w) · a[r, f′, w]) · ((r ⋆ φ) · ϱ ⋆ w) · ν
            using assms(1) 1 whisker-left [of r ϑ inv φ ⋆ w] comp-assoc by auto
          also have ... = (r ⋆ ϑ) · (a[r, f, w] · ((r ⋆ inv φ) ⋆ w)) · ((r ⋆ φ) · ϱ ⋆ w) · ν
            using assms assoc-naturality [of r inv φ w]
            by (metis 1 T′.tab-simps(1) base-simps(3) base-simps(4) T1.w-simps(5−6)
                cod-inv dom-inv hseqE in-homE seqE trg-inv)
          also have ... = (r ⋆ ϑ) · a[r, f, w] · ((((r ⋆ inv φ) ⋆ w) · ((r ⋆ φ) ⋆ w)) · (ϱ ⋆ w)) · ν
            using whisker-right [of w r ⋆ φ ϱ] comp-assoc T1.ide-w vseq-implies-hpar(1)
            by auto
          also have ... = composite-cell w ϑ · ν
          proof −
            have (((r ⋆ inv φ) ⋆ w) · ((r ⋆ φ) ⋆ w)) · (ϱ ⋆ w) = ϱ ⋆ w
            proof −
              have «r ⋆ φ : r ⋆ f ⇒ r ⋆ f′»
                using assms(1) by (intro hcomp-in-vhom, auto)
              moreover have «r ⋆ inv φ : r ⋆ f′ ⇒ r ⋆ f»
                using assms by (intro hcomp-in-vhom, auto)
              ultimately show ?thesis
                using comp-cod-arr
              by (metis T1.w-in-hom(2) tab-simps(1) tab-simps(5) assms(1−2) comp-inv-arr′
                        in-homE leg0-simps(2) interchange base-in-hom(2) seqI′)
```

qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \omega$
using $w\vartheta\nu$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed
thus $\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : f' \star w \Rightarrow u» \wedge «\nu : dom\ \omega \Rightarrow g \star w» \wedge iso\ \nu \wedge$
$T'.composite\text{-}cell\ w\ \vartheta \cdot \nu = \omega$
by *blast*
qed
show $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.\ [\![\ ide\ w;\ ide\ w';\ «\vartheta : f' \star w \Rightarrow u»;\ «\vartheta' : f' \star w' \Rightarrow u»;$
$«\beta : g \star w \Rightarrow g \star w'»;$
$T'.composite\text{-}cell\ w\ \vartheta = T'.composite\text{-}cell\ w'\ \vartheta' \cdot \beta\ ]\!] \Longrightarrow$
$\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f' \star \gamma)$
proof −
fix $u\ w\ w'\ \vartheta\ \vartheta'\ \beta$
assume w: $ide\ w$ **and** w': $ide\ w'$
and ϑ: $«\vartheta : f' \star w \Rightarrow u»$ **and** ϑ': $«\vartheta' : f' \star w' \Rightarrow u»$
and β: $«\beta : g \star w \Rightarrow g \star w'»$
and eq: $T'.composite\text{-}cell\ w\ \vartheta = T'.composite\text{-}cell\ w'\ \vartheta' \cdot \beta$
interpret $uw\vartheta w'\vartheta'\beta$ $V\ H$ a i $src\ trg\ r$ ‹$(r \star \varphi) \cdot \varrho$› $f'\ g\ u\ w\ \vartheta\ w'\ \vartheta'\ \beta$
using $w\ w'\ \vartheta\ \vartheta'\ \beta\ eq\ comp\text{-}assoc$ **by** (*unfold-locales*, *auto*)
show $\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f' \star \gamma)$
proof −
have φ-w: $«\varphi \star w : f \star w \Rightarrow f' \star w»$
using *assms*(*1*) **by** (*intro hcomp-in-vhom*, *auto*)
have φ-w': $«\varphi \star w' : f \star w' \Rightarrow f' \star w'»$
using *assms*(*1*) **by** (*intro hcomp-in-vhom*, *auto*)
have $«\vartheta \cdot (\varphi \star w) : f \star w \Rightarrow u»$
using ϑ *assms*(*1*) **by** *fastforce*
moreover have $«\vartheta' \cdot (\varphi \star w') : f \star w' \Rightarrow u»$
using ϑ' *assms*(*1*) **by** *fastforce*
moreover have $composite\text{-}cell\ w\ (\vartheta \cdot (\varphi \star w)) = composite\text{-}cell\ w'\ (\vartheta' \cdot (\varphi \star w')) \cdot \beta$
proof −
have $composite\text{-}cell\ w\ (\vartheta \cdot (\varphi \star w)) =$
$(r \star \vartheta) \cdot ((r \star \varphi \star w) \cdot \mathrm{a}[r, f, w]) \cdot (\varrho \star w)$
using *assms*(*2*) φ-w ϑ *whisker-left comp-assoc* **by** *auto*
also have ... $= (r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot ((r \star \varphi) \star w) \cdot (\varrho \star w)$
using *assms*(*1*) *assoc-naturality* [*of* $r\ \varphi\ w$] *comp-assoc*
by (*metis* φ-w $T'.tab\text{-}simps(1)$ *base-simps*(*3*) *base-simps*(*4*) *hseq-char*
in-homE seqE $uw\vartheta.w\text{-}simps(5)$ $uw\vartheta.w\text{-}simps(6)$)
also have ... $= T'.composite\text{-}cell\ w\ \vartheta$
using *assms*(*2*) w *whisker-right* [*of* w] **by** *simp*
also have ... $= T'.composite\text{-}cell\ w'\ \vartheta' \cdot \beta$
using *eq* **by** *simp*
also have ... $= (r \star \vartheta') \cdot (\mathrm{a}[r, f', w'] \cdot ((r \star \varphi) \star w')) \cdot (\varrho \star w') \cdot \beta$

using *assms*(*2*) w' *whisker-right* [*of* w'] *comp-assoc* **by** *simp*
also have ... $= ((r \star \vartheta') \cdot (r \star \varphi \star w')) \cdot \mathrm{a}[r, f, w'] \cdot (\varrho \star w') \cdot \beta$
using *assms*(*1*) *assoc-naturality* [*of* r φ w'] *comp-assoc*
by (*metis* φ-w' T'.*tab-simps*(*1*) *base-simps*(*3*) *base-simps*(*4*) *hseqE in-homE seqE* $uw'\vartheta'$.*w-simps*(*5*) $uw'\vartheta'$.*w-simps*(*6*))
also have ... $=$ *composite-cell* w' $(\vartheta' \cdot (\varphi \star w')) \cdot \beta$
using *assms*(*2*) *whisker-left* [*of* r] ‹«$\vartheta' \cdot (\varphi \star w') : f \star w' \Rightarrow u$»› *comp-assoc*
by *auto*
finally show *?thesis* **by** *simp*
qed
ultimately have $*$: $\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = g \star \gamma \wedge$
$\vartheta \cdot (\varphi \star w) = (\vartheta' \cdot (\varphi \star w')) \cdot (f \star \gamma)$
using w w' β *T2* **by** *auto*
show *?thesis*
proof −
have $**$: $\bigwedge\gamma.$ «$\gamma : w \Rightarrow w'$» $\Longrightarrow \vartheta' \cdot (\varphi \star w') \cdot (f \star \gamma) \cdot (inv\ \varphi \star w) = \vartheta' \cdot (f' \star \gamma)$
proof −
fix γ
assume γ: «$\gamma : w \Rightarrow w'$»
have $\vartheta' \cdot (\varphi \star w') \cdot (f \star \gamma) \cdot (inv\ \varphi \star w) = \vartheta' \cdot (\varphi \star w') \cdot (f \cdot inv\ \varphi \star \gamma \cdot w)$
using γ *assms*(*1*−*2*) *interchange*
by (*metis arr-inv cod-inv in-homE leg0-simps*(*2*) *leg0-simps*(*4*) *uw*ϑ.*w-in-hom*(*2*) *seqI*)
also have ... $= \vartheta' \cdot (\varphi \cdot f \cdot inv\ \varphi \star w' \cdot \gamma \cdot w)$
using *assms*(*1*−*2*) *interchange*
by (*metis* γ *arr-inv cod-inv comp-arr-dom comp-cod-arr in-homE seqI*)
also have ... $= \vartheta' \cdot (f' \star \gamma)$
proof −
have $\varphi \cdot f \cdot inv\ \varphi = f'$
using *assms*(*1*−*2*) *comp-cod-arr comp-arr-inv'* **by** *auto*
moreover have $w' \cdot \gamma \cdot w = \gamma$
using γ *comp-arr-dom comp-cod-arr* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
finally show $\vartheta' \cdot (\varphi \star w') \cdot (f \star \gamma) \cdot (inv\ \varphi \star w) = \vartheta' \cdot (f' \star \gamma)$ **by** *simp*
qed
obtain γ **where** γ: «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = g \star \gamma \wedge$
$\vartheta \cdot (\varphi \star w) = (\vartheta' \cdot (\varphi \star w')) \cdot (f \star \gamma)$
using $*$ **by** *blast*
have $\vartheta = \vartheta' \cdot (\varphi \star w') \cdot (f \star \gamma) \cdot (inv\ \varphi \star w)$
proof −
have *seq* $(\vartheta' \cdot (\varphi \star w'))$ $(f \star \gamma)$
using *assms*(*2*) φ-w φ-w' γ β ϑ
apply (*intro seqI*)
apply *auto*
by (*metis seqE seqI'*)
thus *?thesis*
using *assms* φ-w γ *comp-assoc invert-side-of-triangle*(*2*) *iso-hcomp*
by (*metis hcomp-in-vhomE ide-is-iso inv-hcomp inv-ide* w)

```
        qed
        hence ϑ = ϑ' · (f' ⋆ γ)
          using γ ** by simp
        hence ∃γ. «γ : w ⇒ w'» ∧ β = g ⋆ γ ∧ ϑ = ϑ' · (f' ⋆ γ)
          using γ by auto
        moreover have ⋀γ γ'. «γ : w ⇒ w'» ∧ β = g ⋆ γ ∧ ϑ = ϑ' · (f' ⋆ γ) ∧
                            «γ' : w ⇒ w'» ∧ β = g ⋆ γ' ∧ ϑ = ϑ' · (f' ⋆ γ')
                              ⟹ γ = γ'
        proof −
          fix γ γ'
          assume A: «γ : w ⇒ w'» ∧ β = g ⋆ γ ∧ ϑ = ϑ' · (f' ⋆ γ) ∧
                    «γ' : w ⇒ w'» ∧ β = g ⋆ γ' ∧ ϑ = ϑ' · (f' ⋆ γ')
          have ϑ · (φ ⋆ w) = (ϑ' · (φ ⋆ w')) · (f ⋆ γ)
          proof −
            have ϑ = ((ϑ' · (φ ⋆ w')) · (f ⋆ γ)) · (inv φ ⋆ w)
              using A ** comp-assoc by simp
            thus ?thesis
              using assms(1−2) A iso-inv-iso
              by (metis comp-arr-dom comp-cod-arr in-homE comp-assoc interchange)
          qed
          moreover have ϑ · (φ ⋆ w) = (ϑ' · (φ ⋆ w')) · (f ⋆ γ')
          proof −
            have ϑ = ((ϑ' · (φ ⋆ w')) · (f ⋆ γ')) · (inv φ ⋆ w)
              using A ** comp-assoc by auto
            thus ?thesis
              using assms(1−2) A iso-inv-iso
              by (metis comp-arr-dom comp-cod-arr in-homE comp-assoc interchange)
          qed
         ultimately show γ = γ'
            using A * by blast
        qed
        ultimately show ∃!γ.  «γ : w ⇒ w'» ∧ β = g ⋆ γ ∧ ϑ = ϑ' · (f' ⋆ γ)
          by metis
      qed
    qed
  qed
 qed
qed
```

Similarly, tabulations are preserved under composition of an isomorphism with the "output leg".

```
lemma preserved-by-output-iso:
assumes «φ : g' ⇒ g» and iso φ
shows tabulation V H a i src trg r (ϱ · φ) f g'
proof −
  have τφ: «ϱ · φ : g' ⇒ r ⋆ f»
    using assms by auto
  interpret T': tabulation-data V H a i src trg r ‹ϱ · φ› f g'
    using assms(2) τφ by (unfold-locales, auto)
```

have φ-*in-hhom*: «φ : *src f* $\rightarrow$ *trg r*»
  using *assms src-cod* [*of* φ] *trg-cod* [*of* φ]
  by (*elim in-homE*, *simp*)
show *?thesis*
proof
  fix *u* ω
  assume *u*: *ide u* **and** ω: «ω : *dom* $\omega \Rightarrow r \star u$»
  show $\exists\, w\ \vartheta\ \nu'$. *ide w* $\wedge$ «ϑ : $f \star w \Rightarrow u$» $\wedge$ «ν' : *dom* $\omega \Rightarrow g' \star w$» $\wedge$ *iso* ν' $\wedge$
              T'.*composite-cell w* $\vartheta \cdot \nu' = \omega$
  proof −
    obtain *w* ϑ ν **where** $w\vartheta\nu$: *ide w* $\wedge$ «ϑ : $f \star w \Rightarrow u$» $\wedge$ «ν : *dom* $\omega \Rightarrow g \star w$» $\wedge$
                iso ν $\wedge$ *composite-cell w* $\vartheta \cdot \nu = \omega$
      using *u* ω *T1* [*of u* ω] **by** *auto*
    interpret $uw\vartheta\omega\nu$: $uw\vartheta\omega\nu$ *V H* a i *src trg r* ϱ *f g u w* ϑ ω ν
      using $w\vartheta\nu$ *comp-assoc* **by** (*unfold-locales*, *auto*)
    let $?\nu' = (inv\ \varphi \star w) \cdot \nu$
    have ν': «$?\nu'$: *dom* $\omega \Rightarrow g' \star w$»
      using *assms* φ-*in-hhom* $uw\vartheta\omega\nu$.ν-*in-hom*
      by (*intro comp-in-homI*, *auto*)
    moreover have *iso* $?\nu'$
      using *assms* ν' $w\vartheta\nu$ φ-*in-hhom*
      by (*intro iso-hcomp isos-compose*) *auto*
    moreover have T'.*composite-cell w* $\vartheta \cdot ?\nu' = \omega$
    proof −
      have *composite-cell w* $\vartheta \cdot ((\varphi \star w) \cdot ?\nu') = \omega$
      proof −
        have $(\varphi \star w) \cdot ?\nu' = \nu$
          using *assms* ν' φ-*in-hhom whisker-right comp-cod-arr comp-assoc*
          by (*metis comp-arr-inv' in-homE leg1-simps*(*2*) $uw\vartheta\omega\nu$.$uw\vartheta\omega\nu$)
        thus *?thesis*
          using $w\vartheta\nu$ **by** *simp*
      qed
      moreover have $(\varrho \cdot \varphi \star w) \cdot ?\nu' = (\varrho \star w) \cdot ((\varphi \star w) \cdot ?\nu')$
        using *assms* φ-*in-hhom whisker-right comp-assoc* **by** *simp*
      ultimately show *?thesis*
        using *comp-assoc* **by** *simp*
    qed
    ultimately show *?thesis*
      using $w\vartheta\nu$ **by** *blast*
  qed
  next
  fix *u w* w' ϑ ϑ' β'
  assume *w*: *ide w* **and** w': *ide* w'
  and ϑ: «ϑ : $f \star w \Rightarrow u$» **and** ϑ': «ϑ' : $f \star w' \Rightarrow u$»
  and β': «β' : $g' \star w \Rightarrow g' \star w'$»
  and *eq*': T'.*composite-cell w* ϑ = T'.*composite-cell* w' $\vartheta' \cdot \beta'$
  interpret $uw\vartheta w'\vartheta'\beta$: $uw\vartheta w'\vartheta'\beta$ *V H* a i *src trg r* ‹$\varrho \cdot \varphi$› *f* g' *u w* ϑ w' ϑ' β'
    using *assms w* w' ϑ ϑ' β' *eq*' *comp-assoc* **by** (*unfold-locales*, *auto*)
  let $?\beta = (\varphi \star w') \cdot \beta' \cdot (inv\ \varphi \star w)$

have β: «$?\beta : g \star w \Rightarrow g \star w'$»
  using *assms* φ*-in-hhom* β'
  by (*intro comp-in-homI hcomp-in-vhom*, *auto*)
have *eq*: *composite-cell* $w\ \vartheta$ = *composite-cell* $w'\ \vartheta' \cdot ((\varphi \star w') \cdot \beta' \cdot (inv\ \varphi \star w))$
proof −
  have *composite-cell* $w\ \vartheta = (r \star \vartheta) \cdot \mathrm{a}[r, f, w] \cdot ((\varrho \star w) \cdot (\varphi \star w)) \cdot (inv\ \varphi \star w)$
  proof −
    have $\varrho \star w = (\varrho \star w) \cdot (\varphi \star w) \cdot (inv\ \varphi \star w)$
      using *assms w* φ*-in-hhom whisker-right comp-arr-dom comp-arr-inv'*
      by (*metis tab-simps*(*1*) *tab-simps*(*4*) *in-homE leg1-simps*(*2*))
    thus *?thesis*
      using *comp-assoc* **by** *simp*
  qed
  also have ... = T'*.composite-cell* $w\ \vartheta \cdot (inv\ \varphi \star w)$
    using *assms* φ*-in-hhom whisker-right comp-assoc* **by** *simp*
  also have ... = T'*.composite-cell* $w'\ \vartheta' \cdot (\beta' \cdot (inv\ \varphi \star w))$
    using *eq' comp-assoc* **by** *simp*
  also have ... = *composite-cell* $w'\ \vartheta' \cdot ((\varphi \star w') \cdot \beta' \cdot (inv\ \varphi \star w))$
    using *assms* φ*-in-hhom whisker-right comp-assoc* **by** *simp*
  finally show *?thesis* **by** *simp*
qed
show $\exists!\gamma$. «$\gamma : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
proof −
  obtain γ **where** γ: «$\gamma : w \Rightarrow w'$» $\wedge\ ?\beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
    using *assms w w'* $\vartheta\ \vartheta'\ \beta$ *eq* φ*-in-hhom T2* [*of w w'* ϑ *u* ϑ' *?*β] **by** *auto*
  have $\beta' = g' \star \gamma$
  proof −
    have $g \star \gamma = (\varphi \star w') \cdot \beta' \cdot (inv\ \varphi \star w)$
      using γ **by** *simp*
    hence $(inv\ \varphi \star w') \cdot (g \star \gamma) = \beta' \cdot (inv\ \varphi \star w)$
      using *assms w'* β φ*-in-hhom invert-side-of-triangle arrI iso-hcomp*
        hseqE ide-is-iso inv-hcomp inv-ide seqE
      by *metis*
    hence $\beta' = (inv\ \varphi \star w') \cdot (g \star \gamma) \cdot (\varphi \star w)$
      using *assms w* β φ*-in-hhom invert-side-of-triangle comp-assoc seqE*
      by (*metis comp-arr-dom in-homE local.uw*ϑ*w'*$\vartheta'\beta$*.*β*-simps*(*4*) *whisker-right*)
    also have ... = $(inv\ \varphi \star w') \cdot (\varphi \star \gamma)$
      using *assms* φ*-in-hhom* γ *interchange comp-arr-dom comp-cod-arr*
      by (*metis in-homE*)
    also have ... = $g' \star \gamma$
      using *assms* φ*-in-hhom* γ *interchange comp-inv-arr inv-is-inverse comp-cod-arr*
      by (*metis arr-dom calculation in-homE*)
    finally show *?thesis* **by** *simp*
  qed
  hence $\exists\gamma$. «$\gamma : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
    using β γ **by** *auto*
  moreover have $\bigwedge\gamma\ \gamma'$. $[\![$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$;
      «$\gamma' : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma' \wedge \vartheta = \vartheta' \cdot (f \star \gamma')$ $]\!] \Longrightarrow \gamma = \gamma'$
  proof −

have $*$: $\bigwedge\gamma$. «$\gamma : w \Rightarrow w'$» $\Longrightarrow (\varphi \star w') \cdot (g' \star \gamma) \cdot (inv\ \varphi \star w) = g \star \gamma$
proof −
fix γ
assume γ: «$\gamma : w \Rightarrow w'$»
have $(\varphi \star w') \cdot (g' \star \gamma) \cdot (inv\ \varphi \star w) = (\varphi \star w') \cdot (inv\ \varphi \star \gamma)$
using *assms* φ*-in-hhom* γ *interchange comp-arr-dom comp-cod-arr*
by (*metis arr-dom comp-inv-arr′ in-homE invert-side-of-triangle*(*2*))
also have ... $= g \star \gamma$
using *assms* φ*-in-hhom interchange comp-arr-inv inv-is-inverse comp-cod-arr*
by (*metis* γ *comp-arr-inv′ in-homE leg1-simps*(*2*))
finally show $(\varphi \star w') \cdot (g' \star \gamma) \cdot (inv\ \varphi \star w) = g \star \gamma$ **by** *blast*
qed
fix $\gamma\ \gamma'$
assume γ: «$\gamma : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
and γ': «$\gamma' : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma' \wedge \vartheta = \vartheta' \cdot (f \star \gamma')$
show $\gamma = \gamma'$
using $w\ w'\ \vartheta\ \vartheta'\ \beta\ \gamma\ \gamma'$ *eq* $*$ *T2* **by** *metis*
qed
ultimately show $\exists!\gamma$. «$\gamma : w \Rightarrow w'$» $\wedge\ \beta' = g' \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$ **by** *blast*
qed
qed
qed

Finally, tabulations are preserved by composition with an isomorphism on the "base".

lemma *is-preserved-by-base-iso*:
assumes «$\varphi : r \Rightarrow r'$» **and** *iso* φ
shows *tabulation* $V\ H$ a i *src trg* r' $((\varphi \star f) \cdot \varrho)\ f\ g$
proof −
have φf: «$\varphi \star f : r \star f \Rightarrow r' \star f$»
using *assms ide-leg0* **by** *auto*
interpret T: *tabulation-data* $V\ H$ a i *src trg* r' ‹$(\varphi \star f) \cdot \varrho$› f
proof
show *ide-r′*: *ide* r' **using** *assms* **by** *auto*
show *ide* f **using** *ide-leg0* **by** *auto*
show «$(\varphi \star f) \cdot \varrho : g \Rightarrow r' \star f$»
using *tab-in-hom* φf **by** *force*
qed
show *?thesis*
proof
have $*$: $\bigwedge u\ v\ w\ \vartheta\ \nu$. $[\![$ *ide* u; *ide* v; *ide* w; «$\vartheta : f \star w \Rightarrow u$»; «$\nu : v \Rightarrow g \star w$» $]\!] \Longrightarrow$
$((\varphi \star u) \cdot (r \star \vartheta)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu =$
T*.composite-cell* $w\ \vartheta \cdot \nu$
proof −
fix $u\ v\ w\ \vartheta\ \nu$
assume u: *ide* u **and** v: *ide* v **and** w: *ide* w
and ϑ: «$\vartheta : f \star w \Rightarrow u$» **and** ν: «$\nu : v \Rightarrow g \star w$»
have *fw*: *hseq* $f\ w$
using ϑ *ide-dom* [*of* ϑ] **by** *fastforce*
have $r\vartheta$: *hseq* $r\ \vartheta$

using ϑ *ide-base ide-dom* [*of* ϑ] *trg-dom* [*of* ϑ]
using *arrI fw vconn-implies-hpar*(*2*) **by** *auto*
have $((\varphi \star u) \cdot (r \star \vartheta)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu =$
$((r' \star \vartheta) \cdot (\varphi \star f \star w)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu$
using *assms u w ide-base ide-leg0* ϑ *interchange comp-arr-dom comp-cod-arr*
by (*metis* $r\vartheta$ *hseq-char in-homE*)
also have ... $= (r' \star \vartheta) \cdot ((\varphi \star f \star w) \cdot \mathrm{a}[r, f, w]) \cdot (\varrho \star w) \cdot \nu$
using *comp-assoc* **by** *simp*
also have ... $= (r' \star \vartheta) \cdot \mathrm{a}[r', f, w] \cdot (((\varphi \star f) \star w) \cdot (\varrho \star w)) \cdot \nu$
proof −
have $(\varphi \star f \star w) \cdot \mathrm{a}[r, f, w] = \mathrm{a}[r', f, w] \cdot ((\varphi \star f) \star w)$
using *assms ide-leg0 w assoc-naturality* [*of* φ *f w*] *fw* **by** *fastforce*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= T.$*composite-cell* $w\ \vartheta \cdot \nu$
using *assms ide-leg0 whisker-right fw T.tab-in-hom arrI w comp-assoc* **by** *auto*
finally show $((\varphi \star u) \cdot (r \star \vartheta)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu = T.$*composite-cell* $w\ \vartheta \cdot \nu$
by *simp*
qed
show $\bigwedge u\ \omega'.\ [\![\ ide\ u;\ «\omega' : dom\ \omega' \Rightarrow r' \star u»\ ]\!] \Longrightarrow$
$\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : f \star w \Rightarrow u» \wedge «\nu : dom\ \omega' \Rightarrow g \star w» \wedge iso\ \nu \wedge$
$T.$*composite-cell* $w\ \vartheta \cdot \nu = \omega'$
proof −
fix $u\ v\ \omega'$
assume u: $ide\ u$ **and** ω': $«\omega' : v \Rightarrow r' \star u»$
have ω: $«(inv\ \varphi \star u) \cdot \omega' : v \Rightarrow r \star u»$
proof
show $«\omega' : v \Rightarrow r' \star u»$ **by** *fact*
show $«inv\ \varphi \star u : r' \star u \Rightarrow r \star u»$
proof −
have $ide\ (r' \star u)$
using ω' *ide-cod* **by** *fastforce*
hence $hseq\ r'\ u$ **by** *simp*
thus *?thesis*
using *assms u* **by** *auto*
qed
qed
have φu: $hseq\ \varphi\ u$
using *assms* ω *hseqI*
by (*metis arrI ide-is-iso iso-hcomp iso-is-arr seqE seq-if-composable*
src-inv u)
obtain $w\ \vartheta\ \nu$ **where** $w\vartheta\nu$: $ide\ w \wedge «\vartheta : f \star w \Rightarrow u» \wedge «\nu : v \Rightarrow g \star w» \wedge iso\ \nu \wedge$
composite-cell $w\ \vartheta \cdot \nu = (inv\ \varphi \star u) \cdot \omega'$
using $u\ \omega$ *T1* [*of* $u\ (inv\ \varphi \star u) \cdot \omega'$] φf *in-homE seqI'* **by** *auto*

interpret $uw\vartheta\omega\nu$ $V\ H$ a i $src\ trg\ r\ \varrho\ f\ g\ u\ w\ \vartheta$ ‹$(inv\ \varphi \star u) \cdot \omega'$› ν
using $w\vartheta\nu$ ω *comp-assoc* **by** (*unfold-locales*, *auto*)

have *ide w* ∧ «ϑ : $f \star w \Rightarrow u$» ∧ «ν : $v \Rightarrow g \star w$» ∧ *iso* ν ∧
T.composite-cell w $\vartheta \cdot \nu = \omega'$
proof −
have $\omega' = ((\varphi \star u) \cdot (r \star \vartheta)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu$
proof −
have *seq* $(r \star \vartheta)$ $(\mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot \nu)$ **by** *fastforce*
moreover have *iso* (*inv* $\varphi \star u$)
using *assms u* φu **by** *auto*
moreover have *inv* (*inv* $\varphi \star u$) $= \varphi \star u$
using *assms u* φu **by** *auto*
ultimately show *?thesis*
using *invert-side-of-triangle*(*1*) $w\vartheta\nu$ *comp-assoc* **by** *metis*
qed
also have ... = *T.composite-cell w* $\vartheta \cdot \nu$
using *u* $w\vartheta\nu$ ∗ [*of u v w* ϑ ν] **by** *force*
finally have $\omega' =$ *T.composite-cell w* $\vartheta \cdot \nu$ **by** *simp*
thus *?thesis*
using $w\vartheta\nu$ **by** *simp*
qed
thus ∃ *w* ϑ ν. *ide w* ∧ «ϑ : $f \star w \Rightarrow u$» ∧ «ν : $v \Rightarrow g \star w$» ∧ *iso* ν ∧
T.composite-cell w $\vartheta \cdot \nu = \omega'$
by *blast*
qed
show ⋀*u w w′* ϑ ϑ' β. ⟦ *ide w*; *ide w′*; «ϑ : $f \star w \Rightarrow u$»; «ϑ' : $f \star w' \Rightarrow u$»;
«β : $g \star w \Rightarrow g \star w'$»;
T.composite-cell w ϑ = *T.composite-cell w′* $\vartheta' \cdot \beta$ ⟧ ⟹
∃!γ. «γ : $w \Rightarrow w'$» ∧ $\beta = g \star \gamma$ ∧ $\vartheta = \vartheta' \cdot (f \star \gamma)$
proof −
fix *u w w′* ϑ ϑ' β
assume *w*: *ide w* **and** *w′*: *ide w′*
and ϑ: «ϑ : $f \star w \Rightarrow u$» **and** ϑ': «ϑ' : $f \star w' \Rightarrow u$»
and β: «β : $g \star w \Rightarrow g \star w'$»
and *eq′*: *T.composite-cell w* ϑ = *T.composite-cell w′* $\vartheta' \cdot \beta$
interpret *T*: $uw\vartheta w'\vartheta'\beta$ *V H* a i *src trg r′* ‹$(\varphi \star f) \cdot \varrho$› *f g u w* ϑ *w′* ϑ' β
using *w w′* ϑ ϑ' β *eq′ comp-assoc*
by (*unfold-locales*, *auto*)
have *eq*: *composite-cell w* ϑ = *composite-cell w′* $\vartheta' \cdot \beta$
proof −
have $(\varphi \star u) \cdot$ *composite-cell w* $\vartheta = (\varphi \star u) \cdot$ *composite-cell w′* $\vartheta' \cdot \beta$
proof −
have $(\varphi \star u) \cdot$ *composite-cell w* ϑ =
$((\varphi \star u) \cdot (r \star \vartheta)) \cdot \mathrm{a}[r, f, w] \cdot (\varrho \star w) \cdot (g \star w)$
proof −
have «$\varrho \star w : g \star w \Rightarrow (r \star f) \star w$»
using *w* **by** *auto*
thus *?thesis*
using *comp-arr-dom comp-assoc* **by** *auto*
qed
also have ... = *T.composite-cell w* $\vartheta \cdot (g \star w)$

```
        using ∗ [of u g ⋆ w w ϑ g ⋆ w] by fastforce
      also have ... = T.composite-cell w ϑ
      proof −
        have «(φ ⋆ f) · ϱ ⋆ w : g ⋆ w ⇒ (r′ ⋆ f) ⋆ w»
          using assms by fastforce
        thus ?thesis
          using comp-arr-dom comp-assoc by auto
      qed
      also have ... = T.composite-cell w′ ϑ′ · β
        using eq′ by simp
      also have ... = ((φ ⋆ u) · (r ⋆ ϑ′)) · a[r, f, w′] · (ϱ ⋆ w′) · β
        using ∗ [of u g ⋆ w w′ ϑ′ β] by fastforce
      finally show ?thesis
        using comp-assoc by simp
    qed
    moreover have iso (φ ⋆ u)
      using assms by auto
    moreover have seq (φ ⋆ u) ((r ⋆ ϑ) · a[r, f, w] · (ϱ ⋆ w))
    proof −
      have «φ ⋆ u : r ⋆ u ⇒ r′ ⋆ u»
        using assms by (intro hcomp-in-vhom, auto)
      thus ?thesis
        using composite-cell-in-hom [of w u ϑ] by auto
    qed
    moreover have seq (φ ⋆ u) (composite-cell w′ ϑ′ · β)
      using assms ide-leg0 w w′ ϑ ϑ′ β calculation(1) calculation(3) by auto
    ultimately show ?thesis
      using mono-cancel section-is-mono iso-is-section by metis
  qed
  show ∃!γ. «γ : w ⇒ w′» ∧ β = g ⋆ γ ∧ ϑ = ϑ′ · (f ⋆ γ)
    using w w′ ϑ ϑ′ β eq T2 by simp
 qed
qed
qed

end
```

2.2.5 Canonical Tabulations

If the 1-cell $g \star f^*$ has any tabulation (f, ϱ, g), then it has the canonical tabulation obtained as the adjoint transpose of (the identity on) $g \star f^*$.

```
context map-in-bicategory
begin

  lemma canonical-tabulation:
  assumes ide g and src f = src g
  and ∃ϱ. tabulation V H a i src trg (g ⋆ f∗) ϱ f g
  shows tabulation V H a i src trg (g ⋆ f∗) (trnr_η g (g ⋆ f∗)) f g
  proof −
```

```
    have 1: ide (g ⋆ f*)
      using assms(1−2) ide-right antipar by simp
    obtain ϱ where ϱ: tabulation V H a i src trg (g ⋆ f*) ϱ f g
      using assms(3) by auto
    interpret ϱ: tabulation V H a i src trg ‹g ⋆ f*› ϱ f g
      using ϱ by auto
    let ?ψ = trnr_ε (g ⋆ f*) ϱ
    have 3: «?ψ : g ⋆ f* ⇒ g ⋆ f*» ∧ iso ?ψ
      using ϱ.yields-isomorphic-representation by blast
    hence tabulation (·) (⋆) a i src trg (g ⋆ f*) ((inv ?ψ ⋆ f) · ϱ) f g
      using ϱ.is-preserved-by-base-iso [of inv ?ψ g ⋆ f*] by simp
    moreover have (inv ?ψ ⋆ f) · ϱ = trnr_η g (g ⋆ f*)
    proof −
      have (inv ?ψ ⋆ f) · ϱ = ((inv ?ψ ⋆ f) · (?ψ ⋆ f)) · trnr_η g (g ⋆ f*)
        using ϱ.ϱ-in-terms-of-rep comp-assoc by simp
      also have ... = ((g ⋆ f*) ⋆ f) · trnr_η g (g ⋆ f*)
      proof −
        have src (inv ?ψ) = trg f
          using 3 antipar
          by (metis ϱ.leg0-simps(3) ϱ.base-in-hom(2) seqI' src-inv vseq-implies-hpar(1))
        hence (inv ?ψ ⋆ f) · (?ψ ⋆ f) = (g ⋆ f*) ⋆ f
          using 3 whisker-right [of f inv ?ψ ?ψ] inv-is-inverse comp-inv-arr by auto
        thus ?thesis
          using comp-cod-arr by simp
      qed
      also have ... = trnr_η g (g ⋆ f*)
      proof −
        have src (g ⋆ f*) = trg f by simp
        moreover have ide g by simp
        ultimately have «trnr_η g (g ⋆ f*) : g ⇒ (g ⋆ f*) ⋆ f»
          using 1 adjoint-transpose-right(1) ide-in-hom antipar by blast
        thus ?thesis
          using comp-cod-arr by blast
      qed
      finally show ?thesis by simp
    qed
    ultimately show ?thesis by simp
  qed

end
```

2.2.6 Uniqueness of Tabulations

We now intend to show that a tabulation of r is "unique up to equivalence", which is a property that any proper bicategorical limit should have. What do we mean by this, exactly? If we have two tabulations (f, ϱ) and (f', ϱ') of the same 1-cell r, then this induces «w : $src\ f' \to src\ f$», «w' : $src\ f \to src\ f'$», «$\vartheta : f \star w \Rightarrow f'$», and «$\vartheta : f \star w \Rightarrow f'$», such that ϱ' is recovered up to isomorphism «$\nu : g' \Rightarrow g \star w$» from (w, ϑ) by composition with ϱ and ϱ is recovered up to isomorphism «$\nu' : g \Rightarrow g' \star w'$» from $(w',$

ϑ') by composition with ϱ'. This means that we obtain isomorphisms «$(\nu' \star w') \cdot \nu : g' \Rightarrow g' \star w' \star w$» and «$(\nu \star w') \cdot \nu' : g \Rightarrow g \star w \star w'$». These isomorphisms then induce, via *T2*, unique 2-cells from *src* f' to $w' \star w$ and from *src* f to $w \star w'$, which must be isomorphisms, thus showing w and w' are equivalence maps.

context *tabulation*
begin

We will need the following technical lemma.

lemma *apex-equivalence-lemma*:
assumes «$\varrho' : g' \Rightarrow r \star f'$»
and *ide* $w \wedge$ «$\vartheta : f' \star w \Rightarrow f$» $\wedge$ «$\nu : g \Rightarrow g' \star w$» $\wedge$ *iso* $\nu \wedge$
$(r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot (\varrho' \star w) \cdot \nu = \varrho$
and *ide* $w' \wedge$ «$\vartheta' : f \star w' \Rightarrow f'$» $\wedge$ «$\nu' : g' \Rightarrow g \star w'$» $\wedge$ *iso* $\nu' \wedge$
$(r \star \vartheta') \cdot \mathrm{a}[r, f, w'] \cdot (\varrho \star w') \cdot \nu' = \varrho'$
shows $\exists \varphi.$ «φ : *src* $f \Rightarrow w' \star w$» $\wedge$ *iso* φ
proof −
interpret *T'*: *uwϑων V H* a i *src trg r ϱ f g f' w' ϑ' ϱ' ν'*
using *assms*(*1*,*3*) **apply** *unfold-locales* **by** *auto*
interpret *T*: *tabulation-data V H* a i *src trg r ϱ' f' g'*
using *assms*(*1*,*2*) **apply** *unfold-locales* **by** *auto*
interpret *T*: *uwϑων V H* a i *src trg r ϱ' f' g' f w ϑ ϱ ν*
using *assms*(*1*,*2*) **apply** *unfold-locales* **by** *auto*

have *dom-ν* [*simp*]: *dom* ν = *dom* ϱ
using *assms*(*2*) **by** *auto*
have *dom-ν'* [*simp*]: *dom* ν' = *dom* ϱ'
using *assms*(*3*) **by** *auto*

let *?ν'ν* = $\mathrm{a}[dom\ \varrho, w', w] \cdot (\nu' \star w) \cdot \nu$
have *ν'ν*: «*?ν'ν* : *dom* $\varrho \Rightarrow$ *dom* $\varrho \star w' \star w$»
by *fastforce*
have «ν : *src* $\varrho \rightarrow$ *trg* r» **by** *simp*
let *?ϑϑ'* = $\vartheta \cdot (\vartheta' \star w) \cdot \mathrm{a}^{-1}[f, w', w]$
have *ϑϑ'*: «*?ϑϑ'* : $f \star w' \star w \Rightarrow f$»
by *fastforce*
have *iso-ν'ν-r*: *iso* (*?ν'ν* $\cdot$ r[g])
using *iso-runit ν'ν*
apply (*intro isos-compose*) **by** *auto*

have *eq*: *composite-cell* (*src f*) r[f] = *composite-cell* ($w' \star w$) *?ϑϑ'* $\cdot$ (*?ν'ν* $\cdot$ r[g])
proof −
have *composite-cell* ($w' \star w$) *?ϑϑ'* $\cdot$ (*?ν'ν* $\cdot$ r[g]) =
$((r \star \vartheta) \cdot (r \star \vartheta' \star w) \cdot (r \star \mathrm{a}^{-1}[f, w', w])) \cdot$
$\mathrm{a}[r, f, w' \star w] \cdot ((\varrho \star w' \star w) \cdot \mathrm{a}[g, w', w]) \cdot (\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$
using *whisker-left comp-assoc* **by** *simp*
also have ... = $((r \star \vartheta) \cdot (r \star \vartheta' \star w) \cdot (r \star \mathrm{a}^{-1}[f, w', w])) \cdot$
$\mathrm{a}[r, f, w' \star w] \cdot (\mathrm{a}[r \star f, w', w] \cdot$
$((\varrho \star w') \star w)) \cdot (\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$

using *assoc-naturality* [*of* ϱ w' w] **by** *simp*
also have ... $= (r \star \vartheta) \cdot (r \star \vartheta' \star w) \cdot$
$((r \star \mathrm{a}^{-1}[f, w', w]) \cdot \mathrm{a}[r, f, w' \star w] \cdot \mathrm{a}[r \star f, w', w]) \cdot$
$((\varrho \star w') \star w) \cdot (\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$
using *comp-assoc* **by** *simp*
also have ... $= (r \star \vartheta) \cdot ((r \star \vartheta' \star w) \cdot \mathrm{a}[r, f \star w', w]) \cdot$
$(\mathrm{a}[r, f, w'] \star w) \cdot$
$((\varrho \star w') \star w) \cdot (\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$
proof −
have *seq* $\mathrm{a}[r, f, w' \star w]$ $\mathrm{a}[r \star f, w', w]$ **by** *simp*
moreover have *inv* $(r \star \mathrm{a}[f, w', w]) = r \star \mathrm{a}^{-1}[f, w', w]$
by *simp*
moreover have $(r \star \mathrm{a}[f, w', w]) \cdot \mathrm{a}[r, f \star w', w] \cdot (\mathrm{a}[r, f, w'] \star w) =$
$\mathrm{a}[r, f, w' \star w] \cdot \mathrm{a}[r \star f, w', w]$
using *pentagon* **by** *simp*
ultimately have $(r \star \mathrm{a}^{-1}[f, w', w]) \cdot \mathrm{a}[r, f, w' \star w] \cdot \mathrm{a}[r \star f, w', w] =$
$\mathrm{a}[r, f \star w', w] \cdot (\mathrm{a}[r, f, w'] \star w)$
using *iso-assoc* [*of f w′ w*] *iso-hcomp*
invert-side-of-triangle(*1*)
[*of* $\mathrm{a}[r, f, w' \star w] \cdot \mathrm{a}[r \star f, w', w]$ $r \star \mathrm{a}[f, w', w]$
$\mathrm{a}[r, f \star w', w] \cdot (\mathrm{a}[r, f, w'] \star w)$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot$
$(((r \star \vartheta') \star w) \cdot (\mathrm{a}[r, f, w'] \star w) \cdot ((\varrho \star w') \star w)) \cdot$
$(\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$
proof −
have $(r \star \vartheta' \star w) \cdot \mathrm{a}[r, f \star w', w] = \mathrm{a}[r, f', w] \cdot ((r \star \vartheta') \star w)$
using *assoc-naturality* [*of r ϑ′ w*] **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot (\textit{composite-cell}\ w'\ \vartheta' \star w) \cdot (\nu' \star w) \cdot \nu \cdot \mathrm{r}[g]$
using *whisker-right*
by (*metis T′.uwϑω T′.w-in-hom*(*1*) *composite-cell-in-hom T′.ϑ-simps*(*2*) *T′.ide-w T.ide-w arrI seqE*)
also have ... $= (r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot ((\varrho' \cdot \textit{inv}\ \nu' \star w) \cdot (\nu' \star w)) \cdot \nu \cdot \mathrm{r}[g]$
proof −
have *composite-cell* $w'\ \vartheta' = \varrho' \cdot \textit{inv}\ \nu'$
using *assms invert-side-of-triangle*(*2*) *T.tab-simps*(*1*) *comp-assoc* **by** *presburger*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (T.\textit{composite-cell}\ w\ \vartheta \cdot \nu) \cdot \mathrm{r}[g]$
using *whisker-right* [*of w* $\varrho' \cdot \textit{inv}\ \nu'$ ν'] *dom-ν′ comp-assoc comp-inv-arr′*
comp-arr-dom
by *simp*

also have ... $= \varrho \cdot \mathrm{r}[g]$
using *assms*(*2*) *comp-assoc* **by** *simp*
also have ... $=$ *composite-cell* (*src f*) $\mathrm{r}[f]$
using *comp-assoc runit-hcomp runit-naturality* [*of* ϱ] **by** *simp*
finally show *?thesis* **by** *simp*
qed
have *eq'*: $(r \star \mathrm{r}[f]) \cdot \mathrm{a}[r, f, src\ f] \cdot (\varrho \star src\ f) \cdot (inv\ (?\nu'\nu \cdot \mathrm{r}[g])) =$
composite-cell $(w' \star w)$ $?\vartheta\vartheta'$
proof −
have *1*: *composite-cell* (*src f*) $\mathrm{r}[f] =$ (*composite-cell* $(w' \star w)$ $?\vartheta\vartheta'$) $\cdot$ $?\nu'\nu \cdot \mathrm{r}[g]$
using *eq comp-assoc* **by** *simp*
have *composite-cell* (*src f*) $\mathrm{r}[f] \cdot (inv\ (?\nu'\nu \cdot \mathrm{r}[g])) =$ *composite-cell* $(w' \star w)$ $?\vartheta\vartheta'$
proof −
have *seq* $(r \star \mathrm{r}[f])$ $(\mathrm{a}[r, f, src\ f] \cdot (\varrho \star src\ f))$
by *fastforce*
thus *?thesis*
using *iso-ν'ν-r 1 invert-side-of-triangle*(*2*) **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed

have *ν'ν-r*: «$?\nu'\nu \cdot \mathrm{r}[g] : g \star src\ f \Rightarrow g \star w' \star w$»
by *force*
have *inv-ν'ν-r*: «$inv\ (?\nu'\nu \cdot \mathrm{r}[g]) : g \star w' \star w \Rightarrow g \star src\ f$»
using *ν'ν iso-ν'ν-r* **by** *auto*

let $?P = \lambda\gamma.$ «$\gamma : src\ f \Rightarrow w' \star w$» $\wedge$ $?\nu'\nu \cdot \mathrm{r}[g] = dom\ \varrho \star \gamma \wedge \mathrm{r}[f] = ?\vartheta\vartheta' \cdot (f \star \gamma)$
let $?\gamma = THE\ \gamma.\ ?P\ \gamma$
have $?P\ ?\gamma$
proof −
have $\exists!\gamma.\ ?P\ \gamma$
using *ν'ν-r ϑϑ' eq T2* [*of src f* $w' \star w$ $\mathrm{r}[f]$ *f* $?\vartheta\vartheta'$ $?\nu'\nu \cdot \mathrm{r}[g]$] **by** *simp*
thus *?thesis*
using *the1-equality* [*of ?P*] **by** *blast*
qed
hence γ: «$?\gamma : src\ f \rightarrow src\ f$» $\wedge$ $?P\ ?\gamma$
using *vconn-implies-hpar*(*1*−*2*) **by** *auto*

let $?P' = \lambda\gamma.$ «$\gamma : w' \star w \Rightarrow src\ f$» $\wedge$ $inv\ (?\nu'\nu \cdot \mathrm{r}[g]) = g \star \gamma \wedge ?\vartheta\vartheta' = \mathrm{r}[f] \cdot (f \star \gamma)$
let $?\gamma' = THE\ \gamma.\ ?P'\ \gamma$
have $?P'\ ?\gamma'$
proof −
have $\exists!\gamma.\ ?P'\ \gamma$
using *inv-ν'ν-r ϑϑ' eq'*
T2 [*of* $w' \star w$ *src f* $\vartheta \cdot (\vartheta' \star w) \cdot \mathrm{a}^{-1}[f, w', w]$ *f*] *comp-assoc*
by *simp*
thus *?thesis*
using *the1-equality* [*of ?P'*] **by** *blast*

```
qed
hence γ′: «?γ′ : src f → src f» ∧ ?P′ ?γ′
  using vconn-implies-hpar(1−2) by auto

have inverse-arrows ?γ ?γ′
proof
  let ?Q = λγ. «γ : src f ⇒ src f» ∧ dom ϱ ⋆ src f = g ⋆ γ ∧ r[f] = r[f] · (f ⋆ γ)
  have ∃!γ. ?Q γ
  proof −
    have ide (src f) by simp
    moreover have «r[f] : f ⋆ src f ⇒ f» by simp
    moreover have «dom ϱ ⋆ src f : g ⋆ src f ⇒ g ⋆ src f» by auto
    moreover have (ϱ ⋆ src f) · (dom ϱ ⋆ src f) = ϱ ⋆ src f
    proof −
      have (ϱ ⋆ src ϱ) · (dom ϱ ⋆ src (dom ϱ)) = ϱ ⋆ src ϱ
        using R.as-nat-trans.naturality1 arr-dom tab-simps(1) by presburger
      thus ?thesis
        by simp
    qed
    ultimately show ?thesis
      using comp-arr-dom T2 [of src f src f r[f] f r[f] dom ϱ ⋆ src f]
            comp-assoc
      by metis
  qed
  moreover have ?Q (src f)
    using comp-arr-dom by auto
  moreover have ?Q (?γ′ · ?γ)
  proof (intro conjI)
    show «?γ′ · ?γ : src f ⇒ src f»
      using γ γ′ by auto
    show dom ϱ ⋆ src f = g ⋆ ?γ′ · ?γ
    proof −
      have g ⋆ ?γ′ · ?γ = (g ⋆ ?γ′) · (g ⋆ ?γ)
        using γ γ′ whisker-left by fastforce
      also have ... = inv (?ν′ν · r[g]) · (?ν′ν · r[g])
        using γ γ′ by simp
      also have ... = dom ϱ ⋆ src f
        using ν′ν iso-ν′ν-r comp-inv-arr inv-is-inverse by auto
      finally show ?thesis by simp
    qed
    show r[f] = r[f] · (f ⋆ ?γ′ · ?γ)
    proof −
      have r[f] · (f ⋆ ?γ′ · ?γ) = r[f] · (f ⋆ ?γ′) · (f ⋆ ?γ)
        using γ γ′ whisker-left by fastforce
      also have ... = (r[f] · (f ⋆ ?γ′)) · (f ⋆ ?γ)
        using comp-assoc by simp
      also have ... = r[f]
        using γ γ′ by simp
      finally show ?thesis by simp
```

```
  qed
qed
ultimately have ?γ' · ?γ = src f by blast
thus ide (?γ' · ?γ) by simp

let ?Q' = λγ. «γ : w' ⋆ w ⇒ w' ⋆ w» ∧ g ⋆ w' ⋆ w = g ⋆ γ ∧ ?ϑϑ' = ?ϑϑ' · (f ⋆ γ)
have ∃!γ. ?Q' γ
proof −
  have ide (w' ⋆ w) by simp
  moreover have «?ϑϑ' : f ⋆ w' ⋆ w ⇒ f»
    using ϑϑ' by simp
  moreover have «g ⋆ w' ⋆ w : g ⋆ w' ⋆ w ⇒ g ⋆ w' ⋆ w»
    by auto
  moreover have
    composite-cell (w' ⋆ w) ?ϑϑ' = composite-cell (w' ⋆ w) ?ϑϑ' · (g ⋆ w' ⋆ w)
  proof −
    have «ϱ ⋆ w' ⋆ w : g ⋆ w' ⋆ w ⇒ (r ⋆ f) ⋆ w' ⋆ w»
      by (intro hcomp-in-vhom, auto)
    hence (ϱ ⋆ w' ⋆ w) · (g ⋆ w' ⋆ w) = ϱ ⋆ w' ⋆ w
      using comp-arr-dom by auto
    thus ?thesis
      using comp-assoc by simp
  qed
  ultimately show ?thesis
    using T2 by presburger
qed
moreover have ?Q' (w' ⋆ w)
  using ϑϑ' comp-arr-dom by auto
moreover have ?Q' (?γ · ?γ')
proof (intro conjI)
  show «?γ · ?γ' : w' ⋆ w ⇒ w' ⋆ w»
    using γ γ' by auto
  show g ⋆ w' ⋆ w = g ⋆ ?γ · ?γ'
  proof −
    have g ⋆ ?γ · ?γ' = (g ⋆ ?γ) · (g ⋆ ?γ')
      using γ γ' whisker-left by fastforce
    also have ... = (?ν'ν · r[g]) · inv (?ν'ν · r[g])
      using γ γ' by simp
    also have ... = g ⋆ w' ⋆ w
      using ν'ν iso-ν'ν-r comp-arr-inv inv-is-inverse by auto
    finally show ?thesis by simp
  qed
  show ?ϑϑ' = ?ϑϑ' · (f ⋆ ?γ · ?γ')
  proof −
    have ?ϑϑ' · (f ⋆ ?γ · ?γ') = ?ϑϑ' · (f ⋆ ?γ) · (f ⋆ ?γ')
      using γ γ' whisker-left by fastforce
    also have ... = (?ϑϑ' · (f ⋆ ?γ)) · (f ⋆ ?γ')
      using comp-assoc by simp
    also have ... = ?ϑϑ'
```

```
        using γ γ' by simp
      finally show ?thesis by simp
    qed
  qed
  ultimately have ?γ · ?γ' = w' ⋆ w by blast
  thus ide (?γ · ?γ') by simp
qed
hence «?γ : src f ⇒ w' ⋆ w» ∧ iso ?γ
  using γ by auto
thus ?thesis by auto
qed
```

Now we can show that, given two tabulations of the same 1-cell, there is an equivalence map between the apexes that extends to a transformation of one tabulation into the other.

```
lemma apex-unique-up-to-equivalence:
assumes tabulation V H a i src trg r ϱ' f' g'
shows ∃ w w' φ ψ ϑ ν ϑ' ν'.
        equivalence-in-bicategory V H a i src trg w' w ψ φ ∧
        «w : src f → src f'» ∧ «w' : src f' → src f» ∧
        «ϑ : f' ⋆ w ⇒ f» ∧ «ν : g ⇒ g' ⋆ w» ∧ iso ν ∧
        ϱ = (r ⋆ ϑ) · a[r, f', w] · (ϱ' ⋆ w) · ν ∧
        «ϑ' : f ⋆ w' ⇒ f'» ∧ «ν' : g' ⇒ g ⋆ w'» ∧ iso ν' ∧
        ϱ' = (r ⋆ ϑ') · a[r, f, w'] · (ϱ ⋆ w') · ν'
proof −
  interpret T': tabulation V H a i src trg r ϱ' f' g'
    using assms by auto
  obtain w ϑ ν
  where wϑν: ide w ∧ «ϑ : f' ⋆ w ⇒ f» ∧ «ν : g ⇒ g' ⋆ w» ∧ iso ν ∧
             ϱ = T'.composite-cell w ϑ · ν
    using T'.T1 [of f ϱ] ide-leg0 tab-in-hom by auto
  obtain w' ϑ' ν'
  where w'ϑ'ν': ide w' ∧ «ϑ' : f ⋆ w' ⇒ f'» ∧ «ν' : g' ⇒ g ⋆ w'» ∧ iso ν' ∧
                ϱ' = composite-cell w' ϑ' · ν'
    using T1 [of f' ϱ'] T'.ide-leg0 T'.tab-in-hom by auto
  obtain φ where φ: «φ : src f ⇒ w' ⋆ w» ∧ iso φ
    using wϑν w'ϑ'ν' apex-equivalence-lemma T'.tab-in-hom comp-assoc by metis
  obtain ψ where ψ: «ψ : src f' ⇒ w ⋆ w'» ∧ iso ψ
    using wϑν w'ϑ'ν' T'.apex-equivalence-lemma tab-in-hom comp-assoc by metis
  have 1: src f = src w
    using φ src-dom [of φ] hcomp-simps(1) [of w' w]
    by (metis arr-cod in-homE leg0-simps(2) src-hcomp src-src vconn-implies-hpar(3))
  have 2: src f' = src w'
    using ψ src-dom [of ψ] hcomp-simps(1) [of w w']
    by (metis T'.leg0-simps(2) arr-cod in-homE src-hcomp src-src vconn-implies-hpar(3))
  interpret E: equivalence-in-bicategory V H a i src trg w' w ψ ‹inv φ›
    using φ ψ 1 2 wϑν w'ϑ'ν' by unfold-locales auto
  have «w : src f → src f'»
    using ψ wϑν 1 2 trg-cod hcomp-simps(2) E.antipar(1) by simp
  moreover have «w' : src f' → src f»
```

using φ $w'\vartheta'\nu'$ *1 2 E.antipar(2)* **by** *simp*
ultimately show *?thesis*
using *E.equivalence-in-bicategory-axioms* $w\vartheta\nu$ $w'\vartheta'\nu'$ *comp-assoc* **by** *metis*
qed

end

2.2.7 'Tabulation' is Bicategorical

In this section we show that "tabulation" is a truly bicategorical notion, in the sense that tabulations are preserved and reflected by equivalence pseudofunctors. The proofs given here is are elementary proofs from first principles. It should also be possible to give a proof based on birepresentations, but for this to actually save work it would first be necessary to carry out a general development of birepresentations and bicategorical limits, and I have chosen not to attempt this here.

context *equivalence-pseudofunctor*
begin

lemma *preserves-tabulation*:
assumes *tabulation* $(\cdot_C)$ $(\star_C)$ a_C i_C src_C trg_C r ϱ f g
shows *tabulation* $(\cdot_D)$ $(\star_D)$ a_D i_D src_D trg_D $(F\ r)$ $(D.inv\ (\Phi\ (r, f))\ \cdot_D\ F\ \varrho)$ $(F\ f)$ $(F\ g)$
proof −
let $?\varrho' = D.inv\ (\Phi\ (r, f))\ \cdot_D\ F\ \varrho$
interpret T: *tabulation* V_C H_C a_C i_C src_C trg_C r ϱ f
using *assms* **by** *auto*
interpret T': *tabulation-data* V_D H_D a_D i_D src_D trg_D ‹$F\ r$› $?\varrho'$ ‹$F\ f$› ‹$F\ g$›
using *cmp-in-hom* Φ*.components-are-iso* $C.VV.ide\text{-}char_{SbC}$ $C.VV.arr\text{-}char_{SbC}$
apply *unfold-locales*
apply *auto*
by (*intro D.comp-in-homI*, *auto*)
interpret T': *tabulation* V_D H_D a_D i_D src_D trg_D ‹$F\ r$› $?\varrho'$ ‹$F\ f$› ‹$F\ g$›

How bad can it be to just show this directly from first principles? It is worse than it at first seems, once you start filling in the details!

proof
fix u' ω'
assume u': $D.ide\ u'$
assume ω': «$\omega' : D.dom\ \omega' \Rightarrow_D F\ r \star_D u'$»
show $\exists\, w'\ \vartheta'\ \nu'.\ D.ide\ w' \wedge$ «$\vartheta' : F\ f \star_D w' \Rightarrow_D u'$» $\wedge$
«$\nu' : D.dom\ \omega' \Rightarrow_D F\ g \star_D w'$» $\wedge$ $D.iso\ \nu' \wedge$
$T'.composite\text{-}cell\ w'\ \vartheta' \cdot_D \nu' = \omega'$
proof −

First, obtain ω in C such that $F\ \omega$ is related to ω' by an equivalence in D.

define v' **where** $v' = D.dom\ \omega'$
have v': $D.ide\ v'$
using *assms* v'*-def D.ide-dom* ω' **by** *blast*
have ω': «$\omega' : v' \Rightarrow_D F\ r \star_D u'$»

using v'-*def* ω' **by** *simp*
define a' **where** $a' = src_D\ \omega'$

have [*simp*]: $src_D\ u' = a'$
using a'-*def* ω'
by (*metis D.arr-cod D.ide-char D.in-homE D.src.preserves-cod D.src-dom D.src-hcomp* v')
have [*simp*]: $trg_D\ u' = src_D\ (F\ r)$
using ω'
by (*metis D.cod-trg D.in-homE D.not-arr-null D.seq-if-composable D.trg.extensionality D.trg.preserves-arr D.trg.preserves-cod*)
have [*simp*]: $src_D\ v' = a'$
using v'-*def* ω' a'-*def* **by** *auto*
have [*simp*]: $trg_D\ v' = trg_D\ (F\ r)$
using v'-*def D.vconn-implies-hpar*(*4*) ω' u' **by** *force*

have [*simp*]: $src_D\ \omega' = a'$
using ω' a'-*def* **by** *blast*
have [*simp*]: $trg_D\ \omega' = trg_D\ (F\ r)$
using ω' v'-*def* ‹$trg_D\ v' = trg_D\ (F\ r)$› **by** *auto*

obtain a **where** a: $C.obj\ a \wedge D.equivalent\text{-}objects\ (map_0\ a)\ a'$
using u' ω' a'-*def biessentially-surjective-on-objects D.obj-src* **by** *blast*
obtain e' **where** e': «$e' : map_0\ a \to_D a'$» $\wedge$ *D.equivalence-map* e'
using a *D.equivalent-objects-def* **by** *auto*

have u'-*in-hhom*: «$u' : a' \to_D map_0\ (src_C\ r)$»
by (*simp add*: u')
hence *1*: «$u' \star_D e' : map_0\ a \to_D map_0\ (src_C\ r)$»
using e' **by** *blast*
have v'-*in-hhom*: «$v' : a' \to_D map_0\ (trg_C\ r)$»
by (*simp add*: v')
hence *2*: «$v' \star_D e' : map_0\ a \to_D map_0\ (trg_C\ r)$»
using e' **by** *blast*

obtain $d'\ \eta'\ \varepsilon'$
where $d'\eta'\varepsilon'$: *adjoint-equivalence-in-bicategory* $(\cdot_D)$ $(\star_D)$ a_D i_D src_D trg_D e' d' η' ε'
using e' *D.equivalence-map-extends-to-adjoint-equivalence* **by** *blast*
interpret e': *adjoint-equivalence-in-bicategory* ‹$(\cdot_D)$› ‹$(\star_D)$› a_D i_D src_D trg_D e' d' η' ε'
using $d'\eta'\varepsilon'$ **by** *auto*
interpret d': *adjoint-equivalence-in-bicategory* ‹$(\cdot_D)$› ‹$(\star_D)$› a_D i_D src_D trg_D d' e' *D.inv* ε' *D.inv* η'
using e'.*dual-adjoint-equivalence* **by** *simp*
have [*simp*]: $src_D\ e' = map_0\ a$
using e' **by** *auto*
have [*simp*]: $trg_D\ e' = a'$
using e' **by** *auto*
have [*simp*]: $src_D\ d' = a'$
by (*simp add*: e'.*antipar*(*2*))

have [*simp*]: $trg_D\ d' = map_0\ a$
  using e'.*antipar* **by** *simp*

obtain u **where** u: «$u : a \rightarrow_C src_C\ r$» $\wedge$ $C.ide\ u \wedge D.isomorphic\ (F\ u)\ (u' \star_D e')$
  using $a\ e'\ u'$ *1* u'-*in-hhom locally-essentially-surjective* [*of* $a\ src_C\ r\ u' \star_D e'$]
    C.*obj-src* D.*equivalence-map-is-ide* T.*base-simps*(*2*)
  by *blast*
obtain φ **where** φ: «$\varphi : u' \star_D e' \Rightarrow_D F\ u$» $\wedge$ $D.iso\ \varphi$
  using u D.*isomorphic-symmetric* **by** *blast*
obtain v **where** v: «$v : a \rightarrow_C trg_C\ r$» $\wedge$ $C.ide\ v \wedge D.isomorphic\ (F\ v)\ (v' \star_D e')$
  using $a\ e'\ v'$ v'-*in-hhom locally-essentially-surjective* [*of* $a\ trg_C\ r\ v' \star_D e'$]
    C.*obj-trg* D.*equivalence-map-is-ide* T.*base-simps*(*2*)
  by *blast*
obtain ψ **where** ψ: «$\psi : F\ v \Rightarrow_D v' \star_D e'$» $\wedge$ $D.iso\ \psi$
  using v **by** *blast*

have [*simp*]: $src_C\ u = a$ **using** u **by** *auto*
have [*simp*]: $trg_C\ u = src_C\ r$ **using** u **by** *auto*
have [*simp*]: $src_C\ v = a$ **using** v **by** *auto*
have [*simp*]: $trg_C\ v = trg_C\ r$ **using** v **by** *auto*
have [*simp*]: $src_D\ \varphi = map_0\ a$
  using φ **by** (*metis 1* D.*dom-src* D.*in-hhomE* D.*in-homE* D.*src.preserves-dom*)
have [*simp*]: $trg_D\ \varphi = trg_D\ u'$
  using φ
  by (*metis* D.*cod-trg* D.*hseqI* D.*in-homE* D.*isomorphic-implies-hpar*(*4*)
    D.*trg.preserves-cod* D.*trg-hcomp* $e'\ u$ u'-*in-hhom*)
have [*simp*]: $src_D\ \psi = map_0\ a$
  using ψ
  by (*metis* C.*in-hhomE* D.*in-homE* D.*src-dom* ‹$src_D\ e' = map_0\ a$› *preserves-src* v)
have [*simp*]: $trg_D\ \psi = trg_D\ v'$
  using ψ
  by (*metis 2* D.*cod-trg* D.*in-hhomE* D.*in-homE* D.*trg.preserves-cod* T.*base-simps*(*2*)
    ‹$trg_D\ v' = trg_D\ (F\ r)$› *preserves-trg*)

define $F\omega$ **where** $F\omega = \Phi\ (r,\ u) \cdot_D (F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D e') \cdot_D \psi$
have $F\omega$: «$F\omega : F\ v \Rightarrow_D F\ (r \star_C u)$»
proof (*unfold* $F\omega$-*def*, *intro* D.*comp-in-homI*)
  show «$\psi : F\ v \Rightarrow_D v' \star_D e'$»
    using ψ **by** *simp*
  show «$\omega' \star_D e' : v' \star_D e' \Rightarrow_D (F\ r \star_D u') \star_D e'$»
    using $e'\ \omega'$ D.*equivalence-map-is-ide* v'-*in-hhom* **by** *blast*
  show «$\mathrm{a}_D[F\ r,\ u',\ e'] : (F\ r \star_D u') \star_D e' \Rightarrow_D F\ r \star_D u' \star_D e'$»
    using $e'\ u'$ D.*equivalence-map-is-ide* D.*in-hhom-def* u'-*in-hhom* **by** *auto*
  show «$F\ r \star_D \varphi : F\ r \star_D u' \star_D e' \Rightarrow_D F\ r \star_D F\ u$»
    using $e'\ u'\ u\ \varphi$
    by (*metis* C.*in-hhomE* D.*hcomp-in-vhom* D.*isomorphic-implies-hpar*(*4*)
      T'.*base-in-hom*(*2*) T.*base-simps*(*2*) *preserves-src preserves-trg*)
  show «$\Phi\ (r,\ u) : F\ r \star_D F\ u \Rightarrow_D F\ (r \star_C u)$»
    using u *cmp-in-hom*(*2*) [*of* $r\ u$] **by** *auto*

qed

obtain ω **where** ω: «ω : $v \Rightarrow_C r \star_C u$» $\wedge$ $F\ \omega = F\omega$
  using $u\ v\ \omega'\ \varphi\ \psi\ F\omega$ *locally-full* [*of* $v\ r \star_C u\ F\omega$]
  by (*metis C.ide-hcomp C.hseqI C.in-hhomE C.src-hcomp C.trg-hcomp*
    T.ide-base T.base-in-hom(*1*))
have [*simp*]: $src_C\ \omega = src_C\ u$
  using ω
  by (*metis C.hseqI C.in-homE C.src-cod C.src-hcomp T.base-in-hom*(*1*) u)
have [*simp*]: $trg_C\ \omega = trg_C\ r$
  using ω
  by (*metis C.ide-char C.ide-trg C.in-homE C.trg.preserves-hom* ‹$trg_C\ v = trg_C\ r$›)

Apply *T.T1* to u and ω to obtain w, ϑ, ν.

obtain $w\ \vartheta\ \nu$
where $w\vartheta\nu$: $C.ide\ w \wedge$ «$\vartheta : f \star_C w \Rightarrow_C u$» $\wedge$ «$\nu : C.dom\ \omega \Rightarrow_C g \star_C w$» $\wedge$
    $C.iso\ \nu \wedge T.composite\text{-}cell\ w\ \vartheta \cdot_C \nu = \omega$
  using $u\ \omega$ *T.T1* [*of* $u\ \omega$] **by** *auto*

Combining ω and $w\vartheta\nu$ yields the situation depicted in the diagram below. In this as well as subsequent diagrams, canonical isomorphisms have been suppressed in the interests of clarity.

$F($

$) \quad = \quad$

have [*simp*]: $src_C\ w = src_C\ u$
  by (*metis C.arrI C.seqE C.src-hcomp C.src-vcomp C.vseq-implies-hpar*(*1*)
    ω ‹$src_C\ \omega = src_C\ u$› $w\vartheta\nu$)
have [*simp*]: $trg_C\ w = src_C\ f$
  by (*metis C.arrI C.hseq-char C.seqE T.tab-simps*(*2*) ω $w\vartheta\nu$)
have [*simp*]: $src_D\ (F\ u) = map_0\ a$
  using e'*.antipar*(*1*) u **by** *auto*
have [*simp*]: $src_D\ (F\ v) = map_0\ a$
  using $v\ e'\ e'$*.antipar* **by** *force*
have [*simp*]: $src_D\ (F\ w) = map_0\ a$
  by (*simp add*: $w\vartheta\nu$)

have $*$: $F\ (T.\textit{composite-cell}\ w\ \vartheta\ \cdot_C\ \nu) =$
$\Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ F\ \vartheta\ \cdot_D\ \Phi\ (f,\ w))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w]\ \cdot_D$
$(D.\textit{inv}\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ F\ w)\ \cdot_D\ D.\textit{inv}\ (\Phi\ (g,\ w))\ \cdot_D\ F\ \nu$

$F($

$$\begin{tikzcd}
 & a \arrow[d, "w"] \arrow[dl, bend right, "v"'] \arrow[dr, bend left, "u"] \arrow[dl, phantom, shift right=3ex, "\overset{\nu}{\Rightarrow}"] \arrow[dr, phantom, shift left=3ex, "\overset{\theta}{\Rightarrow}"] & \\
 & \mathrm{src}\ g = \mathrm{src}\ f \arrow[dl, "g"'] \arrow[dr, "f"] \arrow[d, phantom, "\overset{\rho}{\Rightarrow}"] & \\
\mathrm{trg}\ r & & \mathrm{src}\ r \arrow[ll, "r"]
\end{tikzcd}$$

$)\quad =$

$$\begin{tikzcd}
 & \mathrm{src}(Fa) \arrow[d, "Fw"] \arrow[dl, bend right, "Fv"'] \arrow[dr, bend left, "Fu"] \arrow[dl, phantom, shift right=3ex, "\overset{F\nu}{\Rightarrow}"] \arrow[dr, phantom, shift left=3ex, "\overset{F\theta}{\Rightarrow}"] & \\
 & \mathrm{src}(Fg) = \mathrm{src}(Ff) \arrow[dl, "Fg"'] \arrow[dr, "Ff"] \arrow[d, phantom, "\overset{F\rho}{\Rightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ (Fr) \arrow[ll, "Fr"]
\end{tikzcd}$$

proof $-$
have $F\ (T.\textit{composite-cell}\ w\ \vartheta\ \cdot_C\ \nu) = F\ ((r\ \star_C\ \vartheta)\ \cdot_C\ \mathrm{a}_C[r,\ f,\ w]\ \cdot_C\ (\varrho\ \star_C\ w)\ \cdot_C\ \nu)$
using $C.\textit{comp-assoc}$ **by** *simp*
also have ... $= F\ (r\ \star_C\ \vartheta)\ \cdot_D\ F\ \mathrm{a}_C[r,\ f,\ w]\ \cdot_D\ F\ (\varrho\ \star_C\ w)\ \cdot_D\ F\ \nu$
by (*metis* $C.\textit{arr-dom-iff-arr}$ $C.\textit{comp-assoc}$ $C.\textit{in-homE}$ $C.\textit{seqE}$
$\textit{as-nat-trans.preserves-comp-2}\ w\vartheta\nu$)
also have ... $=$
$F\ (r\ \star_C\ \vartheta)\ \cdot_D\ (\Phi\ (r,\ f\ \star_C\ w)\ \cdot_D\ (F\ r\ \star_D\ \Phi\ (f,\ w))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w]\ \cdot_D$
$(D.\textit{inv}\ (\Phi\ (r,\ f))\ \star_D\ F\ w)\ \cdot_D\ D.\textit{inv}\ (\Phi\ (r\ \star_C\ f,\ w)))\ \cdot_D\ F\ (\varrho\ \star_C\ w)\ \cdot_D\ F\ \nu$
using ω $w\vartheta\nu$ *preserves-assoc* [*of r f w*]
by (*metis* $C.\textit{hseqE}$ $C.\textit{in-homE}$ $C.\textit{seqE}$ $T.\textit{tab-simps}(2)$ $T.\textit{ide-leg0}$ $T.\textit{ide-base}$
$T.\textit{leg0-simps}(3)$)
also have ... $=$
$((F\ (r\ \star_C\ \vartheta)\ \cdot_D\ \Phi\ (r,\ f\ \star_C\ w))\ \cdot_D\ (F\ r\ \star_D\ \Phi\ (f,\ w)))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w]\ \cdot_D$
$((D.\textit{inv}\ (\Phi\ (r,\ f))\ \star_D\ F\ w)\ \cdot_D\ D.\textit{inv}\ (\Phi\ (r\ \star_C\ f,\ w)))\ \cdot_D\ F\ (\varrho\ \star_C\ w)\ \cdot_D\ F\ \nu$
using $D.\textit{comp-assoc}$ **by** *simp*
also have ... $=$
$\Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ F\ \vartheta\ \cdot_D\ \Phi\ (f,\ w))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w]\ \cdot_D$
$((D.\textit{inv}\ (\Phi\ (r,\ f))\ \star_D\ F\ w)\ \cdot_D\ D.\textit{inv}\ (\Phi\ (r\ \star_C\ f,\ w))\ \cdot_D\ F\ (\varrho\ \star_C\ w))\ \cdot_D\ F\ \nu$
proof $-$
have $(F\ (r\ \star_C\ \vartheta)\ \cdot_D\ \Phi\ (r,\ f\ \star_C\ w))\ \cdot_D\ (F\ r\ \star_D\ \Phi\ (f,\ w)) =$
$(\Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ F\ \vartheta)\ \cdot_D\ (F\ r\ \star_D\ \Phi\ (f,\ w)))$
proof $-$
have $F\ (r\ \star_C\ \vartheta)\ \cdot_D\ \Phi\ (r,\ f\ \star_C\ w) = \Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ F\ \vartheta)$
using ω $\Phi.\textit{naturality}$ [*of* $(r,\ \vartheta)$] *FF-def* $w\vartheta\nu$ $C.VV.\textit{arr-char}_{SbC}$
$C.VV.\textit{dom-simp}$ $C.VV.\textit{cod-simp}$
apply *simp*

by (*metis* (*no-types*, *lifting*) *C.hseqE C.in-homE C.seqE*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w))$
proof −
have $(F\ r \star_D F\ \vartheta) \cdot_D (F\ r \star_D \Phi\ (f,\ w)) = F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)$
using ω $w\vartheta\nu$ *D.whisker-right* [*of* $F\ r\ F\ \vartheta\ \Phi\ (f,\ w)$]
by (*metis C.hseqE C.in-homE C.seqE D.comp-ide-self D.interchange D.seqI′ T′.ide-base T′.base-in-hom*(*2*) *T.tab-simps*(*2*) *T.ide-leg0 cmp-in-hom*(*2*) *preserves-hom*)
thus *?thesis* **by** *simp*
qed
finally have $(F\ (r \star_C \vartheta) \cdot_D \Phi\ (r,\ f \star_C w)) \cdot_D (F\ r \star_D \Phi\ (f,\ w)) =$
$\Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w))$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D D.inv\ (\Phi\ (g,\ w))) \cdot_D F\ \nu$
proof −
have $(D.inv\ (\Phi\ (r,\ f)) \star_D F\ w) \cdot_D D.inv\ (\Phi\ (r \star_C f,\ w)) \cdot_D F\ (\varrho \star_C w) =$
$((D.inv\ (\Phi\ (r,\ f)) \star_D F\ w) \cdot_D (F\ \varrho \star_D F\ w)) \cdot_D D.inv\ (\Phi\ (g,\ w))$
proof −
have $D.inv\ (\Phi\ (r \star_C f,\ w)) \cdot_D F\ (\varrho \star_C w) = (F\ \varrho \star_D F\ w) \cdot_D D.inv\ (\Phi\ (g,\ w))$
proof −
have $src_C\ (r \star_C f) = trg_C\ w$
using ω $w\vartheta\nu$
by (*metis C.arrI C.hseq-char C.seqE C.hcomp-simps*(*1*) *T.tab-simps*(*2*) *T.leg0-simps*(*2*) *T.leg0-simps*(*3*))
hence $D.seq\ (\Phi\ (r \star_C f,\ w))\ (F\ \varrho \star_D F\ w)$
using ω $w\vartheta\nu$ *cmp-in-hom*(*2*) [*of* $r \star_C f\ w$] $C.VV.arr\text{-}char_{SbC}$ *FF-def* **by** *auto*
moreover have $\Phi\ (r \star_C f,\ w) \cdot_D (F\ \varrho \star_D F\ w) = F\ (\varrho \star_C w) \cdot_D \Phi\ (g,\ w)$
using ω $w\vartheta\nu$ Φ*.naturality* [*of* $(\varrho,\ w)$] *cmp-components-are-iso FF-def* $C.VV.arr\text{-}char_{SbC}$ *C.VV.dom-simp C.VV.cod-simp*
by *simp*
moreover have $D.iso\ (\Phi\ (r \star_C f,\ w))$
using $w\vartheta\nu$ *cmp-components-are-iso*
by (*metis C.arrI C.ide-hcomp C.hseqE C.hseqI′ C.seqE C.src-hcomp T.tab-simps*(*2*) *T.ide-leg0 T.ide-base T.leg0-simps*(*2*−*3*) ω)
moreover have $D.iso\ (\Phi\ (g,\ w))$
using $w\vartheta\nu$ *cmp-components-are-iso*
by (*metis C.arrI C.hseqE C.seqE T.tab-simps*(*2*) *T.ide-leg1 T.leg1-simps*(*3*) ω)
ultimately show *?thesis*
using ω $w\vartheta\nu$ Φ*.naturality cmp-components-are-iso FF-def* $C.VV.arr\text{-}char_{SbC}$ *D.invert-opposite-sides-of-square*
by *presburger*
qed

```
      thus ?thesis
        using D.comp-assoc by simp
    qed
    also have ... = (D.inv (Φ (r, f)) ·D F ϱ ⋆D F w) ·D D.inv (Φ (g, w))
      using ω wϑν D.whisker-right cmp-components-are-iso cmp-in-hom D.comp-assoc
      by auto
    finally show ?thesis
      using D.comp-assoc by simp
  qed
  finally show ?thesis
    using D.comp-assoc by simp
qed
```

We can now define the w', ϑ', and ν' that we are required to exhibit.

```
define φ' where φ' = e'.trnrε u' (D.inv φ)
have φ' = rD[u'] ·D (u' ⋆D ε') ·D aD[u', e', d'] ·D (D.inv φ ⋆D d')
  unfolding φ'-def e'.trnrε-def by simp
have φ': «φ' : F u ⋆D d' ⇒D u'»
  using φ φ'-def u u' e'.adjoint-transpose-right(2) [of u' F u] by auto

have [simp]: srcD φ' = srcD u'
  using φ' by fastforce
have [simp]: trgD φ' = trgD u'
  using φ' by fastforce

define ψ' where ψ' = d'.trnrη v' (D.inv ψ)
have ψ'-eq: ψ' = (D.inv ψ ⋆D d') ·D aD⁻¹[v', e', d'] ·D (v' ⋆D D.inv ε') ·D rD⁻¹[v']
  unfolding ψ'-def d'.trnrη-def by simp
have ψ': «ψ' : v' ⇒D F v ⋆D d'»
  using ψ ψ'-def v v' d'.adjoint-transpose-right(1) [of F v v'] by auto
have iso-ψ': D.iso ψ'
  unfolding ψ'-def d'.trnrη-def
  using ψ e'.counit-is-iso
  by (metis D.arrI D.iso-hcomp D.hseq-char D.ide-is-iso D.iso-assoc'
      D.iso-inv-iso D.iso-runit' D.isos-compose D.seqE ψ'-eq
      ψ' d'.unit-simps(5) e'.antipar(1) e'.antipar(2) e'.ide-left e'.ide-right v')

have [simp]: srcD ψ' = srcD v'
  using ψ' by fastforce
have [simp]: trgD ψ' = trgD v'
  using ψ' by fastforce

define w' where w' = F w ⋆D d'
define ϑ' where ϑ' = φ' ·D (F ϑ ·D Φ (f, w) ⋆D d') ·D aD⁻¹[F f, F w, d']
define ν' where ν' = aD[F g, F w, d'] ·D (D.inv (Φ (g, w)) ·D F ν ⋆D d') ·D ψ'
have w': D.ide w' ∧ «w' : srcD u' →D srcD (F f)»
  using w'-def ω wϑν by simp
have ϑ': «ϑ' : F f ⋆D w' ⇒D u'»
  unfolding ϑ'-def w'-def
```

using φ' ω $w\vartheta\nu$ *cmp-in-hom*
apply (*intro D.comp-in-homI D.hcomp-in-vhom*)
apply *auto*
by (*intro D.comp-in-homI D.hcomp-in-vhom*, *auto*)
have ν': «$\nu' : v' \Rightarrow_D F\ g \star_D w'$»
unfolding ν'*-def* w'*-def*
using ψ' ω $w\vartheta\nu$ *cmp-in-hom cmp-components-are-iso*
apply (*intro D.comp-in-homI*)
apply *auto*
by (*intro D.hcomp-in-vhom D.comp-in-homI*, *auto*)
have *iso-*ν': *D.iso* ν'
using ν'*-def iso-*ψ' *cmp-in-hom D.isos-compose preserves-iso*
by (*metis* (*no-types*, *lifting*) *C.ideD*(*1*) *D.arrI D.iso-hcomp D.hseqE D.ide-is-iso D.iso-assoc D.iso-inv-iso D.seqE T.ide-leg1 T.leg1-simps*(*3*) *cmp-components-are-iso* ν' ‹src_D $(F\ w) = map_0\ a$› ‹$src_D\ e' = map_0\ a$› ‹$trg_C\ w = src_C\ f$› *e'.antipar*(*1*) *e'.ide-right preserves-ide preserves-src preserves-trg* $w\vartheta\nu$)

have T'*.composite-cell* w' ϑ' $\cdot_D$ $\nu' = \omega'$

$$
\begin{tikzcd}
 & a' \arrow[d, "d'"] \arrow[ddl, bend right=60, "v'"'] \arrow[ddr, bend left=60, "u'"] & \\
 & \mathrm{src}(Fg) = \mathrm{src}(Ff) \arrow[d, "Fw"] \arrow[dl, bend right=40, "Fv"'] \arrow[dr, bend left=40, "Fu"] \arrow[dl, phantom, "\overset{\psi'}{\Rrightarrow}"] \arrow[dr, phantom, "\overset{\phi'}{\Rrightarrow}"] & \\
 & a' \arrow[dl, "Fg"'] \arrow[dr, "Ff"] \arrow[d, phantom, "\overset{F\rho}{\Rrightarrow}"] & \\
\mathrm{trg}\ (Fr) \arrow[rr, leftarrow, "Fr"'] \arrow[uuu, phantom, "\overset{F\nu}{\Rrightarrow}"] \arrow[uuu, phantom, "\overset{F\theta}{\Rrightarrow}"] & & \mathrm{src}\ (Fr)
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
 & a' \arrow[dl, "v'"'] \arrow[dr, "u'"] \arrow[d, phantom, "\overset{\omega'}{\Rrightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ (Fr) \arrow[ll, "Fr"]
\end{tikzcd}
$$

proof −
have *1*: «T'*.composite-cell* w' ϑ' $\cdot_D$ ν' : $v' \Rightarrow_D F\ r \star_D u'$»
using w' ϑ' ν' $w\vartheta\nu$ T'*.composite-cell-in-hom* **by** *blast*
have T'*.composite-cell* w' ϑ' $\cdot_D$ ν' =
$(F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D$
$(F\ (T.composite\text{-}cell\ w\ \vartheta \cdot_C \nu) \star_D d') \cdot_D \psi'$
proof −
have T'*.composite-cell* w' ϑ' $\cdot_D$ ν' =

$(F\ r \star_D \varphi' \cdot_D (F\ \vartheta \cdot_D \Phi\ (f,\ w) \star_D d') \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ w'] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D w') \cdot_D \mathrm{a}_D[F\ g,\ F\ w,\ d'] \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
using *ϑ'-def ν'-def D.comp-assoc* **by** *simp*
also have
$... = (F\ r \star_D \varphi') \cdot_D (F\ r \star_D (F\ \vartheta \cdot_D \Phi\ (f,\ w) \star_D d') \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w \star_D d') \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ w,\ d'] \cdot_D (D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
using *ϑ' ϑ'-def w'-def D.comp-assoc D.whisker-left* **by** *auto*
also have
$... = (F\ r \star_D \varphi') \cdot_D (F\ r \star_D (F\ \vartheta \star_D d') \cdot_D (\Phi\ (f,\ w) \star_D d') \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w \star_D d') \cdot_D$
$\mathrm{a}_D[F\ g,\ F\ w,\ d']) \cdot_D (D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
using *ϑ' ϑ'-def D.whisker-right cmp-in-hom D.comp-assoc* **by** *fastforce*
also have
$... = (F\ r \star_D \varphi') \cdot_D (F\ r \star_D (F\ \vartheta \star_D d') \cdot_D (\Phi\ (f,\ w) \star_D d') \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D$
$\mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d'] \cdot_D ((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have $(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w \star_D d') \cdot_D \mathrm{a}_D[F\ g,\ F\ w,\ d'] =$
$\mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d'] \cdot_D ((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d')$
using *D.assoc-naturality* [*of D.inv* $(\Phi\ (r,\ f)) \cdot_D F\ \varrho\ F\ w\ d'$]
cmp-in-hom cmp-components-are-iso
by (*simp add*: *$w\vartheta\nu$*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = (F\ r \star_D \varphi') \cdot_D (F\ r \star_D F\ \vartheta \star_D d') \cdot_D (F\ r \star_D \Phi\ (f,\ w) \star_D d') \cdot_D$
$((F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D \mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d']) \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
using *1 D.whisker-left D.comp-assoc*
by (*metis D.arrI D.hseq-char D.seqE T'.ide-base calculation*)
also have $... = (F\ r \star_D \varphi') \cdot_D (F\ r \star_D F\ \vartheta \star_D d') \cdot_D ((F\ r \star_D \Phi\ (f,\ w) \star_D d') \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f \star_D F\ w,\ d'])\ \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d') \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have *D.seq* $\mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d']\ \mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d']$
by (*metis 1 D.arrI D.seqE calculation*)
hence $(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ F\ w,\ d'])\ \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D$
$\mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d'] =$
$\mathrm{a}_D[F\ r,\ F\ f \star_D F\ w,\ d'] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d')$
using *$w\vartheta\nu$ D.pentagon*
D.invert-side-of-triangle(*1*)
[*of* $\mathrm{a}_D[F\ r,\ F\ f,\ F\ w \star_D d'] \cdot_D \mathrm{a}_D[F\ r \star_D F\ f,\ F\ w,\ d']$

$F\ r \star_D \mathrm{a}_D[F\ f,\ F\ w,\ d']$
$\mathrm{a}_D[F\ r,\ F\ f \star_D F\ w,\ d'] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d')]$
by (*simp add*: $w\vartheta\nu$)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D ((F\ r \star_D F\ \vartheta \star_D d') \cdot_D \mathrm{a}_D[F\ r,\ F\ (f \star_C w),\ d']) \cdot_D$
$((F\ r \star_D \Phi\ (f,\ w)) \star_D d') \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d') \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have $(F\ r \star_D \Phi\ (f,\ w) \star_D d') \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D F\ w,\ d'] =$
$\mathrm{a}_D[F\ r,\ F\ (f \star_C w),\ d'] \cdot_D ((F\ r \star_D \Phi\ (f,\ w)) \star_D d')$
using *1* $w\vartheta\nu$ *D.assoc-naturality* [*of* $F\ r\ \Phi\ (f,\ w)\ d'$]
‹$trg_C\ w = src_C\ f$› *e′.ide-right*
by (*metis D.arrI D.hseq-char D.ide-char D.seqE T′.base-simps*(*3*)
T′.base-simps(*4*) *T′.leg0-simps*(*3*) *T.ide-leg0 cmp-simps*(*1*−*5*) *w′-def*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (((F\ r \star_D F\ \vartheta) \star_D d') \cdot_D$
$((F\ r \star_D \Phi\ (f,\ w)) \star_D d') \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d') \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d')) \cdot_D \psi'$
proof −
have $src_D\ (F\ r) = trg_D\ (F\ \vartheta)$
using $w\vartheta\nu$ **by** (*metis C.arrI C.hseqE C.seqE* ω *preserves-hseq*)
moreover have $src_D\ (F\ \vartheta) = trg_D\ d'$
using $w\vartheta\nu$ *C.arrI C.vconn-implies-hpar*(*1*) **by** *auto*
ultimately
have $(F\ r \star_D F\ \vartheta \star_D d') \cdot_D \mathrm{a}_D[F\ r,\ F\ (f \star_C w),\ d'] =$
$\mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D ((F\ r \star_D F\ \vartheta) \star_D d')$
using $w\vartheta\nu$ *D.assoc-naturality* [*of* $F\ r\ F\ \vartheta\ d'$] **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$(((F\ r \star_D F\ \vartheta) \cdot_D (F\ r \star_D \Phi\ (f,\ w))) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have $((F\ r \star_D F\ \vartheta) \star_D d') \cdot_D$
$((F\ r \star_D \Phi\ (f,\ w)) \star_D d') \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \star_D d') \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \star_D d') \cdot_D$
$(D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') =$
$(F\ r \star_D F\ \vartheta) \cdot_D (F\ r \star_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu$
$\star_D d'$
proof −

have «$(F\ r \star_D F\ \vartheta) \cdot_D (F\ r \star_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu$:
$F\ v \Rightarrow_D F\ r \star_D F\ u$»
using *wϑν ω cmp-in-hom*
apply (*intro D.comp-in-homI*)
apply *auto*
by (*intro D.hcomp-in-vhom*, *auto*)
hence $D.arr\ ((F\ r \star_D F\ \vartheta) \cdot_D (F\ r \star_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu)$
by *auto*
thus *?thesis*
using *D.whisker-right* **by** *fastforce*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
using *wϑν D.whisker-left cmp-in-hom*
by (*metis D.seqI′ T′.ide-base T.ide-leg0* ‹$trg_C\ w = src_C\ f$› *preserves-hom*)
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u) \cdot_D$
$(F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w))) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have $(D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u)) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) =$
$F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)$
proof −
have $(D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u)) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) =$
$(F\ r \star_D F\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w))$
using *u cmp-components-are-iso*
by (*simp add*: *D.comp-inv-arr′*)
also have ... $= F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)$
using *u ω wϑν cmp-in-hom* ‹$trg_C\ u = src_C\ r$›
D.comp-cod-arr [*of* $F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)\ F\ r \star_D F\ u$]
by (*metis* (*full-types*) ∗ *D.arrI D.cod-comp D.seqE Fω T.ide-base*
cmp-simps(*4*))
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$(D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$

using *D.comp-assoc* **by** *simp*
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D$
$(\Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d') \cdot_D \psi'$
proof −
have $D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d' =$
$(D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D (\Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta \cdot_D \Phi\ (f,\ w)) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D F\ w) \cdot_D$
$D.inv\ (\Phi\ (g,\ w)) \cdot_D F\ \nu \star_D d')$
using *D.whisker-right cmp-in-hom cmp-components-are-iso*
by (*metis* ∗ *D.arrI D.invert-side-of-triangle*(*1*) *Fω T.ide-base ω*
‹$trg_C\ u = src_C\ r$› *e′.ide-right u wϑν*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D$
$(F\ (T.composite\text{-}cell\ w\ \vartheta \cdot_C \nu) \star_D d') \cdot_D \psi'$
using *D.comp-assoc* ∗ **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D$
$(F\ \omega \star_D d') \cdot_D \psi'$
using *wϑν* **by** *simp*
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D d') \cdot_D$
$(\Phi\ (r,\ u) \cdot_D (F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D e') \cdot_D \psi \star_D d') \cdot_D$
ψ'
using *ω Fω-def* **by** *simp*

$$
\begin{tikzcd}
 & a' \arrow[dd, dotted, "d'"] \arrow[ddl, bend right, "Fv"'] \arrow[ddr, bend left, "Fu"] \arrow[d, phantom, "\overset{\psi'}{\Rightarrow}"] \arrow[d, phantom, "\overset{\phi'}{\Rightarrow}"] & \\
 & \mathrm{src}(Fa) \arrow[dl, "Fv"'] \arrow[dr, "Fu"] \arrow[d, phantom, "\overset{F\omega}{\Rightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ (Fr) \arrow[ll, "Fr"]
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
 & a' \arrow[dd, dotted, "d'"] \arrow[ddl, bend right, "v'"'] \arrow[ddr, bend left, "u'"] \arrow[d, phantom, "\overset{\psi'}{\Rightarrow}"] \arrow[d, phantom, "\overset{\phi'}{\Rightarrow}"] & \\
 & \mathrm{src}(Fa) \arrow[dl, bend right, "Fv"'] \arrow[dr, bend left, "Fu"] \arrow[d, dotted, "e'"] \arrow[d, phantom, "\overset{\psi}{\Rightarrow}"] \arrow[d, phantom, "\overset{\phi}{\Rightarrow}"] & \\
 & a' \arrow[dl, "v'"'] \arrow[dr, "u'"] \arrow[d, phantom, "\overset{\omega'}{\Rightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ (Fr) \arrow[ll, "Fr"]
\end{tikzcd}
$$

also have ... $= \omega'$

$$
\begin{tikzcd}
 & a' \arrow[d, "d'"] \arrow[ddl, bend right, "v'"'] \arrow[ddr, bend left, "u'"] \arrow[d, phantom, "\overset{\psi'}{\Rightarrow}"] \arrow[d, phantom, "\overset{\phi'}{\Rightarrow}"] & \\
 & \mathrm{src}(Fa) \arrow[dl, bend right, "Fv"'] \arrow[d, "e'"] \arrow[dr, bend left, "Fu"] \arrow[d, phantom, "\overset{\psi}{\Rightarrow}"] \arrow[d, phantom, "\overset{\phi}{\Rightarrow}"] & \\
 & a' \arrow[dl, "v'"'] \arrow[dr, "u'"] \arrow[d, phantom, "\overset{\omega'}{\Rightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ (Fr) \arrow[ll, "Fr"]
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
 & a' \arrow[dl, "v'"'] \arrow[dr, "u'"] \arrow[d, phantom, "\overset{\omega'}{\Rightarrow}"] & \\
\mathrm{trg}\ (Fr) & & \mathrm{src}\ )(Fr) \arrow[ll, "Fr"]
\end{tikzcd}
$$

proof –

have $(F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D (D.inv\ (\Phi\ (r,\ u)) \star_D\ d') \cdot_D$
$(\Phi\ (r,\ u) \cdot_D (F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi' =$
$(F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((D.inv\ (\Phi\ (r,\ u)) \star_D\ d') \cdot_D (\Phi\ (r,\ u) \star_D\ d')) \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi'$
using *D.whisker-right cmp-in-hom D.comp-assoc*
by (*metis D.arrI Fω Fω-def e′.ide-right*)
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi'$
proof −
have $(D.inv\ (\Phi\ (r,\ u)) \star_D\ d') \cdot_D (\Phi\ (r,\ u) \star_D\ d') =$
$D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u) \star_D\ d'$
using *cmp-in-hom cmp-components-are-iso D.whisker-right*
by (*metis C.hseqI D.comp-arr-inv′ D.in-homE D.invert-opposite-sides-of-square D.iso-inv-iso T.ide-base T.base-in-hom(1)* ‹$trg_C\ u = src_C\ r$› *e′.ide-right preserves-arr u*)
also have ... $= (F\ r \star_D\ F\ u) \star_D\ d'$
using *u cmp-components-are-iso D.comp-inv-arr′* **by** *simp*
finally have $(F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((D.inv\ (\Phi\ (r,\ u)) \star_D\ d') \cdot_D (\Phi\ (r,\ u) \star_D\ d')) \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi' =$
$(F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D ((F\ r \star_D\ F\ u) \star_D\ d') \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi'$
by *simp*
also have ... $= (F\ r \star_D \varphi') \cdot_D (\mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D ((F\ r \star_D\ F\ u) \star_D\ d')) \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi'$
using *D.comp-assoc* **by** *auto*
also have ... $= (F\ r \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d') \cdot_D \psi'$
using *u D.comp-arr-dom* **by** *simp*
finally show *?thesis* **by** *blast*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D (\mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D$
$((F\ r \star_D \varphi) \star_D\ d')) \cdot_D (\mathrm{a}_D[F\ r,\ u',\ e'] \star_D\ d') \cdot_D$
$((\omega' \star_D\ e') \star_D\ d') \cdot_D (\psi \star_D\ d') \cdot_D \psi'$
proof −
have
$(F\ r \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e'] \cdot_D (\omega' \star_D\ e') \cdot_D \psi \star_D\ d' =$
$((F\ r \star_D \varphi) \star_D\ d') \cdot_D (\mathrm{a}_D[F\ r,\ u',\ e'] \star_D\ d') \cdot_D ((\omega' \star_D\ e') \star_D\ d') \cdot_D (\psi \star_D\ d')$
using *D.whisker-right φ φ′ e′ e′.antipar(1) u′ u′-in-hhom*
by (*metis D.arrI D.seqE Fω Fω-def e′.ide-right*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \varphi') \cdot_D (F\ r \star_D \varphi \star_D\ d') \cdot_D \mathrm{a}_D[F\ r,\ u' \star_D\ e',\ d'] \cdot_D$
$((\mathrm{a}_D[F\ r,\ u',\ e'] \star_D\ d') \cdot_D ((\omega' \star_D\ e') \star_D\ d')) \cdot_D (\psi \star_D\ d') \cdot_D \psi'$
proof −
have $\mathrm{a}_D[F\ r,\ F\ u,\ d'] \cdot_D ((F\ r \star_D \varphi) \star_D\ d') =$
$(F\ r \star_D \varphi \star_D\ d') \cdot_D \mathrm{a}_D[F\ r,\ u' \star_D\ e',\ d']$

using *D.assoc-naturality* [*of* $F\ r\ \varphi\ d'$] φ **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $(F\ r \star_D \varphi') \cdot_D (F\ r \star_D \varphi \star_D d') \cdot_D \mathrm{a}_D[F\ r,\ u' \star_D e',\ d'] \cdot_D$
$((\mathrm{a}_D[F\ r,\ u',\ e'] \star_D d') \cdot_D (\mathrm{a}_D{}^{-1}[F\ r \star_D u',\ e',\ d'] \cdot_D$
$(\omega' \star_D e' \star_D d') \cdot_D \mathrm{a}_D[v',\ e',\ d'])) \cdot_D (\psi \star_D d') \cdot_D \psi'$
using *Fω Fω-def ω′ D.comp-assoc D.hcomp-reassoc*(*1*) [*of* $\omega'\ e'\ d'$]
by (*elim D.in-homE*, *simp*)
also have ... = $(F\ r \star_D \varphi') \cdot_D (F\ r \star_D \varphi \star_D d') \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u',\ e',\ d']) \cdot_D$
$\mathrm{a}_D[F\ r,\ u',\ e' \star_D d'] \cdot_D (\omega' \star_D e' \star_D d') \cdot_D \mathrm{a}_D[v',\ e',\ d'] \cdot_D$
$(\psi \star_D d') \cdot_D \psi'$
proof −
have *D.seq* $(F\ r \star_D \mathrm{a}_D[u',\ e',\ d'])$
$(\mathrm{a}_D[F\ r,\ u' \star_D e',\ d'] \cdot_D (\mathrm{a}_D[F\ r,\ u',\ e'] \star_D d'))$
using u' **by** *simp*
moreover have $(F\ r \star_D \mathrm{a}_D[u',\ e',\ d']) \cdot_D \mathrm{a}_D[F\ r,\ u' \star_D e',\ d'] \cdot_D$
$(\mathrm{a}_D[F\ r,\ u',\ e'] \star_D d') =$
$\mathrm{a}_D[F\ r,\ u',\ e' \star_D d'] \cdot_D \mathrm{a}_D[F\ r \star_D u',\ e',\ d']$
using u' *D.pentagon* **by** *simp*
moreover have *D.iso* $(F\ r \star_D \mathrm{a}_D[u',\ e',\ d'])$
using u' **by** *simp*
moreover have *D.inv* $(F\ r \star_D \mathrm{a}_D[u',\ e',\ d']) = F\ r \star_D \mathrm{a}_D{}^{-1}[u',\ e',\ d']$
using u' **by** *simp*
ultimately
have $\mathrm{a}_D[F\ r,\ u' \star_D e',\ d'] \cdot_D (\mathrm{a}_D[F\ r,\ u',\ e'] \star_D d') \cdot_D \mathrm{a}_D{}^{-1}[F\ r \star_D u',\ e',\ d'] =$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[u',\ e',\ d']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e' \star_D d']$
using u' *D.comp-assoc*
D.invert-opposite-sides-of-square
[*of* $F\ r \star_D \mathrm{a}_D[u',\ e',\ d']$
$\mathrm{a}_D[F\ r,\ u' \star_D e',\ d'] \cdot_D (\mathrm{a}_D[F\ r,\ u',\ e'] \star_D d')$
$\mathrm{a}_D[F\ r,\ u',\ e' \star_D d']\ \mathrm{a}_D[F\ r \star_D u',\ e',\ d']$]
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *metis*
qed
also have
... = $(F\ r \star_D \mathrm{r}_D[u'] \cdot_D (u' \star_D \varepsilon') \cdot_D \mathrm{a}_D[u',\ e',\ d'] \cdot_D (D.inv\ \varphi \star_D d')) \cdot_D$
$(F\ r \star_D \varphi \star_D d') \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u',\ e',\ d']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e' \star_D d'] \cdot_D$
$(\omega' \star_D e' \star_D d') \cdot_D \mathrm{a}_D[v',\ e',\ d'] \cdot_D (\psi \star_D d') \cdot_D (D.inv\ \psi \star_D d') \cdot_D$
$\mathrm{a}_D{}^{-1}[v',\ e',\ d'] \cdot_D (v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
unfolding *φ′-def ψ′-def e′.trnr*$_\varepsilon$*-def d′.trnr*$_\eta$*-def* **by** *simp*
also have
... = $(F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D u' \star_D \varepsilon') \cdot_D (F\ r \star_D \mathrm{a}_D[u',\ e',\ d']) \cdot_D$
$(F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d') \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[u',\ e',\ d']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ e' \star_D d'] \cdot_D (\omega' \star_D e' \star_D d') \cdot_D$
$\mathrm{a}_D[v',\ e',\ d'] \cdot_D (\psi \star_D d') \cdot_D (D.inv\ \psi \star_D d') \cdot_D \mathrm{a}_D{}^{-1}[v',\ e',\ d'] \cdot_D$
$(v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −

have $F\ r \star_D \mathrm{r}_D[u'] \cdot_D (u' \star_D \varepsilon') \cdot_D \mathrm{a}_D[u', e', d'] \cdot_D (D.inv\ \varphi \star_D d') =$
$(F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D u' \star_D \varepsilon') \cdot_D (F\ r \star_D \mathrm{a}_D[u', e', d']) \cdot_D$
$(F\ r \star_D D.inv\ \varphi \star_D d')$
proof −
have $D.ide\ (F\ r)$ **by** *simp*
moreover have $D.seq\ \mathrm{r}_D[u']\ ((u' \star_D \varepsilon') \cdot_D \mathrm{a}_D[u', e', d'] \cdot_D (D.inv\ \varphi \star_D d')) \wedge$
$D.seq\ (u' \star_D \varepsilon')\ (\mathrm{a}_D[u', e', d'] \cdot_D (D.inv\ \varphi \star_D d')) \wedge$
$D.seq\ \mathrm{a}_D[u', e', d']\ (D.inv\ \varphi \star_D d')$
using φ' φ'*-def* **unfolding** $e'.trnr_\varepsilon$*-def* **by** *blast*
ultimately show *?thesis*
using *D.whisker-left* **by** *metis*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = (F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D u' \star_D \varepsilon') \cdot_D (F\ r \star_D \mathrm{a}_D[u', e', d']) \cdot_D$
$(((F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d')) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d'])) \cdot_D \mathrm{a}_D[F\ r, u', e' \star_D d'] \cdot_D (\omega' \star_D e' \star_D d') \cdot_D$
$\mathrm{a}_D[v', e', d'] \cdot_D (((\psi \star_D d') \cdot_D (D.inv\ \psi \star_D d')) \cdot_D \mathrm{a}_D{}^{-1}[v', e', d']) \cdot_D$
$(v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
using *D.comp-assoc* **by** *simp*
also have
$... = (F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D u' \star_D \varepsilon') \cdot_D (F\ r \star_D \mathrm{a}_D[u', e', d']) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d']) \cdot_D \mathrm{a}_D[F\ r, u', e' \star_D d'] \cdot_D (\omega' \star_D e' \star_D d') \cdot_D$
$((\mathrm{a}_D[v', e', d'] \cdot_D \mathrm{a}_D{}^{-1}[v', e', d']) \cdot_D (v' \star_D D.inv\ \varepsilon')) \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $((F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d')) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d']) =$
$F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d']$
proof −
have $(F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d') = F\ r \star_D D.inv\ \varphi \cdot_D \varphi \star_D d'$
using *u u' φ 1 2 D.src-dom e'.antipar D.whisker-left D.whisker-right*
by *auto*
also have $... = F\ r \star_D (u' \star_D e') \star_D d'$
using φ *D.comp-inv-arr'* **by** *auto*
finally have
$(F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d') = F\ r \star_D (u' \star_D e') \star_D d'$
by *simp*
hence
$((F\ r \star_D D.inv\ \varphi \star_D d') \cdot_D (F\ r \star_D \varphi \star_D d')) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d']) =$
$(F\ r \star_D (u' \star_D e') \star_D d') \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d'])$
using *D.comp-assoc* **by** *simp*
also have $... = F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d']$
proof −
have «$F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d'] :$
$F\ r \star_D u' \star_D e' \star_D d' \Rightarrow_D F\ r \star_D (u' \star_D e') \star_D d'$»
using u' $e'.antipar$ φ' $D.assoc'$*-in-hom*
unfolding $e'.trnr_\varepsilon$*-def*
by (*intro D.hcomp-in-vhom*, *auto*)

thus *?thesis*
using *D.comp-cod-arr* **by** *blast*
qed
finally show *?thesis* **by** *simp*
qed
moreover have
$((\psi \star_D d') \cdot_D (D.inv\ \psi \star_D d')) \cdot_D \mathrm{a}_D{}^{-1}[v', e', d'] = \mathrm{a}_D{}^{-1}[v', e', d']$
proof −
have $(\psi \star_D d') \cdot_D (D.inv\ \psi \star_D d') = (v' \star_D e') \star_D d'$
using ψ *e'.antipar D.src-cod v' e'.antipar* ψ' *d'.trnr*$_\eta$*-def*
D.whisker-right [*of d'* ψ *D.inv* ψ] *D.comp-arr-inv'*
by *auto*
moreover have «$\mathrm{a}_D{}^{-1}[v', e', d'] : v' \star_D e' \star_D d' \Rightarrow_D (v' \star_D e') \star_D d'$»
using *v' e'.antipar* ψ' *D.assoc'-in-hom*
unfolding *d'.trnr*$_\eta$*-def*
by *fastforce*
ultimately show *?thesis*
using *D.comp-cod-arr* **by** *auto*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D u' \star_D \varepsilon') \cdot_D (((F\ r \star_D \mathrm{a}_D[u', e', d']) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d'])) \cdot_D \mathrm{a}_D[F\ r, u', e' \star_D d']) \cdot_D$
$(\omega' \star_D e' \star_D d') \cdot_D (v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $(\mathrm{a}_D[v', e', d'] \cdot_D \mathrm{a}_D{}^{-1}[v', e', d']) \cdot_D (v' \star_D D.inv\ \varepsilon') = v' \star_D D.inv\ \varepsilon'$
proof −
have *1*: *D.hseq v' e'*
using *v' e'.antipar* ψ' **unfolding** *d'.trnr*$_\eta$*-def* **by** *fastforce*
have $\mathrm{a}_D[v', e', d'] \cdot_D \mathrm{a}_D{}^{-1}[v', e', d'] = v' \star_D e' \star_D d'$
using *v' e'.antipar 1 D.comp-assoc-assoc'* **by** *auto*
moreover have «$v' \star_D D.inv\ \varepsilon' : v' \star_D trg_D\ e' \Rightarrow_D v' \star_D e' \star_D d'$»
using *v' e'.antipar 1*
apply (*intro D.hcomp-in-vhom*)
apply *auto*
by (*metis D.ideD*(*1*) *D.trg-src* ‹$trg_D\ e' = a'$› *e'.antipar*(*2*) *e'.ide-right*)
ultimately show *?thesis*
using *D.comp-cod-arr* **by** *auto*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D ((F\ r \star_D u' \star_D \varepsilon') \cdot_D \mathrm{a}_D[F\ r, u', e' \star_D d']) \cdot_D$
$(\omega' \star_D e' \star_D d') \cdot_D (v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $((F\ r \star_D \mathrm{a}_D[u', e', d']) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[u', e', d'])) \cdot_D$
$\mathrm{a}_D[F\ r, u', e' \star_D d'] =$
$\mathrm{a}_D[F\ r, u', e' \star_D d']$

using φ u' e'.*antipar 1 D.comp-cod-arr D.comp-assoc-assoc'*
D.whisker-left [*of F r* $\mathrm{a}_D[u', e', d']$ $\mathrm{a}_D{}^{-1}[u', e', d']$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ trg_D\ e'] \cdot_D (((F\ r \star_D u') \star_D \varepsilon') \cdot_D$
$(\omega' \star_D e' \star_D d')) \cdot_D (v' \star_D D.inv\ \varepsilon') \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $(F\ r \star_D u' \star_D \varepsilon') \cdot_D \mathrm{a}_D[F\ r,\ u',\ e' \star_D d'] =$
$\mathrm{a}_D[F\ r,\ u',\ trg_D\ e'] \cdot_D ((F\ r \star_D u') \star_D \varepsilon')$
using *D.assoc-naturality* [*of F r* u' ε'] e' u' *u'-in-hhom* **by** *force*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ trg_D\ e'] \cdot_D (\omega' \star_D trg_D\ e') \cdot_D$
$((v' \star_D \varepsilon') \cdot_D (v' \star_D D.inv\ \varepsilon')) \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $((F\ r \star_D u') \star_D \varepsilon') \cdot_D (\omega' \star_D e' \star_D d') = (\omega' \star_D trg_D\ e') \cdot_D (v' \star_D \varepsilon')$
proof −
have $((F\ r \star_D u') \star_D \varepsilon') \cdot_D (\omega' \star_D e' \star_D d') =$
$((F\ r \star_D u') \cdot_D \omega' \star_D \varepsilon' \cdot_D (e' \star_D d'))$
using *D.interchange*
by (*metis D.comp-arr-dom D.hcomp-simps*(*3*) *D.hseqI D.ide-char D.in-hhomE D.in-homE D.seqI T'.base-in-hom*(*1*) *T'.base-simps*(*3*) *T.base-simps*(*2*) ω' e'.*counit-simps*(*1*) e'.*counit-simps*(*2*) *preserves-src* u' *u'-in-hhom*)
also have ... $= \omega' \cdot_D v' \star_D trg_D\ e' \cdot_D \varepsilon'$
using ω' *D.comp-arr-dom D.comp-cod-arr* **by** *auto*
also have ... $= (\omega' \star_D trg_D\ e') \cdot_D (v' \star_D \varepsilon')$
using *D.interchange*
by (*metis D.arrI D.comp-cod-arr D.ide-char D.seqI* ω' ‹$trg_D\ e' = a'$› e'.*counit-simps*(*1*) e'.*counit-simps*(*3*) e'.*counit-simps*(*5*) v' *v'-def*)
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D \mathrm{a}_D[F\ r,\ u',\ trg_D\ e'] \cdot_D (\omega' \star_D trg_D\ e') \cdot_D \mathrm{r}_D{}^{-1}[v']$
proof −
have $(v' \star_D \varepsilon') \cdot_D (v' \star_D D.inv\ \varepsilon') = v' \star_D trg_D\ e'$
using v' *D.whisker-left D.comp-arr-inv D.inv-is-inverse*
by (*metis D.comp-arr-inv' D.seqI'* d'.*unit-in-vhom* e'.*counit-in-hom*(*2*) e'.*counit-is-iso* e'.*counit-simps*(*3*))
moreover have «$\mathrm{r}_D{}^{-1}[v'] : v' \Rightarrow_D v' \star_D trg_D\ e'$»
using v' *1* **by** *simp*
ultimately show *?thesis*
using v' *D.comp-cod-arr* **by** *auto*
qed
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D (\mathrm{a}_D[F\ r,\ u',\ trg_D\ e'] \cdot_D \mathrm{r}_D{}^{-1}[F\ r \star_D u']) \cdot_D \omega'$

using u' v' ω' *D.runit'-naturality D.comp-assoc*
by (*metis D.in-hhomE D.in-homE a'-def e'*)
also have ... $= (F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D \mathrm{r}_D{}^{-1}[u']) \cdot_D \omega'$
using *1 T'.ide-base u' D.runit-hcomp* [*of F r u'*] **by** *fastforce*
also have ... $= ((F\ r \star_D \mathrm{r}_D[u']) \cdot_D (F\ r \star_D \mathrm{r}_D{}^{-1}[u'])) \cdot_D \omega'$
using *D.comp-assoc* **by** *simp*
also have ... $= (F\ r \star_D \mathrm{r}_D[u'] \cdot_D \mathrm{r}_D{}^{-1}[u']) \cdot_D \omega'$
using *1 T'.ide-base u' D.whisker-left* **by** *simp*
also have ... $= (F\ r \star_D u') \cdot_D \omega'$
using u'
by (*metis D.comp-ide-self D.ide-in-hom*(*2*) *D.ide-is-iso*
D.invert-opposite-sides-of-square D.invert-side-of-triangle(*1*)
D.iso-runit D.runit-in-vhom D.seqI')
also have ... $= \omega'$
using ω' *D.comp-cod-arr* **by** *auto*
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
thus $\exists\, w'\ \vartheta'\ \nu'.\ D.ide\ w' \wedge\ «\vartheta' : F\ f \star_D w' \Rightarrow_D u'» \wedge$
$«\nu' : D.dom\ \omega' \Rightarrow_D F\ g \star_D w'» \wedge D.iso\ \nu' \wedge\ T'.composite\text{-}cell\ w'\ \vartheta' \cdot_D \nu' = \omega'$
using w' ϑ' ν' *iso-ν' v'-def* **by** *blast*
qed

Now we establish *T'.T2*.

next
fix u w w' ϑ ϑ' β
assume w: $D.ide\ w$
assume w': $D.ide\ w'$
assume ϑ: $«\vartheta : F\ f \star_D w \Rightarrow_D u»$
assume ϑ': $«\vartheta' : F\ f \star_D w' \Rightarrow_D u»$
assume β: $«\beta : F\ g \star_D w \Rightarrow_D F\ g \star_D w'»$
assume *eq*: $T'.composite\text{-}cell\ w\ \vartheta = T'.composite\text{-}cell\ w'\ \vartheta' \cdot_D \beta$
show $\exists!\gamma.\ «\gamma : w \Rightarrow_D w'» \wedge \beta = F\ g \star_D \gamma \wedge \vartheta = \vartheta' \cdot_D (F\ f \star_D \gamma)$
proof −
define a **where** $a = src_D\ w$
have a: $D.obj\ a$
unfolding *a-def* **by** (*simp add*: w)

have [*simp*]: $src_D\ \vartheta = a$
using ϑ *a-def*
by (*metis D.dom-src D.in-homE D.src.preserves-dom D.src.preserves-reflects-arr*
D.src-hcomp)
have [*simp*]: $trg_D\ \vartheta = trg_D\ (F\ f)$
using ϑ
by (*metis D.arr-dom D.in-homE D.trg-hcomp D.vconn-implies-hpar*(*2*))
have [*simp*]: $src_D\ \vartheta' = a$
using ϑ' *a-def*
by (*metis D.horizontal-homs-axioms D.in-homE* ‹$src_D\ \vartheta = a$› ϑ *horizontal-homs.src-cod*)

have [*simp*]: $trg_D\ \vartheta' = trg_D\ (F\ f)$
  using ϑ'
  by (*metis D.vconn-implies-hpar*(*2*) *D.vconn-implies-hpar*(*4*) ‹$trg_D\ \vartheta = trg_D\ (F\ f)$› ϑ)
have [*simp*]: $src_D\ w = a$
  using *a-def* **by** *simp*
have [*simp*]: $trg_D\ w = map_0\ (src_C\ \varrho)$
by (*metis D.horizontal-homs-axioms D.hseq-char D.in-homE T.tab-simps*(*2*) *T.leg0-simps*(*2*)
    ϑ *category.ideD*(*1*) *category.ide-dom horizontal-homs-def preserves-src*)
have [*simp*]: $src_D\ w' = a$
  using *a-def*
  by (*metis D.ideD*(*1*) *D.in-homE D.src-hcomp D.vconn-implies-hpar*(*1*) ‹$src_D\ \vartheta' = a$›
    ϑ' *category.ide-dom horizontal-homs-def weak-arrow-of-homs-axioms*
    weak-arrow-of-homs-def)
have [*simp*]: $trg_D\ w' = map_0\ (src_C\ \varrho)$
by (*metis D.horizontal-homs-axioms D.hseq-char D.in-homE T.tab-simps*(*2*) *T.leg0-simps*(*2*)
    ϑ' *category.ideD*(*1*) *category.ide-dom horizontal-homs-def preserves-src*)

First, reflect the picture back to C, so that we will be able to apply $T.T2$. We need to choose arrows in C carefully, so that their F images will enable the cancellation of the various isomorphisms that appear.

obtain a_C **where** a_C: $C.obj\ a_C \wedge D.equivalent\text{-}objects\ (map_0\ a_C)\ a$
  using w *a-def biessentially-surjective-on-objects D.obj-src D.ideD*(*1*)
  by *presburger*
obtain e **where** e: «$e : map_0\ a_C \rightarrow_D a$» $\wedge$ *D.equivalence-map e*
  using a_C *D.equivalent-objects-def* **by** *auto*
obtain $d\ \eta\ \varepsilon$
  where $d\eta\varepsilon$: *adjoint-equivalence-in-bicategory* $(\cdot_D)\ (\star_D)\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ e\ d\ \eta\ \varepsilon$
    using e *D.equivalence-map-extends-to-adjoint-equivalence* **by** *blast*
interpret e: *adjoint-equivalence-in-bicategory* ‹$(\cdot_D)$› ‹$(\star_D)$› $\mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ e\ d\ \eta\ \varepsilon$
  using $d\eta\varepsilon$ **by** *auto*
interpret d: *adjoint-equivalence-in-bicategory* ‹$(\cdot_D)$› ‹$(\star_D)$› $\mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
        $d\ e$ *D.inv* ε *D.inv* η
  using *e.dual-adjoint-equivalence* **by** *simp*

have [*simp*]: $src_D\ e = map_0\ a_C$
  using e **by** *auto*
have [*simp*]: $trg_D\ e = a$
  using e **by** *auto*
have [*simp*]: $src_D\ d = a$
  using *e.antipar* **by** *simp*
have [*simp*]: $trg_D\ d = map_0\ a_C$
  using *e.antipar* **by** *simp*

have we: «$w \star_D e : map_0\ a_C \rightarrow_D map_0\ (src_C\ \varrho)$»
  using a_C e *D.ideD*(*1*) ‹$trg_D\ w = map_0\ (src_C\ \varrho)$› *a-def* **by** *blast*
obtain w_C **where**
  w_C: $C.ide\ w_C \wedge$ «$w_C : a_C \rightarrow_C src_C\ \varrho$» $\wedge$ $D.isomorphic\ (F\ w_C)\ (w \star_D e)$
  using a_C e *we locally-essentially-surjective* [*of* $a_C\ src_C\ \varrho\ w \star_D e$]
      C.obj-src T.tab-simps(*1*) *e.ide-left* w **by** *blast*

have $w'e$: «$w' \star_D e : map_0\ a_C \rightarrow_D map_0\ (src_C\ \varrho)$»
  using a_C e $D.ideD(1)$ ‹$trg_D\ w' = map_0\ (src_C\ \varrho)$› $a\text{-}def$ ‹$src_D\ w' = a$› w' **by** *blast*
obtain $w_C{}'$ **where**
  $w_C{}'$: $C.ide\ w_C{}' \wedge$ «$w_C{}' : a_C \rightarrow_C src_C\ \varrho$» $\wedge$ $D.isomorphic\ (F\ w_C{}')\ (w' \star_D e)$
  using a_C e $a\text{-}def$ *locally-essentially-surjective*
  by (*metis* $C.obj\text{-}src$ $D.ide\text{-}hcomp$ $D.hseq\text{-}char$ $D.in\text{-}hhomE$ $T.tab\text{-}simps(2)$
    $T.leg0\text{-}simps(2)$ $e.ide\text{-}left$ w' $w'e$)

have [*simp*]: $src_C\ w_C = a_C$
  using w_C **by** *auto*
have [*simp*]: $trg_C\ w_C = src_C\ \varrho$
  using w_C **by** *auto*
have [*simp*]: $src_C\ w_C{}' = a_C$
  using $w_C{}'$ **by** *auto*
have [*simp*]: $trg_C\ w_C{}' = src_C\ \varrho$
  using $w_C{}'$ **by** *auto*

obtain φ **where** φ: «$\varphi : F\ w_C \Rightarrow_D w \star_D e$» $\wedge$ $D.iso\ \varphi$
  using w_C $D.isomorphicE$ **by** *blast*
obtain φ' **where** φ': «$\varphi' : F\ w_C{}' \Rightarrow_D w' \star_D e$» $\wedge$ $D.iso\ \varphi'$
  using $w_C{}'$ $D.isomorphicE$ **by** *blast*

have ue: «$u \star_D e : map_0\ a_C \rightarrow_D map_0\ (trg_C\ f)$» $\wedge$ $D.ide\ (u \star_D e)$
  using a_C e ϑ $e.ide\text{-}left$
  by (*intro* $conjI$, *auto*)
obtain u_C **where**
  u_C: $C.ide\ u_C \wedge$ «$u_C : a_C \rightarrow_C trg_C\ f$» $\wedge$ $D.isomorphic\ (F\ u_C)\ (u \star_D e)$
  using a_C e ue *locally-essentially-surjective* [*of* a_C $trg_C\ f\ u \star_D e$] **by** *auto*

have [*simp*]: $src_C\ u_C = a_C$
  using u_C **by** *auto*
have [*simp*]: $trg_C\ u_C = trg_C\ f$
  using u_C **by** *auto*

obtain ψ **where** ψ: «$\psi : u \star_D e \Rightarrow_D F\ u_C$» $\wedge$ $D.iso\ \psi$
  using u_C $D.isomorphic\text{-}symmetric$ $D.isomorphicE$ **by** *blast*

define $F\vartheta_C$ **where**
  $F\vartheta_C = \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w_C))$
have *1*: «$F\vartheta_C : F\ (f \star_C w_C) \Rightarrow_D F\ u_C$»
proof (*unfold* $F\vartheta_C\text{-}def$, *intro* $D.comp\text{-}in\text{-}homI$)
  show «$D.inv\ (\Phi\ (f,\ w_C)) : F\ (f \star_C w_C) \Rightarrow_D F\ f \star_D F\ w_C$»
    by (*simp add*: $cmp\text{-}in\text{-}hom(2)$ w_C)
  show «$F\ f \star_D \varphi : F\ f \star_D F\ w_C \Rightarrow_D F\ f \star_D w \star_D e$»
    using w w_C φ **by** (*intro* $D.hcomp\text{-}in\text{-}vhom$, *auto*)
  show «$\mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] : F\ f \star_D w \star_D e \Rightarrow_D (F\ f \star_D w) \star_D e$»
    using w $D.assoc'\text{-}in\text{-}hom$ **by** *simp*
  show «$\vartheta \star_D e : (F\ f \star_D w) \star_D e \Rightarrow_D u \star_D e$»
    using w ϑ **by** (*intro* $D.hcomp\text{-}in\text{-}vhom$, *auto*)

show «ψ : $u \star_D e \Rightarrow_D F\ u_C$»
using ψ **by** *simp*
qed
have *2*: $\exists \vartheta_C$. «$\vartheta_C : f \star_C w_C \Rightarrow_C u_C$» $\wedge$ $F\ \vartheta_C = F\vartheta_C$
using u_C w_C *1 e* ϑ φ *locally-full* **by** *simp*
obtain ϑ_C **where** ϑ_C: «$\vartheta_C : f \star_C w_C \Rightarrow_C u_C$» $\wedge$ $F\ \vartheta_C = F\vartheta_C$
using *2* **by** *auto*

define $F\vartheta_C{}'$ **where**
$F\vartheta_C{}' = \psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \varphi') \cdot_D D.inv\ (\Phi\ (f,\ w_C{}'))$
have *1*: «$F\vartheta_C{}' : F\ (f \star_C w_C{}') \Rightarrow_D F\ u_C$»
proof (*unfold* $F\vartheta_C{}'$*-def*, *intro D.comp-in-homI*)
show «$D.inv\ (\Phi\ (f,\ w_C{}')) : F\ (f \star_C w_C{}') \Rightarrow_D F\ f \star_D F\ w_C{}'$»
by (*simp add*: *cmp-in-hom*(*2*) $w_C{}'$)
show «$F\ f \star_D \varphi' : F\ f \star_D F\ w_C{}' \Rightarrow_D F\ f \star_D w' \star_D e$»
using w' $w_C{}'$ φ' **by** (*intro D.hcomp-in-vhom*, *auto*)
show «$\mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] : F\ f \star_D w' \star_D e \Rightarrow_D (F\ f \star_D w') \star_D e$»
using w' *D.assoc'-in-hom* **by** *simp*
show «$\vartheta' \star_D e : (F\ f \star_D w') \star_D e \Rightarrow_D u \star_D e$»
using w' ϑ' **by** (*intro D.hcomp-in-vhom*, *auto*)
show «$\psi : u \star_D e \Rightarrow_D F\ u_C$»
using ψ **by** *simp*
qed
have *2*: $\exists \vartheta_C{}'$. «$\vartheta_C{}' : f \star_C w_C{}' \Rightarrow_C u_C$» $\wedge$ $F\ \vartheta_C{}' = F\vartheta_C{}'$
using u_C $w_C{}'$ *1 e* ϑ φ *locally-full* **by** *simp*
obtain $\vartheta_C{}'$ **where** $\vartheta_C{}'$: «$\vartheta_C{}' : f \star_C w_C{}' \Rightarrow_C u_C$» $\wedge$ $F\ \vartheta_C{}' = F\vartheta_C{}'$
using *2* **by** *auto*

define $F\beta_C$ **where**
$F\beta_C = \Phi\ (g,\ w_C{}') \cdot_D (F\ g \star_D D.inv\ \varphi') \cdot_D \mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
have $F\beta_C$: «$F\beta_C$: $F\ (g \star_C w_C) \Rightarrow_D F\ (g \star_C w_C{}')$»
proof (*unfold* $F\beta_C$*-def*, *intro D.comp-in-homI*)
show «$D.inv\ (\Phi\ (g,\ w_C)) : F\ (g \star_C w_C) \Rightarrow_D F\ g \star_D F\ w_C$»
by (*simp add*: *cmp-in-hom*(*2*) w_C)
show «$F\ g \star_D \varphi : F\ g \star_D F\ w_C \Rightarrow_D F\ g \star_D w \star_D e$»
using w_C φ **apply** (*intro D.hcomp-in-vhom*) **by** *auto*
show «$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] : F\ g \star_D w \star_D e \Rightarrow_D (F\ g \star_D w) \star_D e$»
using w *D.assoc'-in-hom* **by** *simp*
show «$\beta \star_D e : (F\ g \star_D w) \star_D e \Rightarrow_D (F\ g \star_D w') \star_D e$»
using w β **apply** (*intro D.hcomp-in-vhom*) **by** *auto*
show «$\mathrm{a}_D[F\ g,\ w',\ e] : (F\ g \star_D w') \star_D e \Rightarrow_D F\ g \star_D w' \star_D e$»
using w' *e.antipar D.assoc-in-hom* **by** *simp*
show «$F\ g \star_D D.inv\ \varphi' : F\ g \star_D w' \star_D e \Rightarrow_D F\ g \star_D F\ w_C{}'$»
using w' $w_C{}'$ φ' **by** (*intro D.hcomp-in-vhom*, *auto*)
show «$\Phi\ (g,\ w_C{}') : F\ g \star_D F\ w_C{}' \Rightarrow_D F\ (g \star_C w_C{}')$»
using $w_C{}'$ *cmp-in-hom* **by** *simp*
qed

have *1*: $\exists \beta_C.$ «$\beta_C : g \star_C w_C \Rightarrow_C g \star_C w_C{}'$» $\wedge$ $F\ \beta_C = F\beta_C$
  using $w_C\ w_C{}'\ F\beta_C$ *locally-full* **by** *simp*
obtain β_C **where** β_C: «$\beta_C : g \star_C w_C \Rightarrow_C g \star_C w_C{}'$» $\wedge$ $F\ \beta_C = F\beta_C$
  using *1* **by** *auto*

The following is the main calculation that needs to be done, to permit us to apply $T.T2$. Once again, it started out looking simple, but once all the necessary isomorphisms are thrown in it looks much more complicated.

have $*$: T.*composite-cell* $w_C\ \vartheta_C = T$.*composite-cell* $w_C{}'\ \vartheta_C{}' \cdot_C \beta_C$
proof −
  have *par*: C.*par* (T.*composite-cell* $w_C\ \vartheta_C$) (T.*composite-cell* $w_C{}'\ \vartheta_C{}' \cdot_C \beta_C$)
  proof −
    have «T.*composite-cell* $w_C\ \vartheta_C : g \star_C w_C \Rightarrow_C r \star_C u_C$»
      using $w_C\ \vartheta_C$ T.*composite-cell-in-hom* **by** *simp*
    moreover have «T.*composite-cell* $w_C{}'\ \vartheta_C{}' \cdot_C \beta_C : g \star_C w_C \Rightarrow_C r \star_C u_C$»
    proof (*intro* C.*comp-in-homI*)
      show «$\beta_C : g \star_C w_C \Rightarrow_C g \star_C w_C{}'$»
        using β_C **by** *simp*
      show «$\varrho \star_C w_C{}' : g \star_C w_C{}' \Rightarrow_C (r \star_C f) \star_C w_C{}'$»
        using $w_C{}'$ **by** (*intro* C.*hcomp-in-vhom*, *auto*)
      show «$\mathrm{a}_C[r, f, w_C{}'] : (r \star_C f) \star_C w_C{}' \Rightarrow_C r \star_C f \star_C w_C{}'$»
        using $w_C{}'$ C.*assoc-in-hom* **by** *simp*
      show «$r \star_C \vartheta_C{}' : r \star_C f \star_C w_C{}' \Rightarrow_C r \star_C u_C$»
        using $w_C{}'\ \vartheta_C{}'$ **by** (*intro* C.*hcomp-in-vhom*, *auto*)
    qed
    ultimately show *?thesis*
      by (*metis* C.*in-homE*)
  qed
  moreover have F (T.*composite-cell* $w_C\ \vartheta_C$) = F (T.*composite-cell* $w_C{}'\ \vartheta_C{}' \cdot_C \beta_C$)
  proof −
   have F (T.*composite-cell* $w_C\ \vartheta_C$) = $F\ (r \star_C \vartheta_C) \cdot_D F\ \mathrm{a}_C[r, f, w_C] \cdot_D F\ (\varrho \star_C w_C)$
      using *par* **by** *auto*
    also have ... = $(\Phi\ (r, u_C) \cdot_D (F\ r \star_D F\ \vartheta_C) \cdot_D D.inv\ (\Phi\ (r, f \star_C w_C))) \cdot_D$
      $(\Phi\ (r, f \star_C w_C) \cdot_D (F\ r \star_D \Phi\ (f, w_C)) \cdot_D \mathrm{a}_D[F\ r, F\ f, F\ w_C] \cdot_D$
      $(D.inv\ (\Phi\ (r, f)) \star_D F\ w_C) \cdot_D D.inv\ (\Phi\ (r \star_C f, w_C))) \cdot_D$
      $(\Phi\ (r \star_C f, w_C) \cdot_D (F\ \varrho \star_D F\ w_C) \cdot_D D.inv\ (\Phi\ (g, w_C)))$
    proof −
      have $src_C\ f = trg_C\ w_C \wedge C.hseq\ r\ \vartheta_C \wedge C.hseq\ \varrho\ w_C$
      using *par* **by** *auto*
      thus *?thesis*
        using $w_C\ \vartheta_C$ *preserves-assoc preserves-hcomp*
      by (*metis* $C.ideD(2)$ $C.ideD(3)$ $C.in\text{-}homE$ $T.ide\text{-}base$ $T.ide\text{-}leg0$ $T.leg0\text{-}simps(3)$ $T.tab\text{-}simps(4)$ $T.tab\text{-}simps(5)$)
    qed
    also have
      ... = $\Phi\ (r, u_C) \cdot_D (F\ r \star_D F\ \vartheta_C) \cdot_D (((D.inv\ (\Phi\ (r, f \star_C w_C))) \cdot_D$
        $(\Phi\ (r, f \star_C w_C))) \cdot_D (F\ r \star_D \Phi\ (f, w_C))) \cdot_D \mathrm{a}_D[F\ r, F\ f, F\ w_C] \cdot_D$
        $(D.inv\ (\Phi\ (r, f)) \star_D F\ w_C) \cdot_D ((D.inv\ (\Phi\ (r \star_C f, w_C))) \cdot_D$
        $(\Phi\ (r \star_C f, w_C)) \cdot_D (F\ \varrho \star_D F\ w_C)) \cdot_D D.inv\ (\Phi\ (g, w_C))$

using *D.comp-assoc* **by** *simp*
also have
... = Φ $(r,\ u_C)$ $\cdot_D$ $((F\ r \star_D F\ \vartheta_C) \cdot_D (F\ r \star_D \Phi\ (f,\ w_C))) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C]$ $\cdot_D$ $((D.inv\ (\Phi\ (r,\ f)) \star_D F\ w_C) \cdot_D (F\ \varrho \star_D F\ w_C)) \cdot_D$
$D.inv\ (\Phi\ (g,\ w_C))$
proof −
have
$(D.inv\ (\Phi\ (r \star_C f,\ w_C)) \cdot_D \Phi\ (r \star_C f,\ w_C)) \cdot_D (F\ \varrho \star_D F\ w_C) = F\ \varrho \star_D F\ w_C$
using w_C ‹$trg_C\ w_C = src_C\ \varrho$› *D.comp-inv-arr′ cmp-in-hom cmp-components-are-iso D.comp-cod-arr*
by *simp*
moreover have
$((D.inv\ (\Phi\ (r,\ f \star_C w_C))) \cdot_D (\Phi\ (r,\ f \star_C w_C))) \cdot_D (F\ r \star_D \Phi\ (f,\ w_C)) =$
$F\ r \star_D \Phi\ (f,\ w_C)$
using w_C *D.comp-cod-arr D.comp-inv-arr′ cmp-simps*(*1*,*4*) *C.VV.cod-simp*
by *auto*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
... = Φ $(r,\ u_C)$ $\cdot_D$ $(F\ r \star_D F\ \vartheta_C \cdot_D \Phi\ (f,\ w_C)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C] \cdot_D$
$(?\varrho' \star_D F\ w_C) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $(F\ r \star_D F\ \vartheta_C) \cdot_D (F\ r \star_D \Phi\ (f,\ w_C)) = F\ r \star_D F\ \vartheta_C \cdot_D \Phi\ (f,\ w_C)$
using ϑ_C w_C *D.whisker-left cmp-in-hom*
by (*metis C.hseqE C.seqE D.seqI′ T′.ide-base T.tab-simps*(*2*) *T.ide-leg0 par preserves-hom*)
moreover have $(D.inv\ (\Phi\ (r,\ f)) \star_D F\ w_C) \cdot_D (F\ \varrho \star_D F\ w_C) = ?\varrho' \star_D F\ w_C$
using *D.whisker-right* **by** (*simp add*: w_C)
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
... = Φ $(r,\ u_C)$ $\cdot_D$ $(F\ r \star_D \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] \cdot_D (F\ f \star_D \varphi) \cdot_D$
$D.inv\ (\Phi\ (f,\ w_C)) \cdot_D \Phi\ (f,\ w_C)) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C] \cdot_D$
$(?\varrho' \star_D F\ w_C) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
using ϑ_C $F\vartheta_C$*-def D.comp-assoc* **by** *simp*
also have
... = Φ $(r,\ u_C)$ $\cdot_D$ $(F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta \star_D e) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D$
$((F\ r \star_D F\ f \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C]) \cdot_D (?\varrho' \star_D F\ w_C) \cdot_D$
$D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $F\ r \star_D \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] \cdot_D (F\ f \star_D \varphi) \cdot_D$
$D.inv\ (\Phi\ (f,\ w_C)) \cdot_D \Phi\ (f,\ w_C) =$
$F\ r \star_D \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] \cdot_D (F\ f \star_D \varphi)$
using *cmp-in-hom cmp-components-are-iso D.comp-arr-dom*
by (*metis C.arrI D.cod-inv D.comp-inv-arr′ D.seqE* $F\vartheta_C$*-def T.tab-simps*(*2*) *T.ide-leg0* ‹$trg_C\ w_C = src_C\ \varrho$› ϑ_C *preserves-arr* w_C)
also have ... = $(F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta \star_D e) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D$

$(F\ r \star_D F\ f \star_D \varphi)$
using *D.whisker-left*
by (*metis* (*no-types*, *lifting*) *C.in-homE D.comp-assoc D.seqE* $F\vartheta_C$*-def T′.ide-base* ϑ_C *preserves-arr*)
finally have $F\ r \star_D \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] \cdot_D (F\ f \star_D \varphi) \cdot_D$
$D.inv\ (\Phi\ (f,\ w_C)) \cdot_D \Phi\ (f,\ w_C) =$
$(F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta \star_D e) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D$
$(F\ r \star_D F\ f \star_D \varphi)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta \star_D e) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e] \cdot_D (((F\ r \star_D F\ f) \star_D \varphi) \cdot_D (?\varrho' \star_D F\ w_C)) \cdot_D$
$D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $(F\ r \star_D F\ f \star_D \varphi) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C] =$
$\mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e] \cdot_D ((F\ r \star_D F\ f) \star_D \varphi)$
using $w_C\ \varphi$ ‹$trg_C\ w_C = src_C\ \varrho$› *D.assoc-naturality* [*of* $F\ r\ F\ f\ \varphi$]
by (*metis* (*mono-tags*, *lifting*) *C.ideD*(*1*) *D.in-homE D.vconn-implies-hpar*(*2*) *T′.base-simps*(*2*−*4*) *T′.leg0-simps*(*2*−*5*) *T.leg0-simps*(*2*) *T.tab-simps*(*2*) *preserves-src preserves-trg*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta \star_D e) \cdot_D ((F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e]) \cdot_D (?\varrho' \star_D w \star_D e) \cdot_D (F\ g \star_D \varphi) \cdot_D$
$D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ r \star_D F\ f) \star_D \varphi) \cdot_D (?\varrho' \star_D F\ w_C) = ?\varrho' \star_D \varphi \cdot_D F\ w_C$
using φ *D.interchange*
by (*metis D.comp-arr-dom D.comp-cod-arr D.in-homE T′.tab-simps*(*1*,*5*))
also have $... = ?\varrho' \star_D (w \star_D e) \cdot_D \varphi$
using $\varphi\ w_C$ *D.comp-arr-dom D.comp-cod-arr* **by** *auto*
also have $... = (?\varrho' \star_D w \star_D e) \cdot_D (F\ g \star_D \varphi)$
using φ *D.interchange*
by (*metis D.comp-arr-ide D.comp-cod-arr D.in-homE D.seqI′ T′.ide-leg1 T′.leg1-in-hom*(*2*) *T′.tab-in-vhom′*)
finally have
$((F\ r \star_D F\ f) \star_D \varphi) \cdot_D (?\varrho' \star_D F\ w_C) = (?\varrho' \star_D w \star_D e) \cdot_D (F\ g \star_D \varphi)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D ((F\ r \star_D \vartheta \star_D e) \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e]) \cdot_D$
$(\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e) \cdot_D (\mathrm{a}_D{}^{-1}[F\ r \star_D F\ f,\ w,\ e] \cdot_D$

$(?\varrho' \star_D w \star_D e)) \cdot_D (F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $D.inv\ (F\ r \star_D \mathrm{a}_D[F\ f,\ w,\ e]) = F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]$
using w **by** *simp*
moreover have $D.seq\ (F\ r \star_D \mathrm{a}_D[F\ f,\ w,\ e])$
$(\mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e))$
using w **by** *simp*
moreover have
$(F\ r \star_D \mathrm{a}_D[F\ f,\ w,\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e) =$
$\mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e] \cdot_D \mathrm{a}_D[F\ r \star_D F\ f,\ w,\ e]$
using w *D.pentagon* **by** *simp*
ultimately
have $(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e] =$
$\mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ r \star_D F\ f,\ w,\ e]$
using w *D.comp-assoc*
D.invert-opposite-sides-of-square
[*of* $F\ r \star_D \mathrm{a}_D[F\ f,\ w,\ e]\ \mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e)$
$\mathrm{a}_D[F\ r,\ F\ f,\ w \star_D e]\ \ \mathrm{a}_D[F\ r \star_D F\ f,\ w,\ e]$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D (((F\ r \star_D \vartheta) \star_D e) \cdot_D$
$(\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e) \cdot_D ((?\varrho' \star_D w) \star_D e)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have
$(F\ r \star_D \vartheta \star_D e) \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D w,\ e] = \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D ((F\ r \star_D \vartheta) \star_D e)$
using *D.assoc-naturality* [*of* $F\ r\ \vartheta\ e$] ϑ **by** *auto*
moreover have $\mathrm{a}_D{}^{-1}[F\ r \star_D F\ f,\ w,\ e] \cdot_D (?\varrho' \star_D w \star_D e) =$
$((?\varrho' \star_D w) \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$
using *w we e.ide-left D.assoc′-naturality* [*of* $?\varrho'\ w\ e$] **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D$
$(T'.composite\text{-}cell\ w\ \vartheta \star_D e) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ r \star_D \vartheta) \star_D e) \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w] \star_D e) \cdot_D ((?\varrho' \star_D w) \star_D e) =$
$T'.composite\text{-}cell\ w\ \vartheta \star_D e$
proof −
have «$T'.composite\text{-}cell\ w\ \vartheta : F\ g \star_D w \Rightarrow_D F\ r \star_D u$»
using *w we* ϑ ‹$src_D\ \vartheta = a$› ‹$trg_D\ e = a$› $T'.composite\text{-}cell\text{-}in\text{-}hom$
by (*metis D.ideD*(*1*) *D.ide-in-hom*(*1*) *D.not-arr-null D.seq-if-composable*
$T'.leg1\text{-}simps(3)$ $T.leg1\text{-}simps(2-3)$ $T.tab\text{-}simps(2)$
‹$trg_D\ w = map_0\ (src_C\ \varrho)$› *a-def preserves-src ue*)
thus *?thesis*
using *D.whisker-right D.arrI* **by** *auto*

qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally have *L*: F (T.*composite-cell* w_C ϑ_C) =
Φ (r, u_C) $\cdot_D$ (F r $\star_D$ ψ) $\cdot_D$ $\mathrm{a}_D[F\ r,\ u,\ e]$ $\cdot_D$
(T'.*composite-cell* w ϑ $\star_D$ e) $\cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$ $\cdot_D$ (F g $\star_D$ φ) $\cdot_D$ D.*inv* (Φ (g, w_C))
by *simp*

have F (T.*composite-cell* $w_C{}'$ $\vartheta_C{}'$ $\cdot_C$ β_C) =
F ((r $\star_C$ $\vartheta_C{}'$) $\cdot_C$ $\mathrm{a}_C[r,\ f,\ w_C{}']$ $\cdot_C$ (ϱ $\star_C$ $w_C{}'$) $\cdot_C$ β_C)
using *C.comp-assoc* **by** *simp*
also have ... = F(r $\star_C$ $\vartheta_C{}'$) $\cdot_D$ F $\mathrm{a}_C[r,\ f,\ w_C{}']$ $\cdot_D$ F (ϱ $\star_C$ $w_C{}'$) $\cdot_D$ F β_C
using *C.comp-assoc par* **by** *fastforce*
also have ... = (Φ (r, u_C) $\cdot_D$ (F r $\star_D$ F $\vartheta_C{}'$) $\cdot_D$ D.*inv* (Φ (r, f $\star_C$ $w_C{}'$))) $\cdot_D$
(Φ (r, f $\star_C$ $w_C{}'$) $\cdot_D$ (F r $\star_D$ Φ (f, $w_C{}'$)) $\cdot_D$ $\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}']$ $\cdot_D$
(D.*inv* (Φ (r, f)) $\star_D$ F $w_C{}'$) $\cdot_D$ D.*inv* (Φ (r $\star_C$ f, $w_C{}'$))) $\cdot_D$
(Φ (r $\star_C$ f, $w_C{}'$) $\cdot_D$ (F ϱ $\star_D$ F $w_C{}'$) $\cdot_D$ D.*inv* (Φ (g, $w_C{}'$))) $\cdot_D$
Φ (g, $w_C{}'$) $\cdot_D$ (F g $\star_D$ D.*inv* φ') $\cdot_D$ $\mathrm{a}_D[F\ g,\ w',\ e]$ $\cdot_D$ (β $\star_D$ e) $\cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$ $\cdot_D$ (F g $\star_D$ φ) $\cdot_D$ D.*inv* (Φ (g, w_C))
proof −
have C.*hseq* r $\vartheta_C{}'$ $\wedge$ C.*hseq* ϱ $w_C{}'$
using *par* **by** *blast*
thus *?thesis*
using $w_C{}'$ $\vartheta_C{}'$ β_C $F\beta_C$*-def preserves-assoc* [*of* r f $w_C{}'$] *preserves-hcomp*
by *force*
qed
also have ... = Φ (r, u_C) $\cdot_D$ (F r $\star_D$ F $\vartheta_C{}'$) $\cdot_D$ ((D.*inv* (Φ (r, f $\star_C$ $w_C{}'$))) $\cdot_D$
(Φ (r, f $\star_C$ $w_C{}'$)) $\cdot_D$ (F r $\star_D$ Φ (f, $w_C{}'$))) $\cdot_D$ $\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}']$ $\cdot_D$
(D.*inv* (Φ (r, f)) $\star_D$ F $w_C{}'$) $\cdot_D$ ((D.*inv* (Φ (r $\star_C$ f, $w_C{}'$)) $\cdot_D$
Φ (r $\star_C$ f, $w_C{}'$)) $\cdot_D$ (F ϱ $\star_D$ F $w_C{}'$)) $\cdot_D$ ((D.*inv* (Φ (g, $w_C{}'$)) $\cdot_D$
Φ (g, $w_C{}'$)) $\cdot_D$ (F g $\star_D$ D.*inv* φ')) $\cdot_D$ $\mathrm{a}_D[F\ g,\ w',\ e]$ $\cdot_D$ (β $\star_D$ e) $\cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$ $\cdot_D$ (F g $\star_D$ φ) $\cdot_D$ D.*inv* (Φ (g, w_C))
using *D.comp-assoc* **by** *simp*
also have
... = Φ (r, u_C) $\cdot_D$ (F r $\star_D$ F $\vartheta_C{}'$) $\cdot_D$ (F r $\star_D$ Φ (f, $w_C{}'$)) $\cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}']$ $\cdot_D$
((D.*inv* (Φ (r, f)) $\star_D$ F $w_C{}'$) $\cdot_D$ (F ϱ $\star_D$ F $w_C{}'$)) $\cdot_D$ (F g $\star_D$ D.*inv* φ') $\cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e]$ $\cdot_D$ (β $\star_D$ e) $\cdot_D$ $\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$ $\cdot_D$
(F g $\star_D$ φ) $\cdot_D$ D.*inv* (Φ (g, w_C))
proof −
have (D.*inv* (Φ (r, f $\star_C$ $w_C{}'$))) $\cdot_D$ (Φ (r, f $\star_C$ $w_C{}'$)) $\cdot_D$ (F r $\star_D$ Φ (f, $w_C{}'$)) =
F r $\star_D$ Φ (f, $w_C{}'$)
proof −
have D.*seq* (Φ (r, f $\star_C$ $w_C{}'$)) (F r $\star_D$ Φ (f, $w_C{}'$)) $\wedge$
D.*arr* (D.*inv* (Φ (r, f $\star_C$ $w_C{}'$))) $\wedge$
D.*dom* (D.*inv* (Φ (r, f $\star_C$ $w_C{}'$))) =
D.*cod* (Φ (r, f $\star_C$ $w_C{}'$) $\cdot_D$ (F r $\star_D$ Φ (f, $w_C{}'$)))

by (*metis D.seqE calculation par preserves-arr*)
thus *?thesis*
using *C.ide-hcomp C.ideD*(*1*) *C.trg-hcomp D.invert-side-of-triangle*(*1*)
T.ide-base T.ide-leg0 T.leg0-simps(*3*) *T.tab-simps*(*2*) *cmp-components-are-iso*
‹trg_C $w_C{}'$ = src_C ϱ› $w_C{}'$
by *presburger*
qed
moreover have
($D.inv$ (Φ ($r \star_C f$, $w_C{}'$)) $\cdot_D$ Φ ($r \star_C f$, $w_C{}'$)) $\cdot_D$ (F $\varrho \star_D F$ $w_C{}'$) =
F $\varrho \star_D F$ $w_C{}'$
proof −
have $D.seq$ (F $\varrho \star_D F$ $w_C{}'$) ($D.inv$ (Φ ($C.dom$ ϱ, $C.dom$ $w_C{}'$))) ∧
$D.arr$ (Φ ($r \star_C f$, $w_C{}'$)) ∧
$D.dom$ (Φ ($r \star_C f$, $w_C{}'$)) =
$D.cod$ ((F $\varrho \star_D F$ $w_C{}'$) $\cdot_D$ $D.inv$ (Φ ($C.dom$ ϱ, $C.dom$ $w_C{}'$)))
by (*metis C.hseqI′ C.ide-char D.seqE T.tab-simps*(*1*) *T.tab-simps*(*5*)
‹trg_C $w_C{}'$ = src_C ϱ› *preserves-arr preserves-hcomp* $w_C{}'$)
thus *?thesis*
by (*metis* (*no-types*) *C.ide-hcomp C.ide-char C.hcomp-simps*(*1*)
D.cod-comp D.comp-inv-arr′ D.seqE T.ide-base T.ide-leg0 T.leg0-simps(*3*)
T.tab-simps(*2*) *cmp-components-are-iso D.comp-cod-arr*
‹trg_C $w_C{}'$ = src_C ϱ› $w_C{}'$)
qed
moreover have ($D.inv$ (Φ (g, $w_C{}'$)) $\cdot_D$ Φ (g, $w_C{}'$)) $\cdot_D$ (F $g \star_D D.inv$ φ') =
F $g \star_D D.inv$ φ'
proof −
have ($D.inv$ (Φ (g, $w_C{}'$)) $\cdot_D$ Φ (g, $w_C{}'$)) $\cdot_D$ (F $g \star_D D.inv$ φ') =
(F $g \star_D F$ $w_C{}'$) $\cdot_D$ (F $g \star_D D.inv$ φ')
using $w_C{}'$ β_C $F\beta_C$*-def cmp-components-are-iso D.comp-inv-arr′* **by** *simp*
also have ... = F $g \star_D D.inv$ φ'
using *D.comp-cod-arr* [*of* F $g \star_D D.inv$ φ' F $g \star_D F$ $w_C{}'$]
by (*metis D.cod-inv D.null-is-zero*(*2*) *D.hseq-char′ D.in-homE*
D.is-weak-composition T′.leg1-simps(*6*) φ'
weak-composition.hcomp-simps$_{WC}$(*3*))
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = Φ (r, u_C) $\cdot_D$ (F $r \star_D F$ $\vartheta_C{}'$) $\cdot_D$ (F $r \star_D$ Φ (f, $w_C{}'$)) $\cdot_D$
a_D[F r, F f, F $w_C{}'$] $\cdot_D$ (?$\varrho' \star_D F$ $w_C{}'$) $\cdot_D$ (F $g \star_D D.inv$ φ') $\cdot_D$
a_D[F g, w', e] $\cdot_D$ ($\beta \star_D e$) $\cdot_D$ $\mathrm{a}_D{}^{-1}$[F g, w, e] $\cdot_D$
(F $g \star_D \varphi$) $\cdot_D$ $D.inv$ (Φ (g, w_C))
using $w_C{}'$ *D.whisker-right cmp-in-hom cmp-components-are-iso* **by** *simp*
also have ... = Φ (r, u_C) $\cdot_D$
(F $r \star_D \psi$ $\cdot_D$ ($\vartheta' \star_D e$) $\cdot_D$ $\mathrm{a}_D{}^{-1}$[F f, w', e] $\cdot_D$ (F $f \star_D \varphi'$) $\cdot_D$
$D.inv$ (Φ (f, $w_C{}'$))) $\cdot_D$
(F $r \star_D$ Φ (f, $w_C{}'$)) $\cdot_D$
a_D[F r, F f, F $w_C{}'$] $\cdot_D$ (?$\varrho' \star_D F$ $w_C{}'$) $\cdot_D$ (F $g \star_D D.inv$ φ') $\cdot_D$

$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
using $\vartheta_C{}'\ F\vartheta_C{}'$*-def* **by** *simp*
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D (F\ r \star_D F\ f \star_D \varphi') \cdot_D$
$(((F\ r \star_D D.inv\ (\Phi\ (f,\ w_C{}'))) \cdot_D (F\ r \star_D \Phi\ (f,\ w_C{}'))) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}'] \cdot_D (?\varrho' \star_D F\ w_C{}') \cdot_D (F\ g \star_D D.inv\ \varphi') \cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $F\ r \star_D \psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \varphi') \cdot_D$
$D.inv\ (\Phi\ (f,\ w_C{}')) =$
$(F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D (F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D$
$(F\ r \star_D F\ f \star_D \varphi') \cdot_D (F\ r \star_D D.inv\ (\Phi\ (f,\ w_C{}')))$
using *D.whisker-left cmp-in-hom cmp-components-are-iso*
by (*metis C.arrI D.src.preserves-reflects-arr D.src-vcomp D.vseq-implies-hpar*(*1*)
$F\vartheta_C{}'$*-def T'.ide-base* $\vartheta_C{}'$ *preserves-arr*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D ((F\ r \star_D F\ f \star_D \varphi') \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}'] \cdot_D (?\varrho' \star_D F\ w_C{}') \cdot_D (F\ g \star_D D.inv\ \varphi') \cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ r \star_D D.inv\ (\Phi\ (f,\ w_C{}'))) \cdot_D (F\ r \star_D \Phi\ (f,\ w_C{}'))) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}'] =$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}']$
using *cmp-in-hom cmp-components-are-iso D.comp-cod-arr*
D.whisker-left [*of F r D.inv* $(\Phi\ (f,\ w_C{}')\ \Phi\ (f,\ w_C{}')$]
by (*simp add*: *D.comp-inv-arr'* $w_C{}'$)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D e] \cdot_D$
$(((F\ r \star_D F\ f) \star_D \varphi') \cdot_D (?\varrho' \star_D F\ w_C{}')) \cdot_D (F\ g \star_D D.inv\ \varphi') \cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $(F\ r \star_D F\ f \star_D \varphi') \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w_C{}'] =$
$\mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D e] \cdot_D ((F\ r \star_D F\ f) \star_D \varphi')$
using $w_C{}'\ \varphi'$ *D.assoc-naturality* [*of F r F f* φ']
by (*metis C.ideD*(*1*) *D.dom-trg D.in-homE D.trg.preserves-dom*
T'.leg0-simps(*2*−*5*) *T'.base-simps*(*2*−*4*) *T.tab-simps*(*2*) *T.leg0-simps*(*2*)
‹$trg_C\ w_C{}' = src_C\ \varrho$› *preserves-src preserves-trg*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*

qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D$
$(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D e] \cdot_D$
$(?\varrho' \star_D w' \star_D e) \cdot_D (((F\ g \star_D \varphi') \cdot_D (F\ g \star_D D.inv\ \varphi')) \cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e]) \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ r \star_D F\ f) \star_D \varphi') \cdot_D (?\varrho' \star_D F\ w_C{}') = (?\varrho' \star_D w' \star_D e) \cdot_D (F\ g \star_D \varphi')$
using φ' *D.interchange D.comp-arr-dom D.comp-cod-arr*
by (*metis D.in-homE T′.tab-in-hom*(*2*))
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D (F\ r \star_D \vartheta' \star_D e) \cdot_D$
$((F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D e]) \cdot_D$
$(?\varrho' \star_D w' \star_D e) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ g \star_D \varphi') \cdot_D (F\ g \star_D D.inv\ \varphi')) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e] = \mathrm{a}_D[F\ g,\ w',\ e]$
proof −
have $((F\ g \star_D \varphi') \cdot_D (F\ g \star_D D.inv\ \varphi')) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e] =$
$(F\ g \star_D w' \star_D e) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e]$
by (*metis D.arr-inv D.cod-inv D.comp-arr-inv′ D.in-homE D.seqI*
D.whisker-left T′.ide-leg1 φ')
also have ... $= \mathrm{a}_D[F\ g,\ w',\ e]$
using w' *D.comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D ((F\ r \star_D \vartheta' \star_D e) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f \star_D w',\ e]) \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D e) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ r \star_D F\ f,\ w',\ e] \cdot_D (?\varrho' \star_D w' \star_D e)) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e] \cdot_D$
$(\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $D.inv\ (F\ r \star_D \mathrm{a}_D[F\ f,\ w',\ e]) = F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]$
using w' **by** *simp*
moreover have $D.seq\ (F\ r \star_D \mathrm{a}_D[F\ f,\ w',\ e])$
$(\mathrm{a}_D[F\ r,\ F\ f \star_D w',\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D e))$
using w' **by** *simp*
moreover have $D.iso\ (F\ r \star_D \mathrm{a}_D[F\ f,\ w',\ e])$
using w' **by** *simp*
moreover have $D.iso\ \mathrm{a}_D[F\ r \star_D F\ f,\ w',\ e]$
using w' **by** *simp*
moreover have $(F\ r \star_D \mathrm{a}_D[F\ f,\ w',\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D w',\ e] \cdot_D$
$(\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D e) =$
$\mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D e] \cdot_D \mathrm{a}_D[F\ r \star_D F\ f,\ w',\ e]$
using w' *D.pentagon* **by** *simp*

ultimately
have $(F\ r \star_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e]) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D\ e] =$
$\mathrm{a}_D[F\ r,\ F\ f \star_D\ w',\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\ r \star_D\ F\ f,\ w',\ e]$
using *w′ D.comp-assoc*
D.invert-opposite-sides-of-square
[*of* $F\ r \star_D \mathrm{a}_D[F\ f,\ w',\ e]\ \mathrm{a}_D[F\ r,\ F\ f \star_D\ w',\ e] \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D\ e)$
$\mathrm{a}_D[F\ r,\ F\ f,\ w' \star_D\ e]\ \ \mathrm{a}_D[F\ r \star_D\ F\ f,\ w',\ e]$]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D\ \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D (((F\ r \star_D\ \vartheta') \star_D\ e) \cdot_D$
$(\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D\ e) \cdot_D ((?\varrho' \star_D\ w') \star_D\ e)) \cdot_D$
$((\mathrm{a}_D{}^{-1}[F\ g,\ w',\ e] \cdot_D \mathrm{a}_D[F\ g,\ w',\ e]) \cdot_D (\beta \star_D\ e)) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D$
$(F\ g \star_D\ \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $(F\ r \star_D\ \vartheta' \star_D\ e) \cdot_D \mathrm{a}_D[F\ r,\ F\ f \star_D\ w',\ e] =$
$\mathrm{a}_D[F\ r,\ u,\ e] \cdot_D ((F\ r \star_D\ \vartheta') \star_D\ e)$
using *D.assoc-naturality* [*of* $F\ r\ \vartheta'\ e$] ϑ' **by** *auto*
moreover have $\mathrm{a}_D{}^{-1}[F\ r \star_D\ F\ f,\ w',\ e] \cdot_D (?\varrho' \star_D\ w' \star_D\ e) =$
$((?\varrho' \star_D\ w') \star_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w',\ e]$
using *w′ w′e D.assoc′-naturality* [*of* $?\varrho'\ w'\ e$] **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D\ \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D$
$(T'.composite\text{-}cell\ w'\ \vartheta' \star_D\ e) \cdot_D (\beta \star_D\ e) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D\ \varphi) \cdot_D D.inv\ (\Phi\ (g,\ w_C))$
proof −
have $((F\ r \star_D\ \vartheta') \star_D\ e) \cdot_D (\mathrm{a}_D[F\ r,\ F\ f,\ w'] \star_D\ e) \cdot_D ((?\varrho' \star_D\ w') \star_D\ e) =$
$T'.composite\text{-}cell\ w'\ \vartheta' \star_D\ e$
proof −
have «$T'.composite\text{-}cell\ w'\ \vartheta' : F\ g \star_D\ w' \Rightarrow_D F\ r \star_D\ u$»
using ϑ' *w′ T′.composite-cell-in-hom D.vconn-implies-hpar*(*3*) **by** *simp*
thus *?thesis*
using *D.whisker-right D.arrI* **by** *auto*
qed
moreover have $(\mathrm{a}_D{}^{-1}[F\ g,\ w',\ e] \cdot_D \mathrm{a}_D[F\ g,\ w',\ e]) \cdot_D (\beta \star_D\ e) = \beta \star_D\ e$
using *w′ β e.ide-left* ‹$src_D\ w' = a$› ‹$trg_D\ e = a$› $F\beta_C$ $F\beta_C$*-def D.comp-cod-arr*
D.comp-arr-inv′
by (*metis* (*no-types*, *lifting*) *D.comp-assoc-assoc′*(*2*) *D.hcomp-simps*(*1*)
D.hcomp-simps(*4*) *D.hseqI′ D.ide-char D.in-homE D.vconn-implies-hpar*(*1*)
D.vconn-implies-hpar(*3*) *T′.ide-leg1 T.leg1-simps*(*2*) *T.leg1-simps*(*3*)
T.tab-simps(*2*) ‹$trg_D\ w' = map_0\ (src_C\ \varrho)$› *preserves-src*)
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \Phi\ (r,\ u_C) \cdot_D (F\ r \star_D\ \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D$

$(T'.\textit{composite-cell}\ w'\ \vartheta' \cdot_D \beta \star_D e) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.\textit{inv}\ (\Phi\ (g,\ w_C))$
proof −
have $D.\textit{arr}\ (T'.\textit{composite-cell}\ w'\ \vartheta' \cdot_D \beta)$
by (*metis* (*full-types*) *D.hseq-char D.seqE L eq par preserves-arr*)
thus *?thesis*
using *D.whisker-right* **by** (*metis D.comp-assoc e.ide-left*)
qed
finally have R: $F\ (T.\textit{composite-cell}\ w_C{}'\ \vartheta_C{}' \cdot_C \beta_C) =$
$\Phi\ (r,\ u_C) \cdot_D (F\ r \star_D \psi) \cdot_D \mathrm{a}_D[F\ r,\ u,\ e] \cdot_D$
$(T'.\textit{composite-cell}\ w'\ \vartheta' \cdot_D \beta \star_D e) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \cdot_D (F\ g \star_D \varphi) \cdot_D D.\textit{inv}\ (\Phi\ (g,\ w_C))$
by *simp*

show $F\ (T.\textit{composite-cell}\ w_C\ \vartheta_C) = F\ (T.\textit{composite-cell}\ w_C{}'\ \vartheta_C{}' \cdot_C \beta_C)$
using *eq L R* **by** *simp*
qed
ultimately show *?thesis*
using *is-faithful* [*of* $T.\textit{composite-cell}\ w_C\ \vartheta_C\ T.\textit{composite-cell}\ w_C{}'\ \vartheta_C{}' \cdot_C \beta_C$]
by *simp*
qed
have $**$: $\exists!\gamma.\ «\gamma : w_C \Rightarrow_C w_C{}'» \wedge \beta_C = g \star_C \gamma \wedge \vartheta_C = \vartheta_C{}' \cdot_C (f \star_C \gamma)$
using $*$ $w_C\ w_C{}'\ \vartheta_C\ \vartheta_C{}'\ \beta_C$ *T.T2* [*of* $w_C\ w_C{}'\ \vartheta_C\ u_C\ \vartheta_C{}'\ \beta_C$] **by** *simp*
obtain γ_C **where**
γ_C: $«\gamma_C : w_C \Rightarrow_C w_C{}'» \wedge \beta_C = g \star_C \gamma_C \wedge \vartheta_C = \vartheta_C{}' \cdot_C (f \star_C \gamma_C)$
using $**$ **by** *auto*
have γ_C*-unique*: $\bigwedge\gamma_C{}'.\ «\gamma_C{}' : w_C \Rightarrow_C w_C{}'» \wedge \beta_C = g \star_C \gamma_C{}' \wedge$
$\vartheta_C = \vartheta_C{}' \cdot_C (f \star_C \gamma_C{}') \Longrightarrow \gamma_C{}' = \gamma_C$
using γ_C $**$ **by** *blast*

Now use F to map everything back to D, transport the result along the equivalence map e, and cancel all of the isomorphisms that got introduced.

let $?P = \lambda\gamma.\ «\gamma : w \star_D e \Rightarrow_D w' \star_D e» \wedge$
$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] = F\ g \star_D \gamma \wedge$
$\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] =$
$\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \gamma)$
define γe **where** $\gamma e = \varphi' \cdot_D F\ \gamma_C \cdot_D D.\textit{inv}\ \varphi$
have $P\gamma e$: $?P\ \gamma e$
proof −
have *1*: $«F\ \gamma_C : F\ w_C \Rightarrow_D F\ w_C{}'» \wedge$
$F\ \beta_C = \Phi\ (g,\ w_C{}') \cdot_D (F\ g \star_D F\ \gamma_C) \cdot_D D.\textit{inv}\ (\Phi\ (g,\ w_C)) \wedge$
$F\ \vartheta_C = F\ \vartheta_C{}' \cdot_D \Phi\ (f,\ C.\textit{cod}\ \gamma_C) \cdot_D (F\ f \star_D F\ \gamma_C) \cdot_D D.\textit{inv}\ (\Phi\ (f,\ w_C))$
using $\beta_C\ \vartheta_C\ \gamma_C$ *preserves-hcomp* [*of* $f\ \gamma_C$] *preserves-hcomp* [*of* $g\ \gamma_C$] **by** *force*
have A: $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] =$
$F\ g \star_D \varphi' \cdot_D F\ \gamma_C \cdot_D D.\textit{inv}\ \varphi$
proof −
have $F\ g \star_D F\ \gamma_C = D.\textit{inv}\ (\Phi\ (g,\ w_C{}')) \cdot_D F\ \beta_C \cdot_D \Phi\ (g,\ w_C)$
proof −
have $F\ \beta_C = \Phi\ (g,\ w_C{}') \cdot_D (F\ g \star_D F\ \gamma_C) \cdot_D D.\textit{inv}\ (\Phi\ (g,\ w_C))$

using *1* **by** *simp*
hence $D.inv\ (\Phi\ (g,\ w_C{}'))\ \cdot_D\ F\ \beta_C = (F\ g \star_D F\ \gamma_C)\ \cdot_D\ D.inv\ (\Phi\ (g,\ w_C))$
using $w_C\ w_C{}'$ ‹$trg_C\ w_C = src_C\ \varrho$› ‹$trg_C\ w_C{}' = src_C\ \varrho$› *cmp-components-are-iso D.invert-side-of-triangle*(*1*)
by (*metis D.arrI* $F\beta_C$ *T.ide-leg1 T.leg1-simps*(*3*) *T.tab-simps*(*2*) β_C)
hence $(D.inv\ (\Phi\ (g,\ w_C{}'))\ \cdot_D\ F\ \beta_C)\ \cdot_D\ \Phi\ (g,\ w_C) = F\ g \star_D F\ \gamma_C$
using *cmp-components-are-iso D.invert-side-of-triangle*(*2*)
by (*metis 1 D.arrI D.inv-inv D.iso-inv-iso D.seqE* $F\beta_C$ *T.ide-leg1 T.leg1-simps*(*3*) *T.tab-simps*(*2*) β_C ‹$trg_C\ w_C = src_C\ \varrho$› w_C)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((D.inv\ (\Phi\ (g,\ w_C{}'))\ \cdot_D\ \Phi\ (g,\ w_C{}'))\ \cdot_D\ (F\ g \star_D D.inv\ \varphi'))\ \cdot_D$
$\mathrm{a}_D[F\ g,\ w',\ e]\ \cdot_D\ (\beta \star_D e)\ \cdot_D\ \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]\ \cdot_D\ (F\ g \star_D \varphi)\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w_C))\ \cdot_D\ \Phi\ (g,\ w_C)$
using β_C $F\beta_C$*-def D.comp-assoc* **by** *simp*
also have ... $= (F\ g \star_D D.inv\ \varphi')\ \cdot_D\ \mathrm{a}_D[F\ g,\ w',\ e]\ \cdot_D\ (\beta \star_D e)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]\ \cdot_D\ (F\ g \star_D \varphi)$
proof −
have $(D.inv\ (\Phi\ (g,\ w_C{}'))\ \cdot_D\ \Phi\ (g,\ w_C{}'))\ \cdot_D\ (F\ g \star_D D.inv\ \varphi') = F\ g \star_D D.inv\ \varphi'$
proof −
have $(D.inv\ (\Phi\ (g,\ w_C{}'))\ \cdot_D\ \Phi\ (g,\ w_C{}'))\ \cdot_D\ (F\ g \star_D D.inv\ \varphi') =$
$(F\ g \star_D F\ w_C{}')\ \cdot_D\ (F\ g \star_D D.inv\ \varphi')$
using $w_C{}'\ \varphi'$ *cmp-components-are-iso D.comp-inv-arr′* **by** *simp*
also have ... $= F\ g \star_D D.inv\ \varphi'$
using $w_C{}'\ \varphi'$ *D.comp-cod-arr*
by (*metis D.arr-inv D.cod-inv D.in-homE D.whisker-left T′.ide-leg1*)
finally show *?thesis* **by** *blast*
qed
moreover have $(F\ g \star_D \varphi)\ \cdot_D\ D.inv\ (\Phi\ (g,\ w_C))\ \cdot_D\ \Phi\ (g,\ w_C) = F\ g \star_D \varphi$
proof −
have $(F\ g \star_D \varphi)\ \cdot_D\ D.inv\ (\Phi\ (g,\ w_C))\ \cdot_D\ \Phi\ (g,\ w_C) =$
$(F\ g \star_D \varphi)\ \cdot_D\ (F\ g \star_D F\ w_C)$
using $w_C\ \varphi$ ‹$trg_C\ w_C = src_C\ \varrho$› *cmp-components-are-iso cmp-in-hom D.comp-inv-arr′*
by *simp*
also have ... $= F\ g \star_D \varphi$
using $w_C\ \varphi$ *D.comp-arr-dom*
by (*metis D.hcomp-simps*(*3*) *D.hseqI′ D.in-hhom-def D.in-homE D.vconn-implies-hpar*(*2*) *D.vconn-implies-hpar*(*4*) *T′.leg1-simps*(*2*,*5*) *T.leg1-simps*(*2*−*3*) *T.tab-simps*(*2*) *preserves-src we*)
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
qed
finally have *2*: $(F\ g \star_D D.inv\ \varphi')\ \cdot_D\ (\mathrm{a}_D[F\ g,\ w',\ e]\ \cdot_D\ (\beta \star_D e)\ \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e])\ \cdot_D\ (F\ g \star_D \varphi) =$
$F\ g \star_D F\ \gamma_C$
using *D.comp-assoc* **by** *simp*

have *3*: $(\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]) \cdot_D (F\ g \star_D \varphi) =$
$(F\ g \star_D \varphi') \cdot_D (F\ g \star_D F\ \gamma_C)$
proof −
have *D.hseq* $(F\ g)\ (F\ \gamma_C)$
using *1* $F\beta_C\ \beta_C$ **by** *auto*
moreover have *D.iso* $(F\ g \star_D D.inv\ \varphi')$
by (*metis 2 D.iso-hcomp D.hseqE D.ide-is-iso D.iso-inv-iso D.seqE T′.ide-leg1* φ' *calculation*)
moreover have *D.inv* $(F\ g \star_D D.inv\ \varphi') = F\ g \star_D \varphi'$
by (*metis D.hseqE D.ide-is-iso D.inv-hcomp D.inv-ide D.inv-inv D.iso-inv-iso D.iso-is-arr T′.ide-leg1* φ' *calculation*(*2*))
ultimately show *?thesis*
using *2* $\varphi\ \varphi'$
D.invert-side-of-triangle(*1*)
[*of* $F\ g \star_D F\ \gamma_C\ F\ g \star_D D.inv\ \varphi'$
$(\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]) \cdot_D (F\ g \star_D \varphi)$]
by *auto*
qed
hence $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] =$
$((F\ g \star_D \varphi') \cdot_D (F\ g \star_D F\ \gamma_C)) \cdot_D (F\ g \star_D D.inv\ \varphi)$
proof −
have *D.seq* $(F\ g \star_D \varphi')\ (F\ g \star_D F\ \gamma_C)$
by (*metis 1 2 3 D.arrI D.null-is-zero*(*1*) *D.null-is-zero*(*2*) *D.ext* $F\beta_C\ \beta_C$)
moreover have *D.iso* $(F\ g \star_D \varphi)$
using *D.vconn-implies-hpar*(*2*) *D.vconn-implies-hpar*(*4*) φ *we* **by** *auto*
moreover have *D.inv* $(F\ g \star_D \varphi) = F\ g \star_D D.inv\ \varphi$
by (*metis D.hseqE D.ide-is-iso D.inv-hcomp D.inv-ide D.iso-is-arr T′.ide-leg1* φ *calculation*(*2*))
ultimately show *?thesis*
using *3* $\varphi\ \varphi'$
D.invert-side-of-triangle(*2*)
[*of* $(F\ g \star_D \varphi') \cdot_D (F\ g \star_D F\ \gamma_C)$
$\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]\ F\ g \star_D \varphi$]
by *auto*
qed
also have ... $= F\ g \star_D \varphi' \cdot_D F\ \gamma_C \cdot_D D.inv\ \varphi$
using φ' *D.whisker-left*
by (*metis 1 D.arr-inv D.cod-comp D.cod-inv D.comp-assoc D.in-homE D.seqI T′.ide-leg1* φ)
finally show *?thesis* **by** *simp*
qed
have *B*: $\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] =$
$\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \varphi' \cdot_D F\ \gamma_C \cdot_D D.inv\ \varphi)$
proof −
have $F\ \vartheta_C{}' \cdot_D \Phi\ (f,\ w_C{}') \cdot_D (F\ f \star_D F\ \gamma_C) \cdot_D D.inv\ (\Phi\ (f,\ w_C)) =$
$(\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \varphi') \cdot_D (D.inv\ (\Phi\ (f,\ w_C{}')) \cdot_D$
$\Phi\ (f,\ w_C{}')) \cdot_D (F\ f \star_D F\ \gamma_C)) \cdot_D D.inv\ (\Phi\ (f,\ w_C))$
using $\gamma_C\ \vartheta_C{}'\ F\vartheta_C{}'$*-def D.comp-assoc* **by** *auto*
also have ... $= \psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \varphi') \cdot_D$

$(F\,f \star_D F\,\gamma_C) \cdot_D D.inv\,(\Phi\,(f,\,w_C))$
proof −
have $(D.inv\,(\Phi\,(f,\,w_C{}')) \cdot_D \Phi\,(f,\,w_C{}')) \cdot_D (F\,f \star_D F\,\gamma_C) = F\,f \star_D F\,\gamma_C$
using *D.comp-cod-arr*
by (*metis* (*mono-tags*, *lifting*) *C.in-homE D.cod-comp D.comp-inv-arr′ D.seqE T.tab-simps*(*2*) *T.ide-leg0 cmp-components-are-iso* γ_C *1* ‹$trg_C\ w_C{}' = src_C\ \varrho$› ϑ_C *preserves-arr* $w_C{}'$)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
finally have $F\,\vartheta_C{}' \cdot_D \Phi\,(f,\,w_C{}') \cdot_D (F\,f \star_D F\,\gamma_C) \cdot_D D.inv\,(\Phi\,(f,\,w_C)) =$
$\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D (F\,f \star_D \varphi') \cdot_D$
$(F\,f \star_D F\,\gamma_C) \cdot_D D.inv\,(\Phi\,(f,\,w_C))$
by *simp*
hence *3*: $F\,\vartheta_C{}' \cdot_D \Phi\,(f,\,w_C{}') \cdot_D (F\,f \star_D F\,\gamma_C) =$
$\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D (F\,f \star_D \varphi') \cdot_D (F\,f \star_D F\,\gamma_C)$
using *cmp-components-are-iso D.iso-inv-iso D.iso-is-retraction D.retraction-is-epi D.epi-cancel*
by (*metis C.in-homE D.comp-assoc T.tab-simps*(*2*) *T.ide-leg0* γ_C *1* ‹$trg_C\ w_C = src_C\ \varrho$› ϑ_C *preserves-arr* w_C)
hence $(\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w,\,e] \cdot_D (F\,f \star_D \varphi)) \cdot_D D.inv\,(\Phi\,(f,\,w_C)) =$
$(\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D (F\,f \star_D \varphi') \cdot_D$
$(F\,f \star_D F\,\gamma_C)) \cdot_D D.inv\,(\Phi\,(f,\,w_C))$
using *1* ϑ_C *F*ϑ_C*-def D.comp-assoc* **by** (*metis C.in-homE* γ_C)
hence *2*: $(\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w,\,e]) \cdot_D (F\,f \star_D \varphi) =$
$\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D (F\,f \star_D \varphi') \cdot_D (F\,f \star_D F\,\gamma_C)$
using γ_C *cmp-components-are-iso D.iso-inv-iso D.iso-is-retraction D.retraction-is-epi D.epi-cancel*
by (*metis* (*mono-tags*, *lifting*) *1 3 C.in-homE D.comp-assoc T.tab-simps*(*2*) *T.ide-leg0* ‹$trg_C\ w_C = src_C\ \varrho$› ϑ_C *preserves-arr* w_C)
hence $\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w,\,e] =$
$(\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e]) \cdot_D$
$(F\,f \star_D \varphi') \cdot_D (F\,f \star_D F\,\gamma_C) \cdot_D (F\,f \star_D D.inv\,\varphi)$
proof −
have $D.inv\,(F\,f \star_D \varphi) = F\,f \star_D D.inv\,\varphi$
using φ
by (*metis C.arrI D.hseq-char D.ide-is-iso D.inv-hcomp D.inv-ide D.seqE F*ϑ_C*-def T′.ide-leg0 preserves-arr* ϑ_C)
thus *?thesis*
using φ φ' ϑ ϑ' γ_C *D.invert-side-of-triangle*(*2*)
by (*metis 2 C.arrI D.comp-assoc D.iso-hcomp D.hseqE D.ide-is-iso D.seqE F*ϑ_C*-def T′.ide-leg0* ϑ_C *preserves-arr*)
qed
also have ... $= \psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D$
$(F\,f \star_D \varphi') \cdot_D (F\,f \star_D F\,\gamma_C) \cdot_D (F\,f \star_D D.inv\,\varphi)$
using *D.comp-assoc* **by** *simp*
also have
... $= \psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\,w',\,e] \cdot_D (F\,f \star_D \varphi' \cdot_D F\,\gamma_C \cdot_D D.inv\,\varphi)$
proof −

```
      have D.arr (φ' ·D F γC ·D D.inv φ)
        using 1 φ φ' by blast
      thus ?thesis
        using D.whisker-left by auto
    qed
    finally show ?thesis by simp
  qed
  have C: «φ' ·D F γC ·D D.inv φ : w ⋆D e ⇒D w' ⋆D e»
    using φ φ' γC 1 by (meson D.comp-in-homI D.inv-in-hom)
  show ?thesis
    unfolding γe-def
    using A B C by simp
qed
have UN: ⋀γ. ?P γ ⟹ γ = γe
proof −
  fix γ
  assume γ: ?P γ
  show γ = γe
  proof −
    let ?γ' = D.inv φ' ·D γ ·D φ
    have γ': «?γ' : F wC ⇒D F wC'»
      using γ φ φ' by auto
    obtain γC' where γC': «γC' : wC ⇒C wC'» ∧ F γC' = ?γ'
      using wC wC' γ γ' locally-full by fastforce
    have 1: βC = g ⋆C γC'
    proof −
      have F βC = F (g ⋆C γC')
      proof −
        have F βC =
              Φ (g, wC') ·D (F g ⋆D D.inv φ') ·D aD[F g, w', e] ·D (β ⋆D e) ·D
                aD⁻¹[F g, w, e] ·D (F g ⋆D φ) ·D D.inv (Φ (g, wC))
          using βC Fβ_C-def by simp
        have F (g ⋆C γC') =
              Φ (g, wC') ·D (F g ⋆D D.inv φ' ·D γ ·D φ) ·D D.inv (Φ (g, wC))
          using γC' preserves-hcomp
          by (metis C.hseqI' C.in-homE C.trg-dom T.tab-simps(2) T.leg1-simps(2)
              T.leg1-simps(3,5−6) ‹trgC wC = srcC ϱ›)
        also have ... = Φ (g, wC') ·D (F g ⋆D D.inv φ') ·D (F g ⋆D γ) ·D
                        (F g ⋆D φ) ·D D.inv (Φ (g, wC))
          using φ φ' D.whisker-left D.comp-assoc
          by (metis D.arrI D.seqE Fβ_C-def T'.ide-leg1 γ γ')
        also have ... = Φ (g, wC') ·D (F g ⋆D D.inv φ') ·D
                        (aD[F g, w', e] ·D (β ⋆D e) ·D aD⁻¹[F g, w, e]) ·D
                        (F g ⋆D φ) ·D D.inv (Φ (g, wC))
          using γ D.comp-assoc by simp
        also have ... = F βC
          using βC Fβ_C-def D.comp-assoc by simp
        finally show ?thesis by simp
      qed
```

```
  moreover have C.par β_C (g ⋆_C γ_C')
  proof (intro conjI)
    show C.arr β_C
      using β_C by blast
    show 2: C.hseq g γ_C'
      using Fβ_C β_C calculation by fastforce
    show C.dom β_C = C.dom (g ⋆_C γ_C')
      using 2 β_C γ_C' by fastforce
    show C.cod β_C = C.cod (g ⋆_C γ_C')
      using 2 β_C γ_C' by fastforce
  qed
  ultimately show ?thesis using is-faithful by blast
qed
have 2: ϑ_C = ϑ_C' ·_C (f ⋆_C γ_C')
proof −
  have F ϑ_C = F (ϑ_C' ·_C (f ⋆_C γ_C'))
  proof −
    have F (ϑ_C' ·_C (f ⋆_C γ_C')) = F ϑ_C' ·_D F (f ⋆_C γ_C')
      using ϑ_C' γ_C' by force
    also have
     ... = F ϑ_C' ·_D Φ (f, w_C') ·_D (F f ⋆_D D.inv φ' ·_D γ ·_D φ) ·_D D.inv (Φ (f, w_C))
      using w_C w_C' ϑ_C' γ_C' preserves-hcomp
      by (metis C.hseqI' C.in-homE C.trg-dom T.tab-simps(2) T.leg0-simps(2)
          T.leg0-simps(4−5) ‹trg_C w_C = src_C ϱ›)
    also have ... = F ϑ_C' ·_D Φ (f, w_C') ·_D
                    ((F f ⋆_D D.inv φ') ·_D (F f ⋆_D γ) ·_D (F f ⋆_D φ)) ·_D
                    D.inv (Φ (f, w_C))
      using D.whisker-left
      by (metis D.arrI D.seqE T'.ide-leg0 γ')
    also have ... = ψ ·_D (ϑ' ⋆_D e) ·_D a_D⁻¹[F f, w', e] ·_D (((F f ⋆_D φ') ·_D
                    (D.inv (Φ (f, w_C')) ·_D Φ (f, w_C')) ·_D (F f ⋆_D D.inv φ')) ·_D
                    (F f ⋆_D γ)) ·_D (F f ⋆_D φ) ·_D D.inv (Φ (f, w_C))
      using ϑ_C' Fϑ_C'-def D.comp-assoc by simp
    also have ... = (ψ ·_D (ϑ' ⋆_D e) ·_D a_D⁻¹[F f, w', e] ·_D (F f ⋆_D γ)) ·_D
                    (F f ⋆_D φ) ·_D D.inv (Φ (f, w_C))
    proof −
      have D.inv (Φ (f, w_C')) ·_D Φ (f, w_C') = F f ⋆_D F w_C'
        using w_C' cmp-in-hom cmp-components-are-iso
        by (simp add: D.comp-inv-arr')
      moreover have D.hseq (F f) (D.inv φ')
        using φ' D.hseqI'
        by (metis D.ide-is-iso D.in-hhom-def D.iso-inv-iso D.iso-is-arr
            D.trg-inv D.vconn-implies-hpar(2) D.vconn-implies-hpar(4)
            T'.ide-leg0 T'.leg1-simps(3) T.leg1-simps(2−3)
            T.tab-simps(2) γ preserves-src we)
      ultimately have (D.inv (Φ (f, w_C')) ·_D Φ (f, w_C')) ·_D (F f ⋆_D D.inv φ') =
                      F f ⋆_D D.inv φ'
        using w_C' φ' D.comp-cod-arr [of F f ⋆_D D.inv φ' F f ⋆_D F w_C']
        by fastforce
```

```
        hence ((F f ⋆D φ') ·D (D.inv (Φ (f, wC')) ·D Φ (f, wC')) ·D
                (F f ⋆D D.inv φ')) ·D (F f ⋆D γ) =
              ((F f ⋆D φ') ·D (F f ⋆D D.inv φ')) ·D (F f ⋆D γ)
          by simp
        also have ... = F f ⋆D γ
          using γ φ' ϑC' FϑC'-def D.comp-cod-arr D.whisker-left D.hseqI'
          by (metis D.comp-arr-inv' D.in-hhom-def D.in-homE T'.ide-leg0 w'e)
        finally have ((F f ⋆D φ') ·D (D.inv (Φ (f, wC')) ·D Φ (f, wC')) ·D
                      (F f ⋆D D.inv φ')) ·D (F f ⋆D γ) =
                     F f ⋆D γ
          by simp
        thus ?thesis
          using D.comp-assoc by simp
      qed
      also have ... = ψ ·D (ϑ ⋆D e) ·D aD⁻¹[F f, w, e] ·D
                      (F f ⋆D φ) ·D D.inv (Φ (f, wC))
        using γ D.comp-assoc by metis
      also have ... = F ϑC
        using ϑC FϑC-def by simp
      finally show ?thesis by simp
    qed
    moreover have C.par ϑC (ϑC' ·C (f ⋆C γC'))
    proof (intro conjI)
      show C.arr ϑC
        using ϑC by auto
      show 1: C.seq ϑC' (f ⋆C γC')
        using ϑC' γC'
        by (metis C.arrI ϑC calculation preserves-reflects-arr)
      show C.dom ϑC = C.dom (ϑC' ·C (f ⋆C γC'))
        using 1 ϑC γC' by fastforce
      show C.cod ϑC = C.cod (ϑC' ·C (f ⋆C γC'))
        using 1 ϑC γC' γC by auto
    qed
    ultimately show ?thesis
      using is-faithful by blast
  qed
  have F γC' = F γC
    using ** γC γC' 1 2 by blast
  hence ?γ' = F γC
    using γC' by simp
  thus γ = γe
    unfolding γe-def
    by (metis D.arrI D.comp-assoc D.inv-inv D.invert-side-of-triangle(1)
        D.invert-side-of-triangle(2) D.iso-inv-iso γ' φ φ')
 qed
qed
```

We are now in a position to exhibit the 2-cell γ and show that it is unique with the required properties.

```
show ?thesis
proof
  let ?γ = r_D[w'] ·_D (w' ⋆_D ε) ·_D a_D[w', e, d] ·_D (γe ⋆_D d) ·_D a_D^-1[w, e, d] ·_D
            (w ⋆_D D.inv ε) ·_D r_D^-1[w]
  have γ: «?γ : w ⇒_D w'»
    using Pγe w w' e.counit-in-hom(2) e.counit-is-iso
    apply (intro D.comp-in-homI)
          apply auto[2]
         apply fastforce
        apply auto[3]
      apply fastforce
    by auto
  moreover have β = F g ⋆_D ?γ
  proof −
    have F g ⋆_D ?γ =
          (F g ⋆_D r_D[w']) ·_D (F g ⋆_D w' ⋆_D ε) ·_D (F g ⋆_D a_D[w', e, d]) ·_D
            (F g ⋆_D γe ⋆_D d) ·_D
            (F g ⋆_D a_D^-1[w, e, d]) ·_D (F g ⋆_D w ⋆_D D.inv ε) ·_D (F g ⋆_D r_D^-1[w])
      using w w' γ Pγe D.whisker-left e.antipar
      by (metis D.arrI D.seqE T'.ide-leg1)
    also have ... =
              (F g ⋆_D r_D[w']) ·_D (F g ⋆_D w' ⋆_D ε) ·_D (F g ⋆_D a_D[w', e, d]) ·_D
                (a_D[F g, w' ⋆_D e, d] ·_D ((F g ⋆_D γe) ⋆_D d) ·_D a_D^-1[F g, w ⋆_D e, d]) ·_D
                (F g ⋆_D a_D^-1[w, e, d]) ·_D (F g ⋆_D w ⋆_D D.inv ε) ·_D (F g ⋆_D r_D^-1[w])
    proof −
      have a_D[F g, w' ⋆_D e, d] ·_D ((F g ⋆_D γe) ⋆_D d) ·_D a_D^-1[F g, w ⋆_D e, d] =
            a_D[F g, w' ⋆_D e, d] ·_D a_D^-1[F g, w' ⋆_D e, d] ·_D (F g ⋆_D γe ⋆_D d)
        using w w' e.antipar Pγe D.assoc'-naturality [of F g γe d]
        by (metis D.dom-trg D.ideD(1−3) D.in-hhomE D.in-homE
            D.src-dom D.trg.preserves-dom T'.leg1-simps(2) T'.leg1-simps(3,5−6)
            T.tab-simps(2) T.leg0-simps(2) e e.ide-right preserves-src we)
      also have
        ... = (a_D[F g, w' ⋆_D e, d] ·_D a_D^-1[F g, w' ⋆_D e, d]) ·_D (F g ⋆_D γe ⋆_D d)
        using D.comp-assoc by simp
      also have ... = F g ⋆_D γe ⋆_D d
      proof −
        have (a_D[F g, w' ⋆_D e, d] ·_D a_D^-1[F g, w' ⋆_D e, d]) ·_D (F g ⋆_D γe ⋆_D d) =
              (F g ⋆_D (w' ⋆_D e) ⋆_D d) ·_D (F g ⋆_D γe ⋆_D d)
          using w'e D.isomorphic-implies-ide(2) w_C' D.comp-assoc-assoc'(1) by auto
        also have ... = F g ⋆_D γe ⋆_D d
        proof −
          have «F g ⋆_D γe ⋆_D d : F g ⋆_D (w ⋆_D e) ⋆_D d ⇒_D F g ⋆_D (w' ⋆_D e) ⋆_D d»
            using we e.ide-right e.antipar Pγe by fastforce
          thus ?thesis
            using D.comp-cod-arr by auto
        qed
        finally show ?thesis by blast
      qed
      finally have
```

$\mathrm{a}_D[F\ g,\ w' \star_D\ e,\ d] \cdot_D ((F\ g \star_D\ \gamma e) \star_D\ d) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w \star_D\ e,\ d] =$
$F\ g \star_D\ \gamma e \star_D\ d$
by *simp*
thus *?thesis* **by** *simp*
qed
also have ... =
$(F\ g \star_D\ \mathrm{r}_D[w']) \cdot_D (F\ g \star_D\ w' \star_D\ \varepsilon) \cdot_D (F\ g \star_D\ \mathrm{a}_D[w',\ e,\ d]) \cdot_D$
$(\mathrm{a}_D[F\ g,\ w' \star_D\ e,\ d] \cdot_D$
$(\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D\ d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w \star_D\ e,\ d]) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D (F\ g \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\ g \star_D\ \mathrm{r}_D{}^{-1}[w])$
using $P\gamma e$ **by** *simp*
also have
... =
$(F\ g \star_D\ \mathrm{r}_D[w']) \cdot_D (F\ g \star_D\ w' \star_D\ \varepsilon) \cdot_D$
$(F\ g \star_D\ \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\ g,\ w' \star_D\ e,\ d] \cdot_D (\mathrm{a}_D[F\ g,\ w',\ e] \star_D\ d) \cdot_D$
$((\beta \star_D\ e) \star_D\ d) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D\ d) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w \star_D\ e,\ d] \cdot_D (F\ g \star_D\ \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\ g \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\ g \star_D\ \mathrm{r}_D{}^{-1}[w])$
proof −
have $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D\ d =$
$(\mathrm{a}_D[F\ g,\ w',\ e] \star_D\ d) \cdot_D ((\beta \star_D\ e) \star_D\ d) \cdot_D (\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D\ d)$
proof −
have $D.arr\ (\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e])$
using *D.arrI D.in-hhom-def D.vconn-implies-hpar*(*2*) $P\gamma e$ *we* **by** *auto*
thus *?thesis*
using *D.whisker-right* **by** *auto*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
... =
$(F\ g \star_D\ \mathrm{r}_D[w']) \cdot_D (F\ g \star_D\ w' \star_D\ \varepsilon) \cdot_D$
$((F\ g \star_D\ \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\ g,\ w' \star_D\ e,\ d] \cdot_D (\mathrm{a}_D[F\ g,\ w',\ e] \star_D\ d) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ g \star_D\ w',\ e,\ d]) \cdot_D (\beta \star_D\ e \star_D\ d) \cdot_D (\mathrm{a}_D[F\ g \star_D\ w,\ e,\ d]) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D\ d) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w \star_D\ e,\ d] \cdot_D (F\ g \star_D\ \mathrm{a}_D{}^{-1}[w,\ e,\ d])) \cdot_D$
$(F\ g \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\ g \star_D\ \mathrm{r}_D{}^{-1}[w])$
proof −
have $(\beta \star_D\ e) \star_D\ d =$
$\mathrm{a}_D{}^{-1}[F\ g \star_D\ w',\ e,\ d] \cdot_D (\beta \star_D\ e \star_D\ d) \cdot_D \mathrm{a}_D[F\ g \star_D\ w,\ e,\ d]$
proof −
have $src_D\ \beta = trg_D\ e$
using β
by (*metis D.dom-trg D.hseq-char′ D.in-homE D.src-dom D.src-hcomp*
D.trg.extensionality D.trg.preserves-arr D.trg.preserves-dom
‹$trg_D\ e = a$› *a-def*)
moreover have $src_D\ (F\ g) = trg_D\ w$
by *simp*

moreover have $src_D\ (F\ g) = trg_D\ w'$
by *simp*
moreover have
«$(\beta \star_D e) \star_D d : ((F\ g \star_D w) \star_D e) \star_D d \Rightarrow_D ((F\ g \star_D w') \star_D e) \star_D d$»
using $\beta\ w\ w'\ e$ *e.antipar*
by (*intro D.hcomp-in-vhom*, *auto*)
ultimately have
$\mathrm{a}_D{}^{-1}[F\ g \star_D w',\ e,\ d] \cdot_D (\beta \star_D e \star_D d) \cdot_D \mathrm{a}_D[F\ g \star_D w,\ e,\ d] =$
$\mathrm{a}_D{}^{-1}[F\ g \star_D w',\ e,\ d] \cdot_D \mathrm{a}_D[F\ g \star_D w',\ e,\ d] \cdot_D ((\beta \star_D e) \star_D d)$
using $w'\ e$ *e.ide-left e.ide-right e.antipar* β *D.assoc′-naturality*
by (*metis D.assoc-naturality D.in-homE e.triangle-equiv-form*(*1*)
e.triangle-in-hom(*3*) *e.triangle-in-hom*(*4*) *e.triangle-right*
e.triangle-right′ e.triangle-right-implies-left)
also have
$... = (\mathrm{a}_D{}^{-1}[F\ g \star_D w',\ e,\ d] \cdot_D \mathrm{a}_D[F\ g \star_D w',\ e,\ d]) \cdot_D ((\beta \star_D e) \star_D d)$
using *D.comp-assoc* **by** *simp*
also have $... = (((F\ g \star_D w') \star_D e) \star_D d) \cdot_D ((\beta \star_D e) \star_D d)$
using $w'\ e$ *e.antipar* β *D.comp-assoc-assoc′* **by** *simp*
also have $... = (\beta \star_D e) \star_D d$
proof −
have «$(\beta \star_D e) \star_D d : ((F\ g \star_D w) \star_D e) \star_D d \Rightarrow_D ((F\ g \star_D w') \star_D e) \star_D d$»
using $w\ e$ *e.antipar* β
by (*intro D.hcomp-in-vhom*, *auto*)
thus *?thesis*
using *D.comp-cod-arr* **by** *auto*
qed
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = (F\ g \star_D \mathrm{r}_D[w']) \cdot_D ((F\ g \star_D w' \star_D \varepsilon) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e \star_D d]) \cdot_D$
$(\beta \star_D e \star_D d) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e \star_D d] \cdot_D (F\ g \star_D w \star_D D.inv\ \varepsilon)) \cdot_D (F\ g \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(F\ g \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\ g,\ w' \star_D e,\ d] \cdot_D (\mathrm{a}_D[F\ g,\ w',\ e] \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g \star_D w',\ e,\ d] =$
$\mathrm{a}_D[F\ g,\ w',\ e \star_D d]$
proof −
have *D.seq* $(F\ g \star_D \mathrm{a}_D[w',\ e,\ d])$
$(\mathrm{a}_D[F\ g,\ w' \star_D e,\ d] \cdot_D (\mathrm{a}_D[F\ g,\ w',\ e] \star_D d))$
using $w\ w'\ e$ *e.antipar* **by** *simp*
thus *?thesis*
using $w\ w'\ e$ *e.antipar D.pentagon* [*of* $F\ g\ w'\ e\ d$] *D.invert-side-of-triangle*(*2*)
D.assoc′-eq-inv-assoc D.comp-assoc D.ide-hcomp D.ideD(*1*)
D.iso-assoc D.hcomp-simps(*1*) *T′.ide-leg1 T.leg1-simps*(*2*−*3*)
T.tab-simps(*2*) ‹$src_D\ w' = a$› ‹$trg_D\ e = a$› ‹$trg_D\ w' = map_0\ (src_C\ \varrho)$›
e.ide-left e.ide-right preserves-src

by *metis*
qed
moreover have
$\mathrm{a}_D[F\ g \star_D w,\ e,\ d] \cdot_D (\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w \star_D e,\ d] \cdot_D$
$(F\ g \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) =$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e \star_D d]$
proof −
have *D.seq* $(\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] \star_D d)$
$(\mathrm{a}_D{}^{-1}[F\ g,\ w \star_D e,\ d] \cdot_D (F\ g \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]))$
using *w w′ e e.antipar* **by** *simp*
moreover have *D.inv* $\mathrm{a}_D{}^{-1}[F\ g \star_D w,\ e,\ d] = \mathrm{a}_D[F\ g \star_D w,\ e,\ d]$
using *w w′ e e.antipar* **by** *simp*
ultimately show *?thesis*
using *w w′ e e.antipar D.pentagon′* [*of F g w e d*]
D.iso-inv-iso D.inv-inv D.comp-assoc D.invert-side-of-triangle(*1*)
by (*metis D.assoc′-simps*(*3*) *D.null-is-zero*(*2*) *D.ide-hcomp D.ideD*(*1*)
D.iso-assoc′ D.not-arr-null D.seq-if-composable D.src-hcomp T′.ide-leg1
‹$trg_D\ e = a$› *a-def e.ide-left e.ide-right*)
qed
ultimately show *?thesis*
using *w w′ e e.antipar* β *D.comp-assoc* **by** *metis*
qed
also have ... $= (F\ g \star_D \mathrm{r}_D[w']) \cdot_D \mathrm{a}_D[F\ g,\ w',\ trg_D\ e] \cdot_D$
$(((F\ g \star_D w') \star_D \varepsilon) \cdot_D (\beta \star_D e \star_D d) \cdot_D ((F\ g \star_D w) \star_D D.inv\ \varepsilon)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\ g,\ w,\ trg_D\ e] \cdot_D (F\ g \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(F\ g \star_D w' \star_D \varepsilon) \cdot_D \mathrm{a}_D[F\ g,\ w',\ e \star_D d] =$
$\mathrm{a}_D[F\ g,\ w',\ trg_D\ e] \cdot_D ((F\ g \star_D w') \star_D \varepsilon)$
using *w′ e e.antipar D.assoc-naturality* [*of F g w′ ε*] **by** *simp*
moreover have $\mathrm{a}_D{}^{-1}[F\ g,\ w,\ e \star_D d] \cdot_D (F\ g \star_D w \star_D D.inv\ \varepsilon) =$
$((F\ g \star_D w) \star_D D.inv\ \varepsilon) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ trg_D\ e]$
using *w e e.antipar D.assoc′-naturality* [*of F g w D.inv ε*]
e.counit-is-iso e.counit-in-hom
by *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= ((F\ g \star_D \mathrm{r}_D[w']) \cdot_D \mathrm{a}_D[F\ g,\ w',\ trg_D\ e]) \cdot_D$
$(\beta \star_D trg_D\ e) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\ g,\ w,\ trg_D\ e] \cdot_D (F\ g \star_D \mathrm{r}_D{}^{-1}[w]))$
proof −
have $((F\ g \star_D w') \star_D \varepsilon) \cdot_D (\beta \star_D e \star_D d) \cdot_D ((F\ g \star_D w) \star_D D.inv\ \varepsilon) =$
$\beta \star_D trg_D\ e$
proof −
have $((F\ g \star_D w') \star_D \varepsilon) \cdot_D (\beta \star_D e \star_D d) \cdot_D ((F\ g \star_D w) \star_D D.inv\ \varepsilon) =$
$((F\ g \star_D w') \star_D \varepsilon) \cdot_D (\beta \star_D D.inv\ \varepsilon)$
using *w w′ e e.antipar D.interchange* [*of* β $F\ g \star_D w\ e \star_D d\ D.inv\ \varepsilon$]
D.comp-arr-dom D.comp-cod-arr e.counit-is-iso
by (*metis D.in-homE* β *d.unit-simps*(*1*) *d.unit-simps*(*3*))

also have ... = $((F\ g \star_D w') \star_D \varepsilon) \cdot_D ((F\ g \star_D w') \star_D D.inv\ \varepsilon) \cdot_D (\beta \star_D trg_D\ e)$
using *w w′ e e.antipar β D.interchange* [*of F g* $\star_D$ *w′ β D.inv ε* trg_D *e*]
D.comp-arr-dom D.comp-cod-arr e.counit-is-iso
by *auto*
also have
... = $(((F\ g \star_D w') \star_D \varepsilon) \cdot_D ((F\ g \star_D w') \star_D D.inv\ \varepsilon)) \cdot_D (\beta \star_D trg_D\ e)$
using *D.comp-assoc* **by** *simp*
also have ... = $((F\ g \star_D w') \star_D \varepsilon \cdot_D D.inv\ \varepsilon) \cdot_D (\beta \star_D trg_D\ e)$
using *w′ D.whisker-left* [*of F g* $\star_D$ *w′*] **by** *simp*
also have ... = $((F\ g \star_D w') \star_D trg_D\ e) \cdot_D (\beta \star_D trg_D\ e)$
by (*simp add*: *D.comp-arr-inv′*)
also have ... = $\beta \star_D trg_D\ e$
using *β D.comp-cod-arr D.hseqI′*
by (*metis D.cod-cod D.hcomp-simps*(*1*) *D.hcomp-simps*(*4*)
D.in-homE D.trg.preserves-reflects-arr D.vconn-implies-hpar(*1*)
D.vconn-implies-hpar(*2*) *D.vconn-implies-hpar*(*3*) *D.vconn-implies-hpar*(*4*)
‹$src_D\ w' = a$› ‹$trg_D\ e = a$› *e.counit-in-hom*(*2*) *e.counit-simps*(*5*))
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = $\mathrm{r}_D[F\ g \star_D w'] \cdot_D (\beta \star_D trg_D\ e) \cdot_D \mathrm{r}_D{}^{-1}[F\ g \star_D w]$
using *w w′ D.runit-hcomp D.runit-hcomp* [*of F g w*] **by** *simp*
also have ... = $\mathrm{r}_D[F\ g \star_D w'] \cdot_D \mathrm{r}_D{}^{-1}[F\ g \star_D w'] \cdot_D \beta$
using *β D.runit′-naturality*
by (*metis D.arr-cod D.arr-dom D.cod-dom D.in-homE D.src.preserves-cod*
D.src-dom D.src-hcomp ‹$src_D\ w' = a$› ‹$trg_D\ e = a$›)
also have ... = $(\mathrm{r}_D[F\ g \star_D w'] \cdot_D \mathrm{r}_D{}^{-1}[F\ g \star_D w']) \cdot_D \beta$
using *D.comp-assoc* **by** *simp*
also have ... = β
using *w′ β D.comp-cod-arr D.comp-arr-inv′ D.iso-runit* **by** *auto*
finally show *?thesis* **by** *simp*
qed
moreover have $\vartheta = \vartheta' \cdot_D (F\ f \star_D ?\gamma)$
proof −
have $\vartheta' \cdot_D (F\ f \star_D ?\gamma) =$
$\vartheta' \cdot_D (F\ f \star_D \mathrm{r}_D[w']) \cdot_D (F\ f \star_D w' \star_D \varepsilon) \cdot_D (F\ f \star_D \mathrm{a}_D[w', e, d]) \cdot_D$
$(F\ f \star_D \gamma e \star_D d) \cdot_D$
$(F\ f \star_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D (F\ f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\ f \star_D \mathrm{r}_D{}^{-1}[w])$
using *w ϑ γ D.whisker-left*
by (*metis D.arrI D.seqE T′.ide-leg0*)
also have
... = $(\vartheta' \cdot_D (F\ f \star_D \mathrm{r}_D[w'])) \cdot_D (F\ f \star_D w' \star_D \varepsilon) \cdot_D (F\ f \star_D \mathrm{a}_D[w', e, d]) \cdot_D$
$(\mathrm{a}_D[F\ f, w' \star_D e, d] \cdot_D ((F\ f \star_D \gamma e) \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\ f, w \star_D e, d]) \cdot_D$
$(F\ f \star_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D (F\ f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\ f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have *1*: $\mathrm{a}_D[F\ f, w' \star_D e, d] \cdot_D ((F\ f \star_D \gamma e) \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\ f, w \star_D e, d] =$
$\mathrm{a}_D[F\ f, w' \star_D e, d] \cdot_D \mathrm{a}_D{}^{-1}[F\ f, w' \star_D e, d] \cdot_D (F\ f \star_D \gamma e \star_D d)$

using *w w′ e we w′e e.antipar Pγe D.assoc′-naturality* [*of F f γe d*]
by (*metis D.cod-trg D.in-hhomE D.in-homE D.src-cod D.trg.preserves-cod*
T′.leg0-simps(*2*,*4*−*5*) *T.tab-simps*(*2*) *T.leg0-simps*(*2*)
e.triangle-in-hom(*4*) *e.triangle-right′ preserves-src*)
also have
2: ... = $(\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w' \star_D e,\ d]) \cdot_D (F\,f \star_D \gamma e \star_D d)$
using *D.comp-assoc* **by** *simp*
also have ... = $F\,f \star_D \gamma e \star_D d$
proof −
have $(\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w' \star_D e,\ d]) \cdot_D (F\,f \star_D \gamma e \star_D d) =$
$(F\,f \star_D (w' \star_D e) \star_D d) \cdot_D (F\,f \star_D \gamma e \star_D d)$
using *1 2 e.antipar D.isomorphic-implies-ide*(*2*) $w_C{}'$ *w′e D.comp-assoc-assoc′*
by *auto*
also have ... = $F\,f \star_D \gamma e \star_D d$
proof −
have «$F\,f \star_D \gamma e \star_D d : F\,f \star_D (w \star_D e) \star_D d \Rightarrow_D F\,f \star_D (w' \star_D e) \star_D d$»
using *we 1 2 e.antipar Pγe* **by** *fastforce*
thus *?thesis*
using *D.comp-cod-arr* **by** *blast*
qed
finally show *?thesis* **by** *blast*
qed
finally have
$\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d]) =$
$F\,f \star_D \gamma e \star_D d$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
... = $((\vartheta' \cdot_D \mathrm{r}_D[F\,f \star_D w'\rceil) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w',\ src_D\ w'\rceil) \cdot_D (F\,f \star_D w' \star_D \varepsilon) \cdot_D$
$(F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D (\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using *w′ D.runit-hcomp*(*3*) [*of F f w′*] *D.comp-assoc* **by** *simp*
also have ... = $\mathrm{r}_D[u] \cdot_D (\vartheta' \star_D src_D\ w') \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w',\ src_D\ w'\rceil \cdot_D$
$(F\,f \star_D w' \star_D \varepsilon)) \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D$
$(\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using *ϑ′ D.runit-naturality* [*of ϑ′*] *D.comp-assoc* **by** *fastforce*
also have ... = $\mathrm{r}_D[u] \cdot_D ((\vartheta' \star_D src_D\ w') \cdot_D ((F\,f \star_D w') \star_D \varepsilon)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D d] \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D$
$\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using *w′ D.assoc′-naturality* [*of F f w′ ε*] *D.comp-assoc* **by** *simp*
also have ... = $\mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D e \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D d] \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D$

$(\mathrm{a}_D[F\,f,\ w' \star_D\ e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D\ d)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D\ e,\ d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$

proof −
have $(\vartheta' \star_D src_D\ w') \cdot_D ((F\,f \star_D\ w') \star_D \varepsilon) = \vartheta' \star_D \varepsilon$
using *D.interchange D.comp-arr-dom D.comp-cod-arr*
by (*metis D.in-homE* ‹$src_D\ w' = a$› ‹$trg_D\ e = a$› ϑ' *e.counit-simps*(*1*) *e.counit-simps*(*3*))
also have ... $= (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D\ e \star_D\ d)$
using ϑ' *D.interchange* [*of u* ϑ' ε $e \star_D\ d$] *D.comp-arr-dom D.comp-cod-arr*
by *auto*
finally have $(\vartheta' \star_D src_D\ w') \cdot_D ((F\,f \star_D\ w') \star_D \varepsilon) = (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D\ e \star_D\ d)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D\ e \star_D\ d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D\ d] \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D$
$(F\,f \star_D \gamma e \star_D\ d) \cdot_D ((\mathrm{a}_D[F\,f,\ w \star_D\ e,\ d] \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D\ e,\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d])) \cdot_D$
$(F\,f \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$

proof −
have $\mathrm{a}_D[F\,f,\ w' \star_D\ e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D\ d) =$
$(F\,f \star_D \gamma e \star_D\ d) \cdot_D \mathrm{a}_D[F\,f,\ w \star_D\ e,\ d]$
using *D.assoc-naturality* [*of F f* γe *d*]
by (*metis D.cod-trg D.in-hhomE D.in-homE D.src-cod D.trg.preserves-cod* $P\gamma e$ *T′.leg0-simps*(*2*,*4*−*5*) *T.tab-simps*(*2*) *T.leg0-simps*(*2*) *e e.antipar*(*1*) *e.triangle-in-hom*(*4*) *e.triangle-right′ preserves-src* $w'e$)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D\ e \star_D\ d) \cdot_D$
$(\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D$
$(F\,f \star_D \gamma e \star_D\ d) \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$

proof −
have $(\mathrm{a}_D[F\,f,\ w \star_D\ e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w \star_D\ e,\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) =$
$F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]$
using *w D.comp-cod-arr D.comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D (\vartheta' \star_D\ e \star_D\ d) \cdot_D$
$((\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D\ d]) \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\,f,\ w' \star_D\ e,\ d]) \cdot_D$
$((F\,f \star_D \gamma e) \star_D\ d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D\ e,\ d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D\ w \star_D\ D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$

proof −

have $F\,f \star_D \gamma e \star_D d =$
$\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d]$
proof −
have $\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D ((F\,f \star_D \gamma e) \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d] =$
$\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w' \star_D e,\ d] \cdot_D (F\,f \star_D \gamma e \star_D d)$
using *Pγe e.antipar D.assoc′-naturality*
by (*metis D.in-hhom-def D.in-homE D.vconn-implies-hpar(1) D.vconn-implies-hpar(2) T′.leg0-simps(2,4−5) T.leg0-simps(2) T.tab-simps(2)* ‹$src_D\ e = map_0\ a_C$› *d.triangle-equiv-form(1) d.triangle-in-hom(3) d.triangle-left preserves-src we*)
also have
$... = (\mathrm{a}_D[F\,f,\ w' \star_D e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w' \star_D e,\ d]) \cdot_D (F\,f \star_D \gamma e \star_D d)$
using *D.comp-assoc* **by** *simp*
also have $... = (F\,f \star_D (w' \star_D e) \star_D d) \cdot_D (F\,f \star_D \gamma e \star_D d)$
using *w′e D.isomorphic-implies-ide(2) w_C′ D.comp-assoc-assoc′* **by** *auto*
also have $... = F\,f \star_D \gamma e \star_D d$
using *D.comp-cod-arr*
by (*metis D.comp-cod-arr D.null-is-zero(2) D.hseq-char D.hseq-char′ D.in-homE D.whisker-left D.whisker-right Pγe T′.ide-leg0 e.ide-right*)
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D ((\vartheta' \star_D e \star_D d) \cdot_D$
$\mathrm{a}_D[F\,f \star_D w',\ e,\ d]) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e] \star_D d) \cdot_D$
$((F\,f \star_D \gamma e) \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e \star_D d]) \cdot_D (F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\,f,\ w' \star_D e,\ d] =$
$\mathrm{a}_D[F\,f \star_D w',\ e,\ d] \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w',\ e] \star_D d)$
proof −
have $\mathrm{a}_D[F\,f,\ w',\ e \star_D d] \cdot_D \mathrm{a}_D[F\,f \star_D w',\ e,\ d] =$
$((F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\,f,\ w' \star_D e,\ d]) \cdot_D (\mathrm{a}_D[F\,f,\ w',\ e] \star_D d)$
using *w′ D.pentagon D.comp-assoc* **by** *simp*
moreover have *D.seq* $\mathrm{a}_D[F\,f,\ w',\ e \star_D d]\ \mathrm{a}_D[F\,f \star_D w',\ e,\ d]$
using *w′* **by** *simp*
moreover have *D.inv* $(\mathrm{a}_D[F\,f,\ w',\ e] \star_D d) = \mathrm{a}_D{}^{-1}[F\,f,\ w',\ e] \star_D d$
using *w′* **by** *simp*
ultimately show *?thesis*
using *w′ D.comp-assoc*
D.invert-opposite-sides-of-square
[*of* $\mathrm{a}_D[F\,f,\ w',\ e \star_D d]\ \mathrm{a}_D[F\,f \star_D w',\ e,\ d]$
$(F\,f \star_D \mathrm{a}_D[w',\ e,\ d]) \cdot_D \mathrm{a}_D[F\,f,\ w' \star_D e,\ d]$
$\mathrm{a}_D[F\,f,\ w',\ e] \star_D d$]
by *simp*
qed

thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have
$... = \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D \mathrm{a}_D[u, e, d] \cdot_D$
$(((\vartheta' \star_D e) \star_D d) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f, w', e] \star_D d) \cdot_D ((F\,f \star_D \gamma e) \star_D d)) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f, w \star_D e, d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(\vartheta' \star_D e \star_D d) \cdot_D \mathrm{a}_D[F\,f \star_D w', e, d] = \mathrm{a}_D[u, e, d] \cdot_D ((\vartheta' \star_D e) \star_D d)$
using *w' ϑ' e.ide-left e.ide-right e.antipar D.assoc-naturality* [*of ϑ' e d*]
by *auto*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D \mathrm{a}_D[u, e, d] \cdot_D$
$((\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w', e] \cdot_D (F\,f \star_D \gamma e) \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f, w \star_D e, d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $((\vartheta' \star_D e) \star_D d) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f, w', e] \star_D d) \cdot_D ((F\,f \star_D \gamma e) \star_D d) =$
$(\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w', e] \cdot_D (F\,f \star_D \gamma e) \star_D d$
using *w' w'e ϑ' ϑ_C e.ide-left e.ide-right e.antipar D.whisker-right*
by (*metis* (*full-types*) *C.arrI D.cod-comp D.seqE D.seqI Fϑ_C-def Pγe preserves-arr*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D \mathrm{a}_D[u, e, d] \cdot_D$
$((\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w, e] \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f, w \star_D e, d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w', e] \cdot_D (F\,f \star_D \gamma e) =$
$\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w, e]$
using *Pγe* **by** *simp*
moreover have $D.arr\ (\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w', e] \cdot_D (F\,f \star_D \gamma e))$
by (*metis C.in-homE D.comp-assoc D.null-is-zero*(*1*) *D.ext Fϑ_C-def Pγe ϑ_C preserves-arr*)
moreover have $D.arr\ (\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w, e])$
using *Pγe calculation*(*2*) **by** *auto*
ultimately have $(\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w', e] \cdot_D (F\,f \star_D \gamma e) =$
$(\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f, w, e]$
using *ψ ϑ_C Fϑ_C-def D.iso-is-section D.section-is-mono*
by (*metis D.mono-cancel*)
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have $... = \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D \mathrm{a}_D[u, e, d] \cdot_D$

$((\vartheta \star_D e) \star_D d) \cdot_D ((\mathrm{a}_D{}^{-1}[F\,f,\ w,\ e] \star_D d) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w \star_D e,\ d] \cdot_D (F\,f \star_D \mathrm{a}_D{}^{-1}[w,\ e,\ d])) \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$

proof −
have $(\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ e] \star_D d =$
$((\vartheta \star_D e) \star_D d) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w,\ e] \star_D d)$
proof −
have $D.arr\ ((\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ e])$
by (*metis C.arrI D.cod-comp D.seqE D.seqI* $F\vartheta_C$*-def* ϑ_C *preserves-arr*)
thus *?thesis*
using *D.whisker-right e.ide-right* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D \mathrm{a}_D[u,\ e,\ d] \cdot_D$
$(((\vartheta \star_D e) \star_D d) \cdot_D \mathrm{a}_D{}^{-1}[F\,f \star_D w,\ e,\ d]) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ e \star_D d] \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using *w D.pentagon′ D.comp-assoc* **by** *simp*
also have ... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D ((\mathrm{a}_D[u,\ e,\ d] \cdot_D$
$\mathrm{a}_D{}^{-1}[u,\ e,\ d]) \cdot_D (\vartheta \star_D e \star_D d)) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ e \star_D d] \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using ϑ *e.antipar D.assoc′-naturality* [*of* ϑ *e d*] *D.comp-assoc* **by** *fastforce*
also have ... $= \mathrm{r}_D[u] \cdot_D (u \star_D \varepsilon) \cdot_D (\vartheta \star_D e \star_D d) \cdot_D (\mathrm{a}_D{}^{-1}[F\,f,\ w,\ e \star_D d] \cdot_D$
$(F\,f \star_D w \star_D D.inv\ \varepsilon)) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(\mathrm{a}_D[u,\ e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[u,\ e,\ d]) \cdot_D (\vartheta \star_D e \star_D d) = \vartheta \star_D e \star_D d$
proof −
have $(\mathrm{a}_D[u,\ e,\ d] \cdot_D \mathrm{a}_D{}^{-1}[u,\ e,\ d]) \cdot_D (\vartheta \star_D e \star_D d) =$
$(u \star_D e \star_D d) \cdot_D (\vartheta \star_D e \star_D d)$
using ϑ *ue e.ide-left e.ide-right e.antipar D.comp-arr-inv′ D.comp-cod-arr*
by *auto*
also have ... $= \vartheta \star_D e \star_D d$
using *ue e.ide-left e.ide-right e.antipar D.hcomp-simps*(*4*) *D.hseq-char′* ϑ
D.comp-cod-arr [*of* $\vartheta \star_D e \star_D d\ u \star_D e \star_D d$]
by *force*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[u] \cdot_D ((u \star_D \varepsilon) \cdot_D (\vartheta \star_D e \star_D d)) \cdot_D ((F\,f \star_D w) \star_D D.inv\ \varepsilon) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e] \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
using *w e.antipar D.assoc′-naturality* [*of F f w D.inv* ε] *D.comp-assoc* **by** *simp*
also have
... $= \mathrm{r}_D[u] \cdot_D (\vartheta \star_D trg_D\ e) \cdot_D (((F\,f \star_D w) \star_D \varepsilon) \cdot_D ((F\,f \star_D w) \star_D D.inv\ \varepsilon) \cdot_D$
$\mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e]) \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(u \star_D \varepsilon) \cdot_D (\vartheta \star_D e \star_D d) = (\vartheta \star_D trg_D\ e) \cdot_D ((F\,f \star_D w) \star_D \varepsilon)$

using ϑ *e.antipar D.interchange D.comp-arr-dom D.comp-cod-arr*
by (*metis D.in-homE* ‹$trg_D\ e = a$› *e.counit-simps*(*1*−*3*,*5*))
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[u] \cdot_D (\vartheta \star_D trg_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e] \cdot_D (F\,f \star_D \mathrm{r}_D{}^{-1}[w])$
proof −
have $(((F\,f \star_D w) \star_D \varepsilon) \cdot_D ((F\,f \star_D w) \star_D D.inv\ \varepsilon)) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e] = \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e]$
proof −
have $(((F\,f \star_D w) \star_D \varepsilon) \cdot_D ((F\,f \star_D w) \star_D D.inv\ \varepsilon)) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e] = ((F\,f \star_D w) \star_D trg_D\ e) \cdot_D \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e]$
using *w e.ide-left e.ide-right e.antipar e.counit-is-iso D.comp-arr-inv′ D.comp-assoc D.whisker-left*
by (*metis D.ide-hcomp D.seqI′ T′.ide-leg0 T′.leg1-simps*(*3*) *T.leg1-simps*(*2*−*3*) *T.tab-simps*(*2*) ‹$trg_D\ w = map_0\ (src_C\ \varrho)$› *d.unit-in-vhom e.counit-in-hom*(*2*) *e.counit-simps*(*3*) *preserves-src*)
also have ... $= \mathrm{a}_D{}^{-1}[F\,f,\ w,\ trg_D\ e]$
using *w D.comp-cod-arr D.assoc′-in-hom*(*2*) [*of F f w* $trg_D\ e$] ‹$trg_D\ e = a$› ‹$trg_D\ w = map_0\ (src_C\ \varrho)$›
by (*metis D.assoc′-naturality1 D.ideD*(*1*) *D.ideD*(*2*) *D.trg.preserves-ide D.trg-trg T′.leg0-simps*(*2*,*4*) *T′.leg1-simps*(*3*) *T.leg1-simps*(*2*−*3*) *T.tab-simps*(*2*) *a-def e.ide-left preserves-src*)
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{r}_D[u] \cdot_D (\vartheta \star_D trg_D\ e)) \cdot_D \mathrm{r}_D{}^{-1}[F\,f \star_D w]$
using *w D.runit-hcomp*(*2*) [*of F f w*] *D.comp-assoc* **by** *simp*
also have *1*: ... $= (\vartheta \cdot_D \mathrm{r}_D[F\,f \star_D w]) \cdot_D \mathrm{r}_D{}^{-1}[F\,f \star_D w]$
using ϑ *D.runit-naturality* [*of* ϑ] **by** *auto*
also have ... $= \vartheta$
using *w* ϑ *D.comp-arr-dom D.comp-assoc*
by (*metis D.hcomp-arr-obj*(*2*) *D.in-homE D.obj-src 1* ‹$src_D\ \vartheta = a$› ‹$trg_D\ e = a$›)
finally show *?thesis* **by** *simp*
qed
ultimately show «$?\gamma : w \Rightarrow_D w'$» $\wedge\ \beta = F\,g \star_D ?\gamma \wedge \vartheta = \vartheta' \cdot_D (F\,f \star_D ?\gamma)$
by *simp*

show $\bigwedge\gamma'.$ «$\gamma' : w \Rightarrow_D w'$» $\wedge\ \beta = F\,g \star_D \gamma' \wedge \vartheta = \vartheta' \cdot_D (F\,f \star_D \gamma') \Longrightarrow \gamma' = ?\gamma$
proof −
fix γ'
assume γ': «$\gamma' : w \Rightarrow_D w'$» $\wedge\ \beta = F\,g \star_D \gamma' \wedge \vartheta = \vartheta' \cdot_D (F\,f \star_D \gamma')$
show $\gamma' = ?\gamma$
proof −
have $?\gamma = \mathrm{r}_D[w'] \cdot_D (w' \star_D \varepsilon) \cdot_D (\mathrm{a}_D[w',\ e,\ d] \cdot_D ((\gamma' \star_D e) \star_D d)) \cdot_D \mathrm{a}_D{}^{-1}[w,\ e,\ d] \cdot_D (w \star_D D.inv\ \varepsilon) \cdot_D \mathrm{r}_D{}^{-1}[w]$

proof −
  have $\gamma e = \gamma' \star_D e$
  proof −
    have «$\gamma' \star_D e : w \star_D e \Rightarrow_D w' \star_D e$»
      using γ' **by** (*intro D.hcomp-in-vhom*, *auto*)
    moreover have
      $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] = F\ g \star_D \gamma' \star_D e$
    proof −
      have $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D (\beta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e] =$
          $\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D ((F\ g \star_D \gamma') \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w,\ e]$
        using γ' **by** *simp*
      also have ... $= \mathrm{a}_D[F\ g,\ w',\ e] \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w',\ e] \cdot_D (F\ g \star_D \gamma' \star_D e)$
        using γ' *D.assoc-naturality*
        by (*metis D.assoc'-naturality D.hcomp-in-vhomE D.ideD(2) D.ideD(3)*
          D.in-homE T'.leg1-simps(5−6) β
          ‹«$\gamma' \star_D e : w \star_D e \Rightarrow_D w' \star_D e$»› *e.ide-left*)
      also have ... $= (\mathrm{a}_D[F\ g,\ w',\ e] \cdot_D \mathrm{a}_D{}^{-1}[F\ g,\ w',\ e]) \cdot_D (F\ g \star_D \gamma' \star_D e)$
        using *D.comp-assoc* **by** *simp*
      also have ... $= F\ g \star_D \gamma' \star_D e$
       by (*metis D.hcomp-reassoc(2) D.in-homE D.not-arr-null D.seq-if-composable*
          T'.leg1-simps(2,5−6) β γ' calculation
          ‹«$\gamma' \star_D e : w \star_D e \Rightarrow_D w' \star_D e$»› *e.triangle-equiv-form(1)*
          e.triangle-in-hom(3) e.triangle-right e.triangle-right-implies-left)
      finally show *?thesis* **by** *simp*
    qed
    moreover have $\psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e] =$
          $\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \gamma' \star_D e)$
    proof −
      have $\psi \cdot_D (\vartheta' \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w',\ e] \cdot_D (F\ f \star_D \gamma' \star_D e) =$
          $\psi \cdot_D (\vartheta' \star_D e) \cdot_D ((F\ f \star_D \gamma') \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]$
        using γ' ϑ *e.ide-left D.assoc'-naturality*
        by (*metis D.hcomp-in-vhomE D.ideD(2) D.ideD(3) D.in-homE*
          T'.leg0-simps(2,4−5) T'.leg1-simps(3) β calculation(1))
      also have ... $= \psi \cdot_D ((\vartheta' \star_D e) \cdot_D ((F\ f \star_D \gamma') \star_D e)) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]$
        using *D.comp-assoc* **by** *simp*
      also have ... $= \psi \cdot_D (\vartheta' \cdot_D (F\ f \star_D \gamma') \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]$
        using *D.whisker-right* γ' ϑ **by** *auto*
      also have ... $= \psi \cdot_D (\vartheta \star_D e) \cdot_D \mathrm{a}_D{}^{-1}[F\ f,\ w,\ e]$
        using γ' **by** *simp*
      finally show *?thesis* **by** *simp*
    qed
    ultimately show *?thesis*
      using *UN* **by** *simp*
  qed
  thus *?thesis*
    using *D.comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{r}_D[w'] \cdot_D ((w' \star_D \varepsilon) \cdot_D (\gamma' \star_D e \star_D d)) \cdot_D \mathrm{a}_D[w,\ e,\ d] \cdot_D$
          $\mathrm{a}_D{}^{-1}[w,\ e,\ d] \cdot_D (w \star_D D.inv\ \varepsilon) \cdot_D \mathrm{r}_D{}^{-1}[w]$

using w' γ' *D.comp-assoc D.assoc-naturality*
by (*metis D.in-homE D.src-dom* ‹trg_D $e = a$› *a-def e.antipar*(*1*)
e.triangle-equiv-form(*1*) *e.triangle-in-hom*(*3*−*4*)
e.triangle-right e.triangle-right′ e.triangle-right-implies-left)
also have ... $= (\mathrm{r}_D[w'] \cdot_D (\gamma' \star_D trg_D\ e)) \cdot_D (w \star_D \varepsilon) \cdot_D \mathrm{a}_D[w, e, d] \cdot_D$
$\mathrm{a}_D{}^{-1}[w, e, d] \cdot_D (w \star_D D.inv\ \varepsilon) \cdot_D \mathrm{r}_D{}^{-1}[w]$
proof −
have $(w' \star_D \varepsilon) \cdot_D (\gamma' \star_D e \star_D d) = \gamma' \star_D \varepsilon$
using w' γ' *e.antipar D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* w' γ' ε $e \star_D d$]
by *auto*
also have ... $= (\gamma' \star_D trg_D\ e) \cdot_D (w \star_D \varepsilon)$
using w γ' *e.antipar D.comp-arr-dom D.comp-cod-arr D.interchange*
by (*metis D.in-homE* ‹trg_D $e = a$› *e.counit-simps*(*1*) *e.counit-simps*(*3*,*5*))
finally have $(w' \star_D \varepsilon) \cdot_D (\gamma' \star_D e \star_D d) = (\gamma' \star_D trg_D\ e) \cdot_D (w \star_D \varepsilon)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \gamma' \cdot_D \mathrm{r}_D[w] \cdot_D (w \star_D \varepsilon) \cdot_D \mathrm{a}_D[w, e, d] \cdot_D \mathrm{a}_D{}^{-1}[w, e, d] \cdot_D$
$(w \star_D D.inv\ \varepsilon) \cdot_D \mathrm{r}_D{}^{-1}[w]$
using γ' *D.runit-naturality D.comp-assoc*
by (*metis D.in-homE D.src-dom* ‹trg_D $e = a$› *a-def*)
also have ... $= \gamma'$
proof −
have $\mathrm{r}_D[w] \cdot_D (w \star_D \varepsilon) \cdot_D \mathrm{a}_D[w, e, d] \cdot_D \mathrm{a}_D{}^{-1}[w, e, d] \cdot_D (w \star_D D.inv\ \varepsilon) \cdot_D$
$\mathrm{r}_D{}^{-1}[w] =$
$\mathrm{r}_D[w] \cdot_D ((w \star_D \varepsilon) \cdot_D (\mathrm{a}_D[w, e, d] \cdot_D \mathrm{a}_D{}^{-1}[w, e, d]) \cdot_D (w \star_D D.inv\ \varepsilon)) \cdot_D$
$\mathrm{r}_D{}^{-1}[w]$
using *D.comp-assoc* **by** *simp*
also have ... $= \mathrm{r}_D[w] \cdot_D ((w \star_D \varepsilon) \cdot_D (w \star_D e \star_D d) \cdot_D (w \star_D D.inv\ \varepsilon)) \cdot_D$
$\mathrm{r}_D{}^{-1}[w]$
using w γ *e.ide-left e.ide-right we e.antipar D.comp-assoc-assoc′*(*1*)
‹trg_D $e = a$› *a-def*
by *presburger*
also have ... $= \mathrm{r}_D[w] \cdot_D ((w \star_D \varepsilon) \cdot_D (w \star_D D.inv\ \varepsilon)) \cdot_D \mathrm{r}_D{}^{-1}[w]$
using w γ *e.ide-left e.ide-right we e.antipar D.comp-cod-arr*
by (*metis D.whisker-left d.unit-simps*(*1*,*3*))
also have ... $= \mathrm{r}_D[w] \cdot_D (w \star_D src_D\ w) \cdot_D \mathrm{r}_D{}^{-1}[w]$
using w *e.counit-is-iso C.comp-arr-inv′*
by (*metis D.comp-arr-inv′ D.seqI′ D.whisker-left* ‹trg_D $e = a$› *a-def*
d.unit-in-vhom e.counit-in-hom(*2*) *e.counit-simps*(*3*))
also have ... $= \mathrm{r}_D[w] \cdot_D \mathrm{r}_D{}^{-1}[w]$
using w *e.antipar D.comp-cod-arr* **by** *simp*
also have ... $= w$
using w
by (*simp add*: *D.comp-arr-inv′*)
finally have $\mathrm{r}_D[w] \cdot_D (w \star_D \varepsilon) \cdot_D \mathrm{a}_D[w, e, d] \cdot_D \mathrm{a}_D{}^{-1}[w, e, d] \cdot_D$
$(w \star_D D.inv\ \varepsilon) \cdot_D \mathrm{r}_D{}^{-1}[w] = w$

```
            by simp
          thus ?thesis
            using γ′ D.comp-arr-dom by auto
        qed
        finally show ?thesis by simp
      qed
    qed
   qed
  qed
 qed
 show ?thesis ..
qed

lemma reflects-tabulation:
assumes C.ide r and C.ide f and «ϱ : g ⇒C r ⋆C f»
assumes tabulation VD HD aD iD srcD trgD (F r) (D.inv (Φ (r, f)) ·D F ϱ) (F f) (F g)
shows tabulation VC HC aC iC srcC trgC r ϱ f g
proof −
 interpret ϱ′: tabulation VD HD aD iD srcD trgD
              ‹F r› ‹D.inv (Φ (r, f)) ·D F ϱ› ‹F f› ‹F g›
   using assms by auto
 interpret ϱ: tabulation-data VC HC aC iC srcC trgC r ϱ f g
   using assms by (unfold-locales, simp-all)
 interpret ϱ: tabulation VC HC aC iC srcC trgC r ϱ f g
 proof
   show ⋀u ω. ⟦ C.ide u; «ω : C.dom ω ⇒C r ⋆C u» ⟧ ⟹
              ∃ w ϑ ν. C.ide w ∧ «ϑ : f ⋆C w ⇒C u» ∧ «ν : C.dom ω ⇒C g ⋆C w» ∧
                     C.iso ν ∧ ϱ.composite-cell w ϑ ·C ν = ω
   proof −
     fix u ω
     assume u: C.ide u
     assume ω: «ω : C.dom ω ⇒C r ⋆C u»
     have hseq-ru: srcC r = trgC u
       using ω C.ide-cod C.ideD(1) by fastforce
     hence 1: «D.inv (Φ (r, u)) ·D F ω : F (C.dom ω) ⇒D F r ⋆D F u»
       using assms u ω cmp-in-hom cmp-components-are-iso
       by (intro D.comp-in-homI, auto)
     hence 2: D.dom (D.inv (Φ (r, u)) ·D F ω) = F (C.dom ω)
       by auto
     obtain w ϑ ν
       where wϑν: D.ide w ∧ «ϑ : F f ⋆D w ⇒D F u» ∧
                  «ν : F (C.dom ω) ⇒D F g ⋆D w» ∧ D.iso ν ∧
                  ϱ′.composite-cell w ϑ ·D ν = D.inv (Φ (r, u)) ·D F ω
       using 1 2 u ϱ′.T1 [of F u D.inv (Φ (r, u)) ·D F ω] by auto
     have hseq-Ff-w: srcD (F f) = trgD w
       using u ω wϑν
       by (metis 1 D.arrI D.not-arr-null D.seqE D.seq-if-composable ϱ′.tab-simps(2))
     have hseq-Fg-w: srcD (F g) = trgD w
       using u ω wϑν by (simp add: hseq-Ff-w)
```

have *w*: «w : map_0 (src_C ω) $\rightarrow_D$ map_0 (src_C f)»
  using *u ω wϑν hseq-Fg-w*
  by (*metis 1 C.arrI D.arrI D.hseqI′ D.ideD(1) D.in-hhom-def D.src-hcomp D.src-vcomp D.vconn-implies-hpar(1) D.vconn-implies-hpar(3) D.vseq-implies-hpar(1) ϱ′.leg1-simps(2) ϱ.leg0-simps(2) hseq-Ff-w preserves-src*)
obtain w' **where** w': «w' : src_C ω $\rightarrow_C$ src_C f» $\wedge$ *C.ide* w' $\wedge$ *D.isomorphic* (F w') w
 using *assms w ω wϑν locally-essentially-surjective* **by** *force*
obtain φ **where** φ: «φ : F w' $\Rightarrow_D$ w» $\wedge$ *D.iso* φ
 using *w′ D.isomorphic-def* **by** *blast*
have *src-fw′*: src_C (f $\star_C$ w') = src_C u
 using *u w′ ω*
 by (*metis C.hseqI′ C.ideD(1) C.in-hhomE C.src-hcomp C.vconn-implies-hpar(1) C.vconn-implies-hpar(3) ϱ.base-simps(2) ϱ.leg0-in-hom(1) hseq-ru*)
have *3*: «ϑ $\cdot_D$ (F f $\star_D$ φ) $\cdot_D$ *D.inv* (Φ (f, w')) : F (f $\star_C$ w') $\Rightarrow_D$ F u»
proof (*intro D.comp-in-homI*)
 show «*D.inv* (Φ (f, w')) : F (f $\star_C$ w') $\Rightarrow_D$ F f $\star_D$ F w'»
  using *assms w′ cmp-in-hom cmp-components-are-iso* **by** *auto*
 show «F f $\star_D$ φ : F f $\star_D$ F w' $\Rightarrow_D$ F f $\star_D$ w»
  using *φ ϱ′.leg0-in-hom(2) w′* **by** *fastforce*
 show «ϑ : F f $\star_D$ w $\Rightarrow_D$ F u»
  using *wϑν* **by** *simp*
qed
have *4*: $\exists\vartheta'$. «ϑ' : f $\star_C$ w' $\Rightarrow_C$ u» $\wedge$ F ϑ' = ϑ $\cdot_D$ (F f $\star_D$ φ) $\cdot_D$ *D.inv* (Φ (f, w'))
 using *w′ u hseq-ru src-fw′ 3 locally-full* **by** *auto*
obtain ϑ' **where**
 ϑ': «ϑ' : f $\star_C$ w' $\Rightarrow_C$ u» $\wedge$ F ϑ' = ϑ $\cdot_D$ (F f $\star_D$ φ) $\cdot_D$ *D.inv* (Φ (f, w'))
 using *4* **by** *auto*
have *5*: «Φ (g, w') $\cdot_D$ (F g $\star_D$ *D.inv* φ) $\cdot_D$ ν : F (*C.dom* ω) $\Rightarrow_D$ F (g $\star_C$ w')»
proof (*intro D.comp-in-homI*)
 show «ν : F (*C.dom* ω) $\Rightarrow_D$ F g $\star_D$ w»
  using *wϑν* **by** *simp*
 show «F g $\star_D$ *D.inv* φ : F g $\star_D$ w $\Rightarrow_D$ F g $\star_D$ F w'»
  using *assms φ*
  by (*meson D.hcomp-in-vhom D.inv-in-hom ϱ′.leg1-in-hom(2) hseq-Fg-w*)
 show «Φ (g, w') : F g $\star_D$ F w' $\Rightarrow_D$ F (g $\star_C$ w')»
  using *assms w′ cmp-in-hom* **by** *auto*
qed
have *6*: $\exists\nu'$. «ν' : *C.dom* ω $\Rightarrow_C$ g $\star_C$ w'» $\wedge$
 F ν' = Φ(g, w') $\cdot_D$ (F g $\star_D$ *D.inv* φ) $\cdot_D$ ν
 using *u w′ ω C.in-hhom-def hseq-ru C.hseqI′ C.hcomp-simps(1−2)*
 by (*metis 5 C.arrI C.ide-hcomp C.ideD(1) C.ide-dom C.vconn-implies-hpar(1,4) ϱ.base-simps(2) ϱ.ide-leg1 ϱ.leg1-in-hom(1) locally-full*)
obtain ν' **where**
 ν': «ν' : *C.dom* ω $\Rightarrow_C$ g $\star_C$ w'» $\wedge$ F ν' = Φ(g, w') $\cdot_D$ (F g $\star_D$ *D.inv* φ) $\cdot_D$ ν
 using *6* **by** *auto*
have *C.ide* w' $\wedge$ «ϑ' : f $\star_C$ w' $\Rightarrow_C$ u» $\wedge$ «ν' : *C.dom* ω $\Rightarrow_C$ g $\star_C$ w'» $\wedge$ *C.iso* ν' $\wedge$
 ϱ.composite-cell w' ϑ' $\cdot_C$ ν' = ω
 using *w′ ϑ′ ν′*

```
  apply (intro conjI)
      apply auto
proof −
  show C.iso ν′
  proof −
    have D.iso (F ν′)
    proof −
      have D.iso (Φ(g, w′))
        using w′ cmp-components-are-iso by auto
      moreover have D.iso (F g ⋆D D.inv φ)
        using φ
        by (meson 5 D.arrI D.iso-hcomp D.hseq-char′ D.ide-is-iso D.iso-inv-iso
            D.seqE D.seq-if-composable ϱ′.ide-leg1)
      moreover have D.iso ν
        using wϑν by simp
      ultimately show ?thesis
        using ν′ D.isos-compose
        by (metis 5 D.arrI D.seqE)
    qed
    thus ?thesis using reflects-iso by blast
  qed
  have 7: «ϱ.composite-cell w′ ϑ′ : g ⋆C w′ ⇒C r ⋆C u»
    using u w′ ϑ′ ϱ.composite-cell-in-hom hseq-ru src-fw′ C.hseqI′
    by (metis C.in-hhomE C.hcomp-simps(1) ϱ.leg0-simps(2))
  hence 8: «ϱ.composite-cell w′ ϑ′ ·C ν′ : C.dom ω ⇒C r ⋆C u»
    using ν′ by blast
  show ϱ.composite-cell w′ ϑ′ ·C ν′ = ω
  proof −
    have 1: C.par (ϱ.composite-cell w′ ϑ′ ·C ν′) ω
      using ω 8 hseq-ru C.hseqI′ C.in-homE by metis
    moreover have F (ϱ.composite-cell w′ ϑ′ ·C ν′) = F ω
    proof −
      have F (ϱ.composite-cell w′ ϑ′ ·C ν′) =
            F (r ⋆C ϑ′) ·D F aC[r, f, w′] ·D F (ϱ ⋆C w′) ·D F ν′
        using w′ ϑ′ ν′ 1 C.comp-assoc
        by (metis C.seqE preserves-comp)
      also have ... = Φ (r, u) ·D (F r ⋆D F ϑ′) ·D ((D.inv (Φ (r, f ⋆C w′)) ·D
                      Φ (r, f ⋆C w′)) ·D (F r ⋆D Φ (f, w′))) ·D
                      aD[F r, F f, F w′] ·D (D.inv (Φ (r, f)) ⋆D F w′) ·D
                      ((D.inv (Φ (r ⋆C f, w′)) ·D
                      Φ (r ⋆C f, w′)) ·D (F ϱ ⋆D F w′)) ·D D.inv (Φ (g, w′)) ·D F ν′
      proof −
        have F aC[r, f, w′] =
              Φ (r, f ⋆C w′) ·D (F r ⋆D Φ (f, w′)) ·D aD[F r, F f, F w′] ·D
              (D.inv (Φ (r, f)) ⋆D F w′) ·D D.inv (Φ (r ⋆C f, w′))
          using assms w′
          by (simp add: C.in-hhom-def preserves-assoc(1))
        moreover have
          F (r ⋆C ϑ′) = Φ (r, u) ·D (F r ⋆D F ϑ′) ·D D.inv (Φ (r, f ⋆C w′))
```

using *assms* ϑ' *preserves-hcomp* [*of r* ϑ']
by (*metis 1 C.in-homE C.seqE* ϱ*.base-simps*(*3*) ϱ*.base-simps*(*4*))
moreover have
$F\ (\varrho \star_C w') = \Phi\ (r \star_C f,\ w') \cdot_D (F\ \varrho \star_D F\ w') \cdot_D D.inv\ (\Phi\ (g,\ w'))$
using w' *preserves-hcomp* [*of* ϱ w'] **by** *auto*
ultimately show *?thesis*
by (*simp add*: *D.comp-assoc*)
qed
also have ... $= \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta') \cdot_D (F\ r \star_D \Phi\ (f,\ w')) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w'] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \star_D F\ w') \cdot_D$
$(F\ \varrho \star_D F\ w') \cdot_D D.inv\ (\Phi\ (g,\ w')) \cdot_D F\ \nu'$
proof −
have $(D.inv\ (\Phi\ (r,\ f \star_C w')) \cdot_D \Phi\ (r,\ f \star_C w')) \cdot_D (F\ r \star_D \Phi\ (f,\ w')) =$
$F\ r \star_D \Phi\ (f,\ w')$
using w' *cmp-components-are-iso D.comp-cod-arr C.hseqI′ D.hseqI′*
C.in-hhom-def C.trg-hcomp D.comp-inv-arr′ C.ide-hcomp
by (*metis C.ideD*(*1*) *D.hcomp-simps*(*4*) *cmp-simps*(*1*,*3*−*5*)
ϱ'*.leg0-simps*(*3*) ϱ'*.base-simps*(*2*,*4*) ϱ*.ide-leg0* ϱ*.ide-base*
ϱ*.leg0-simps*(*3*))
moreover have $(D.inv\ (\Phi\ (r \star_C f,\ w')) \cdot_D \Phi\ (r \star_C f,\ w')) \cdot_D (F\ \varrho \star_D F\ w') =$
$F\ \varrho \star_D F\ w'$
using w' *D.comp-inv-arr′ hseq-Fg-w D.comp-cod-arr* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u) \cdot_D ((F\ r \star_D \vartheta \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w'))) \cdot_D$
$(F\ r \star_D \Phi\ (f,\ w'))) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w'] \cdot_D$
$((D.inv\ (\Phi\ (r,\ f)) \star_D F\ w') \cdot_D (F\ \varrho \star_D F\ w')) \cdot_D$
$D.inv\ (\Phi\ (g,\ w')) \cdot_D \Phi\ (g,\ w') \cdot_D (F\ g \star_D D.inv\ \varphi) \cdot_D \nu$
using w' ϑ' ν' *D.comp-assoc* **by** *simp*
also have ... $= \Phi\ (r,\ u) \cdot_D (F\ r \star_D \vartheta \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w')) \cdot_D$
$\Phi\ (f,\ w')) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w'] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \cdot_D$
$F\ \varrho \star_D F\ w') \cdot_D ((D.inv\ (\Phi\ (g,\ w')) \cdot_D \Phi\ (g,\ w')) \cdot_D$
$(F\ g \star_D D.inv\ \varphi)) \cdot_D \nu$
proof −
have $(F\ r \star_D \vartheta \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w'))) \cdot_D (F\ r \star_D \Phi\ (f,\ w')) =$
$F\ r \star_D (\vartheta \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w'))) \cdot_D \Phi\ (f,\ w')$
proof −
have $D.seq\ (\vartheta \cdot_D (F\ f \star_D \varphi) \cdot_D D.inv\ (\Phi\ (f,\ w'))) (\Phi\ (f,\ w'))$
using *assms 3* ϱ*.ide-base* w' $w\vartheta\nu$ *cmp-in-hom* [*of f* w'] *cmp-components-are-iso*
C.in-hhom-def
apply (*intro D.seqI*)
using *C.in-hhom-def*
apply *auto*[*3*]
apply *blast*
by *auto*
thus *?thesis*
using *assms* w' $w\vartheta\nu$ *cmp-in-hom cmp-components-are-iso D.whisker-left*
by *simp*
qed

moreover have $(D.inv\ (\Phi\ (r,\ f))\ \star_D\ F\ w')\ \cdot_D\ (F\ \varrho\ \star_D\ F\ w') =$
$D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ F\ w'$
using *w′ D.whisker-right* **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ \vartheta\ \cdot_D\ (F\ f\ \star_D\ \varphi))\ \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D\ ((D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ F\ w')\ \cdot_D$
$(F\ g\ \star_D\ D.inv\ \varphi))\ \cdot_D\ \nu$
proof −
have $(F\ f\ \star_D\ \varphi)\ \cdot_D\ D.inv\ (\Phi\ (f,\ w'))\ \cdot_D\ \Phi\ (f,\ w') = F\ f\ \star_D\ \varphi$
using *assms(2) w′ φ 3 cmp-components-are-iso cmp-in-hom D.hseqI′ D.comp-inv-arr′*
D.comp-arr-dom
by (*metis C.in-hhom-def D.arrI D.cod-inv D.seqE*)
moreover have $(D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ \Phi\ (g,\ w'))\ \cdot_D\ (F\ g\ \star_D\ D.inv\ \varphi) =$
$F\ g\ \star_D\ D.inv\ \varphi$
using *assms w′ φ 3 cmp-components-are-iso cmp-in-hom D.hseqI′*
D.comp-inv-arr′ D.comp-cod-arr
by (*metis 5 C.in-hhom-def D.arrI D.comp-assoc D.seqE ϱ.ide-leg1*
ϱ.leg1-simps(3))
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u)\ \cdot_D\ (F\ r\ \star_D\ \vartheta\ \cdot_D\ (F\ f\ \star_D\ \varphi))\ \cdot_D$
$(\mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D\ ((F\ r\ \star_D\ F\ f)\ \star_D\ D.inv\ \varphi))\ \cdot_D$
$(D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ w)\ \cdot_D\ \nu$
proof −
have $(D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ F\ w')\ \cdot_D\ (F\ g\ \star_D\ D.inv\ \varphi) =$
$D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ D.inv\ \varphi$
using *assms w′ φ cmp-in-hom cmp-components-are-iso D.comp-arr-dom*
D.comp-cod-arr
D.interchange [*of D.inv* $(\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ F\ g\ F\ w'\ D.inv\ \varphi$]
by *auto*
also have ... $= ((F\ r\ \star_D\ F\ f)\ \star_D\ D.inv\ \varphi)\ \cdot_D\ (D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ w)$
using *assms w′ φ cmp-components-are-iso D.comp-arr-dom D.comp-cod-arr*
D.interchange [*of* $F\ r\ \star_D\ F\ f\ D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ D.inv\ \varphi\ w$]
by *auto*
finally have $(D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ F\ w')\ \cdot_D\ (F\ g\ \star_D\ D.inv\ \varphi) =$
$((F\ r\ \star_D\ F\ f)\ \star_D\ D.inv\ \varphi)\ \cdot_D\ (D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ w)$
by *simp*
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u)\ \cdot_D\ ((F\ r\ \star_D\ \vartheta\ \cdot_D\ (F\ f\ \star_D\ \varphi))\ \cdot_D$
$(F\ r\ \star_D\ F\ f\ \star_D\ D.inv\ \varphi))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ w]\ \cdot_D$
$(D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho\ \star_D\ w)\ \cdot_D\ \nu$
proof −
have $\mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D\ ((F\ r\ \star_D\ F\ f)\ \star_D\ D.inv\ \varphi) =$

$(F\ r \star_D F\ f \star_D D.inv\ \varphi) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w]$
proof −
have $src_D\ (F\ r) = trg_D\ (F\ f)$
by *simp*
moreover have $src_D\ (F\ f) = trg_D\ (D.inv\ \varphi)$
using φ
by (*metis 5 D.arrI D.hseqE D.seqE* ϱ'*.leg1-simps*(*3*))
ultimately show *?thesis*
using *assms* w' φ *D.assoc-naturality* [*of F r F f D.inv* φ] **by** *auto*
qed
thus *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= \Phi\ (r,\ u) \cdot_D (F\ r \star_D \vartheta) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ w] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \cdot_D F\ \varrho \star_D w) \cdot_D \nu$
using *assms* φ $w\vartheta\nu$ *D.comp-arr-inv′ D.comp-arr-dom D.comp-cod-arr D.whisker-left D.whisker-left D.comp-assoc*
by (*metis D.ideD*(*1*) *D.in-homE* ϱ'*.ide-base tabulation-data.leg0-simps*(*1*) *tabulation-def*)
also have ... $= (\Phi\ (r,\ u) \cdot_D D.inv\ (\Phi\ (r,\ u))) \cdot_D F\ \omega$
using $w\vartheta\nu$ *D.comp-assoc* **by** *simp*
also have ... $= F\ \omega$
using u ω *cmp-in-hom D.comp-arr-inv′*
by (*metis C.in-homE cmp-components-are-iso cmp-simps*(*5*) ϱ*.ide-base as-nat-trans.naturality1 as-nat-trans.naturality hseq-ru*)
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *is-faithful* [*of* ϱ*.composite-cell* w' $\vartheta' \cdot_C \nu'$ ω] **by** *simp*
qed
qed
thus $\exists\, w\ \vartheta\ \nu.\ C.ide\ w \wedge «\vartheta : f \star_C w \Rightarrow_C u» \wedge «\nu : C.dom\ \omega \Rightarrow_C g \star_C w» \wedge$
$C.iso\ \nu \wedge \varrho.composite\text{-}cell\ w\ \vartheta \cdot_C \nu = \omega$
by *auto*
qed

show $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.\ [\![\ C.ide\ w;\ C.ide\ w';\ «\vartheta : f \star_C w \Rightarrow_C u»;\ «\vartheta' : f \star_C w' \Rightarrow_C u»;$
$«\beta : g \star_C w \Rightarrow_C g \star_C w'»;$
$\varrho.composite\text{-}cell\ w\ \vartheta = \varrho.composite\text{-}cell\ w'\ \vartheta' \cdot_C \beta\ ]\!]$
$\Longrightarrow \exists!\gamma.\ «\gamma : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma)$
proof −
fix u w w' ϑ ϑ' β
assume w: $C.ide\ w$
assume w': $C.ide\ w'$
assume ϑ: $«\vartheta : f \star_C w \Rightarrow_C u»$
assume ϑ': $«\vartheta' : f \star_C w' \Rightarrow_C u»$
assume β: $«\beta : g \star_C w \Rightarrow_C g \star_C w'»$
assume *eq*: $\varrho.composite\text{-}cell\ w\ \vartheta = \varrho.composite\text{-}cell\ w'\ \vartheta' \cdot_C \beta$
show $\exists!\gamma.\ «\gamma : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma)$

proof −
have *hseq-ru*: $src_C\ r = trg_C\ u$
using $w\ \vartheta$
by (*metis C.hseq-char′ C.in-homE C.trg.extensionality C.trg.preserves-hom C.trg-hcomp C.vconn-implies-hpar(2) C.vconn-implies-hpar(4) ϱ.leg0-simps(3)*)
have *hseq-fw*: $src_C\ f = trg_C\ w \wedge src_C\ f = trg_C\ w'$
using $w\ w'$ *ϱ.ide-leg0* $\vartheta\ \vartheta'$
by (*metis C.horizontal-homs-axioms C.ideD(1) C.in-homE C.not-arr-null C.seq-if-composable category.ide-dom horizontal-homs-def*)
have *hseq-gw*: $src_C\ g = trg_C\ w \wedge src_C\ g = trg_C\ w'$
using $w\ w'$ *ϱ.ide-leg0* $\vartheta\ \vartheta'$ ‹$src_C\ f = trg_C\ w \wedge src_C\ f = trg_C\ w'$› **by** *auto*
have $*$: $\exists!\gamma.$ «$\gamma : F\ w \Rightarrow_D F\ w'$» $\wedge$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w) = F\ g \star_D \gamma\ \wedge$
$F\ \vartheta\ \cdot_D\ \Phi\ (f,\ w) = (F\ \vartheta'\ \cdot_D\ \Phi\ (f,\ w'))\ \cdot_D\ (F\ f \star_D \gamma)$
proof −
have $D.ide\ (F\ w) \wedge D.ide\ (F\ w')$
using $w\ w'$ **by** *simp*
moreover have *1*: «$F\ \vartheta\ \cdot_D\ \Phi\ (f,\ w) : F\ f \star_D F\ w \Rightarrow_D F\ u$»
using $w\ \vartheta$ *cmp-in-hom ϱ.ide-leg0 hseq-fw* **by** *blast*
moreover have *2*: «$F\ \vartheta'\ \cdot_D\ \Phi\ (f,\ w') : F\ f \star_D F\ w' \Rightarrow_D F\ u$»
using $w'\ \vartheta'$ *cmp-in-hom ϱ.ide-leg0 hseq-fw* **by** *blast*
moreover have
«$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w) : F\ g \star_D F\ w \Rightarrow_D F\ g \star_D F\ w'$»
using $w\ w'\ \beta$ *ϱ.ide-leg1 cmp-in-hom cmp-components-are-iso hseq-gw preserves-hom*
by *fastforce*
moreover have $\varrho'.composite\text{-}cell\ (F\ w)\ (F\ \vartheta\ \cdot_D\ \Phi\ (f,\ w)) =$
$\varrho'.composite\text{-}cell\ (F\ w')\ (F\ \vartheta'\ \cdot_D\ \Phi\ (f,\ w'))\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w)$
proof −
have $\varrho'.composite\text{-}cell\ (F\ w')\ (F\ \vartheta'\ \cdot_D\ \Phi\ (f,\ w'))\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w) =$
$(F\ r \star_D F\ \vartheta'\ \cdot_D\ \Phi\ (f,\ w'))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D$
$(D.inv\ (\Phi\ (r,\ f))\ \cdot_D\ F\ \varrho \star_D F\ w')\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w)$
using *D.comp-assoc* **by** *simp*
also have ... =
$(F\ r \star_D F\ \vartheta')\ \cdot_D\ (F\ r \star_D \Phi\ (f,\ w'))\ \cdot_D\ \mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \star_D F\ w')\ \cdot_D\ (F\ \varrho \star_D F\ w')\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w)$
using $w'\ \vartheta'$ *2 D.whisker-left D.whisker-right D.comp-assoc* **by** *auto*
also have ... = $(F\ r \star_D F\ \vartheta')\ \cdot_D\ ((D.inv\ (\Phi\ (r,\ f \star_C w'))\ \cdot_D$
$\Phi\ (r,\ f \star_C w'))\ \cdot_D\ (F\ r \star_D \Phi\ (f,\ w')))\ \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w']\ \cdot_D\ (D.inv\ (\Phi\ (r,\ f)) \star_D F\ w')\ \cdot_D$
$((D.inv\ (\Phi\ (r \star_C f,\ w'))\ \cdot_D$
$\Phi\ (r \star_C f,\ w'))\ \cdot_D\ (F\ \varrho \star_D F\ w'))\ \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))\ \cdot_D\ F\ \beta\ \cdot_D\ \Phi\ (g,\ w)$
proof −
have $(D.inv\ (\Phi\ (r,\ f \star_C w'))\ \cdot_D\ \Phi\ (r,\ f \star_C w'))\ \cdot_D\ (F\ r \star_D \Phi\ (f,\ w')) =$
$F\ r \star_D \Phi\ (f,\ w')$

using w' *cmp-components-are-iso D.comp-cod-arr C.hseqI′ D.hseqI′*
C.in-hhom-def C.trg-hcomp D.comp-inv-arr′ C.ide-hcomp
by (*metis C.ideD*(*1*) *D.hcomp-simps*(*4*) *cmp-simps*(*1*) *cmp-simps*(*3*−*5*)
ϱ'*.leg0-simps*(*3*) ϱ'*.base-simps*(*2*,*4*) ϱ*.ide-leg0* ϱ*.ide-base*
ϱ*.leg0-simps*(*3*) *hseq-fw*)
moreover have $(D.inv\ (\Phi\ (r \star_C f,\ w')) \cdot_D \Phi\ (r \star_C f,\ w')) \cdot_D (F\ \varrho \star_D F\ w') =$
$F\ \varrho \star_D F\ w'$
using w' *D.comp-inv-arr′ D.comp-cod-arr hseq-fw* **by** *auto*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... $= D.inv\ (\Phi\ (r,\ u)) \cdot_D$
$(\Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta') \cdot_D (D.inv\ (\Phi\ (r,\ f \star_C w'))) \cdot_D$
$(\Phi\ (r,\ f \star_C w')) \cdot_D (F\ r \star_D \Phi\ (f,\ w')) \cdot_D$
$\mathrm{a}_D[F\ r,\ F\ f,\ F\ w'] \cdot_D (D.inv\ (\Phi\ (r,\ f)) \star_D F\ w') \cdot_D$
$(D.inv\ (\Phi\ (r \star_C f,\ w')) \cdot_D$
$(\Phi\ (r \star_C f,\ w')) \cdot_D (F\ \varrho \star_D F\ w')) \cdot_D$
$D.inv\ (\Phi\ (g,\ w'))) \cdot_D F\ \beta \cdot_D \Phi\ (g,\ w)$
proof −
have $(D.inv\ (\Phi\ (r,\ u)) \cdot_D \Phi\ (r,\ u)) \cdot_D (F\ r \star_D F\ \vartheta') = F\ r \star_D F\ \vartheta'$
using *assms*(*1*) ϑ' *D.comp-cod-arr hseq-ru D.comp-inv-arr′* **by** *auto*
thus *?thesis*
using *D.comp-assoc* **by** *metis*
qed
also have ... $= D.inv\ (\Phi\ (r,\ u)) \cdot_D$
$(F\ (r \star_C \vartheta') \cdot_D F\ \mathrm{a}_C[r,\ f,\ w'] \cdot_D F\ (\varrho \star_C w')) \cdot_D$
$F\ \beta \cdot_D \Phi\ (g,\ w)$
proof −
have $F\ (r \star_C \vartheta') = \Phi\ (r,\ u) \cdot_D (F\ r \star_D F\ \vartheta') \cdot_D D.inv\ (\Phi\ (r,\ f \star_C w'))$
using w' ϑ' *preserves-hcomp hseq-ru* **by** *auto*
moreover have $F\ \mathrm{a}_C[r,\ f,\ w'] =$
$\Phi\ (r,\ f \star_C w') \cdot_D (F\ r \star_D \Phi\ (f,\ w')) \cdot_D \mathrm{a}_D[F\ r,\ F\ f,\ F\ w'] \cdot_D$
$(D.inv\ (\Phi\ (r,\ f)) \star_D F\ w') \cdot_D D.inv\ (\Phi\ (r \star_C f,\ w'))$
using w' *preserves-assoc*(*1*) *hseq-fw* **by** *force*
moreover have
$F\ (\varrho \star_C w') = \Phi\ (r \star_C f,\ w') \cdot_D (F\ \varrho \star_D F\ w') \cdot_D D.inv\ (\Phi\ (g,\ w'))$
using w' *preserves-hcomp hseq-fw* **by** *fastforce*
ultimately show *?thesis*
using *D.comp-assoc* **by** *auto*
qed
also have ... $= D.inv\ (\Phi\ (r,\ u)) \cdot_D F\ (\varrho$*.composite-cell* $w'\ \vartheta') \cdot_D F\ \beta \cdot_D \Phi\ (g,\ w)$
using w' ϑ' *C.comp-assoc hseq-ru hseq-fw* **by** *auto*
also have ... $= D.inv\ (\Phi\ (r,\ u)) \cdot_D (F\ (\varrho$*.composite-cell* $w'\ \vartheta') \cdot_D F\ \beta) \cdot_D \Phi\ (g,\ w)$
using *D.comp-assoc* **by** *simp*
also have ... $= D.inv\ (\Phi\ (r,\ u)) \cdot_D F\ (\varrho$*.composite-cell* $w'\ \vartheta' \cdot_C \beta) \cdot_D \Phi\ (g,\ w)$
proof −
have $F\ (\varrho$*.composite-cell* $w'\ \vartheta') \cdot_D F\ \beta = F\ (\varrho$*.composite-cell* $w'\ \vartheta' \cdot_C \beta)$
using w w' ϑ' β ϱ*.composite-cell-in-hom*
preserves-comp [*of* ϱ*.composite-cell* $w'\ \vartheta'\ \beta$]

by (*metis C.dom-comp C.hcomp-simps*(*3*) *C.ide-char C.in-homE C.seqE C.seqI D.ext D.seqE* ϱ*.tab-simps*(*4*) *extensionality preserves-reflects-arr*)
thus *?thesis* **by** *simp*
qed
also have ... = *D.inv* (Φ (r, u)) $\cdot_D$ F (ϱ*.composite-cell* w ϑ) $\cdot_D$ Φ (g, w)
using *eq* **by** *simp*
also have ... = *D.inv* (Φ (r, u)) $\cdot_D$
F ($r \star_C \vartheta$) $\cdot_D$ F $\mathrm{a}_C[r, f, w]$ $\cdot_D$ F ($\varrho \star_C w$) $\cdot_D$ Φ (g, w)
using w ϑ *C.comp-assoc hseq-ru hseq-fw D.comp-assoc* **by** *auto*
also have ... = ((*D.inv* (Φ (r, u)) $\cdot_D$
Φ (r, u)) $\cdot_D$ (F $r \star_D F\ \vartheta$)) $\cdot_D$ ((*D.inv* (Φ (r, $f \star_C w$)) $\cdot_D$
Φ (r, $f \star_C w$)) $\cdot_D$ (F $r \star_D \Phi$ (f, w))) $\cdot_D$
$\mathrm{a}_D[F\ r, F\ f, F\ w]$ $\cdot_D$ (*D.inv* (Φ (r, f)) $\star_D$ F w) $\cdot_D$
((*D.inv* (Φ ($r \star_C f$, w)) $\cdot_D$
Φ ($r \star_C f$, w)) $\cdot_D$ (F $\varrho \star_D F\ w$)) $\cdot_D$ *D.inv* (Φ (g, w)) $\cdot_D$ Φ (g, w)
proof −
have F ($r \star_C \vartheta$) = Φ (r, u) $\cdot_D$ (F $r \star_D F\ \vartheta$) $\cdot_D$ *D.inv* (Φ (r, $f \star_C w$))
using w ϑ *preserves-hcomp hseq-ru* **by** *auto*
moreover have F $\mathrm{a}_C[r, f, w]$ =
Φ (r, $f \star_C w$) $\cdot_D$ (F $r \star_D \Phi$ (f, w)) $\cdot_D$ $\mathrm{a}_D[F\ r, F\ f, F\ w]$ $\cdot_D$
(*D.inv* (Φ (r, f)) $\star_D$ F w) $\cdot_D$ *D.inv* (Φ ($r \star_C f$, w))
using w *preserves-assoc*(*1*) *hseq-fw* **by** *force*
moreover have
F ($\varrho \star_C w$) = Φ ($r \star_C f$, w) $\cdot_D$ (F $\varrho \star_D F\ w$) $\cdot_D$ *D.inv* (Φ (g, w))
using w *preserves-hcomp hseq-fw* **by** *fastforce*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
also have ... = (F $r \star_D F\ \vartheta$) $\cdot_D$ (F $r \star_D \Phi$ (f, w)) $\cdot_D$ $\mathrm{a}_D[F\ r, F\ f, F\ w]$ $\cdot_D$
(*D.inv* (Φ (r, f)) $\star_D$ F w) $\cdot_D$ (F $\varrho \star_D F\ w$)
proof −
have (*D.inv* (Φ (r, u)) $\cdot_D$ Φ (r, u)) $\cdot_D$ (F $r \star_D F\ \vartheta$) = F $r \star_D F\ \vartheta$
using ϑ *D.comp-cod-arr hseq-ru D.comp-inv-arr′* **by** *auto*
moreover have
(*D.inv* (Φ (r, $f \star_C w$)) $\cdot_D$ Φ (r, $f \star_C w$)) $\cdot_D$ (F $r \star_D \Phi$ (f, w)) =
F $r \star_D \Phi$ (f, w)
using w *cmp-components-are-iso D.comp-cod-arr C.hseqI′ D.hseqI′ C.in-hhom-def C.trg-hcomp D.comp-inv-arr′ C.ide-hcomp*
by (*metis C.ideD*(*1*) *D.hcomp-simps*(*4*) *cmp-simps*(*1*) *cmp-simps*(*3*−*5*) ϱ'*.leg0-simps*(*3*) ϱ'*.base-simps*(*2*,*4*) ϱ*.ide-leg0* ϱ*.ide-base* ϱ*.leg0-simps*(*3*) *hseq-fw*)
moreover have (*D.inv* (Φ ($r \star_C f$, w)) $\cdot_D$ Φ ($r \star_C f$, w)) $\cdot_D$ (F $\varrho \star_D F\ w$) =
F $\varrho \star_D F\ w$
using w *D.comp-inv-arr′ D.comp-cod-arr hseq-fw* **by** *simp*
moreover have (F $\varrho \star_D F\ w$) $\cdot_D$ *D.inv* (Φ (g, w)) $\cdot_D$ Φ (g, w) = F $\varrho \star_D F\ w$
using w ϑ *D.comp-arr-dom D.comp-inv-arr′ hseq-gw* **by** *simp*
ultimately show *?thesis*
using *D.comp-assoc* **by** *simp*
qed

```
    also have ... = ϱ'.composite-cell (F w) (F ϑ ·D Φ (f, w))
      using w ϑ 1 D.whisker-left D.whisker-right D.comp-assoc by auto
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using w w' ϑ ϑ' β eq
          ϱ'.T2 [of F w F w' F ϑ ·D Φ (f, w) F u F ϑ' ·D Φ (f, w')
                   D.inv (Φ (g, w')) ·D F β ·D Φ (g, w)]
    by blast
qed

obtain γ' where γ': «γ' : F w ⇒D F w'» ∧
                   D.inv (Φ (g, w')) ·D F β ·D Φ (g, w) = F g ⋆D γ' ∧
                   F ϑ ·D Φ (f, w) = (F ϑ' ·D Φ (f, w')) ·D (F f ⋆D γ')
  using ∗ by auto
obtain γ where γ: «γ : w ⇒C w'» ∧ F γ = γ'
  using ϑ ϑ w w' γ' locally-full [of w w' γ']
  by (metis C.hseqI' C.ideD(1) C.src-hcomp C.vconn-implies-hpar(3)
      ϱ.leg0-simps(2) ϑ' hseq-fw)
have ϑ = ϑ' ·C (f ⋆C γ)
proof −
  have F ϑ = F (ϑ' ·C (f ⋆C γ))
  proof −
    have F ϑ = F ϑ' ·D Φ (f, w') ·D (F f ⋆D γ') ·D D.inv (Φ (f, w))
      using w' ϑ' γ' preserves-hcomp hseq-fw D.comp-assoc D.invert-side-of-triangle
      by (metis C.in-homE D.comp-arr-inv' cmp-components-are-iso cmp-simps(5)
          ϱ.ide-leg0 ϑ as-nat-trans.naturality1 w)
    also have ... = F ϑ' ·D F (f ⋆C γ)
      using w' D.comp-assoc hseq-fw preserves-hcomp cmp-components-are-iso
            D.comp-arr-inv'
      by (metis C.hseqI' C.in-homE C.trg-cod γ ϱ.leg0-in-hom(2))
    also have ... = F (ϑ' ·C (f ⋆C γ))
      using γ ϑ ϑ' hseq-fw C.hseqI' preserves-comp by force
    finally show ?thesis by simp
  qed
  moreover have C.par ϑ (ϑ' ·C (f ⋆C γ))
    using γ ϑ ϑ' hseq-fw by fastforce
  ultimately show ?thesis
    using is-faithful by blast
qed
moreover have β = g ⋆C γ
proof −
  have F β = F (g ⋆C γ)
  proof −
    have F β = Φ (g, w') ·D (F g ⋆D γ') ·D D.inv (Φ (g, w))
      by (metis (no-types) C.in-homE D.comp-arr-inv' D.comp-assoc
          cmp-components-are-iso cmp-simps(5) β γ' ϱ.ide-leg1 hseq-gw
          as-nat-trans.naturality1 as-nat-trans.naturality w w')
    also have ... = F (g ⋆C γ)
```

using w γ γ' *preserves-hcomp hseq-gw*
by (*metis C.hseqE C.hseqI′ C.in-homE C.seqE* ‹$\vartheta = \vartheta' \cdot_C (f \star_C \gamma)$›
ϱ.leg1-simps(*2*) *ϱ.leg1-simps*(*5*) *ϱ.leg1-simps*(*6*) ϑ *hseq-fw*)
finally show *?thesis* **by** *simp*
qed
moreover have $C.par\ \beta\ (g \star_C \gamma)$
proof (*intro conjI*)
show $C.arr\ \beta$
using β **by** *blast*
show *1*: $C.hseq\ g\ \gamma$
using γ *hseq-gw* **by** *fastforce*
show $C.dom\ \beta = C.dom\ (g \star_C \gamma)$
using γ β *1* **by** *fastforce*
show $C.cod\ \beta = C.cod\ (g \star_C \gamma)$
using γ β *1* **by** *fastforce*
qed
ultimately show *?thesis*
using *is-faithful* **by** *blast*
qed
ultimately have $\exists\gamma.\ «\gamma : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma)$
using γ **by** *blast*
moreover have $\bigwedge\gamma_1\ \gamma_2.\ «\gamma_1 : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma_1 \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma_1) \Longrightarrow$
$«\gamma_2 : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma_2 \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma_2) \Longrightarrow \gamma_1 = \gamma_2$
proof −
fix γ_1 γ_2
assume γ_1: $«\gamma_1 : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma_1 \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma_1)$
assume γ_2: $«\gamma_2 : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma_2 \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma_2)$
have $F\beta_1$: $F\ \beta = \Phi\ (g,\ w') \cdot_D (F\ g \star_D F\ \gamma_1) \cdot_D D.inv\ (\Phi\ (g,\ w))$
using w w' β *hseq-gw* γ_1 *preserves-hcomp* [*of* g γ_1] *cmp-components-are-iso*
by *auto*
have $F\beta_2$: $F\ \beta = \Phi\ (g,\ w') \cdot_D (F\ g \star_D F\ \gamma_2) \cdot_D D.inv\ (\Phi\ (g,\ w))$
using w w' β *hseq-gw* γ_2 *preserves-hcomp* [*of* g γ_2] *cmp-components-are-iso*
by *auto*
have $D.inv\ (\Phi\ (g,\ w')) \cdot_D F\ \beta \cdot_D \Phi\ (g,\ w) = F\ g \star_D F\ \gamma_1$
proof −
have $F\ \beta \cdot_D \Phi\ (g,\ w) = \Phi\ (g,\ w') \cdot_D (F\ g \star_D F\ \gamma_1)$
using w w' β *hseq-gw* γ_1 $F\beta_1$ *preserves-hcomp cmp-components-are-iso*
D.invert-side-of-triangle D.iso-inv-iso
by (*metis C.arrI D.comp-assoc D.inv-inv ϱ.ide-leg1 preserves-reflects-arr*)
thus *?thesis*
using w w' β *hseq-gw* γ_1 *preserves-hcomp cmp-components-are-iso*
D.invert-side-of-triangle
by (*metis C.arrI D.cod-comp D.seqE D.seqI* $F\beta_1$ *ϱ.ide-leg1 preserves-arr*)
qed
moreover have $D.inv\ (\Phi\ (g,\ w')) \cdot_D F\ \beta \cdot_D \Phi\ (g,\ w) = F\ g \star_D F\ \gamma_2$
proof −
have $F\ \beta \cdot_D \Phi\ (g,\ w) = \Phi\ (g,\ w') \cdot_D (F\ g \star_D F\ \gamma_2)$
using w w' β *hseq-gw* γ_2 $F\beta_2$ *preserves-hcomp cmp-components-are-iso*
D.invert-side-of-triangle D.iso-inv-iso

by (*metis C.arrI D.comp-assoc D.inv-inv* ϱ*.ide-leg1 preserves-reflects-arr*)
thus *?thesis*
using w w' β *hseq-gw* γ_2 *preserves-hcomp cmp-components-are-iso D.invert-side-of-triangle*
by (*metis C.arrI D.cod-comp D.seqE D.seqI* $F\beta_2$ ϱ*.ide-leg1 preserves-arr*)
qed
moreover have $F\ \vartheta \cdot_D \Phi\ (f,\ w) = (F\ \vartheta' \cdot_D \Phi\ (f,\ w')) \cdot_D (F\ f \star_D F\ \gamma_1)$
proof −
have $F\ \vartheta \cdot_D \Phi\ (f,\ w) = F\ (\vartheta' \cdot_C (f \star_C \gamma_1)) \cdot_D \Phi\ (f,\ w)$
using γ_1 **by** *blast*
also have ... $= (F\ \vartheta' \cdot_D F\ (f \star_C \gamma_1)) \cdot_D \Phi\ (f,\ w)$
using γ_1 ϑ **by** *auto*
also have
... $= (F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_1) \cdot_D D.inv\ (\Phi\ (f,\ w))) \cdot_D \Phi\ (f,\ w)$
using γ_1 *hseq-fw preserves-hcomp* **by** *auto*
also have
... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_1) \cdot_D D.inv\ (\Phi\ (f,\ w)) \cdot_D \Phi\ (f,\ w)$
using *D.comp-assoc* **by** *simp*
also have ... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_1) \cdot_D (F\ f \star_D F\ w)$
by (*simp add*: *D.comp-inv-arr′ hseq-fw* w)
also have ... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_1)$
using w γ_1 *D.whisker-left* [*of* F f F γ_1 F w] *D.comp-arr-dom* **by** *auto*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
moreover have $F\ \vartheta \cdot_D \Phi\ (f,\ w) = (F\ \vartheta' \cdot_D \Phi\ (f,\ w')) \cdot_D (F\ f \star_D F\ \gamma_2)$
proof −
have $F\ \vartheta \cdot_D \Phi\ (f,\ w) = F\ (\vartheta' \cdot_C (f \star_C \gamma_2)) \cdot_D \Phi\ (f,\ w)$
using γ_2 **by** *blast*
also have ... $= (F\ \vartheta' \cdot_D F\ (f \star_C \gamma_2)) \cdot_D \Phi\ (f,\ w)$
using γ_2 ϑ **by** *auto*
also have
... $= (F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_2) \cdot_D D.inv\ (\Phi\ (f,\ w))) \cdot_D \Phi\ (f,\ w)$
using γ_2 *hseq-fw preserves-hcomp* **by** *auto*
also have
... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_2) \cdot_D D.inv\ (\Phi\ (f,\ w)) \cdot_D \Phi\ (f,\ w)$
using *D.comp-assoc* **by** *simp*
also have ... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_2) \cdot_D (F\ f \star_D F\ w)$
by (*simp add*: *D.comp-inv-arr′ hseq-fw* w)
also have ... $= F\ \vartheta' \cdot_D \Phi\ (f,\ w') \cdot_D (F\ f \star_D F\ \gamma_2)$
using w γ_2 *D.whisker-left* [*of* F f F γ_2 F w] *D.comp-arr-dom* **by** *auto*
finally show *?thesis*
using *D.comp-assoc* **by** *simp*
qed
ultimately have $F\ \gamma_1 = F\ \gamma_2$
using γ_1 γ_2 $*$ **by** *blast*
thus $\gamma_1 = \gamma_2$
using γ_1 γ_2 *is-faithful* [*of* γ_1 γ_2] **by** *auto*
qed

ultimately show $\exists!\gamma.\ «\gamma : w \Rightarrow_C w'» \wedge \beta = g \star_C \gamma \wedge \vartheta = \vartheta' \cdot_C (f \star_C \gamma)$
by *blast*
qed
qed
qed
show *?thesis* **..**
qed

end

end

2.3 Bicategories of Spans

theory *BicategoryOfSpans*
imports *Category3.ConcreteCategory IsomorphismClass CanonicalIsos EquivalenceOfBicategories SpanBicategory Tabulation*
begin

In this section, we prove CKS Theorem 4, which characterizes up to equivalence the bicategories of spans in a category with pullbacks. The characterization consists of three conditions: BS1: "Every 1-cell is isomorphic to a composition $g \star f^*$, where f and g are maps"; BS2: "For every span of maps $(f,\ g)$ there is a 2-cell ϱ such that $(f,\ \varrho,\ g)$ is a tabulation"; and BS3: "Any two 2-cells between the same pair of maps are equal and invertible." One direction of the proof, which is the easier direction once it is established that BS1 and BS3 are respected by equivalence of bicategories, shows that if a bicategory B is biequivalent to the bicategory of spans in some category C with pullbacks, then it satisfies BS1 – BS3. The other direction, which is harder, shows that a bicategory B satisfying BS1 – BS3 is biequivalent to the bicategory of spans in a certain category with pullbacks that is constructed from the sub-bicategory of maps of B.

2.3.1 Definition

We define a *bicategory of spans* to be a bicategory that satisfies the conditions *BS1* – *BS3* stated informally above.

locale *bicategory-of-spans* =
bicategory + *chosen-right-adjoints* +
assumes *BS1*: *ide r* $\Longrightarrow \exists f\ g.$ *is-left-adjoint f* $\wedge$ *is-left-adjoint g* $\wedge$ *isomorphic r* $(g \star f^*)$
and *BS2*: $[\![$ *is-left-adjoint f*; *is-left-adjoint g*; *src f* = *src g* $]\!]$
$\Longrightarrow \exists r\ \varrho.$ *tabulation V H* a i *src trg r* ϱ *f g*
and *BS3*: $[\![$ *is-left-adjoint f*; *is-left-adjoint f'*; $«\mu : f \Rightarrow f'»$; $«\mu' : f \Rightarrow f'»$ $]\!]$
$\Longrightarrow$ *iso* $\mu \wedge$ *iso* $\mu' \wedge \mu = \mu'$

Using the already-established fact *equivalence-pseudofunctor.reflects-tabulation* that tabulations are reflected by equivalence pseudofunctors, it is not difficult to prove that the notion 'bicategory of spans' respects equivalence of bicategories.

lemma *bicategory-of-spans-respects-equivalence*:

assumes *equivalent-bicategories* V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D
and *bicategory-of-spans* V_C H_C a_C i_C src_C trg_C
shows *bicategory-of-spans* V_D H_D a_D i_D src_D trg_D
proof −
  interpret C: *bicategory-of-spans* V_C H_C a_C i_C src_C trg_C
    using *assms* **by** *simp*
  interpret C: *chosen-right-adjoints* V_C H_C a_C i_C src_C trg_C **..**
  interpret D: *bicategory* V_D H_D a_D i_D src_D trg_D
    using *assms equivalent-bicategories-def equivalence-pseudofunctor.axioms*(*1*)
      pseudofunctor.axioms(*2*)
    by *fast*
  interpret D: *chosen-right-adjoints* V_D H_D a_D i_D src_D trg_D **..**
  obtain F Φ **where** F: *equivalence-pseudofunctor*
      V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ
    using *assms equivalent-bicategories-def* **by** *blast*
  interpret F: *equivalence-pseudofunctor*
      V_C H_C a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D F Φ
    using F **by** *simp*
  interpret E: *equivalence-of-bicategories*
      V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
      F Φ $F.right\text{-}map$ $F.right\text{-}cmp$ $F.unit_0$ $F.unit_1$ $F.counit_0$ $F.counit_1$
    using $F.extends\text{-}to\text{-}equivalence\text{-}of\text{-}bicategories$ **by** *simp*
  interpret E': *converse-equivalence-of-bicategories*
      V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
      F Φ $F.right\text{-}map$ $F.right\text{-}cmp$ $F.unit_0$ $F.unit_1$ $F.counit_0$ $F.counit_1$
    ..
  interpret G: *equivalence-pseudofunctor*
      V_D H_D a_D i_D src_D trg_D V_C H_C a_C i_C src_C trg_C
      $F.right\text{-}map$ $F.right\text{-}cmp$
    using $E'.equivalence\text{-}pseudofunctor\text{-}left$ **by** *simp*

  write V_C (**infixr** ‹$\cdot_C$› *55*)
  write V_D (**infixr** ‹$\cdot_D$› *55*)
  write H_C (**infixr** ‹$\star_C$› *53*)
  write H_D (**infixr** ‹$\star_D$› *53*)
  write a_C (‹a_C[-, -, -]›)
  write a_D (‹a_D[-, -, -]›)
  write $C.in\text{-}hhom$ (‹«- : - $\rightarrow_C$ -»›)
  write $C.in\text{-}hom$ (‹«- : - $\Rightarrow_C$ -»›)
  write $D.in\text{-}hhom$ (‹«- : - $\rightarrow_D$ -»›)
  write $D.in\text{-}hom$ (‹«- : - $\Rightarrow_D$ -»›)
  write $C.isomorphic$ (**infix** ‹$\cong_C$› *50*)
  write $D.isomorphic$ (**infix** ‹$\cong_D$› *50*)
  write $C.some\text{-}right\text{-}adjoint$ (‹-^{*C}› [*1000*] *1000*)
  write $D.some\text{-}right\text{-}adjoint$ (‹-^{*D}› [*1000*] *1000*)

  show *bicategory-of-spans* V_D H_D a_D i_D src_D trg_D
  proof
    show $\bigwedge r'$. $D.ide$ $r' \Longrightarrow$

$\exists f'\ g'.\ D.\textit{is-left-adjoint}\ f' \wedge D.\textit{is-left-adjoint}\ g' \wedge r' \cong_D g' \star_D (f')^{*D}$

proof −
  fix r'
  assume r': $D.\textit{ide}\ r'$
  obtain $f\ g$
    where fg: $C.\textit{is-left-adjoint}\ f \wedge C.\textit{is-left-adjoint}\ g \wedge F.\textit{right-map}\ r' \cong_C g \star_C f^{*C}$
    using r' $C.BS1$ [*of* $F.\textit{right-map}\ r'$] **by** *auto*
  have $\textit{trg-g}$: $\textit{trg}_C\ g = E.G.\textit{map}_0\ (\textit{trg}_D\ r')$
    using fg r' $C.\textit{isomorphic-implies-ide}$ $C.\textit{isomorphic-implies-hpar}$
    by (*metis* $C.\textit{ideD}(1)$ $C.\textit{trg-hcomp}$ $D.\textit{ideD}(1)$ $E.G.\textit{preserves-trg}$)
  have $\textit{trg-f}$: $\textit{trg}_C\ f = E.G.\textit{map}_0\ (\textit{src}_D\ r')$
    using fg r' $C.\textit{isomorphic-implies-ide}$ $C.\textit{isomorphic-implies-hpar}$
  by (*metis* $C.\textit{ideD}(1)$ $C.\textit{right-adjoint-simps}(2)$ $C.\textit{src-hcomp}$ $D.\textit{ideD}(1)$ $E.G.\textit{preserves-src}$)

  define $\textit{d-src}$ **where** $\textit{d-src} \equiv F.\textit{counit}_0\ (\textit{src}_D\ r')$
  define $\textit{e-src}$ **where** $\textit{e-src} \equiv (F.\textit{counit}_0\ (\textit{src}_D\ r'))^{\sim D}$
  have $\textit{d-src}$: «$\textit{d-src} : F.\textit{map}_0\ (E.G.\textit{map}_0\ (\textit{src}_D\ r')) \rightarrow_D \textit{src}_D\ r'$» $\wedge$
      $D.\textit{equivalence-map}\ \textit{d-src}$
    using $\textit{d-src-def}$ r' $E.\varepsilon.\textit{map}_0\textit{-in-hhom}$ $E.\varepsilon.\textit{components-are-equivalences}$ **by** *simp*
  have $\textit{e-src}$: «$\textit{e-src} : \textit{src}_D\ r' \rightarrow_D F.\textit{map}_0\ (E.G.\textit{map}_0\ (\textit{src}_D\ r'))$» $\wedge$
      $D.\textit{equivalence-map}\ \textit{e-src}$
    using $\textit{e-src-def}$ r' $E.\varepsilon.\textit{map}_0\textit{-in-hhom}$ $E.\varepsilon.\textit{components-are-equivalences}$ **by** *simp*
  obtain $\eta\textit{-src}$ $\varepsilon\textit{-src}$
  where $\textit{eq-src}$: $\textit{equivalence-in-bicategory}\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ \textit{src}_D\ \textit{trg}_D\ \textit{d-src}\ \textit{e-src}\ \eta\textit{-src}\ \varepsilon\textit{-src}$
    using $\textit{d-src-def}$ $\textit{e-src-def}$ $\textit{d-src}$ $\textit{e-src}$ $D.\textit{quasi-inverses-some-quasi-inverse}$
      $D.\textit{quasi-inverses-def}$
    by *blast*
  interpret $\textit{eq-src}$: $\textit{equivalence-in-bicategory}$
      $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ \textit{src}_D\ \textit{trg}_D\ \textit{d-src}\ \textit{e-src}\ \eta\textit{-src}\ \varepsilon\textit{-src}$
    using $\textit{eq-src}$ **by** *simp*

  define $\textit{d-trg}$ **where** $\textit{d-trg} \equiv F.\textit{counit}_0\ (\textit{trg}_D\ r')$
  define $\textit{e-trg}$ **where** $\textit{e-trg} \equiv (F.\textit{counit}_0\ (\textit{trg}_D\ r'))^{\sim D}$
  have $\textit{d-trg}$: «$\textit{d-trg} : F.\textit{map}_0\ (E.G.\textit{map}_0\ (\textit{trg}_D\ r')) \rightarrow_D \textit{trg}_D\ r'$» $\wedge$
      $D.\textit{equivalence-map}\ \textit{d-trg}$
    using $\textit{d-trg-def}$ r' $E.\varepsilon.\textit{map}_0\textit{-in-hhom}$ $E.\varepsilon.\textit{components-are-equivalences}$ **by** *simp*
  have $\textit{e-trg}$: «$\textit{e-trg} : \textit{trg}_D\ r' \rightarrow_D F.\textit{map}_0\ (E.G.\textit{map}_0\ (\textit{trg}_D\ r'))$» $\wedge$
      $D.\textit{equivalence-map}\ \textit{e-trg}$
    using $\textit{e-trg-def}$ r' $E.\varepsilon.\textit{map}_0\textit{-in-hhom}$ $E.\varepsilon.\textit{components-are-equivalences}$ **by** *simp*
  obtain $\eta\textit{-trg}$ $\varepsilon\textit{-trg}$
  where $\textit{eq-trg}$: $\textit{equivalence-in-bicategory}\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ \textit{src}_D\ \textit{trg}_D\ \textit{d-trg}\ \textit{e-trg}\ \eta\textit{-trg}\ \varepsilon\textit{-trg}$
    using $\textit{d-trg-def}$ $\textit{e-trg-def}$ $\textit{d-trg}$ $\textit{e-trg}$ $D.\textit{quasi-inverses-some-quasi-inverse}$
      $D.\textit{quasi-inverses-def}$
    by *blast*
  interpret $\textit{eq-trg}$: $\textit{equivalence-in-bicategory}\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ \textit{src}_D\ \textit{trg}_D\ \textit{d-trg}\ \textit{e-trg}\ \eta\textit{-trg}\ \varepsilon\textit{-trg}$
    using $\textit{eq-trg}$ **by** *simp*

  interpret $\textit{eqs}$: $\textit{two-equivalences-in-bicategory}\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ \textit{src}_D\ \textit{trg}_D$
      $\textit{d-src}\ \textit{e-src}\ \eta\textit{-src}\ \varepsilon\textit{-src}\ \textit{d-trg}\ \textit{e-trg}\ \eta\textit{-trg}\ \varepsilon\textit{-trg}$

..

interpret *hom*: *subcategory* V_D ‹$\lambda\mu$. «μ : trg_D *d-src* $\rightarrow_D$ trg_D *d-trg*»›
  using *D.hhom-is-subcategory* **by** *simp*
interpret *hom'*: *subcategory* V_D ‹$\lambda\mu$. «μ : src_D *d-src* $\rightarrow_D$ src_D *d-trg*»›
  using *D.hhom-is-subcategory* **by** *simp*
interpret *e*: *equivalence-of-categories hom.comp hom'.comp eqs.F eqs.G eqs.φ eqs.ψ*
  using *eqs.induces-equivalence-of-hom-categories* **by** *simp*

have *r'-in-hhom*: *D.in-hhom r'* (src_D *e-src*) (src_D *e-trg*)
  using *r' e-src e-trg* **by** (*simp add*: *D.in-hhom-def*)

define *g'*
where *g'* = *d-trg* $\star_D$ *F g*
have *g'*: *D.is-left-adjoint g'*
  unfolding *g'-def*
  using *fg r' d-trg trg-g C.left-adjoint-is-ide D.equivalence-is-adjoint*
    D.left-adjoints-compose F.preserves-left-adjoint C.ideD(*1*) *D.in-hhom-def*
    F.preserves-trg
  by *metis*
have *1*: *D.is-right-adjoint* (*F* f^{*C} $\star_D$ *e-src*)
proof −
  have *D.is-right-adjoint e-src*
    using *r' e-src D.equivalence-is-adjoint* **by** *simp*
  moreover have *D.is-right-adjoint* (*F* f^{*C})
    using *fg C.left-adjoint-extends-to-adjoint-pair F.preserves-adjoint-pair* **by** *blast*
  moreover have src_D (*F* f^{*C}) = trg_D *e-src*
    using *fg r' trg-f C.right-adjoint-is-ide e-src* **by** *auto*
  ultimately show *?thesis*
    using *fg r' D.right-adjoints-compose F.preserves-right-adjoint* **by** *blast*
qed
obtain *f'* **where** *f'*: *D.adjoint-pair f'* (*F* f^{*C} $\star_D$ *e-src*)
  using *1* **by** *auto*
have *f'*: *D.is-left-adjoint f'* $\wedge$ *F* f^{*C} $\star_D$ *e-src* $\cong_D$ $(f')^{*D}$
  using *f' D.left-adjoint-determines-right-up-to-iso D.left-adjoint-extends-to-adjoint-pair*
  by *blast*

have *r'* $\cong_D$ *d-trg* $\star_D$ (*e-trg* $\star_D$ *r'* $\star_D$ *d-src*) $\star_D$ *e-src*
  using *r' r'-in-hhom D.isomorphic-def eqs.ψ-in-hom eqs.ψ-components-are-iso*
    D.isomorphic-symmetric D.ide-char eq-src.antipar(*2*) *eq-trg.antipar*(*2*)
  by *metis*
also have *1*: ... $\cong_D$ *d-trg* $\star_D$ *F* (*F.right-map r'*) $\star_D$ *e-src*
proof −
  have *e-trg* $\star_D$ *r'* $\star_D$ *d-src* $\cong_D$ *F* (*F.right-map r'*)
  proof −
    have *D.in-hom* ($F.counit_1$ *r'*)
      (*r'* $\star_D$ *d-src*) ($F.counit_0$ (trg_D *r'*) $\star_D$ *F* (*F.right-map r'*))
      unfolding *d-src-def*
      using *r' E.ε.map_1-in-hom*(*2*) [*of r'*] **by** *simp*

hence $r' \star_D$ *d-src* $\cong_D$ *F.counit*$_0$ (*trg*$_D$ r') $\star_D$ *F* (*F.right-map* r')
using r' *D.isomorphic-def E.ε.iso-map*$_1$*-ide* **by** *auto*
thus *?thesis*
using r' *e-trg-def E.ε.components-are-equivalences D.isomorphic-symmetric D.quasi-inverse-transpose*(*2*)
by (*metis D.isomorphic-implies-hpar*(*1*) *F.preserves-isomorphic d-trg d-trg-def eq-trg.ide-left fg*)
qed
thus *?thesis*
using *D.hcomp-ide-isomorphic D.hcomp-isomorphic-ide D.in-hhom-def D.isomorphic-implies-hpar*(*4*) *d-trg e-src eq-src.antipar*(*1*−*2*) *eq-trg.antipar*(*2*) r'
by *force*
qed
also have *2*: ... $\cong_D$ *d-trg* $\star_D$ (*F g* $\star_D$ *F* f^{*C}) $\star_D$ *e-src*
proof −
have *F* (*F.right-map* r') $\cong_D$ *F g* $\star_D$ *F* f^{*C}
by (*meson C.hseq-char C.ideD*(*1*) *C.isomorphic-implies-ide*(*2*) *C.left-adjoint-is-ide C.right-adjoint-simps*(*1*) *D.isomorphic-symmetric D.isomorphic-transitive F.preserves-isomorphic F.weakly-preserves-hcomp fg*)
thus *?thesis*
using *D.hcomp-ide-isomorphic D.hcomp-isomorphic-ide*
by (*metis 1 D.hseqE D.ideD*(*1*) *D.isomorphic-implies-hpar*(*2*) *eq-src.ide-right eq-trg.ide-left*)
qed
also have *3*: ... $\cong_D$ (*d-trg* $\star_D$ *F g*) $\star_D$ *F* f^{*C} $\star_D$ *e-src*
proof −
have ... $\cong_D$ *d-trg* $\star_D$ *F g* $\star_D$ *F* f^{*C} $\star_D$ *e-src*
by (*metis C.left-adjoint-is-ide C.right-adjoint-simps*(*1*) *D.hcomp-assoc-isomorphic D.hcomp-ide-isomorphic D.hcomp-simps*(*1*) *D.hseq-char D.ideD*(*1*) *D.isomorphic-implies-hpar*(*2*) *F.preserves-ide calculation eq-src.ide-right eq-trg.ide-left fg*)
also have ... $\cong_D$ (*d-trg* $\star_D$ *F g*) $\star_D$ *F* f^{*C} $\star_D$ *e-src*
by (*metis C.left-adjoint-is-ide D.hcomp-assoc-isomorphic D.hcomp-simps*(*2*) *D.hseq-char D.ideD*(*1*) *D.isomorphic-implies-ide*(*1*) *D.isomorphic-symmetric F.preserves-ide calculation eq-trg.ide-left* f' *fg*)
finally show *?thesis* **by** *blast*
qed
also have ... $\cong_D$ $g' \star_D$ f'^{*D}
using *g'-def* f'
by (*metis 3 D.adjoint-pair-antipar*(*1*) *D.hcomp-ide-isomorphic D.hseq-char D.ideD*(*1*) *D.isomorphic-implies-ide*(*2*) g')
finally have *D.isomorphic* r' ($g' \star_D$ f'^{*D})
by *simp*
thus $\exists f'\ g'$. *D.is-left-adjoint* f' $\wedge$ *D.is-left-adjoint* g' $\wedge$ $r' \cong_D g' \star_D f'^{*D}$
using f' g' **by** *auto*
qed
show $\bigwedge f\ f'\ \mu\ \mu'$. ⟦ *D.is-left-adjoint* f; *D.is-left-adjoint* f';
«$\mu : f \Rightarrow_D f'$»; «$\mu' : f \Rightarrow_D f'$» ⟧ $\Longrightarrow$ *D.iso* μ $\wedge$ *D.iso* μ' $\wedge$ $\mu = \mu'$

proof −
  fix $f\ f'\ \mu\ \mu'$
  assume f: *D.is-left-adjoint f*
  and f': *D.is-left-adjoint f′*
  and μ: «$\mu : f \Rightarrow_D f'$»
  and μ': «$\mu' : f \Rightarrow_D f'$»
  have *C.is-left-adjoint* (*F.right-map f*) ∧ *C.is-left-adjoint* (*F.right-map f′*)
    using *f f′ E.G.preserves-left-adjoint* **by** *blast*
  moreover have «*F.right-map μ* : *F.right-map f* $\Rightarrow_C$ *F.right-map f′*» ∧
              «*F.right-map μ′* : *F.right-map f* $\Rightarrow_C$ *F.right-map f′*»
    using *μ μ′ E.G.preserves-hom* **by** *simp*
  ultimately have *C.iso* (*F.right-map μ*) ∧ *C.iso* (*F.right-map μ′*) ∧
              F.right-map μ = *F.right-map μ′*
    using *C.BS3* **by** *blast*
  thus *D.iso μ* ∧ *D.iso μ′* ∧ *μ* = *μ′*
    using *μ μ′ G.reflects-iso G.is-faithful* **by** *blast*
qed
show ⋀*f g*. ⟦ *D.is-left-adjoint f*; *D.is-left-adjoint g*; src_D *f* = src_D *g* ⟧
          ⟹ ∃ *r ϱ*. *tabulation* V_D H_D a_D i_D src_D trg_D *r ϱ f g*
proof −
  fix *f g*
  assume *f*: *D.is-left-adjoint f*
  assume *g*: *D.is-left-adjoint g*
  assume *fg*: src_D *f* = src_D *g*
  have *C.is-left-adjoint* (*F.right-map f*)
    using *f E.G.preserves-left-adjoint* **by** *blast*
  moreover have *C.is-left-adjoint* (*F.right-map g*)
    using *g E.G.preserves-left-adjoint* **by** *blast*
  moreover have src_C (*F.right-map f*) = src_C (*F.right-map g*)
    using *f g D.left-adjoint-is-ide fg* **by** *simp*
  ultimately have
      1: ∃ *r ϱ*. *tabulation* V_C H_C a_C i_C src_C trg_C *r ϱ* (*F.right-map f*) (*F.right-map g*)
    using *C.BS2* **by** *simp*
  obtain *r ϱ* **where**
      ϱ: *tabulation* V_C H_C a_C i_C src_C trg_C *r ϱ* (*F.right-map f*) (*F.right-map g*)
    using *1* **by** *auto*
  interpret *ϱ*: *tabulation* V_C H_C a_C i_C src_C trg_C *r ϱ* ‹*F.right-map f*› ‹*F.right-map g*›
    using *ϱ* **by** *simp*
  obtain *r′* **where**
    r′: *D.ide r′* ∧ *D.in-hhom r′* (trg_D *f*) (trg_D *g*) ∧ *C.isomorphic* (*F.right-map r′*) *r*
    using *f g ϱ.ide-base ϱ.tab-in-hom G.locally-essentially-surjective*
    by (*metis D.obj-trg E.G.preserves-reflects-arr E.G.preserves-trg ϱ.leg0-simps*(*2*−*3*)
      ϱ.leg1-simps(*2*,*4*) *ϱ.base-in-hom*(*1*))
  obtain *φ* **where** *φ*: «$\varphi : r \Rightarrow_C$ *F.right-map r′*» ∧ *C.iso φ*
    using *r′ C.isomorphic-symmetric* **by** *blast*
  have *σ*: *tabulation* V_C H_C a_C i_C src_C trg_C
          (*F.right-map r′*) ((*φ* $\star_C$ *F.right-map f*) $\cdot_C$ *ϱ*) (*F.right-map f*) (*F.right-map g*)
    using *φ ϱ.is-preserved-by-base-iso* **by** *simp*
  have *1*: ∃ *ϱ′*. «$\varrho' : g \Rightarrow_D H_D\ r'\ f$» ∧

$F.right\text{-}map\ \varrho' = F.right\text{-}cmp\ (r', f) \cdot_C (\varphi \star_C F.right\text{-}map\ f) \cdot_C \varrho$
proof −
have $D.ide\ g$
by (*simp add*: *D.left-adjoint-is-ide g*)
moreover have $D.ide\ (H_D\ r'\ f)$
using *f r′ D.left-adjoint-is-ide* **by** *auto*
moreover have $src_D\ g = src_D\ (H_D\ r'\ f)$
using *fg* **by** (*simp add*: *calculation*(*2*))
moreover have $trg_D\ g = trg_D\ (H_D\ r'\ f)$
using *calculation*(*2*) *r′* **by** *auto*
moreover have «$F.right\text{-}cmp\ (r', f) \cdot_C (\varphi \star_C F.right\text{-}map\ f) \cdot_C \varrho$:
$F.right\text{-}map\ g \Rightarrow_C F.right\text{-}map\ (r' \star_D f)$»
using *f g r′ φ D.left-adjoint-is-ide ϱ.ide-base*
by (*intro C.comp-in-homI*, *auto*)
ultimately show *?thesis*
using *G.locally-full* **by** *simp*
qed
obtain ϱ' **where** ϱ': «$\varrho' : g \Rightarrow_D H_D\ r'\ f$» ∧
$F.right\text{-}map\ \varrho' = F.right\text{-}cmp\ (r', f) \cdot_C (\varphi \star_C F.right\text{-}map\ f) \cdot_C \varrho$
using *1* **by** *auto*
have $tabulation\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ r'\ \varrho'\ f\ g$
proof −
have $C.inv\ (F.right\text{-}cmp\ (r', f)) \cdot_C F.right\text{-}map\ \varrho' = (\varphi \star_C F.right\text{-}map\ f) \cdot_C \varrho$
using *r′ f ϱ′ C.comp-assoc C.comp-cod-arr C.invert-side-of-triangle*(*1*)
by (*metis D.adjoint-pair-antipar*(*1*) *D.arrI D.in-hhomE E.G.cmp-components-are-iso E.G.preserves-arr*)
thus *?thesis*
using *σ ϱ′ G.reflects-tabulation*
by (*simp add*: *D.left-adjoint-is-ide f r′*)
qed
thus ∃ *r′ ϱ′*. $tabulation\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ r'\ \varrho'\ f\ g$
by *auto*
qed
qed
qed

2.3.2 Span(C) is a Bicategory of Spans

We first consider an arbitrary 1-cell r in $Span(C)$, and show that it has a tabulation as a span of maps. This is CKS Proposition 3 (stated more strongly to assert that the "output leg" can also be taken to be a map, which the proof shows already).

locale *identity-arrow-in-span-bicategory* =
span-bicategory C prj0 prj1 +
r: *identity-arrow-of-spans C r*
for *C* :: *′a comp* (**infixr** ‹·› *55*)
and *prj0* :: *′a* ⇒ *′a* ⇒ *′a* (‹p_0[-, -]›)
and *prj1* :: *′a* ⇒ *′a* ⇒ *′a* (‹p_1[-, -]›)
and *r* :: *′a arrow-of-spans-data*
begin

CKS say: "Suppose $r = (r_0,\ R,\ r_1)\colon A \to B$ and put $f = (1,\ R,\ r_0)$, $g = (1,\ R,\ r_1)$. Let k_0, k_1 form a kernel pair for r_0 and define ϱ by $k_0\varrho = k_1\varrho = 1_R$."

abbreviation *ra* **where** *ra* ≡ *r.dom.apex*
abbreviation *r0* **where** *r0* ≡ *r.dom.leg0*
abbreviation *r1* **where** *r1* ≡ *r.dom.leg1*
abbreviation *f* **where** *f* ≡ *mkIde ra r0*
abbreviation *g* **where** *g* ≡ *mkIde ra r1*
abbreviation *k0* **where** *k0* ≡ $\mathrm{p}_0[r0,\ r0]$
abbreviation *k1* **where** *k1* ≡ $\mathrm{p}_1[r0,\ r0]$

Here *ra* is the apex R of the span $(r_0,\ R,\ r_1)$, and the spans f and g also have that same 0-cell as their apex. The tabulation 2-cell ϱ has to be an arrow of spans from $g = (1,\ R,\ r_1)$ to $r \star f$, which is the span $(k_0,\ r_1 \cdot k_1)$.

abbreviation ϱ **where** $\varrho \equiv (\!|Chn = \langle ra\ [\![r0,\ r0]\!]\ ra\rangle,$
$Dom = (\!|Leg0 = ra,\ Leg1 = r1|\!),$
$Cod = (\!|Leg0 = k0,\ Leg1 = r1 \cdot k1|\!)|\!)$

lemma *has-tabulation*:
shows *tabulation vcomp hcomp assoc unit src trg r* ϱ *f g*
and *is-left-adjoint f* **and** *is-left-adjoint g*
proof −
  have *ide-f*: *ide f*
  using *ide-mkIde r.dom.leg-in-hom*(*1*) *C.arr-dom C.dom-dom r.dom.apex-def r.dom.leg-simps*(*1*)
    by *presburger*
  interpret *f*: *identity-arrow-of-spans C f*
    using *ide-f ide-char′* **by** *auto*
  have *ide-g*: *ide g*
    using *ide-mkIde r.dom.leg-in-hom*
    by (*metis C.arr-dom C.dom-dom r.dom.leg-simps*(*3*) *r.dom.leg-simps*(*4*))
  interpret *g*: *identity-arrow-of-spans C g*
    using *ide-g ide-char′* **by** *auto*

  show *is-left-adjoint f*
    using *is-left-adjoint-char* [*of f*] *ide-f* **by** *simp*
  show *is-left-adjoint g*
    using *is-left-adjoint-char* [*of g*] *ide-g* **by** *simp*

  have *ide-r*: *ide r*
    using *ide-char′ r.identity-arrow-of-spans-axioms* **by** *auto*
  have *src-r*: *src r* = *mkObj* (*C.cod r0*)
    by (*simp add*: *ide-r src-def*)
  have *trg-r*: *trg r* = *mkObj* (*C.cod r1*)
    by (*simp add*: *ide-r trg-def*)
  have *src-f*: *src f* = *mkObj ra*
    using *ide-f src-def* **by** *auto*
  have *trg-f*: *trg f* = *mkObj* (*C.cod r0*)
    using *ide-f trg-def* **by** *auto*
  have *src-g*: *src g* = *mkObj ra*
    using *ide-g src-def* **by** *auto*

```
have trg-g: trg g = mkObj (C.cod r1)
  using ide-g trg-def by auto

have hseq-rf: hseq r f
  using ide-r ide-f src-r trg-f by simp
interpret rf: two-composable-arrows-of-spans C prj0 prj1 r f
  using hseq-rf hseq-char by unfold-locales auto
interpret rf: two-composable-identity-arrows-of-spans C prj0 prj1 r f ..
interpret rf: identity-arrow-of-spans C ‹r ⋆ f›
  using rf.ide-composite ide-char' by auto
let ?rf = r ⋆ f

have rf: ?rf = ⦇Chn = r0 ↓↓ r0,
               Dom = ⦇Leg0 = k0, Leg1 = r1 · k1⦈,
               Cod = ⦇Leg0 = k0, Leg1 = r1 · k1⦈⦈
  unfolding hcomp-def chine-hcomp-def
  using hseq-rf C.comp-cod-arr by auto

interpret Cod-rf: span-in-category C ‹⦇Leg0 = k0, Leg1 = r1 · k1⦈›
  using ide-r ide-f rf C.comp-cod-arr
  by unfold-locales auto

have Dom-g: Dom g = ⦇Leg0 = ra, Leg1 = r1⦈ by simp
interpret Dom-g: span-in-category C ‹⦇Leg0 = ra, Leg1 = r1⦈›
  using Dom-g g.dom.span-in-category-axioms by simp
interpret Dom-ϱ: span-in-category C ‹Dom ϱ›
  using Dom-g g.dom.span-in-category-axioms by simp
interpret Cod-ϱ: span-in-category C ‹Cod ϱ›
  using rf Cod-rf.span-in-category-axioms by simp
interpret ϱ: arrow-of-spans C ϱ
  using Dom-ϱ.apex-def Cod-ϱ.apex-def C.comp-assoc C.comp-arr-dom
  by unfold-locales auto
have ϱ: «ϱ : g ⇒ r ⋆ f»
  using rf ide-g arr-char dom-char cod-char ϱ.arrow-of-spans-axioms ideD(2)
        Cod-rf.apex-def g.dom.leg-simps(4)
  by auto

show tabulation vcomp hcomp assoc unit src trg r ϱ f g
proof −
  interpret T: tabulation-data vcomp hcomp assoc unit src trg r ϱ f g
    using ide-f ϱ by unfold-locales auto
  show ?thesis
  proof
    show T1: ⋀u ω.
               ⟦ ide u; «ω : dom ω ⇒ r ⋆ u» ⟧ ⟹
               ∃ w ϑ ν. ide w ∧ «ϑ : f ⋆ w ⇒ u» ∧ «ν : dom ω ⇒ g ⋆ w» ∧ iso ν ∧
                      T.composite-cell w ϑ · ν = ω
    proof −
      fix u ω
```

assume u: *ide u*
assume ω: «ω : *dom* $\omega \Rightarrow r \star u$»
show $\exists\, w\ \vartheta\ \nu.$ *ide w* $\wedge$ «$\vartheta : f \star w \Rightarrow u$» $\wedge$ «ν : *dom* $\omega \Rightarrow g \star w$» $\wedge$ *iso* ν $\wedge$
T.composite-cell w $\vartheta \cdot \nu = \omega$
proof −
interpret u: *identity-arrow-of-spans C u*
using *u ide-char′* **by** *auto*
have v: *ide* (*dom* ω)
using ω **by** *auto*
interpret v: *identity-arrow-of-spans C* ‹*dom* ω›
using *v ide-char′* **by** *auto*
interpret ω: *arrow-of-spans C* ω
using ω *arr-char* **by** *auto*
have *hseq-ru*: *hseq r u*
using *u* ω *ide-cod* **by** *fastforce*
interpret *ru*: *two-composable-arrows-of-spans C prj0 prj1 r u*
using *hseq-ru hseq-char* **by** *unfold-locales auto*
interpret *ru*: *two-composable-identity-arrows-of-spans C prj0 prj1 r u* **..**

CKS say: "We must show that $(f,\ \varrho,\ g)$ is a wide tabulation of r. Take $u = (u_0,\ U,\ u_1)$: $X \to A$, $v = (v_0,\ V,\ v_1)$: $X \to B$, ω: $v \Rightarrow ru$ as in *T1*. Let P be the pullback of u_1, r_0. Let $w = (v_0,\ V,\ p_1\omega)$: $X \to R$, $\vartheta = p_0\omega$: $fw \Rightarrow u$, $\nu = 1$: $v \Rightarrow gw$; so $\omega = (r\vartheta)(\varrho w)\nu$ as required."

let *?R* = *r.apex*
let *?A* = *C.cod r0*
let *?B* = *C.cod r1*
let *?U* = *u.apex*
let *?u0* = *u.leg0*
let *?u1* = *u.leg1*
let *?X* = *C.cod ?u0*
let *?V* = *v.apex*
let *?v0* = *v.leg0*
let *?v1* = *v.leg1*
let *?*ω = ω*.chine*
let *?P* = *r0* $\downarrow\downarrow$ *?u1*
let *?p0* = p_0[*r0*, *?u1*]
let *?p1* = p_1[*r0*, *?u1*]
let *?w1* = *?p1* · *?*ω
define *w* **where** *w* = *mkIde ?v0 ?w1*
let *?Q* = *?R* $\downarrow\downarrow$ *?w1*
let *?q1* = p_1[*?R*, *?w1*]
let *?*ϱ = ⟨*?R* ⟦*r0*, *r0*⟧ *?R*⟩

have *P*: *?P* = *ru.apex*
using *ru.apex-composite* **by** *auto*

have *Chn-*ω: «*?*ω : *?V* $\to_C$ *?P*»
using *P Chn-in-hom* ω **by** *force*
have *p0*ω: «*?p0* · *?*ω : *?V* $\to_C$ *?U*»

using *Chn-ω ru.legs-form-cospan* **by** *auto*
have *w1*: «*?w1* : *?V* $\rightarrow_C$ *?R*»
using *Chn-ω ru.legs-form-cospan r.dom.apex-def* **by** *blast*
have *r1w1*: *r1* · *?w1* = *?v1*
by (*metis C.comp-assoc T.base-simps*(*3*) *ω ω.leg1-commutes*
arrow-of-spans-data.select-convs(*3*) *cod-char dom-char ideD*(*1*) *ideD*(*2*)
in-homE ru.composite-in-hom ru.leg1-composite u v)

have *w*: *ide w*
unfolding *w-def*
using *P ω w1* **by** (*intro ide-mkIde*, *auto*)
interpret *w*: *identity-arrow-of-spans C w*
using *w-def w ide-char′* **by** *auto*

have *hseq-fw*: *hseq f w*
using *w-def ide-f w src-def trg-def w1* **by** *auto*
interpret *fw*: *two-composable-arrows-of-spans C prj0 prj1 f w*
using *w-def hseq-fw hseq-char* **by** *unfold-locales auto*
interpret *fw*: *two-composable-identity-arrows-of-spans C prj0 prj1 f w* **..**

have *hseq-gw*: *hseq g w*
using *w-def ide-g w src-def trg-def w1* **by** *auto*
interpret *gw*: *two-composable-arrows-of-spans C prj0 prj1 g w*
using *hseq-gw hseq-char* **by** *unfold-locales auto*
interpret *gw*: *two-composable-identity-arrows-of-spans C prj0 prj1 g w* **..**

interpret *rfw*: *three-composable-arrows-of-spans C prj0 prj1 r f w* **..**
interpret *rfw*: *three-composable-identity-arrows-of-spans C prj0 prj1 r f w* **..**
have *arfw*: «a[*r*, *f*, *w*] : (*r* ⋆ *f*) ⋆ *w* ⇒ *r* ⋆ *f* ⋆ *w*»
using *fw.composable ide-f ide-r w rf.composable* **by** *auto*

have *fw-apex-eq*: *fw.apex* = *?R* ⋎⋏ *?w1*
using *w-def fw.dom.apex-def hcomp-def hseq-fw hseq-char ω C.arr-dom-iff-arr*
C.pbdom-def fw.chine-eq-apex fw.chine-simps(*1*)
by *auto*
have *gw-apex-eq*: *gw.apex* = *?R* ⋎⋏ *?w1*
using *w-def ω w1 gw.dom.apex-def hcomp-def hseq-gw hseq-char* **by** *auto*

Well, CKS say take $\vartheta = p_0\omega$, but taking this literally and trying to define ϑ so that *Chn* ϑ = $?p_0$ · *?ω*, does not yield the required *dom* ϑ = *?R* ⋎⋏ *?w1*. We need «*Chn* ϑ : *?R* ⋎⋏ *?w1* $\rightarrow_C$ *?U*», so we have to compose with a projection, which leads to: *Chn* ϑ = *?p0* · *?ω* · p_0[*?R*, *?w1*]. What CKS say is only correct if the projection p_0[*?R*, *?w1*] is an identity, which can't be guaranteed for an arbitrary choice of pullbacks.

define ϑ
where
ϑ = (|*Chn* = *?p0* · *?ω* · p_0[*?R*, *?w1*], *Dom* = *Dom* (*f* ⋆ *w*), *Cod* = *Cod u*|)

interpret *Dom-ϑ*: *span-in-category C* ‹*Dom ϑ*›
using *ϑ-def fw.dom.span-in-category-axioms* **by** *simp*

```
interpret Cod-ϑ: span-in-category C ‹Cod ϑ›
  using ϑ-def u.cod.span-in-category-axioms by simp

have Dom-ϑ-leg0-eq: Dom-ϑ.leg0 = ?v0 · p₀[?R, ?w1]
  using w-def ϑ-def hcomp-def hseq-fw hseq-char by simp
have Dom-ϑ-leg1-eq: Dom-ϑ.leg1 = r0 · ?q1
  using w-def ϑ-def hcomp-def hseq-fw hseq-char by simp
have Cod-ϑ-leg0-eq: Cod-ϑ.leg0 = ?u0
  using w-def ϑ-def hcomp-def hseq-fw hseq-char by simp
have Cod-ϑ-leg1-eq: Cod-ϑ.leg1 = ?u1
  using w-def ϑ-def hcomp-def hseq-fw hseq-char by simp
have Chn-ϑ-eq: Chn ϑ = ?p0 · ?ω · p₀[?R, ?w1]
  using ϑ-def by simp

interpret ϑ: arrow-of-spans C ϑ
proof
  show 1: «Chn ϑ : Dom-ϑ.apex →_C Cod-ϑ.apex»
    using ϑ-def Chn-ω ru.legs-form-cospan fw-apex-eq
    by (intro C.in-homI, auto)
  show Cod-ϑ.leg0 · Chn ϑ = Dom-ϑ.leg0
  proof −
    have Cod-ϑ.leg0 · Chn ϑ = ?u0 · ?p0 · ?ω · p₀[?R, ?w1]
      using Cod-ϑ-leg0-eq Chn-ϑ-eq by simp
    also have ... = ?v0 · p₀[?R, ?w1]
    proof −
      have ?u0 · ?p0 · ?ω · p₀[?R, ?w1] = (?u0 · ?p0 · ?ω) · p₀[?R, ?w1]
        using C.comp-assoc by simp
      also have ... = ?v0 · p₀[?R, ?w1]
      proof −
        have ?u0 · ?p0 · ?ω = (?u0 · ?p0) · ?ω
          using C.comp-assoc by simp
        also have ... = ω.cod.leg0 · ?ω
          by (metis ω arrow-of-spans-data.select-convs(2) cod-char in-homE
              ru.leg0-composite)
        also have ... = ω.dom.leg0
          using ω.leg0-commutes by simp
        also have ... = ?v0
          using ω dom-char by auto
        finally show ?thesis by simp
      qed
      finally show ?thesis by simp
    qed
    also have ... = Dom-ϑ.leg0
      using Dom-ϑ-leg0-eq by simp
    finally show ?thesis by blast
  qed
  show Cod-ϑ.leg1 · Chn ϑ = Dom-ϑ.leg1
  proof −
    have Cod-ϑ.leg1 · Chn ϑ = ?u1 · ?p0 · ?ω · p₀[?R, ?w1]
```

```
      using Cod-ϑ-leg1-eq Chn-ϑ-eq by simp
    also have ... = r0 · ?q1
    proof −
      have ?u1 · ?p0 · ?ω · p0[?R, ?w1] = ((?u1 · ?p0) · ?ω) · p0[?R, ?w1]
        using C.comp-assoc by fastforce
      also have ... = ((r0 · ?p1) · ?ω) · p0[?R, ?w1]
        using C.pullback-commutes′ ru.legs-form-cospan by simp
      also have ... = r0 · ?w1 · p0[?R, ?w1]
        using C.comp-assoc by fastforce
      also have ... = r0 · ?R · ?q1
        using ω C.in-homE C.pullback-commutes′ w1 by auto
      also have ... = r0 · ?q1
        using ω w1 C.comp-cod-arr by auto
      finally show ?thesis by simp
    qed
    also have ... = Dom-ϑ.leg1
      using Dom-ϑ-leg1-eq by simp
    finally show ?thesis by blast
  qed
qed
```

Similarly, CKS say to take $\nu = 1\colon v \Rightarrow gw$, but obviously this can't be interpreted literally, either, because *v.apex* and *gw.apex* are not equal. Instead, we have to define ν so that $Chn\ \nu = C.inv\ \mathrm{p}_0[?R,\ ?w1]$, noting that $\mathrm{p}_0[?R,\ ?w1]$ is the pullback of an identity, hence is an isomorphism. Then $?v0 \cdot \mathrm{p}_0[?R,\ ?w1] \cdot Chn\ \nu = ?v0$ and $?v1 \cdot \mathrm{p}_1[?R,\ ?w1] \cdot Chn\ \nu = ?v1 \cdot ?w1$, showing that ν is an arrow of spans.

```
let ?ν′ = p0[?R, ?w1]
define ν
  where
    ν = ⦇Chn = C.inv ?ν′, Dom = Dom (dom ω), Cod = Cod (g ⋆ w)⦈
have iso-ν: C.inverse-arrows ?ν′ (Chn ν)
  using ν-def ω w1 C.iso-pullback-ide
  by (metis C.inv-is-inverse C.seqE arrow-of-spans-data.select-convs(1)
      r.chine-eq-apex r.chine-simps(1) r.chine-simps(3) r.cod-simps(1)
      r.dom.apex-def r.dom.ide-apex r.dom.is-span r1w1 v.dom.leg-simps(3))
```

have *w1-eq*: *?w1* = *?q1* · *C.inv ?ν′*
proof −
  have *?R* · *?q1* = *?w1* · *?ν′*
    using *iso-ν ω w1 C.pullback-commutes* [*of ?R ?w1*] **by** *blast*
  hence *?q1* = *?w1* · *?ν′*
    using *ω w1 C.comp-cod-arr* **by** *auto*
  thus *?thesis*
    using *iso-ν C.invert-side-of-triangle*(*2*)
    by (*metis C.isoI C.prj1-simps*(*1*) *arrow-of-spans-data.select-convs*(*3*)
      fw.legs-form-cospan(*2*) *span-data.simps*(*1*−*2*) *w-def*)
qed

interpret *Dom-ν*: *span-in-category C* ‹*Dom ν*›
  using *ν-def v.dom.span-in-category-axioms* **by** *simp*
interpret *Cod-ν*: *span-in-category C* ‹*Cod ν*›
  using *ν-def gw.cod.span-in-category-axioms* **by** *simp*
interpret *ν*: *arrow-of-spans C ν*
proof
  show *1*: «*Chn ν* : *Dom-ν.apex* $\rightarrow_C$ *Cod-ν.apex*»
  proof
    show *C.arr* (*Chn ν*)
      using *iso-ν* **by** *auto*
    show *C.dom* (*Chn ν*) = *Dom-ν.apex*
      using *ν-def iso-ν w1 gw-apex-eq* **by** *fastforce*
    show *C.cod* (*Chn ν*) = *Cod-ν.apex*
      using *ν-def iso-ν gw-apex-eq C.comp-inv-arr C.pbdom-def*
      by (*metis C.cod-comp arrow-of-spans-data.select-convs*(*3*)

```
        gw.apex-composite gw.chine-composite gw.chine-simps(1,3))
    qed
    show Cod-ν.leg0 · Chn ν = Dom-ν.leg0
      using w-def ν-def 1 iso-ν hcomp-def hseq-gw C.comp-arr-inv
          C.comp-assoc v.leg0-commutes
      by auto
    show Cod-ν.leg1 · Chn ν = Dom-ν.leg1
      using w-def ν-def hcomp-def hseq-gw C.comp-assoc w1-eq r1w1
      by auto
  qed
```

Now we can proceed to establishing the conclusions of *T1*.

```
  have ide w ∧ «ϑ : f ⋆ w ⇒ u» ∧ «ν : dom ω ⇒ dom ϱ ⋆ w» ∧ iso ν ∧
      T.composite-cell w ϑ · ν = ω
  proof (intro conjI)
    show ide-w: ide w
      using w by blast
    show ϑ: «ϑ : f ⋆ w ⇒ u»
    using ϑ-def ϑ.arrow-of-spans-axioms arr-char dom-char cod-char hseq-fw hseq-char
          hcomp-def fw.chine-eq-apex
      by auto
    show ν: «ν : dom ω ⇒ dom ϱ ⋆ w»
    proof −
      have «ν : dom ω ⇒ g ⋆ w»
        using ν-def ω ν.arrow-of-spans-axioms arr-char dom-char cod-char hseq-gw
              hseq-char hcomp-def gw.chine-eq-apex
        by auto
      thus ?thesis
        using T.tab-in-hom by simp
    qed
    show iso ν
      using iso-ν iso-char arr-char ν.arrow-of-spans-axioms by auto
    show T.composite-cell w ϑ · ν = ω
    proof (intro arr-eqI)
      have ϱw: «ϱ ⋆ w : g ⋆ w ⇒ (r ⋆ f) ⋆ w»
        using w-def ϱ ide-w hseq-rf hseq-fw hseq-gw by auto
      have rϑ: «r ⋆ ϑ : r ⋆ f ⋆ w ⇒ r ⋆ u»
        using arfw ide-r ϑ fw.composite-simps(2) rf.composable by auto
      have 1: «T.composite-cell w ϑ · ν : dom ω ⇒ r ⋆ u»
        using ν ϱw arfw rϑ by auto
      show 3: par (T.composite-cell w ϑ · ν) ω
        using 1 ω by (elim in-homE, auto)
      show Chn (T.composite-cell w ϑ · ν) = ?ω
      proof −
        have 2: Chn (T.composite-cell w ϑ · ν) =
                Chn (r ⋆ ϑ) · Chn a[r, f, w] · Chn (ϱ ⋆ w) · Chn ν
          using 1 3 Chn-vcomp C.comp-assoc
          by (metis (full-types) seqE)
        also have ... = ?ω
```

proof −
  let *?LHS* = *Chn* $(r \star \vartheta)$ · *Chn* $\mathrm{a}[r, f, w]$ · *Chn* $(\varrho \star w)$ · *Chn* ν
  have *Chn-rϑ*: *Chn* $(r \star \vartheta) = \langle r.chine \cdot \mathrm{p}_1[r0,\ r0 \cdot ?q1]$
    $[\![r0,\ ?u1]\!]$
    $\vartheta.chine \cdot \mathrm{p}_0[r0,\ r0 \cdot ?q1]\rangle$
    using *rϑ hcomp-def ϑ-def chine-hcomp-def Dom-ϑ-leg1-eq*
    by (*metis arrI arrow-of-spans-data.select-convs*(*1*,*3*)
      hseq-char r.cod-simps(*2*) *u.cod-simps*(*3*))
  have *Chn-arfw*: *Chn* $\mathrm{a}[r, f, w]$ = *rfw.chine-assoc*
    using *α-ide ide-f rf.composable fw.composable w* **by** *auto*
  have *Chn-ϱw*: *Chn* $(\varrho \star w) = \langle ?\varrho \cdot ?q1\ [\![k0,\ ?w1]\!]\ ?\nu'\rangle$
  proof −
    have *Chn* $(\varrho \star w)$ =
      chine-hcomp
        (|*Chn* = *?ϱ*,
         Dom = (|*Leg0* = *?R*, *Leg1* = *r1*|),
         Cod = (|*Leg0* = *k0*, *Leg1* = *r1* · *k1*|)|)
        (|*Chn* = *v.apex*,
         Dom = (|*Leg0* = *?v0*, *Leg1* = *?w1*|),
         Cod = (|*Leg0* = *?v0*, *Leg1* = *?w1*|)|)
      using *ϱ ide-w hseq-rf hseq-char hcomp-def src-def trg-def*
      by (*metis* (*no-types*, *lifting*) *ϱw arrI arrow-of-spans-data.select-convs*(*1*)
        v.dom.apex-def w-def)
    also have ... = $\langle ?\varrho \cdot ?q1\ [\![k0,\ ?w1]\!]\ ?V \cdot ?\nu'\rangle$
      unfolding *chine-hcomp-def* **by** *simp*
    also have ... = $\langle ?\varrho \cdot ?q1\ [\![k0,\ ?w1]\!]\ ?\nu'\rangle$
    proof −
      have $?V \cdot ?\nu' = ?\nu'$
        using *C.comp-ide-arr v.dom.ide-apex ϱ w1* **by** *auto*
      thus *?thesis* **by** *simp*
    qed
    finally show *?thesis* **by** *blast*
  qed

  have *3*: *C.seq ?p1 ?ω*
    using *w1* **by** *blast*
  moreover have *4*: *C.seq ?p1 ?LHS*
  proof
    show «*?LHS* : *v.apex* $\rightarrow_C$ *ru.apex*»
      by (*metis* (*no-types*, *lifting*) *1 2 Chn-in-hom ru.chine-eq-apex*
        v.chine-eq-apex)
    show «*?p1* : *ru.apex* $\rightarrow_C$ *?R*»
      using *P C.prj1-in-hom ru.legs-form-cospan* **by** *fastforce*
  qed
  moreover have *?p0* · *?LHS* = *?p0* · *?ω*
  proof −
    have *?p0* · *?LHS* =
      (*?p0* · *Chn* $(r \star \vartheta)$) · *Chn* $\mathrm{a}[r, f, w]$ · *Chn* $(\varrho \star w)$ · *Chn* ν
      using *C.comp-assoc* **by** *simp*

also have ... = $(\vartheta.chine \cdot \mathrm{p}_0[r0,\ r0 \cdot ?q1]) \cdot$
$Chn\ \mathrm{a}[r,\ f,\ w] \cdot Chn\ (\varrho \star w) \cdot Chn\ \nu$
proof −
have $?p0 \cdot Chn\ (r \star \vartheta) = \vartheta.chine \cdot \mathrm{p}_0[r0,\ r0 \cdot ?q1]$
by (*metis C.prj-tuple*(*1*) *Chn-rϑ ϑ-def arrI Dom-ϑ-leg1-eq*
arrow-of-spans-data.select-convs(*3*) *chine-hcomp-props*(*2*)
hseq-char r.cod-simps(*2*) *rϑ u.cod-simps*(*3*))
thus *?thesis* **by** *argo*
qed
also have
... = $?p0 \cdot ?\omega \cdot (rfw.Prj_{00} \cdot Chn\ \mathrm{a}[r,\ f,\ w]) \cdot Chn\ (\varrho \star w) \cdot Chn\ \nu$
using *w-def ϑ-def C.comp-assoc* **by** *simp*
also have ... = $?p0 \cdot ?\omega \cdot (rfw.Prj_0 \cdot Chn\ (\varrho \star w)) \cdot Chn\ \nu$
using *Chn-arfw rfw.prj-chine-assoc C.comp-assoc* **by** *simp*
also have ... = $?p0 \cdot ?\omega \cdot ?\nu' \cdot Chn\ \nu$
proof −
have $rfw.Prj_0 \cdot Chn\ (\varrho \star w) = \mathrm{p}_0[k0,\ ?w1] \cdot \langle ?\varrho \cdot ?q1\ [\![k0,\ ?w1]\!]\ ?\nu' \rangle$
using *w-def Chn-ϱw C.comp-cod-arr* **by** *simp*
also have ... = $?\nu'$
by (*metis* (*no-types*, *lifting*) *C.not-arr-null C.prj-tuple*(*1*) *C.seqE*
C.tuple-extensionality Chn-ϱw 4)
finally have $rfw.Prj_0 \cdot Chn\ (\varrho \star w) = ?\nu'$
by *blast*
thus *?thesis* **by** *simp*
qed
also have ... = $?p0 \cdot ?\omega$
using *iso-ν C.comp-arr-dom*
by (*metis* (*no-types*, *lifting*) *C.comp-arr-inv C.dom-comp ν-def*
ω.chine-simps(*1*) *3 arrow-of-spans-data.simps*(*1*) *w1-eq*)
finally show *?thesis* **by** *blast*
qed
moreover have $?p1 \cdot ?LHS = ?w1$
proof −
have $?p1 \cdot ?LHS =$
$(?p1 \cdot Chn\ (r \star \vartheta)) \cdot Chn\ \mathrm{a}[r,\ f,\ w] \cdot Chn\ (\varrho \star w) \cdot Chn\ \nu$
using *C.comp-assoc* **by** *simp*
also have ... = $(r.chine \cdot \mathrm{p}_1[r0,\ r0 \cdot ?q1]) \cdot Chn\ \mathrm{a}[r,\ f,\ w] \cdot$
$Chn\ (\varrho \star w) \cdot Chn\ \nu$
by (*metis* (*no-types*, *lifting*) *C.not-arr-null C.prj-tuple*(*2*) *C.seqE*
C.tuple-extensionality Chn-rϑ 4)
also have ... = $r.chine \cdot (rfw.Prj_1 \cdot Chn\ \mathrm{a}[r,\ f,\ w]) \cdot Chn\ (\varrho \star w) \cdot Chn\ \nu$
using *w-def Dom-ϑ-leg1-eq C.comp-assoc* **by** *simp*
also have ... = $r.chine \cdot (rfw.Prj_{11} \cdot Chn\ (\varrho \star w)) \cdot Chn\ \nu$
using *Chn-arfw rfw.prj-chine-assoc*(*1*) *C.comp-assoc* **by** *simp*
also have ... = $r.chine \cdot ?q1 \cdot Chn\ \nu$
proof −
have $rfw.Prj_{11} \cdot Chn\ (\varrho \star w) =$
$(k1 \cdot \mathrm{p}_1[k0,\ ?w1]) \cdot \langle ?\varrho \cdot ?q1\ [\![k0,\ ?w1]\!]\ ?\nu' \rangle$
using *w-def Chn-ϱw C.comp-cod-arr* **by** *simp*

```
            also have ... = k1 · p1[k0, ?w1] · ⟨?ϱ · ?q1 ⟦k0, ?w1⟧ ?ν'⟩
              using C.comp-assoc by simp
            also have ... = k1 · ?ϱ · ?q1
              by (metis (no-types, lifting) C.not-arr-null C.prj-tuple(2)
                  C.seqE C.tuple-extensionality Chn-ϱw 4)
            also have ... = (k1 · ?ϱ) · ?q1
              using C.comp-assoc by presburger
            also have ... = ?R · ?q1
              by simp
            also have ... = ?q1
              by (metis Dom-ϑ-leg1-eq C.comp-ide-arr C.prj1-simps(3)
                  C.prj1-simps-arr C.seqE C.seqI Dom-ϑ.leg-simps(3)
                  r.dom.ide-apex)
            finally have rfw.Prj11 · Chn (ϱ ⋆ w) = ?q1
              by blast
            thus ?thesis by simp
          qed
          also have ... = (r.chine · ?p1) · ?ω
            using ν-def w1-eq C.comp-assoc by simp
          also have ... = ?w1
            using C.comp-cod-arr r.chine-eq-apex ru.prj-simps(1) by auto
          finally show ?thesis by blast
        qed
        ultimately show ?thesis
          using ru.legs-form-cospan C.prj-joint-monic by blast
      qed
      finally show ?thesis by argo
    qed
  qed
 qed
 thus ?thesis
   using w-def by auto
 qed
qed

show T2: ⋀u w w' ϑ ϑ' β.
          ⟦ ide w; ide w';
            «ϑ : f ⋆ w ⇒ u»; «ϑ' : f ⋆ w' ⇒ u»; «β : g ⋆ w ⇒ g ⋆ w'»;
            T.composite-cell w ϑ = T.composite-cell w' ϑ' · β ⟧ ⟹
          ∃!γ. «γ : w ⇒ w'» ∧ β = g ⋆ γ ∧ ϑ = ϑ' · (f ⋆ γ)
proof −
  fix u w w' ϑ ϑ' β
  assume ide-w: ide w
  assume ide-w': ide w'
  assume ϑ: «ϑ : f ⋆ w ⇒ u»
  assume ϑ': «ϑ' : f ⋆ w' ⇒ u»
  assume β: «β : g ⋆ w ⇒ g ⋆ w'»
  assume E: T.composite-cell w ϑ = T.composite-cell w' ϑ' · β
  interpret T: uwϑw'ϑ'β vcomp hcomp assoc unit src trg r ϱ f g u w ϑ w' ϑ' β
```

using *ide-w ide-w′ ϑ ϑ′ β E comp-assoc*
by *unfold-locales auto*

show $\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$
proof
interpret *u*: *identity-arrow-of-spans C u*
using *T.uwϑ.u-simps*(*1*) *ide-char′* **by** *auto*
interpret *w*: *identity-arrow-of-spans C w*
using *ide-w ide-char′* **by** *auto*
interpret *w′*: *identity-arrow-of-spans C w′*
using *ide-w′ ide-char′* **by** *auto*
let *?u0* = *u.leg0*
let *?u1* = *u.leg1*
let *?w0* = *w.leg0*
let *?w1* = *w.leg1*
let *?wa* = *w.apex*
let *?w0′* = *w′.leg0*
let *?w1′* = *w′.leg1*
let *?wa′* = *w′.apex*
let *?R* = *ra*
let *?p0* = p_0[*?R*, *?w1*]
let *?p0′* = p_0[*?R*, *?w1′*]
let *?p1* = p_1[*?R*, *?w1*]
let *?p1′* = p_1[*?R*, *?w1′*]

interpret *fw*: *two-composable-identity-arrows-of-spans C prj0 prj1 f w*
using *hseq-char* **by** *unfold-locales auto*
interpret *fw′*: *two-composable-identity-arrows-of-spans C prj0 prj1 f w′*
using *hseq-char* **by** *unfold-locales auto*

have *hseq-gw*: *hseq g w*
using *T.leg1-in-hom* **by** *auto*
interpret *gw*: *two-composable-identity-arrows-of-spans C prj0 prj1 g w*
using *hseq-gw hseq-char* **by** *unfold-locales auto*

have *hseq-gw′*: *hseq g w′*
using *T.leg1-in-hom* **by** *auto*
interpret *gw′*: *two-composable-identity-arrows-of-spans C prj0 prj1 g w′*
using *hseq-gw′ hseq-char* **by** *unfold-locales auto*

interpret *rfw*: *three-composable-identity-arrows-of-spans C prj0 prj1 r f w* **..**
interpret *rfw*: *identity-arrow-of-spans C* ‹$r \star f \star w$›
using *rfw.composites-are-identities ide-char′* **by** *auto*
interpret *rfw′*: *three-composable-arrows-of-spans C prj0 prj1 r f w′* **..**
interpret *rfw′*: *three-composable-identity-arrows-of-spans C prj0 prj1 r f w′* **..**
interpret *rfw′*: *identity-arrow-of-spans C* ‹$r \star f \star w'$›
using *rfw′.composites-are-identities ide-char′* **by** *auto*

have *ϱw*: «$\varrho \star w : g \star w \Rightarrow (r \star f) \star w$»

using ϱ *hseq-gw* **by** *blast*
interpret ϱw: *two-composable-arrows-of-spans C prj0 prj1* ϱ *w*
using ϱw **by** *unfold-locales auto*
have $\varrho w'$: «$\varrho \star w' : g \star w' \Rightarrow (r \star f) \star w'$»
using ϱ *hseq-gw'* **by** *blast*
interpret $\varrho w'$: *two-composable-arrows-of-spans C prj0 prj1* ϱ w'
using $\varrho w'$ **by** *unfold-locales auto*

have *arfw*: «$\mathrm{a}[r, f, w] : (r \star f) \star w \Rightarrow r \star f \star w$»
using *fw.composable ide-f ide-r ide-w rf.composable* **by** *auto*
have *arfw'*: «$\mathrm{a}[r, f, w'] : (r \star f) \star w' \Rightarrow r \star f \star w'$»
using *fw'.composable ide-f ide-r ide-w' rf.composable* **by** *auto*

have $r\vartheta$: «$r \star \vartheta : r \star f \star w \Rightarrow r \star u$»
by *fastforce*
interpret *Dom-*ϑ: *span-in-category C* ‹*Dom* ϑ›
using *fw.dom.span-in-category-axioms*
by (*metis* ϑ *arrow-of-spans-data.select-convs*(*2*) *in-homE dom-char*)
interpret *Cod-*ϑ: *span-in-category C* ‹*Cod* ϑ›
using ϑ *u.cod.span-in-category-axioms cod-char* **by** *auto*
interpret ϑ: *arrow-of-spans C* ϑ
using *arr-char T.uw*ϑ*.*ϑ*-simps*(*1*) **by** *auto*
interpret $r\vartheta$: *two-composable-arrows-of-spans C prj0 prj1 r* ϑ
using $r\vartheta$ **by** *unfold-locales auto*

have $r\vartheta'$: «$r \star \vartheta' : r \star f \star w' \Rightarrow r \star u$»
by *fastforce*
interpret *Dom-*ϑ': *span-in-category C* ‹*Dom* ϑ'›
using *fw'.dom.span-in-category-axioms*
by (*metis* ϑ' *arrow-of-spans-data.select-convs*(*2*) *in-homE dom-char*)
interpret *Cod-*ϑ': *span-in-category C* ‹*Cod* ϑ'›
using ϑ' *u.cod.span-in-category-axioms cod-char* **by** *auto*
interpret ϑ': *arrow-of-spans C* ϑ'
using *arr-char T.uw'*ϑ'*.*ϑ*-simps*(*1*) **by** *auto*
interpret $r\vartheta'$: *two-composable-arrows-of-spans C prj0 prj1 r* ϑ'
using $r\vartheta'$ **by** *unfold-locales auto*

have *7*: «*T.composite-cell* w' $\vartheta' \cdot \beta : g \star w \Rightarrow r \star u$»
using β $\varrho w'$ *arfw'* $r\vartheta'$ **by** *auto*
have *8*: «*T.composite-cell* w $\vartheta : g \star w \Rightarrow r \star u$»
using ϱw *arfw* $r\vartheta$ **by** *auto*

interpret *ru*: *two-composable-identity-arrows-of-spans C prj0 prj1 r u*
using *hseq-char* **by** *unfold-locales auto*

interpret *Dom-*β: *span-in-category C* ‹*Dom* β›
using β *fw.dom.span-in-category-axioms arr-char*
by (*metis comp-arr-dom in-homE gw.cod.span-in-category-axioms seq-char*)
interpret *Cod-*β: *span-in-category C* ‹*Cod* β›

using β *fw.cod.span-in-category-axioms arr-char*
by (*metis* (*no-types*, *lifting*) *comp-arr-dom ideD*(*2*) *in-homI*
gw′.cod.span-in-category-axioms gw′.chine-is-identity hseq-gw′ seqI′
seq-char ide-char)
interpret β: *arrow-of-spans C* β
using β *arr-char* **by** *auto*

CKS say: "Take u, w, w', ϑ, ϑ' as in *T2* and note that $fw = (w_0,\ W,\ r_0\ w_1)$, $gw = (w_0,\ W,\ r_1\ w_1)$, *etc.* So β: $W \to W'$ satisfies $w_0 = w_0{}'\ \beta$, $r_1\ w_1 = r_1\ w_1{}'\ \beta$. But the equation $(r\vartheta)(\varrho w) = (r\vartheta')(\varrho w')\beta$ gives $w_1 = w_1{}'$. So $\gamma = \beta : w \Rightarrow w'$ is unique with $\beta = g\ \gamma$, $\vartheta = \vartheta'\ (f\ \gamma)$."

Once again, there is substantial punning in the proof sketch given by CKS. We can express *fw* and *gw* almost in the form they indicate, but projections are required.

have *cospan*: *C.cospan ?R ?w1*
using *hseq-char* [*of* ϱ *w*] *src-def trg-def* **by** *auto*
have *cospan′*: *C.cospan ?R ?w1′*
using *hseq-char* [*of* ϱ *w′*] *src-def trg-def* **by** *auto*
have *fw*: $f \star w = (\!|\mathit{Chn} = ?R \downarrow\downarrow ?w1,$
$\mathit{Dom} = (\!|\mathit{Leg0} = ?w0 \cdot ?p0,\ \mathit{Leg1} = r0 \cdot ?p1|\!),$
$\mathit{Cod} = (\!|\mathit{Leg0} = ?w0 \cdot ?p0,\ \mathit{Leg1} = r0 \cdot ?p1|\!)|\!)$
using *ide-f hseq-char hcomp-def chine-hcomp-def fw.dom.apex-def cospan*
fw.chine-eq-apex
by *auto*
have *gw*: $g \star w = (\!|\mathit{Chn} = ?R \downarrow\downarrow ?w1,$
$\mathit{Dom} = (\!|\mathit{Leg0} = ?w0 \cdot ?p0,\ \mathit{Leg1} = r1 \cdot ?p1|\!),$
$\mathit{Cod} = (\!|\mathit{Leg0} = ?w0 \cdot ?p0,\ \mathit{Leg1} = r1 \cdot ?p1|\!)|\!)$
using *hseq-gw hseq-char hcomp-def chine-hcomp-def gw.dom.apex-def cospan*
gw.chine-eq-apex
by *auto*
have *fw′*: $f \star w' = (\!|\mathit{Chn} = ?R \downarrow\downarrow ?w1',$
$\mathit{Dom} = (\!|\mathit{Leg0} = ?w0' \cdot ?p0',\ \mathit{Leg1} = r0 \cdot ?p1'|\!),$
$\mathit{Cod} = (\!|\mathit{Leg0} = ?w0' \cdot ?p0',\ \mathit{Leg1} = r0 \cdot ?p1'|\!)|\!)$
using *ide-f hseq-char hcomp-def chine-hcomp-def fw′.dom.apex-def cospan′*
fw′.chine-eq-apex
by *auto*
have *gw′*: $g \star w' = (\!|\mathit{Chn} = ?R \downarrow\downarrow ?w1',$
$\mathit{Dom} = (\!|\mathit{Leg0} = ?w0' \cdot ?p0',\ \mathit{Leg1} = r1 \cdot ?p1'|\!),$
$\mathit{Cod} = (\!|\mathit{Leg0} = ?w0' \cdot ?p0',\ \mathit{Leg1} = r1 \cdot ?p1'|\!)|\!)$
using *hseq-gw′ hseq-char hcomp-def chine-hcomp-def gw′.dom.apex-def cospan′*
gw′.chine-eq-apex
by *auto*

Note that *?p0* and *?p0′* are only isomorphisms, not identities, and we have *?p1* (which equals *?w1* · *?p0*) and *?p1′* (which equals *?w1′* · *?p0′*) in place of *?w1* and *?w1′*.

The following diagram summarizes the various given and defined arrows involved in the proof. We have deviated slightly here from the nomenclature used in in CKS. We prefer to use W and W' to denote the apexes of w and w', respectively. We already have the expressions *?R* ↓↓ *?w1* and *?R* ↓↓ *?w1′* for the apexes of *fw* and *fw′* (which are the

same as the apexes of gw and gw', respectively) and we will not use any abbreviation for them.

have *Chn-β*: «*β.chine*: *?R* ⋎ *?w1* $\rightarrow_C$ *?R* ⋎ *?w1'*»
  using *gw gw' Chn-in-hom β gw'.chine-eq-apex gw.chine-eq-apex* **by** *force*
have *β-eq*: *β* = ⦇*Chn* = *β.chine*,
  Dom = ⦇*Leg0* = *?w0* · *?p0*, *Leg1* = *r1* · *?p1*⦈,
  Cod = ⦇*Leg0* = *?w0'* · *?p0'*, *Leg1* = *r1* · *?p1'*⦈⦈
  using *β gw gw' dom-char cod-char* **by** *auto*
have *Dom-β-eq*: *Dom β* = ⦇*Leg0* = *?w0* · *?p0*, *Leg1* = *r1* · *?p1*⦈
  using *β gw gw' dom-char cod-char* **by** *auto*
have *Cod-β-eq*: *Cod β* = ⦇*Leg0* = *?w0'* · *?p0'*, *Leg1* = *r1* · *?p1'*⦈
  using *β gw gw' dom-char cod-char* **by** *auto*

have *β0*: *?w0* · *?p0* = *?w0'* · *?p0'* · *β.chine*
  using *Dom-β-eq Cod-β-eq β.leg0-commutes C.comp-assoc* **by** *simp*
have *β1*: *r1* · *?p1* = *r1* · *?p1'* · *β.chine*
  using *Dom-β-eq Cod-β-eq β.leg1-commutes C.comp-assoc* **by** *simp*

have *Dom-ϑ-0*: *Dom-ϑ.leg0* = *?w0* · *?p0*
  using *arrI dom-char fw T.uwϑ.ϑ-simps*(*4*) **by** *auto*
have *Cod-ϑ-0*: *Cod-ϑ.leg0* = *?u0*
  using *ϑ cod-char* **by** *auto*
have *Dom-ϑ-1*: *Dom-ϑ.leg1* = *r0* · *?p1*
  using *arrI dom-char fw T.uwϑ.ϑ-simps*(*4*) **by** *auto*
have *Cod-ϑ-1*: *Cod-ϑ.leg1* = *?u1*
  using *T.uwϑ.ϑ-simps*(*5*) *cod-char* **by** *auto*

have *Dom-ϑ′-0*: *Dom-ϑ′.leg0* = *?w0′* · *?p0′*
  using *dom-char fw′ T.uw′ϑ′.ϑ-simps*(*4*) **by** *auto*
have *Cod-ϑ′-0*: *Cod-ϑ′.leg0* = *?u0*
  using *T.uw′ϑ′.ϑ-simps*(*5*) *cod-char* **by** *auto*
have *Dom-ϑ′-1*: *Dom-ϑ′.leg1* = *r0* · *?p1′*
  using *dom-char fw′ T.uw′ϑ′.ϑ-simps*(*4*) **by** *auto*
have *Cod-ϑ′-1*: *Cod-ϑ′.leg1* = *?u1*
  using *T.uw′ϑ′.ϑ-simps*(*5*) *cod-char* **by** *auto*
have *Dom-ϱ-0*: *Dom-ϱ.leg0* = *?R*
  by *simp*
have *Dom-ϱ-1*: *Dom-ϱ.leg1* = *r1*
  by *simp*
have *Cod-ϱ-0*: *Cod-ϱ.leg0* = *k0*
  by *simp*
have *Cod-ϱ-1*: *Cod-ϱ.leg1* = *r1* · *k1*
  by *simp*

have *Chn-rϑ*: «*rϑ.chine* : *rfw.chine* $\rightarrow_C$ *ru.chine*»
  using *rϑ.chine-composite-in-hom ru.chine-composite rfw.chine-composite*
    Cod-ϑ-1 Dom-ϑ-1 fw.leg1-composite
  by *auto*
have *Chn-rϑ-eq*: *rϑ.chine* = ⟨p_1[*r0*, *r0* · *?p1*] ⟦*r0*, *?u1*⟧ *ϑ.chine* · p_0[*r0*, *r0* · *?p1*]⟩
  using *rϑ.chine-composite Cod-ϑ-1 Dom-ϑ-1 fw.leg1-composite C.comp-cod-arr*
  by (*metis arrow-of-spans-data.simps*(*2*) *fw r.chine-eq-apex r.cod-simps*(*2*)
    rfw.prj-simps(*10*) *rfw.prj-simps*(*16*) *span-data.simps*(*2*))

have *rϑ-cod-apex-eq*: *rϑ.cod.apex* = *r0* ↓↓ *?u1*
  using *Cod-ϑ-1 rϑ.chine-composite-in-hom* **by** *auto*
hence *rϑ′-cod-apex-eq*: *rϑ′.cod.apex* = *r0* ↓↓ *?u1*
  using *Cod-ϑ′-1 rϑ′.chine-composite-in-hom* **by** *auto*

have *Chn-rϑ′*: «*rϑ′.chine* : *rfw′.chine* $\rightarrow_C$ *ru.chine*»
  using *rϑ′.chine-composite-in-hom ru.chine-composite rfw′.chine-composite*
    Cod-ϑ′-1 Dom-ϑ′-1 fw′.leg1-composite
  by *auto*
have *Chn-rϑ′-eq*: *rϑ′.chine* =
    ⟨p_1[*r0*, *r0* · *?p1′*] ⟦*r0*, *?u1*⟧ *ϑ′.chine* · p_0[*r0*, *r0* · *?p1′*]⟩
  using *rϑ′.chine-composite Cod-ϑ′-1 Dom-ϑ′-1 fw′.leg1-composite C.comp-cod-arr*
  by (*metis arrow-of-spans-data.simps*(*2*) *fw′ r.chine-eq-apex r.cod-simps*(*2*)
    rfw′.prj-simps(*10*) *rfw′.prj-simps*(*16*) *span-data.simps*(*2*))

have *Chn-ϱw*: «*ϱw.chine* : *?R* ↓↓ *?w1* $\rightarrow_C$ *k0* ↓↓ *?w1*»
  using *ϱw.chine-composite-in-hom* **by** *simp*
have *Chn-ϱw-eq*: *ϱw.chine* = ⟨*ϱ.chine* · *?p1* ⟦*k0*, *?w1*⟧ *?p0*⟩
  using *ϱw.chine-composite C.comp-cod-arr ide-w*
  by (*simp add*: *chine-hcomp-arr-ide hcomp-def*)
have *Chn-ϱw′*: «*ϱw′.chine* : *?R* ↓↓ *?w1′* $\rightarrow_C$ *k0* ↓↓ *?w1′*»
  using *ϱw′.chine-composite-in-hom* **by** *simp*
have *Chn-ϱw′-eq*: *ϱw′.chine* = ⟨*ϱ.chine* · *?p1′* ⟦*k0*, *?w1′*⟧ *?p0′*⟩

using *ϱw′.chine-composite C.comp-cod-arr ide-w′ Dom-ϱ-0 Cod-ϱ-0*
by (*metis ϱw′.composite-is-arrow chine-hcomp-arr-ide chine-hcomp-def hseq-char w′.cod-simps*(*3*))

The following are some collected commutativity properties that are used subsequently.

have *C.commutative-square r0 ?u1 ?p1 ϑ.chine*
using *ru.legs-form-cospan*(*1*) *Dom-ϑ.is-span Dom-ϑ-1 Cod-ϑ-1 ϑ.leg1-commutes*
by (*intro C.commutative-squareI*) *auto*
have *C.commutative-square r0 ?u1* (*?p1′* · *β.chine*) (*ϑ′.chine* · *β.chine*)
by (*metis* (*mono-tags*, *lifting*) *C.commutative-square-comp-arr C.dom-comp C.seqE Cod-ϑ′-1 Dom-β.leg-simps*(*3*) *Dom-β-eq Dom-ϑ′.leg-simps*(*3*) *Dom-ϑ′-1 β1 ϑ′.leg1-commutes C.commutative-squareI ru.legs-form-cospan*(*1*) *span-data.simps*(*2*))
have *C.commutative-square r0 ?u1* p_1[*r0*, *r0* · *?p1*] (*ϑ.chine* · p_0[*r0*, *r0* · *?p1*])
using *ru.legs-form-cospan*(*1*) *Dom-ϑ.is-span Dom-ϑ-1 C.comp-assoc C.pullback-commutes′ rϑ.legs-form-cospan*(*1*)
apply (*intro C.commutative-squareI*)
apply *auto*
by (*metis C.comp-assoc Cod-ϑ-1 ϑ.leg1-commutes*)
hence *C.commutative-square r0 ?u1* p_1[*r0*, *r0* · *?p1*] (*ϑ.chine* · p_0[*r0*, *r0* · *?p1*])
using *fw.leg1-composite* **by** *auto*
have *C.commutative-square r0 ?u1* p_1[*r0*, *r0* · *?p1′*] (*ϑ′.chine* · p_0[*r0*, *r0* · *?p1′*])
using *C.tuple-extensionality Chn-rϑ′-eq rϑ′.chine-simps*(*1*) *fw′* **by** *force*
have *C.commutative-square ra ?w1 rfw.Prj*$_{01}$ *rfw.Prj*$_0$
using *C.pullback-commutes′ gw.legs-form-cospan*(*1*) *rfw.prj-simps*(*2*) *C.comp-assoc C.comp-cod-arr*
by (*intro C.commutative-squareI*) *auto*
have *C.commutative-square ?R ?w1′ rfw′.Prj*$_{01}$ *rfw′.Prj*$_0$
by (*metis* (*no-types*, *lifting*) *C.commutative-square-comp-arr C.comp-assoc C.pullback-commutes select-convs*(*2*) *rfw′.cospan-νπ rfw′.prj-chine-assoc*(*2*) *rfw′.prj-chine-assoc*(*3*) *rfw′.prj-simps*(*2*) *span-data.select-convs*(*1*))
have *C.commutative-square r0* (*r0* · *?p1*) *rfw.Prj*$_{11}$ ⟨*rfw.Prj*$_{01}$ ⟦*ra*, *?w1*⟧ *rfw.Prj*$_0$⟩
proof −
have *C.arr rfw.chine-assoc*
by (*metis C.seqE rfw.prj-chine-assoc*(*1*) *rfw.prj-simps*(*1*))
thus *?thesis*
using *C.tuple-extensionality rfw.chine-assoc-def* **by** *fastforce*
qed
have *C.commutative-square r0* (*r0* · *?p1′*) *rfw′.Prj*$_{11}$ ⟨*rfw′.Prj*$_{01}$ ⟦*ra*, *?w1′*⟧ *rfw′.Prj*$_0$⟩
by (*metis* (*no-types*, *lifting*) *C.not-arr-null C.seqE C.tuple-extensionality arrow-of-spans-data.select-convs*(*2*) *rfw′.chine-assoc-def rfw′.prj-chine-assoc*(*1*) *rfw′.prj-simps*(*1*) *span-data.select-convs*(*1*−*2*))
have *C.commutative-square k0 ?w1* (*ϱ.chine* · *?p1*) *?p0*
using *C.tuple-extensionality Chn-ϱw-eq ϱw.chine-simps*(*1*) **by** *fastforce*
have *C.commutative-square k0 ?w1′* (*ϱ.chine* · *?p1′*) (*w′.chine* · *?p0′*)
using *C.tuple-extensionality ϱw′.chine-composite ϱw′.chine-simps*(*1*) **by** *force*
have *C.commutative-square k0 ?w1′* (*ϱ.chine* · *?p1′*) *?p0′*
using *C.tuple-extensionality Chn-ϱw′-eq ϱw′.chine-simps*(*1*) **by** *force*

Now, derive the consequences of the equation:

$$(r \star \vartheta) \cdot \mathrm{a}[r,\ ?f,\ w] \cdot (?\varrho \star w) = (r \star \vartheta') \cdot \mathrm{a}[r,\ ?f,\ w'] \cdot (?\varrho \star w') \cdot \beta$$

The strategy is to expand and simplify the left and right hand side to tuple form, then compose with projections and equate corresponding components.

We first work on the right-hand side.

```
have R: Chn (T.composite-cell w' ϑ' · β) =
        ⟨?p1' · β.chine ⟦r0, ?u1⟧ ϑ'.chine · β.chine⟩
proof −
  have Chn (T.composite-cell w' ϑ' · β) =
       rϑ'.chine · Chn a[r, f, w'] · ϱw'.chine · β.chine
  proof −
    have 1: «T.composite-cell w' ϑ' · β : g ⋆ w ⇒ r ⋆ u»
      using β ϱw' arfw' rϑ' by auto
    have Chn (T.composite-cell w' ϑ' · β) = Chn (T.composite-cell w' ϑ') · β.chine
      using 1 Chn-vcomp by blast
    also have ... = (rϑ'.chine · Chn (a[r, f, w'] · (ϱ ⋆ w'))) · β.chine
    proof −
      have seq (r ⋆ ϑ') (a[r, f, w'] · (ϱ ⋆ w'))
        using 1 by blast
      thus ?thesis
        using 1 Chn-vcomp by presburger
    qed
    also have ... = (rϑ'.chine · Chn a[r, f, w'] · ϱw'.chine) · β.chine
    proof −
      have seq a[r, f, w'] (ϱ ⋆ w')
        using 1 by blast
      thus ?thesis
        using 1 Chn-vcomp by presburger
    qed
    finally show ?thesis
      using C.comp-assoc by auto
  qed
  also have ... = ⟨?p1' · β.chine ⟦r0, ?u1⟧ ϑ'.chine · β.chine⟩
  proof −
    let ?LHS = rϑ'.chine · Chn a[r, f, w'] · ϱw'.chine · β.chine
    let ?RHS = ⟨?p1' · β.chine ⟦r0, ?u1⟧ ϑ'.chine · β.chine⟩

    have LHS: «?LHS : ?R ↓↓ ?w1 →C rϑ'.cod.apex»
    proof (intro C.comp-in-homI)
      show «β.chine : ?R ↓↓ ?w1 →C ?R ↓↓ ?w1'»
        using Chn-β by simp
      show «ϱw'.chine : ?R ↓↓ ?w1' →C Cod-ϱ.leg0 ↓↓ w'.cod.leg1»
        using Chn-ϱw' by simp
      show «Chn a[r, f, w'] : Cod-ϱ.leg0 ↓↓ w'.cod.leg1 →C rfw'.chine»
        using arfw'
        by (metis (no-types, lifting) Chn-in-hom Cod-ϱ-0
            arrow-of-spans-data.simps(2) rf rf.leg0-composite rfw'.chine-composite(1)
```

```
        span-data.select-convs(1) w'.cod-simps(3))
  show «rϑ'.chine : rfw'.chine →C rϑ'.cod.apex»
    using Chn-rϑ' by auto
qed
have 2: C.commutative-square r0 ?u1
          (?p1' · β.chine) (ϑ'.chine · β.chine)
  by fact
have RHS: «?RHS : ?R ↓↓ ?w1 →C rϑ'.cod.apex»
  using 2 Chn-β rϑ'-cod-apex-eq
        C.tuple-in-hom [of r0 ?u1 ?p1' · β.chine ϑ'.chine · β.chine]
  by fastforce

show ?thesis
proof (intro C.prj-joint-monic [of r0 ?u1 ?LHS ?RHS])
  show C.cospan r0 ?u1
    using ru.legs-form-cospan(1) by blast
  show C.seq ru.prj1 ?LHS
    using LHS rϑ'-cod-apex-eq by auto
  show C.seq ru.prj1 ?RHS
    using RHS rϑ'-cod-apex-eq by auto
  show ru.prj0 · ?LHS = ru.prj0 · ?RHS
  proof −
    have ru.prj0 · ?LHS =
          (ru.prj0 · rϑ'.chine) · Chn a[r, f, w'] · ϱw'.chine · β.chine
      using C.comp-assoc by simp
    also have ... = ((ϑ'.chine · p0[r0, r0 · ?p1']) · Chn a[r, f, w']) ·
                    ϱw'.chine · β.chine
      using Chn-rϑ'-eq C.comp-assoc fw'
            ‹C.commutative-square r0 ?u1 p1[r0, r0 · ?p1']
              (ϑ'.chine · p0[r0, r0 · ?p1'])›
      by simp
    also have ... = ϑ'.chine · (p0[r0, r0 · ?p1'] · Chn a[r, f, w']) ·
                    ϱw'.chine · β.chine
      using C.comp-assoc by simp
    also have ... = ϑ'.chine · (⟨rfw'.Prj01 ⟦?R, ?w1'⟧ rfw'.Prj0⟩ · ϱw'.chine) ·
                    β.chine
      using ide-f hseq-rf hseq-char α-ide C.comp-assoc
            rfw'.chine-assoc-def fw'.leg1-composite C.prj-tuple(1)
            ‹C.commutative-square r0 (r0 · ?p1')
              rfw'.Prj11 ⟨rfw'.Prj01 ⟦?R, ?w1'⟧ rfw'.Prj0⟩›
      by simp
    also have ... = ϑ'.chine · β.chine
    proof −
      have ⟨rfw'.Prj01 ⟦?R, ?w1'⟧ rfw'.Prj0⟩ · ϱw'.chine = gw'.apex
      proof (intro C.prj-joint-monic
                   [of ?R ?w1' ⟨rfw'.Prj01 ⟦?R, ?w1'⟧ rfw'.Prj0⟩ · ϱw'.chine
                       gw'.apex])
        show C.cospan ?R ?w1'
          using fw'.legs-form-cospan(1) by simp
```

show *C.seq ?p1′* (⟨*rfw′.Prj*$_{01}$ ⟦*?R*, *?w1′*⟧ *rfw′.Prj*$_0$⟩ · *ϱw′.chine*)
proof (*intro C.seqI′ C.comp-in-homI*)
 show «*ϱw′.chine* : *Dom-ϱ.leg0* ↓↓ *w′.leg1* →$_C$ *Cod-ϱ.leg0* ↓↓ *w′.cod.leg1*»
  using *ϱw′.chine-composite-in-hom* **by** *simp*
 show «⟨*rfw′.Prj*$_{01}$ ⟦*?R*, *w′.leg1*⟧ *rfw′.Prj*$_0$⟩ :
   Cod-ϱ.leg0 ↓↓ *w′.cod.leg1* →$_C$ *?R* ↓↓ *w′.leg1*»
  using ‹*C.commutative-square ?R ?w1′ rfw′.Prj*$_{01}$ *rfw′.Prj*$_0$›
   C.tuple-in-hom [*of ?R ?w1′ rfw′.Prj*$_{01}$ *rfw′.Prj*$_0$]
   rf rf.leg0-composite
  by *auto*
 show «*?p1′* : *?R* ↓↓ *w′.leg1* →$_C$ *f.apex*»
  using *fw′.prj-in-hom*(*1*) **by** *auto*
qed
show *C.seq ?p1′ gw′.apex*
 using *gw′.dom.apex-def gw′.leg0-composite fw′.prj-in-hom* **by** *auto*
show *?p0′* · ⟨*rfw′.Prj*$_{01}$ ⟦*?R*, *?w1′*⟧ *rfw′.Prj*$_0$⟩ · *ϱw′.chine* =
   ?p0′ · *gw′.apex*
proof −
 have *?p0′* · ⟨*rfw′.Prj*$_{01}$ ⟦*?R*, *?w1′*⟧ *rfw′.Prj*$_0$⟩ · *ϱw′.chine* =
   (*?p0′* · ⟨*rfw′.Prj*$_{01}$ ⟦*?R*, *?w1′*⟧ *rfw′.Prj*$_0$⟩) · *ϱw′.chine*
  using *C.comp-assoc* **by** *simp*
 also have ... = *rfw′.Prj*$_0$ · *ϱw′.chine*
  using ‹*C.commutative-square ?R ?w1′ rfw′.Prj*$_{01}$ *rfw′.Prj*$_0$› **by** *auto*
 also have
  ... = p_0[*k0*, *?w1′*] · ⟨*ϱ.chine* · *?p1′* ⟦*k0*, *?w1′*⟧ *w′.chine* · *?p0′*⟩
  using *ϱw′.chine-composite Dom-ϱ-0 Cod-ϱ-0 C.comp-cod-arr* **by** *simp*
 also have ... = *w′.chine* · *?p0′*
  using ‹*C.commutative-square k0 ?w1′*
   (*ϱ.chine* · *?p1′*) (*w′.chine* · *?p0′*)›
  by *simp*
 also have ... = *?p0′* · *gw′.apex*
  using *cospan C.comp-cod-arr C.comp-arr-dom gw′.chine-is-identity*
   gw′.chine-eq-apex gw′.chine-composite fw′.prj-in-hom
  by *auto*
 finally show *?thesis* **by** *simp*
qed
show *?p1′* · ⟨*rfw′.Prj*$_{01}$ ⟦*ra*, *?w1′*⟧ *rfw′.Prj*$_0$⟩ · *ϱw′.chine* =
   ?p1′ · *gw′.apex*
proof −
 have *?p1′* · ⟨*rfw′.Prj*$_{01}$ ⟦*ra*, *?w1′*⟧ *rfw′.Prj*$_0$⟩ · *ϱw′.chine* =
   (*?p1′* · ⟨*rfw′.Prj*$_{01}$ ⟦*ra*, *?w1′*⟧ *rfw′.Prj*$_0$⟩) · *ϱw′.chine*
  using *C.comp-assoc* **by** *simp*
 also have ... = *rfw′.Prj*$_{01}$ · *ϱw′.chine*
  using ‹*C.commutative-square ?R ?w1′ rfw′.Prj*$_{01}$ *rfw′.Prj*$_0$› **by** *simp*
 also have
  ... = *k0* · p_1[*k0*, *?w1′*] · ⟨*ϱ.chine* · *?p1′* ⟦*k0*, *?w1′*⟧ *w′.chine* · *?p0′*⟩
  using *ϱw′.chine-composite Cod-ϱ-0 C.comp-assoc C.comp-cod-arr*
  by *simp*
 also have ... = *k0* · *ϱ.chine* · *?p1′*

```
        using ‹C.commutative-square k0 ?w1'
                (ϱ.chine · ?p1') (w'.chine · ?p0')›
        by simp
      also have ... = (k0 · ϱ.chine) · ?p1'
        using C.comp-assoc by metis
      also have ... = ?p1'
        using ϱ.leg0-commutes C.comp-cod-arr cospan' by simp
      also have ... = ?p1' · gw'.apex
        using C.comp-arr-dom cospan' gw'.chine-eq-apex gw'.chine-composite
        by simp
      finally show ?thesis by simp
    qed
  qed
  thus ?thesis
    using Chn-β C.comp-cod-arr gw'.apex-composite by auto
qed
also have ... = p₀[r0, ?u1] · ?RHS
  using RHS 2 C.prj-tuple [of r0 ?u1] by simp
finally show ?thesis by simp
qed
show ru.prj₁ · ?LHS = ru.prj₁ · ?RHS
proof −
  have ru.prj₁ · ?LHS =
      (ru.prj₁ · rϑ'.chine) · Chn a[r, f, w'] · ϱw'.chine · β.chine
    using C.comp-assoc by simp
  also have ... = p₁[r0, fw'.leg1] · Chn a[r, f, w'] · ϱw'.chine · β.chine
    using Chn-rϑ' Chn-rϑ'-eq fw'
        ‹C.commutative-square r0 ?u1 p₁[r0, r0 · ?p1']
          (ϑ'.chine · p₀[r0, r0 · ?p1'])›
    by simp
  also have ... = (rfw'.Prj₁ · rfw'.chine-assoc) · ϱw'.chine · β.chine
    using ide-f ide-w' hseq-rf hseq-char α-ide fw'.leg1-composite C.comp-assoc
    by auto
  also have ... = (rfw'.Prj₁₁ · ϱw'.chine) · β.chine
    using rfw'.prj-chine-assoc C.comp-assoc by simp
  also have ... = ((k1 · p₁[k0, ?w1']) · ϱw'.chine) · β.chine
    using C.comp-cod-arr by simp
  also have ... = (k1 · p₁[k0, ?w1'] · ϱw'.chine) · β.chine
    using C.comp-assoc by simp
  also have ... = (k1 · ϱ.chine · ?p1') · β.chine
    using Chn-ϱw'-eq Dom-ϱ-0 Cod-ϱ-0
        ‹C.commutative-square k0 ?w1' (ϱ.chine · ?p1') ?p0'›
    by simp
  also have ... = (k1 · ϱ.chine) · ?p1' · β.chine
    using C.comp-assoc by metis
  also have ... = (?R · ?p1') · β.chine
    using C.comp-assoc by simp
  also have ... = ?p1' · β.chine
    using C.comp-cod-arr C.prj1-in-hom [of ?R ?w1'] cospan' by simp
```

```
        also have ... = ru.prj₁ · ?RHS
          using RHS 2 by simp
        finally show ?thesis by simp
      qed
    qed
  qed
  finally show ?thesis by simp
qed
```

Now we work on the left-hand side.

```
have L: Chn (T.composite-cell w ϑ) = ⟨?p1 ⟦r0, ?u1⟧ ϑ.chine⟩
proof −
  have Chn (T.composite-cell w ϑ) = rϑ.chine · Chn a[r, f, w] · ϱw.chine
    using Chn-vcomp arfw C.comp-assoc by auto
  moreover have ... = ⟨?p1 ⟦r0, ?u1⟧ ϑ.chine⟩
  proof −
    let ?LHS = rϑ.chine · Chn a[r, f, w] · ϱw.chine
    let ?RHS = ⟨?p1 ⟦r0, ?u1⟧ ϑ.chine⟩
    have 2: C.commutative-square r0 ?u1 ?p1 ϑ.chine by fact
    have LHS: «?LHS : ?R ↓↓ ?w1 →C r0 ↓↓ ?u1»
      using Chn-rϑ Chn-ϱw rfw.chine-assoc-in-hom
      by (metis (no-types, lifting) 8 Chn-in-hom Dom-ϱ-0
          arrow-of-spans-data.simps(2) calculation gw.chine-composite
          rϑ-cod-apex-eq ru.chine-composite)
    have RHS: «?RHS : ?R ↓↓ ?w1 →C r0 ↓↓ ?u1»
      using 2 C.tuple-in-hom [of r0 ?u1 ?p1 ϑ.chine] cospan rϑ-cod-apex-eq
      by simp
    show ?thesis
    proof (intro C.prj-joint-monic [of r0 ?u1 ?LHS ?RHS])
      show C.cospan r0 ?u1
        using ru.legs-form-cospan(1) by blast
      show C.seq ru.prj₁ ?LHS
        using LHS rϑ-cod-apex-eq by auto
      show C.seq ru.prj₁ ?RHS
        using RHS rϑ-cod-apex-eq by auto
      show ru.prj₀ · ?LHS = ru.prj₀ · ?RHS
      proof −
        have ru.prj₀ · ?LHS = (ru.prj₀ · rϑ.chine) · Chn a[r, f, w] · ϱw.chine
          using C.comp-assoc by simp
        also have ... = (ϑ.chine · p₀[r0, f.leg1 · fw.prj₁]) ·
                        Chn a[r, f, w] · ϱw.chine
          using Chn-rϑ-eq Dom-ϑ-1 Cod-ϑ-1 fw.leg1-composite
                ‹C.commutative-square r0 ?u1 p₁[r0, r0 · ?p1]
                  (ϑ.chine · p₀[r0, r0 · ?p1])›
          by simp
        also have ... = ϑ.chine · (p₀[r0, r0 · ?p1] · Chn a[r, f, w]) · ϱw.chine
          using C.comp-assoc by simp
        also have ... = ϑ.chine · ⟨rfw.Prj₀₁ ⟦?R, ?w1⟧ rfw.Prj₀⟩ · ϱw.chine
        proof −
```

```
  have Chn a[r, f, w] = rfw.chine-assoc
    using ide-f ide-w hseq-rf hseq-char α-ide by auto
  moreover have p0[r0, r0 · ?p1] · rfw.chine-assoc =
                ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩
    using rfw.chine-assoc-def
          ‹C.commutative-square r0 (r0 · ?p1) rfw.Prj11
            ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩›
    by simp
  ultimately show ?thesis by simp
qed
also have ... = ϑ.chine · (?R ↓↓ ?w1)
proof −
  have ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine = ?R ↓↓ ?w1
  proof (intro C.prj-joint-monic
                [of ?R ?w1 ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine
                    ?R ↓↓ ?w1])
    show C.cospan ?R ?w1 by fact
    show C.seq ?p1 (⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine)
    proof −
      have C.seq rfw.Prj01 ϱw.chine
        by (meson C.seqI′ Chn-in-hom ϱw rfw.prj-in-hom(2)
            ‹C.commutative-square ?R ?w1 rfw.Prj01 rfw.Prj0›)
      thus ?thesis
        using ‹C.commutative-square ?R ?w1 rfw.Prj01 rfw.Prj0›
        by (metis (no-types) C.comp-assoc C.prj-tuple(2))
    qed
    show C.seq ?p1 (?R ↓↓ ?w1)
      using gw.dom.apex-def gw.leg0-composite gw.prj-in-hom by auto
    show ?p0 · ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine =
          ?p0 · (?R ↓↓ ?w1)
    proof −
      have ?p0 · ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine =
            (?p0 · ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩) · ϱw.chine
        using C.comp-assoc by simp
      also have ... = rfw.Prj0 · ϱw.chine
        using ‹C.commutative-square ?R ?w1 rfw.Prj01 rfw.Prj0› by simp
      also have ... = p0[k0, ?w1] · ⟨ϱ.chine · ?p1 ⟦k0, ?w1⟧ ?p0⟩
        using Chn-ϱw-eq C.comp-cod-arr by simp
      also have ... = ?p0
        using ‹C.commutative-square k0 ?w1 (ϱ.chine · ?p1) ?p0›
              C.prj-tuple(1)
        by blast
      also have ... = ?p0 · (?R ↓↓ ?w1)
        using C.comp-arr-dom gw.chine-eq-apex gw.chine-is-identity
        by (metis C.arr-dom-iff-arr C.pbdom-def Dom-g gw.chine-composite
            gw.chine-simps(1) span-data.select-convs(1))
      finally show ?thesis by simp
    qed
    show ?p1 · ⟨rfw.Prj01 ⟦?R, ?w1⟧ rfw.Prj0⟩ · ϱw.chine =
```

```
        ?p1 · (?R ↓↓ ?w1)
      proof −
        have ?p1 · ⟨rfw.Prj₀₁ ⟦?R, ?w1⟧ rfw.Prj₀⟩ · ϱw.chine =
              (?p1 · ⟨rfw.Prj₀₁ ⟦?R, ?w1⟧ rfw.Prj₀⟩) · ϱw.chine
          using C.comp-assoc by simp
        also have ... = rfw.Prj₀₁ · ϱw.chine
          using ‹C.commutative-square ?R ?w1 rfw.Prj₀₁ rfw.Prj₀› by simp
        also have ... = (k0 · p₁[k0, ?w1]) · ⟨ϱ.chine · ?p1 ⟦k0, ?w1⟧ ?p0⟩
          using Chn-ϱw-eq C.comp-cod-arr by simp
        also have ... = k0 · p₁[k0, ?w1] · ⟨ϱ.chine · ?p1 ⟦k0, ?w1⟧ ?p0⟩
          using C.comp-assoc by simp
        also have ... = k0 · ϱ.chine · ?p1
          using ‹C.commutative-square k0 ?w1 (ϱ.chine · ?p1) ?p0› by simp
        also have ... = (k0 · ϱ.chine) · ?p1
          using C.comp-assoc by metis
        also have ... = ?p1 · (?R ↓↓ ?w1)
          using C.comp-arr-dom C.comp-cod-arr cospan by simp
        finally show ?thesis by blast
      qed
    qed
    thus ?thesis by simp
  qed
  also have ... = ϑ.chine
   using C.comp-arr-dom ϑ.chine-in-hom gw.chine-eq-apex gw.chine-is-identity
         Dom-ϑ-0 Cod-ϑ-0 Dom-ϑ.apex-def Cod-ϑ.apex-def
    by (metis Dom-g ϑ.chine-simps(1) ϑ.chine-simps(2) gw.chine-composite
        gw.dom.apex-def gw.leg0-composite span-data.select-convs(1))
  also have ... = ru.prj₀ · ?RHS
    using 2 by simp
  finally show ?thesis by blast
qed
show ru.prj₁ · ?LHS = ru.prj₁ · ?RHS
proof −
  have ru.prj₁ · ?LHS = (ru.prj₁ · rϑ.chine) · Chn a[r, f, w] · ϱw.chine
    using C.comp-assoc by simp
  also have ... = (r.chine · p₁[r0, r0 · ?p1]) · Chn a[r, f, w] · ϱw.chine
  proof −
    have rϑ.chine ≠ C.null ⟹
          p₁[r.cod.leg0, Cod-ϑ.leg1] · rϑ.chine =
          r.chine · p₁[r0, Dom-ϑ.leg1]
      by (metis (lifting) C.prj-tuple(2) C.tuple-extensionality r.cod-simps(2)
          rϑ.chine-composite)
    thus ?thesis
      using Cod-ϑ-1 Dom-ϑ-1 rϑ.chine-simps(1) fw by fastforce
  qed
  also have ... = r.chine · (rfw.Prj₁ · Chn a[r, f, w]) · ϱw.chine
    using C.comp-assoc fw.leg1-composite by simp
  also have ... = r.chine · rfw.Prj₁₁ · ϱw.chine
    using ide-f ide-w hseq-rf hseq-char α-ide
```

```
        rfw.prj-chine-assoc(1)
      by auto
    also have ... = r.chine · k1 · p1[k0, ?w1] · ϱw.chine
      using C.comp-cod-arr C.comp-assoc by simp
    also have ... = r.chine · k1 · ϱ.chine · p1[Dom-ϱ.leg0, ?w1]
      using Chn-ϱw-eq
        ‹C.commutative-square k0 ?w1
          (ϱ.chine · p1[ra, w.leg1]) p0[ra, w.leg1]›
      by auto
    also have ... = r.chine · (k1 · ϱ.chine) · ?p1
      using C.comp-assoc Dom-ϱ-0 by metis
    also have ... = r.chine · ra · ?p1
      by simp
    also have ... = r.chine · ?p1
      using C.comp-cod-arr
      by (metis C.comp-assoc r.cod-simps(1) r.chine-eq-apex r.chine-simps(1)
          r.chine-simps(3))
    also have ... = ?p1
      using C.comp-cod-arr r.chine-eq-apex r.chine-is-identity
      by (metis 2 C.commutative-squareE r.dom.apex-def)
    also have ... = ru.prj1 · ?RHS
      using 2 by simp
    finally show ?thesis by simp
  qed
 qed
qed
ultimately show ?thesis
  by simp
qed
```

This is the main point: the equation E boils down to the following:

$$?p1' \cdot \beta.chine = ?p1 \wedge \vartheta'.chine \cdot \beta.chine = \vartheta.chine$$

The first equation gets us close to what we need, but we still need $?p1 \cdot C.inv\ ?p0 = ?w1$, which follows from the fact that ?p0 is the pullback of ?R.

```
have ∗: ⟨?p1′ · β.chine ⟦r0, ?u1⟧ ϑ′.chine · β.chine⟩ = ⟨?p1 ⟦r0, ?u1⟧ ϑ.chine⟩
  using L R E by simp
have ∗∗: ?p1′ · β.chine = ?p1
  by (metis ∗ C.in-homE C.not-arr-null C.prj-tuple(2) C.tuple-in-hom
      C.tuple-extensionality
      ‹C.commutative-square r0 u.leg1
        (p1[ra, w′.leg1] · β.chine) (ϑ′.chine · β.chine)›)
have ∗∗∗: ϑ′.chine · β.chine = ϑ.chine
  by (metis ∗ C.prj-tuple(1) ‹C.commutative-square r0 ?u1
      (?p1′ · β.chine) (ϑ′.chine · β.chine)›
      ‹C.commutative-square r0 ?u1 ?p1 ϑ.chine›)
```

CKS say to take $\gamma = \beta$, but obviously this cannot work as literally described, because «$\beta : g \star w \Rightarrow g \star w'$», whereas we must have «$\gamma : w \Rightarrow w'$». Instead, we have to define

γ by transporting β along the projections from *?R* ⇊ *?w1* to *?W* and *?R* ⇊ *?w1'* to *?W'*. These are isomorphisms by virtue of their being pullbacks of identities, but they are not themselves necessarily identities. Specifically, we take *Chn* γ = *?p0'* · *Chn* β · *C.inv ?p0*.

```
let ?γ = (|Chn = ?p0' · β.chine · C.inv ?p0, Dom = Dom w, Cod = Cod w'|)
interpret Dom-γ: span-in-category C ‹Dom ?γ›
  using w.dom.span-in-category-axioms by simp
interpret Cod-γ: span-in-category C ‹Cod ?γ›
  using w'.cod.span-in-category-axioms by simp
```

It has to be shown that γ is an arrow of spans.

```
interpret γ: arrow-of-spans C ?γ
proof
  show «Chn ?γ : Dom-γ.apex →C Cod-γ.apex»
  proof −
    have «Chn β: gw.apex →C gw'.apex»
      using Chn-in-hom β gw'.chine-eq-apex gw.chine-eq-apex by force
    moreover have «?p0' : gw'.apex →C w'.apex»
      using cospan' hseq-gw' hseq-char hcomp-def gw'.dom.apex-def w'.dom.apex-def
      by auto
    moreover have «C.inv ?p0 : w.apex →C gw.apex»
      using cospan hseq-gw hseq-char hcomp-def gw.dom.apex-def w.dom.apex-def
            C.iso-pullback-ide
      by auto
    ultimately show ?thesis
      using Dom-γ.apex-def Cod-γ.apex-def by auto
  qed
```

The commutativity property for the "input leg" follows directly from that for β.

```
show Cod-γ.leg0 · Chn ?γ = Dom-γ.leg0
  using C.comp-assoc C.comp-arr-dom cospan C.iso-pullback-ide C.comp-arr-inv'
  by (metis C.invert-side-of-triangle(2) Dom-β.leg-simps(1) Dom-β-eq β0
      arrow-of-spans-data.select-convs(1,3) arrow-of-spans-data.simps(2)
      r.dom.ide-apex span-data.select-convs(1) w'.cod-simps(2))
```

The commutativity property for the "output leg" is a bit more subtle.

```
show Cod-γ.leg1 · Chn ?γ = Dom-γ.leg1
proof −
  have Cod-γ.leg1 · Chn ?γ = ((?w1' · ?p0') · β.chine) · C.inv ?p0
    using C.comp-assoc by simp
  also have ... = ((?R · ?p1') · Chn β) · C.inv ?p0
    using cospan' C.pullback-commutes [of ?R ?w1'] by auto
  also have ... = (?p1' · β.chine) · C.inv ?p0
    using cospan' C.comp-cod-arr by simp
  also have ... = ?p1 · C.inv ?p0
    using ** by simp
  also have ... = ?w1
```

Sledgehammer found this at a time when I was still struggling to understand what was going on.

```
      by (metis C.comp-cod-arr C.invert-side-of-triangle(2) C.iso-pullback-ide
          C.prj1-simps(1,3) C.pullback-commutes' cospan r.dom.ide-apex
          r.chine-eq-apex r.chine-simps(2))
    also have ... = Dom-γ.leg1 by auto
    finally show ?thesis by simp
  qed
qed
```

What remains to be shown is that γ is unique with the properties asserted by *T2*; *i.e.* «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$. CKS' assertion that the equation $(r\vartheta)(\varrho w) = (r\vartheta')(\varrho w')\beta$ gives $w_1 = w_1{}'$ does not really seem to be true. The reason γ is unique is because it is obtained by transporting β along isomorphisms.

```
have γ: «?γ : w ⇒ w'»
  using γ.arrow-of-spans-axioms arr-char dom-char cod-char by auto
have hseq-fγ: hseq f ?γ
  using γ src-def trg-def arrI fw.composable rf.are-arrows(2) by auto
have hseq-gγ: hseq g ?γ
  using γ src-def trg-def fw.composable gw.are-arrows(1) src-f by auto
interpret fγ: two-composable-arrows-of-spans C prj0 prj1 f ?γ
  using hseq-fγ hseq-char by (unfold-locales, simp)
interpret fγ: arrow-of-spans C ‹f ⋆ ?γ›
  using fγ.composite-is-arrow arr-char by simp
interpret gγ: two-composable-arrows-of-spans C prj0 prj1 g ?γ
  using hseq-gγ hseq-char by (unfold-locales, simp)
interpret gγ: arrow-of-spans C ‹g ⋆ ?γ›
  using gγ.composite-is-arrow arr-char by simp
have Chn-gγ: Chn (g ⋆ ?γ) = ⟨?p1 ⟦?R, ?w1'⟧ ?p0' · β.chine⟩
proof −
  have Chn (g ⋆ ?γ) = ⟨?R · ?p1 ⟦?R, ?w1'⟧ (?p0' · β.chine · C.inv ?p0) · ?p0⟩
      using gγ.chine-composite by simp
  also have ... = ⟨?p1 ⟦?R, ?w1'⟧ (?p0' · β.chine · C.inv ?p0) · ?p0⟩
    using C.comp-cod-arr cospan by simp
  also have ... = ⟨?p1 ⟦?R, ?w1'⟧ ?p0' · β.chine⟩
  proof −
    have (?p0' · β.chine · C.inv ?p0) · ?p0 = ?p0' · β.chine
      using C.comp-assoc C.iso-pullback-ide [of ?R ?w1] C.comp-inv-arr
            C.comp-arr-dom Chn-β
      by (metis C.comp-inv-arr' C.in-homE C.pbdom-def cospan r.dom.ide-apex)
    thus ?thesis by simp
  qed
  ultimately show ?thesis by simp
qed
have Chn-β-eq: β.chine = Chn (g ⋆ ?γ)
  by (metis ** C.span-prj C.tuple-prj Chn-gγ cospan cospan')
have β-eq-gγ: β = g ⋆ ?γ
proof (intro arr-eqI)
  show par β (g ⋆ ?γ)
  proof −
    have «g ⋆ ?γ : g ⋆ w ⇒ g ⋆ w'»
```

```
      using ide-g γ T.leg1-simps(3)
      by (intro hcomp-in-vhom, auto)
    thus ?thesis
      using β by (elim in-homE, auto)
  qed
  show β.chine = Chn (g ⋆ ?γ)
    using Chn-β-eq by simp
qed
moreover have ϑ = ϑ′ · (f ⋆ ?γ)
proof (intro arr-eqI)
  have fγ: «f ⋆ ?γ : f ⋆ w ⇒ f ⋆ w′»
    using γ ide-f by auto
  show par: par ϑ (ϑ′ · (f ⋆ ?γ))
    using ϑ ϑ′ fγ by (elim in-homE, auto)
  show ϑ.chine = Chn (ϑ′ · (f ⋆ ?γ))
    using par ∗∗∗ Chn-vcomp calculation fγ.chine-composite gγ.chine-composite
    by auto
qed
ultimately show 2: «?γ : w ⇒ w′» ∧ β = g ⋆ ?γ ∧ ϑ = ϑ′ · (f ⋆ ?γ)
  using γ by simp
show ⋀γ′. «γ′ : w ⇒ w′» ∧ β = g ⋆ γ′ ∧ ϑ = ϑ′ · (f ⋆ γ′) ⟹ γ′ = ?γ
proof −
  fix γ′
  assume 1: «γ′ : w ⇒ w′» ∧ β = g ⋆ γ′ ∧ ϑ = ϑ′ · (f ⋆ γ′)
  interpret γ′: arrow-of-spans C γ′
    using 1 arr-char by auto
  have hseq-gγ′: ‹hseq g γ′›
    using 1 β by auto
  interpret gγ′: two-composable-arrows-of-spans C prj0 prj1 g γ′
    using hseq-gγ′ hseq-char by unfold-locales auto
  interpret gγ′: arrow-of-spans C ‹g ⋆ γ′›
    using gγ′.composite-is-arrow arr-char by simp
  show γ′ = ?γ
  proof (intro arr-eqI)
    show par: par γ′ ?γ
      using 1 γ by fastforce
    show γ′.chine = γ.chine
    proof −
      have C.commutative-square ?R ?w1′ (g.chine · ?p1) (γ′.chine · ?p0)
      proof
        show C.cospan ?R ?w1′ by fact
        show 3: C.span (g.chine · ?p1) (γ′.chine · ?p0)
        proof (intro conjI)
          show C.seq g.chine ?p1
            using cospan by auto
          show C.seq γ′.chine ?p0
            using cospan 2 par arrow-of-spans-data.simps(1)
                  dom-char in-homE w.chine-eq-apex
            by auto
```

```
    thus C.dom (g.chine · ?p1) = C.dom (γ'.chine · ?p0)
      using g.chine-eq-apex cospan by simp
  qed
  show C.dom ra = C.cod (g.chine · ?p1)
    using cospan by auto
  show ?R · g.chine · ?p1 = ?w1' · γ'.chine · ?p0
  proof −
    have ?w1' · γ'.chine · ?p0 = (?w1' · γ'.chine) · ?p0
      using C.comp-assoc by simp
    moreover have ... = ?w1 · ?p0
      using 1 γ'.leg1-commutes dom-char cod-char by auto
    also have ... = ?R · ?p1
      using cospan C.pullback-commutes [of ra ?w1] by auto
    also have ... = ?R · g.chine · ?p1
      using 3 C.comp-cod-arr g.chine-is-identity g.chine-eq-apex g.dom.apex-def
      by auto
    finally show ?thesis by auto
  qed
qed
have C.commutative-square ?R ?w1' (g.chine · ?p1) (γ.chine · ?p0)
proof
  show C.cospan ?R ?w1' by fact
  show 3: C.span (g.chine · ?p1) (γ.chine · ?p0)
    using cospan γ.chine-in-hom by auto
  show C.dom ?R = C.cod (g.chine · ?p1)
    using cospan by auto
  show ?R · g.chine · ?p1 = ?w1' · γ.chine · ?p0
  proof −
    have ?w1' · γ.chine · ?p0 = (?w1' · γ.chine) · ?p0
      using C.comp-assoc by simp
    moreover have ... = ?w1 · ?p0
      using 1 γ.leg1-commutes dom-char cod-char by auto
    also have ... = ?R · ?p1
      using cospan C.pullback-commutes [of ra ?w1] by auto
    also have ... = ?R · g.chine · ?p1
      using 3 C.comp-cod-arr g.chine-is-identity g.chine-eq-apex g.dom.apex-def
      by auto
    finally show ?thesis by auto
  qed
qed
have γ'.chine · ?p0 = γ.chine · ?p0
proof −
  have γ'.chine · ?p0 = ?p0' · gγ'.chine
    using 1 dom-char cod-char gγ'.chine-composite
          ‹C.commutative-square ?R ?w1' (g.chine · ?p1) (γ'.chine · ?p0)›
    by auto
  also have ... = ?p0' · β.chine
    using 1 by simp
  also have ... = ?p0' · gγ.chine
```

```
                using Chn-β-eq by simp
              also have ... = γ.chine · ?p0
                using gγ.chine-composite
                      ‹C.commutative-square ?R ?w1' (g.chine · ?p1) (γ.chine · ?p0)›
                by simp
              finally show ?thesis by simp
            qed
            thus ?thesis
              using C.iso-pullback-ide C.iso-is-retraction C.retraction-is-epi
                    C.epi-cancel [of ?p0 γ'.chine γ.chine] cospan γ.chine-in-hom
                    γ'.chine-in-hom
              by auto
          qed
        qed
      qed
    qed
  qed
  qed
  qed
  qed
  qed

end

context span-bicategory
begin

  interpretation chosen-right-adjoints vcomp hcomp assoc unit src trg ..
  notation some-right-adjoint (‹-*› [1000] 1000)
  notation isomorphic (infix ‹≅› 50)
```

Span(C) is a bicategory of spans.

```
  lemma is-bicategory-of-spans:
  shows bicategory-of-spans vcomp hcomp assoc unit src trg
  proof
```

Every 1-cell r is isomorphic to the composition of a map and the right adjoint of a map. The proof is to obtain a tabulation of r as a span of maps (f, g) and then observe that r is isomorphic to $g \star f^*$.

```
    show ⋀r. ide r ⟹ ∃f g. is-left-adjoint f ∧ is-left-adjoint g ∧ r ≅ g ⋆ f*
    proof −
      fix r
      assume r: ide r
      interpret r: identity-arrow-of-spans C r
        using r ide-char' by auto
      interpret r: identity-arrow-in-span-bicategory C prj0 prj1 r ..
      have ϱ: tabulation (·) (⋆) assoc unit src trg r r.ϱ r.f r.g ∧
              is-left-adjoint r.f ∧ is-left-adjoint r.g
        using r r.has-tabulation by blast
      interpret ϱ: tabulation vcomp hcomp assoc unit src trg r r.ϱ r.f r.g
```

using ϱ **by** *fast*
have *1*: $r \cong r.g \star r.f^*$
using ϱ ϱ*.yields-isomorphic-representation′* ϱ*.T0.is-map*
left-adjoint-extends-to-adjoint-pair
isomorphic-def [*of* $r.g \star r.f^*$ r] *isomorphic-symmetric*
by *auto*
thus $\exists f\ g.$ *is-left-adjoint* $f \wedge$ *is-left-adjoint* $g \wedge r \cong g \star f^*$
using ϱ **by** *blast*
qed

Every span of maps extends to a tabulation.

show $\bigwedge f\ g.$ $[\![$ *is-left-adjoint* f; *is-left-adjoint* g; *src* f = *src* g $]\!] \Longrightarrow$
$\exists r\ \varrho.$ *tabulation* $(\cdot)$ $(\star)$ *assoc unit src trg* r ϱ f g
proof −

The proof idea is as follows: Let maps $f = (f_1,\ f_0)$ and $g = (g_1,\ g_0)$ be given. Let $f' = (f_1 \cdot C.inv\ f_0,\ C.cod\ f_0)$ and $g' = (g_1 \cdot C.inv\ g_0,\ C.cod\ g_0)$; then f' and g' are maps isomorphic to f and g, respectively. By a previous result, f' and g' extend to a tabulation $(f',\ \tau,\ g')$ of $r = (f_1 \cdot C.inv\ f_0,\ g_1 \cdot C.inv\ g_0)$. Compose with isomorphisms «$\varphi : f' \Rightarrow f$» and «$\psi : g \Rightarrow g'$» to obtain $(f,\ (r \star \varphi) \cdot \tau \cdot \psi,\ g)$ and show it must also be a tabulation.

fix f g
assume f: *is-left-adjoint* f
assume g: *is-left-adjoint* g
assume fg: *src* f = *src* g
show $\exists r\ \varrho.$ *tabulation* $(\cdot)$ $(\star)$ *assoc unit src trg* r ϱ f g
proof −

We have to unpack the hypotheses to get information about f and g.

obtain f_a η_f ε_f
where ff_a: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* f f_a η_f ε_f
using f *adjoint-pair-def* **by** *auto*
interpret ff_a: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* f f_a η_f ε_f
using ff_a **by** *simp*
interpret f: *arrow-of-spans* C f
using *ide-char* [*of* f] **by** *simp*
interpret f: *identity-arrow-of-spans* C f
using *ide-char* [*of* f] **by** *unfold-locales auto*
obtain g_a η_g ε_g
where G: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* g g_a η_g ε_g
using g *adjoint-pair-def* **by** *auto*
interpret gg_a: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* g g_a η_g ε_g
using G **by** *simp*
interpret g: *arrow-of-spans* C g
using *ide-char* [*of* g] **by** *simp*
interpret g: *identity-arrow-of-spans* C g
using *ide-char* [*of* g] **by** *unfold-locales auto*

let *?f′* = *mkIde* $(C.cod\ f.leg0)$ $(f.dom.leg1 \cdot C.inv\ f.leg0)$

have f': *ide ?f'*
proof −
  have *C.span* (*C.cod f.leg0*) (*f.leg1* · *C.inv f.leg0*)
    using *f is-left-adjoint-char* **by** *auto*
  thus *?thesis*
    using *ide-mkIde* **by** *blast*
qed
interpret f': *arrow-of-spans C ?f'*
  using f' *ide-char* **by** *blast*
interpret f': *identity-arrow-of-spans C ?f'*
  using f' *ide-char* **by** *unfold-locales auto*

let *?g'* = *mkIde* (*C.cod g.leg0*) (*g.dom.leg1* · *C.inv g.leg0*)
have g': *ide ?g'*
proof −
  have *C.span* (*C.cod g.leg0*) (*g.leg1* · *C.inv g.leg0*)
    using *g is-left-adjoint-char* **by** *auto*
  thus *?thesis*
    using *ide-mkIde* **by** *blast*
qed
interpret g': *arrow-of-spans C ?g'*
  using g' *ide-char* **by** *blast*
interpret g': *identity-arrow-of-spans C ?g'*
  using g' *ide-char* **by** *unfold-locales auto*

let *?r* = *mkIde* (f'*.leg1*) (g'*.leg1*)
have r: *ide ?r*
proof −
  have *C.span* (f'*.leg1*) (g'*.leg1*)
    using *f g fg src-def is-left-adjoint-char* **by** *simp*
  thus *?thesis*
    using *ide-mkIde* **by** *blast*
qed
interpret r: *arrow-of-spans C ?r*
  using *r ide-char* **by** *blast*
interpret r: *identity-arrow-of-spans C ?r*
  using *r ide-char* **by** *unfold-locales auto*
interpret r: *identity-arrow-in-span-bicategory C prj0 prj1 ?r* **..**

have *r.f* = *?f'*
  using *f r.chine-eq-apex is-left-adjoint-char* **by** *auto*
have *r.g* = *?g'*
  using *f r.chine-eq-apex fg src-def is-left-adjoint-char* **by** *simp*

interpret ϱ: *tabulation* ‹(•)› ‹(⋆)› *assoc unit src trg ?r r.*ϱ *r.f r.g*
  using *r.has-tabulation* **by** *simp*
have ϱ*-eq*: *r.*ϱ = ⦇*Chn* = ⟨*C.cod f.leg0* ⟦f'*.leg1*, f'*.leg1*⟧ *C.cod f.leg0*⟩,
  Dom = ⦇*Leg0* = *C.cod f.leg0*, *Leg1* = g'*.leg1*⦈,
  Cod = ⦇*Leg0* = p_0[f'*.leg1*, f'*.leg1*],

$Leg1 = g'.leg1 \cdot \mathrm{p}_1[f'.leg1, f'.leg1]\rangle\rangle$
using ‹$r.f = ?f'$› **by** *auto*

Obtain the isomorphism from f' to f.

let $?\varphi = (\!|Chn = C.inv\ f.leg0, Dom = Dom\ ?f', Cod = Dom\ f|\!)$
interpret *Dom-φ*: *span-in-category C*
‹$Dom\ (\!|Chn = C.inv\ f.leg0,$
$Dom = Dom\ (mkIde\ f.dsrc\ (f.leg1 \cdot C.inv\ f.leg0)),$
$Cod = Dom\ f|\!)$›
using $f'.dom.span\text{-}in\text{-}category\text{-}axioms$ **by** *simp*
interpret *Cod-φ*: *span-in-category C*
‹$Cod\ (\!|Chn = C.inv\ f.leg0,$
$Dom = Dom\ (mkIde\ f.dsrc\ (f.leg1 \cdot C.inv\ f.leg0)),$
$Cod = Dom\ f|\!)$›
using $f.dom.span\text{-}in\text{-}category\text{-}axioms$ **by** *simp*
interpret *φ*: *arrow-of-spans C ?φ*
proof
show «$Chn\ (\!|Chn = C.inv\ f.leg0,$
$Dom = Dom\ (mkIde\ f.dsrc\ (f.leg1 \cdot C.inv\ f.leg0)),$
$Cod = Dom\ f|\!) : Dom\text{-}\varphi.apex \rightarrow_C Cod\text{-}\varphi.apex$»
using $f\ f.dom.apex\text{-}def\ f'.dom.apex\text{-}def\ is\text{-}left\text{-}adjoint\text{-}char$ **by** *auto*
show $Cod\text{-}\varphi.leg0 \cdot Chn\ (\!|Chn = C.inv\ f.leg0,$
$Dom = Dom\ (mkIde\ f.dsrc\ (f.leg1 \cdot C.inv\ f.leg0)),$
$Cod = Dom\ f|\!) =$
$Dom\text{-}\varphi.leg0$
using $f\ f.dom.apex\text{-}def\ is\text{-}left\text{-}adjoint\text{-}char\ C.comp\text{-}arr\text{-}inv\ C.inv\text{-}is\text{-}inverse$
by *simp*
show $Cod\text{-}\varphi.leg1 \cdot Chn\ (\!|Chn = C.inv\ f.leg0,$
$Dom = Dom\ (mkIde\ f.dsrc\ (f.leg1 \cdot C.inv\ f.leg0)),$
$Cod = Dom\ f|\!) =$
$Dom\text{-}\varphi.leg1$
by *simp*
qed
have *φ*: «$?\varphi : ?f' \Rightarrow f$» $\wedge$ $iso\ ?\varphi$
using $f\ is\text{-}left\text{-}adjoint\text{-}char\ iso\text{-}char\ arr\text{-}char\ dom\text{-}char\ cod\text{-}char$
$\varphi.arrow\text{-}of\text{-}spans\text{-}axioms\ f'.dom.apex\text{-}def\ f.dom.apex\text{-}def$
by *auto*

Obtain the isomorphism from g to g'. Recall: $g' = mkIde\ (C.cod\ g.leg0)\ (g.dom.leg1 \cdot C.inv\ g.leg0)$. The isomorphism is given by $g.leg0$.

let $?\psi = (\!|Chn = g.leg0, Dom = Dom\ g, Cod = Dom\ ?g'|\!)$
interpret *Dom-ψ*: *span-in-category C*
‹$Dom\ (\!|Chn = g.leg0,$
$Dom = Dom\ g,$
$Cod = Dom\ (mkIde\ g.dsrc\ (g.leg1 \cdot C.inv\ g.leg0))|\!)$›
using $g.dom.span\text{-}in\text{-}category\text{-}axioms$ **by** *simp*
interpret *Cod-ψ*: *span-in-category C*
‹$Cod\ (\!|Chn = g.leg0,$
$Dom = Dom\ g,$

```
                    Cod = Dom (mkIde g.dsrc (g.leg1 · C.inv g.leg0))⦇›
    using g′.dom.span-in-category-axioms by simp
  interpret ψ: arrow-of-spans C ?ψ
  proof
    show «Chn ⦇Chn = g.leg0,
              Dom = Dom g,
              Cod = Dom (mkIde g.dsrc (g.leg1 · C.inv g.leg0))⦈ :
                Dom-ψ.apex →C Cod-ψ.apex»
      using g g.dom.apex-def g′.dom.apex-def is-left-adjoint-char by auto
    show Cod-ψ.leg0 · Chn ⦇Chn = g.leg0,
                    Dom = Dom g,
                    Cod = Dom (mkIde g.dsrc (g.leg1 · C.inv g.leg0))⦈ =
        Dom-ψ.leg0
      using C.comp-cod-arr by simp
    show Cod-ψ.leg1 · Chn ⦇Chn = g.leg0,
                    Dom = Dom g,
                    Cod = Dom (mkIde g.dsrc (g.leg1 · C.inv g.leg0))⦈ =
        Dom-ψ.leg1
      using g g.dom.apex-def is-left-adjoint-char C.comp-inv-arr C.inv-is-inverse
          C.comp-assoc C.comp-arr-dom
      by simp
  qed
  have ψ: «?ψ : g ⇒ ?g′» ∧ iso ?ψ
    using g is-left-adjoint-char iso-char arr-char dom-char cod-char
        ψ.arrow-of-spans-axioms g.dom.apex-def g′.dom.apex-def
    by auto
  have ϱψ: tabulation (·) (⋆) assoc unit src trg ?r (r.ϱ · ?ψ) r.f g
    using ψ ‹r.g = ?g′› r.has-tabulation ϱ.preserved-by-output-iso by simp
  interpret τψ: tabulation vcomp hcomp assoc unit src trg ?r ‹r.ϱ · ?ψ› r.f g
    using ϱψ by auto
  have tabulation (·) (⋆) assoc unit src trg ?r ((?r ⋆ ?φ) · r.ϱ · ?ψ) f g
    using φ ‹r.f = ?f′› τψ.preserved-by-input-iso [of ?φ f] by argo
  thus ?thesis by auto
 qed
qed
```

The sub-bicategory of maps is locally essentially discrete.

```
show ⋀f f′ μ μ′. ⟦ is-left-adjoint f; is-left-adjoint f′; «μ : f ⇒ f′»; «μ′ : f ⇒ f′» ⟧
                 ⟹ iso μ ∧ iso μ′ ∧ μ = μ′
proof −
  fix f f′ μ μ′
  assume f: is-left-adjoint f and f′: is-left-adjoint f′
  assume μ: «μ : f ⇒ f′» and μ′: «μ′ : f ⇒ f′»
  obtain fa η ε
    where fa: adjunction-in-bicategory vcomp hcomp assoc unit src trg f fa η ε
    using f adjoint-pair-def by auto
  obtain f′a η′ ε′
    where f′a: adjunction-in-bicategory vcomp hcomp assoc unit src trg f′ f′a η′ ε′
    using f′ adjoint-pair-def adjunction-def by auto
```

interpret f_a: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* $f\ f_a\ \eta\ \varepsilon$
using f_a **by** *simp*
interpret f'_a: *adjunction-in-bicategory vcomp hcomp assoc unit src trg* $f'\ f'_a\ \eta'\ \varepsilon'$
using f'_a **by** *simp*
interpret f: *identity-arrow-of-spans C f*
using *ide-char'* [*of f*] **by** *simp*
interpret f': *identity-arrow-of-spans C f'*
using *ide-char'* [*of f'*] **by** *simp*
interpret μ: *arrow-of-spans C* μ **using** μ *arr-char* **by** *auto*
interpret μ': *arrow-of-spans C* μ' **using** μ' *arr-char* **by** *auto*
have *1*: *C.iso f.leg0* $\wedge$ *C.iso f'.leg0*
using *f f' is-left-adjoint-char* **by** *simp*
have *2*: μ*.chine = C.inv f'.leg0* $\cdot$ *f.leg0*
using μ *1 dom-char cod-char* μ*.leg0-commutes C.invert-side-of-triangle* **by** *auto*
moreover have μ'*.chine = C.inv f'.leg0* $\cdot$ *f.leg0*
using μ' *1 dom-char cod-char* μ'*.leg0-commutes C.invert-side-of-triangle* **by** *auto*
ultimately have *3*: μ*.chine* = μ'*.chine* **by** *simp*
have *iso* μ
using *1 2* μ *C.isos-compose dom-char cod-char iso-char arr-char* **by** *auto*
hence *iso* μ'
using *3 iso-char arr-char* μ'*.arrow-of-spans-axioms* **by** *simp*
moreover have $\mu = \mu'$
using *3* μ μ' *dom-char cod-char* **by** *fastforce*
ultimately show *iso* μ $\wedge$ *iso* μ' $\wedge$ $\mu = \mu'$
by *simp*
qed
qed

We can now prove the easier half of the main result (CKS Theorem 4): If B is biequivalent to *Span*(C), where C is a category with pullbacks, then B is a bicategory of spans. (Well, it is easier given that we have already done the work to show that the notion "bicategory of spans" is respected by equivalence of bicategories.)

theorem *equivalent-implies-bicategory-of-spans*:
assumes *equivalent-bicategories vcomp hcomp assoc unit src trg* V_1 H_1 a_1 i_1 src_1 trg_1
shows *bicategory-of-spans* V_1 H_1 a_1 i_1 src_1 trg_1
using *assms is-bicategory-of-spans bicategory-of-spans-respects-equivalence* **by** *blast*

end

2.3.3 Properties of Bicategories of Spans

We now develop consequences of the axioms for a bicategory of spans, in preparation for proving the other half of the main result.

context *bicategory-of-spans*
begin

notation *isomorphic* (**infix** ‹≅› *50*)

The following is a convenience version of *BS2* that gives us what we generally want:

given specified f, g obtain ϱ that makes (f, ϱ, g) a tabulation of $g \star f^*$, not a tabulation of some r isomorphic to $g \star f^*$.

lemma *BS2′*:
assumes *is-left-adjoint f* **and** *is-left-adjoint g* **and** *src f* = *src g*
and *isomorphic* $(g \star f^*)$ *r*
shows $\exists\, \varrho$. *tabulation V H* a i *src trg r* ϱ *f g*
proof −
  have *1*: *is-left-adjoint f* ∧ *is-left-adjoint g* ∧ $g \star f^* \cong r$
    using *assms BS1* **by** *simp*
  obtain φ **where** φ: «$\varphi : g \star f^* \Rightarrow r$» ∧ *iso* φ
    using *1 isomorphic-def* **by** *blast*
  obtain r' ϱ' **where** ϱ': *tabulation V H* a i *src trg* r' ϱ' *f g*
    using *assms 1 BS2* **by** *blast*
  interpret ϱ': *tabulation V H* a i *src trg* r' ϱ' *f g*
    using ϱ' **by** *simp*
  let $?\psi = \varrho'.T0.trnr_\varepsilon\ r'\ \varrho'$
  have ψ: «$?\psi : g \star f^* \Rightarrow r'$» ∧ *iso* $?\psi$
    using ϱ'*.yields-isomorphic-representation* **by** *blast*
  have «$\varphi \cdot$ *inv* $?\psi : r' \Rightarrow r$» ∧ *iso* $(\varphi \cdot$ *inv* $?\psi)$
    using φ ψ *isos-compose* **by** *blast*
  hence *3*: *tabulation V H* a i *src trg r* $((\varphi \cdot$ *inv* $?\psi \star f) \cdot \varrho')$ *f g*
    using ϱ'*.is-preserved-by-base-iso* **by** *blast*
  hence $\exists\, \varrho$. *tabulation V H* a i *src trg r* ϱ *f g*
    by *blast*
  thus *?thesis*
    using *someI-ex* [*of* $\lambda\varrho$. *tabulation V H* a i *src trg r* ϱ *f g*] **by** *simp*
qed

The following observation is made by CKS near the beginning of the proof of Theorem 4: If w is an arbitrary 1-cell, and g and $g \star w$ are maps, then w is in fact a map. It is applied frequently.

lemma *BS4*:
assumes *is-left-adjoint g* **and** *ide w* **and** *is-left-adjoint* $(g \star w)$
shows *is-left-adjoint w*
proof −

CKS say: "by (i) there are maps m, n with $w \cong nm^*$, so, by (ii), we have two tabulations $(1, \varrho, gw)$, (m, σ, gn) of gw; since tabulations are unique up to equivalence, m is invertible and $w \cong nm^*$ is a map."

  have *ex-*ϱ: $\exists\, \varrho$. *tabulation V H* a i *src trg* $(g \star w)$ ϱ (*src w*) $(g \star w)$
  proof −
    have $(g \star w) \star$ *src w* $\cong g \star w$
      by (*metis assms*(*3*) *iso-runit ideD*(*1*) *isomorphic-def left-adjoint-is-ide*
        runit-in-hom(*2*) *src-hcomp*)
    moreover have $(g \star w) \star (\mathit{src}\ w)^* \cong g \star w$
    proof −
      have $(g \star w) \star$ *src* $(g \star w) \cong g \star w$
        using *calculation isomorphic-implies-ide*(*2*) **by** *auto*

```
    moreover have src (g ⋆ w) ≅ (src w)*
    proof −
      interpret src-w: map-in-bicategory V H a i src trg ‹src w›
        using assms obj-is-self-adjoint by unfold-locales auto
      interpret src-w: adjunction-in-bicategory V H a i src trg
                    ‹src w› ‹(src w)*› src-w.η src-w.ε
        using src-w.is-map left-adjoint-extends-to-adjunction by simp
      have adjoint-pair (src w) (src w)
        using assms obj-is-self-adjoint by simp
      moreover have adjoint-pair (src w) (src w)*
        using adjoint-pair-def src-w.adjunction-in-bicategory-axioms by auto
      ultimately have src w ≅ (src w)*
        using left-adjoint-determines-right-up-to-iso by simp
      moreover have src w = src (g ⋆ w)
        using assms isomorphic-def hcomp-simps(1) left-adjoint-is-ide by simp
      ultimately show ?thesis by simp
    qed
    moreover have src (g ⋆ w) = trg (src (g ⋆ w))
      using assms left-adjoint-is-ide by simp
    ultimately show ?thesis
      using assms left-adjoint-is-ide isomorphic-transitive isomorphic-symmetric
          hcomp-ide-isomorphic
      by blast
  qed
  ultimately show ?thesis
    using assms obj-is-self-adjoint
        left-adjoint-is-ide BS2' [of src w g ⋆ w g ⋆ w]
    by auto
qed
obtain ϱ where ϱ: tabulation V H a i src trg (g ⋆ w) ϱ (src w) (g ⋆ w)
  using ex-ϱ by auto
obtain m n where mn: is-left-adjoint m ∧ is-left-adjoint n ∧ isomorphic w (n ⋆ m*)
  using assms BS1 [of w] by auto
have m_a: adjoint-pair m m* ∧ isomorphic w (n ⋆ m*)
  using mn adjoint-pair-def left-adjoint-extends-to-adjoint-pair by blast
have ex-σ: ∃σ. tabulation V H a i src trg (g ⋆ w) σ m (g ⋆ n)
proof −
  have hseq n m*
    using mn isomorphic-implies-ide by auto
  have trg (n ⋆ m*) = trg w
    using mn m_a isomorphic-def
    by (metis (no-types, lifting) dom-inv in-homE trg-dom trg-inv)
  hence trg n = trg w
    using mn by (metis assms(2) ideD(1) trg.preserves-reflects-arr trg-hcomp)
  hence hseq g n
    using assms mn left-adjoint-is-ide ideD(1)
    by (metis hseq-char)
  have hseq g w
    using assms left-adjoint-is-ide by simp
```

```
    have src m = src n
      using mn m_a ‹hseq n m*› adjoint-pair-antipar [of m m*] by fastforce

    have is-left-adjoint (g ⋆ n)
      using assms mn left-adjoints-compose ‹hseq g n› by blast
    moreover have src m = src (g ⋆ n)
      using assms mn ‹hseq g n› ‹src m = src n› by simp
    moreover have (g ⋆ n) ⋆ m* ≅ g ⋆ w
    proof −
      have 1: src g = trg (n ⋆ m*)
        using assms ‹trg (n ⋆ m*) = trg w› ‹hseq g w› by fastforce
      hence (g ⋆ n) ⋆ m* ≅ g ⋆ n ⋆ m*
        using assms mn m_a assoc-in-hom iso-assoc ‹hseq g n› ‹hseq n m*›
              isomorphic-def left-adjoint-is-ide right-adjoint-is-ide
        by (metis hseqE ideD(2) ideD(3))
      also have ... ≅ g ⋆ w
        using assms 1 mn m_a isomorphic-symmetric hcomp-ide-isomorphic left-adjoint-is-ide
        by simp
      finally show ?thesis
        using isomorphic-transitive by blast
    qed
    ultimately show ?thesis
      using assms mn m_a BS2' by blast
  qed
  obtain σ where σ: tabulation V H a i src trg (g ⋆ w) σ m (g ⋆ n)
    using ex-σ by auto

  interpret ϱ: tabulation V H a i src trg ‹g ⋆ w› ϱ ‹src w› ‹g ⋆ w›
    using ϱ by auto
  interpret σ: tabulation V H a i src trg ‹g ⋆ w› σ m ‹g ⋆ n›
    using σ by auto
```

As usual, the sketch given by CKS seems more suggestive than it is a precise recipe. We can obtain an equivalence map «e : *src* $w \to$ *src* m» and ϑ such that «ϑ : $m \star e \Rightarrow$ *src* w». We can also obtain an equivalence map «e' : *src* $m \to$ *src* w» and ϑ' such that «ϑ' : *src* $w \star e' \Rightarrow m$». If ϑ' can be taken to be an isomorphism; then we have $e' \cong$ *src* $w \star e' \cong m$. Since e' is an equivalence, this shows m is an equivalence, hence its right adjoint m^* is also an equivalence and therefore a map. But $w = n \star m_a$, so this shows that w is a map.

Now, we may assume without loss of generality that e and e' are part of an adjoint equivalence. We have «ϑ : $m \star e \Rightarrow$ *src* w» and «ϑ' : *src* $w \star e' \Rightarrow m$». We may take the transpose of ϑ to obtain «ζ : $m \Rightarrow$ *src* $w \star e'$»; then «$\vartheta' \cdot \zeta$: $m \Rightarrow m$» and «$\zeta \cdot \vartheta'$: *src* $w \star e' \Rightarrow$ *src* $w \star e'$». Since m and *src* $w \star e'$ are maps, by *BS3* it must be that ζ and ϑ' are inverses.

Note: CKS don't cite *BS3* here. I am not sure whether this result can be proved without *BS3*. For example, I am interested in knowing whether it can still be proved under the the assumption that 2-cells between maps are unique, but not necessarily

invertible, or maybe even in a more general situation. It looks like the invertibility part of *BS3* is not used in the proof below.

have *2*: $\exists\, e\ e'\ \varphi\ \psi\ \vartheta\ \nu\ \vartheta'\ \nu'.$
$\quad$ *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* $e\ e'\ \varphi\ \psi\ \wedge$
$\quad$ «ϑ' : *src* $w \star e' \Rightarrow m$» $\wedge$ «$\nu : g \star n \Rightarrow (g \star w) \star e'$» $\wedge$ *iso* $\nu\ \wedge$
$\quad$ $\sigma = \varrho$.*composite-cell* $e'\ \vartheta' \cdot \nu\ \wedge$
$\quad$ «$\vartheta : m \star e \Rightarrow$ *src* w» $\wedge$ «$\nu' : g \star w \Rightarrow (g \star n) \star e$» $\wedge$ *iso* $\nu'\ \wedge$
$\quad$ $\varrho = ((g \star w) \star \vartheta) \cdot \mathrm{a}[g \star w,\ m,\ e] \cdot (\sigma \star e) \cdot \nu'$
using ϱ σ.*apex-unique-up-to-equivalence* [*of* ϱ *src* w $g \star w$] *comp-assoc*
by *metis*
obtain $e\ e'\ \varphi\ \psi\ \vartheta\ \nu\ \vartheta'\ \nu'$
where $*$: *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* $e\ e'\ \varphi\ \psi\ \wedge$
$\quad$ «ϑ' : *src* $w \star e' \Rightarrow m$» $\wedge$ «$\nu : g \star n \Rightarrow (g \star w) \star e'$» $\wedge$ *iso* $\nu\ \wedge$
$\quad$ $\sigma = \varrho$.*composite-cell* $e'\ \vartheta' \cdot \nu\ \wedge$
$\quad$ «$\vartheta : m \star e \Rightarrow$ *src* w» $\wedge$ «$\nu' : g \star w \Rightarrow (g \star n) \star e$» $\wedge$ *iso* $\nu'\ \wedge$
$\quad$ $\varrho = \sigma$.*composite-cell* $e\ \vartheta \cdot \nu'$
using *2 comp-assoc* **by** *auto*
interpret *ee'*: *equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg* $e\ e'\ \varphi\ \psi$
using $*$ **by** *simp*

have *equiv-e*: *equivalence-map* e
using *ee'.equivalence-in-bicategory-axioms equivalence-map-def* **by** *auto*
obtain ψ' **where** ψ': *adjoint-equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* $e\ e'\ \varphi\ \psi'$
using *equivalence-refines-to-adjoint-equivalence* [*of* $e\ e'\ \varphi$]
$\quad$ *ee'.unit-in-hom*(*2*) *ee'.unit-is-iso ee'.antipar equiv-e*
by *auto*
interpret *ee'*: *adjoint-equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg* $e\ e'\ \varphi\ \psi'$
using ψ' **by** *simp*
interpret *e'e*: *adjoint-equivalence-in-bicategory* ‹$(\cdot)$› ‹$(\star)$› a i *src trg* $e'\ e$ ‹*inv* ψ'› ‹*inv* φ›
using $*$ *ee'.dual-adjoint-equivalence* **by** *simp*
have *equiv-e'*: *equivalence-map* e'
using *e'e.equivalence-in-bicategory-axioms equivalence-map-def* **by** *auto*

have *hseq* $m\ e$
using $*$ *ide-dom* [*of* ϑ]
by (*elim conjE in-homE*) *simp*
have *hseq* (*src* w) e'
using $*$ *ide-dom* [*of* ϑ']
by (*elim conjE in-homE*) *simp*

have $e'e.\mathrm{trnr}_\eta\ m\ \vartheta \in \mathit{hom}\ m\ (\mathit{src}\ w \star e')$
proof $-$
have *src* m = *trg* e
using ‹*hseq* $m\ e$› **by** *auto*
moreover have *src* (*src* w) = *trg* e'
using ‹*hseq* (*src* w) e'› **by** *auto*
moreover have *ide* m
using *mn left-adjoint-is-ide* **by** *simp*
moreover have *ide* (*src* w)

```
      using assms by simp
    ultimately show ?thesis
      using ∗ e′e.adjoint-transpose-right(1) by blast
  qed
  hence 3: «e′e.trnr_η m ϑ : m ⇒ src w ⋆ e′»
    by simp
  hence «ϑ′ · e′e.trnr_η m ϑ : m ⇒ m» ∧ «e′e.trnr_η m ϑ · ϑ′ : src w ⋆ e′ ⇒ src w ⋆ e′»
    using ∗ by auto
  moreover have «m : m ⇒ m» ∧ «src w ⋆ e′ : src w ⋆ e′ ⇒ src w ⋆ e′»
    using mn 3 ide-cod [of e′e.trnr_η m ϑ] left-adjoint-is-ide by fastforce
  moreover have 4: is-left-adjoint (src w ⋆ e′)
  proof −
    have is-left-adjoint (src w)
      using assms obj-is-self-adjoint by simp
    moreover have is-left-adjoint e′
      using e′e.adjunction-in-bicategory-axioms adjoint-pair-def by auto
    ultimately show ?thesis
      using left-adjoints-compose ‹hseq (src w) e′› by auto
  qed
  ultimately have ϑ′ · e′e.trnr_η m ϑ = m ∧ e′e.trnr_η m ϑ · ϑ′ = src w ⋆ e′
    using mn BS3 [of m m ϑ′ · e′e.trnr_η m ϑ m]
          BS3 [of src w ⋆ e′ src w ⋆ e′ e′e.trnr_η m ϑ · ϑ′ src w ⋆ e′]
    by auto
  hence inverse-arrows ϑ′ (e′e.trnr_η m ϑ)
    using mn 4 left-adjoint-is-ide inverse-arrows-def by simp
  hence 5: iso ϑ′
    by auto
  have equivalence-map (src w ⋆ e′)
    using assms obj-is-equivalence-map equiv-e′ ‹hseq (src w) e′› equivalence-maps-compose
    by auto
  hence equivalence-map m
    using ∗ 5 equivalence-map-preserved-by-iso isomorphic-def by auto
  hence equivalence-map m*
    using mn m_a right-adjoint-to-equivalence-is-equivalence by simp
  hence is-left-adjoint m*
    using equivalence-is-left-adjoint by simp
  moreover have hseq n m*
    using mn isomorphic-implies-ide by auto
  ultimately have is-left-adjoint (n ⋆ m*)
    using mn left-adjoints-compose by blast
  thus ?thesis
    using mn left-adjoint-preserved-by-iso isomorphic-def isomorphic-symmetric
    by metis
qed

end
```

2.3.4 Choosing Tabulations

context *bicategory-of-spans*
begin

notation *isomorphic* (**infix** ‹≅› *50*)
notation *iso-class* (‹⟦-⟧›)

We will ultimately need to have chosen a specific tabulation for each 1-cell. This has to be done carefully, to avoid unnecessary choices. We start out by using *BS1* to choose a specific factorization of the form $r \cong tab_1\ r \star (tab_0\ r)^*$ for each 1-cell r. This has to be done in such a way that all elements of an isomorphism class are assigned the same factorization.

abbreviation *isomorphic-rep*
where *isomorphic-rep r f g* ≡ *is-left-adjoint f* ∧ *is-left-adjoint g* ∧ $g \star f^* \cong r$

definition tab_0
where $tab_0\ r \equiv SOME\ f.\ \exists g.$ *isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) *f g*

definition tab_1
where $tab_1\ r \equiv SOME\ g.$ *isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) ($tab_0\ r$) *g*

definition *rep*
where *rep r* ≡ *SOME* φ. «$\varphi : tab_1\ r \star (tab_0\ r)^* \Rightarrow r$» ∧ *iso* φ

lemma *rep-props*:
assumes *ide r*
shows «*rep r* : $tab_1\ r \star (tab_0\ r)^* \Rightarrow r$» **and** *iso* (*rep r*)
and $r \cong$ *iso-class-rep* ⟦*r*⟧
and *isomorphic-rep r* ($tab_0\ r$) ($tab_1\ r$)
and $tab_1\ r \star (tab_0\ r)^* \cong r$
proof −
  have *1*: *isomorphic-rep r* ($tab_0\ r$) ($tab_1\ r$)
  proof −
    have ∃*f g. isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) *f g*
      using *assms BS1 isomorphic-symmetric rep-iso-class isomorphic-transitive*
      by *blast*
    hence *isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) ($tab_0\ r$) ($tab_1\ r$)
      using *assms* tab_0*-def* tab_1*-def*
        someI-ex [*of* λ*f*. ∃*g. isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) *f g*]
        someI-ex [*of* λ*g. isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) ($tab_0\ r$) *g*]
      by *simp*
    thus *?thesis*
      using *assms isomorphic-symmetric isomorphic-transitive rep-iso-class* **by** *blast*
  qed
  hence ∃φ. «$\varphi : tab_1\ r \star (tab_0\ r)^* \Rightarrow r$» ∧ *iso* φ
    using *isomorphic-def* **by** *blast*
  hence *2*: «*rep r* : $tab_1\ r \star (tab_0\ r)^* \Rightarrow r$» ∧ *iso* (*rep r*)
    using *someI-ex* [*of* λφ. «$\varphi : tab_1\ r \star (tab_0\ r)^* \Rightarrow r$» ∧ *iso* φ] *rep-def* **by** *auto*

show «*rep r* : tab_1 *r* ⋆ (tab_0 *r*)* ⇒ *r*»
using *2* **by** *simp*
show *iso* (*rep r*)
using *2* **by** *simp*
show *r* ≅ *iso-class-rep* ⟦*r*⟧
using *assms rep-iso-class isomorphic-symmetric* **by** *simp*
thus *isomorphic-rep r* (tab_0 *r*) (tab_1 *r*)
using *1 isomorphic-transitive* **by** *blast*
thus tab_1 *r* ⋆ (tab_0 *r*)* ≅ *r*
by *simp*
qed

lemma tab_0*-in-hom* [*intro*]:
assumes *ide r*
shows «tab_0 *r* : *src* (tab_0 *r*) → *src r*»
and «tab_0 *r* : tab_0 *r* ⇒ tab_0 *r*»
proof −
show «tab_0 *r* : tab_0 *r* ⇒ tab_0 *r*»
using *assms rep-props left-adjoint-is-ide* **by** *auto*
have *trg* (tab_0 *r*) = *src r*
using *assms rep-props*
by (*metis ideD*(*1*) *isomorphic-implies-hpar*(*1*) *isomorphic-implies-hpar*(*3*)
right-adjoint-simps(*2*) *src-hcomp*)
thus «tab_0 *r* : *src* (tab_0 *r*) → *src r*»
using *assms rep-props left-adjoint-is-ide*
by (*intro in-hhomI*, *auto*)
qed

lemma tab_0*-simps* [*simp*]:
assumes *ide r*
shows *ide* (tab_0 *r*)
and *is-left-adjoint* (tab_0 *r*)
and *trg* (tab_0 *r*) = *src r*
and *dom* (tab_0 *r*) = tab_0 *r* **and** *cod* (tab_0 *r*) = tab_0 *r*
using *assms* tab_0*-in-hom rep-props ide-dom left-adjoint-is-ide* **by** *auto*

lemma tab_1*-in-hom* [*intro*]:
assumes *ide r*
shows «tab_1 *r* : *src* (tab_0 *r*) → *trg r*»
and «tab_1 *r* : tab_1 *r* ⇒ tab_1 *r*»
proof −
show «tab_1 *r* : tab_1 *r* ⇒ tab_1 *r*»
using *assms rep-props left-adjoint-is-ide* **by** *auto*
have *trg* (tab_1 *r*) = *trg r*
using *assms rep-props*
by (*metis ideD*(*1*) *isomorphic-implies-hpar*(*1*) *isomorphic-implies-hpar*(*4*) *trg-hcomp*)
moreover have *src* (tab_0 *r*) = *src* (tab_1 *r*)
using *assms rep-props* **by** *fastforce*
ultimately show «tab_1 *r* : *src* (tab_0 *r*) → *trg r*»

using *assms rep-props left-adjoint-is-ide*
by (*intro in-hhomI*, *auto*)
qed

lemma tab_1*-simps* [*simp*]:
assumes *ide r*
shows *ide* (tab_1 *r*)
and *is-left-adjoint* (tab_1 *r*)
and *src* (tab_1 *r*) = *src* (tab_0 *r*) **and** *trg* (tab_1 *r*) = *trg r*
and *dom* (tab_1 *r*) = tab_1 *r* **and** *cod* (tab_1 *r*) = tab_1 *r*
using *assms* tab_1*-in-hom rep-props ide-dom left-adjoint-is-ide* **by** *auto*

lemma *rep-in-hom* [*intro*]:
assumes *ide r*
shows «*rep r* : *src r* → *trg r*»
and «*rep r* : tab_1 *r* ⋆ (tab_0 *r*)* ⇒ *r*»
proof −
show «*rep r* : tab_1 *r* ⋆ (tab_0 *r*)* ⇒ *r*»
using *assms rep-props* **by** *auto*
thus «*rep r* : *src r* → *trg r*»
using *arrI vconn-implies-hpar*(*1*−*4*) **by** *force*
qed

lemma *rep-simps* [*simp*]:
assumes *ide r*
shows *arr* (*rep r*)
and *src* (*rep r*) = *src r* **and** *trg* (*rep r*) = *trg r*
and *dom* (*rep r*) = tab_1 *r* ⋆ (tab_0 *r*)* **and** *cod* (*rep r*) = *r*
using *assms rep-in-hom* **by** *auto*

lemma *iso-rep*:
assumes *ide r*
shows *iso* (*rep r*)
using *assms rep-props* **by** *simp*

end

Next, we assign a specific tabulation to each 1-cell r. We can't just do this any old way if we ultimately expect to obtain a mapping that is functorial with respect to vertical composition. What we have to do is to assign the representative tab_1 *r* ⋆ (tab_0 *r*)* its canonical tabulation, obtained as the adjoint transpose of the identity, and then translate this to a tabulation of *r* via the chosen isomorphism «*rep r* : tab_1 *r* ⋆ (tab_0 *r*)* ⇒ *r*».

locale *identity-in-bicategory-of-spans* =
bicategory-of-spans +
fixes *r* :: *'a*
assumes *is-ide*: *ide r*
begin

interpretation tab_0: *map-in-bicategory V H* a i *src trg* ‹tab_0 *r*›

using *is-ide rep-props* **by** *unfold-locales auto*
interpretation tab_1: *map-in-bicategory V H* a i *src trg* ‹tab_1 *r*›
using *is-ide rep-props* **by** *unfold-locales auto*

A tabulation (tab_0 *r*, *tab*, tab_1 *r*) of *r* can be obtained as the adjoint transpose of the isomorphism «*rep r* : (tab_1 *r*) ⋆ (tab_0 *r*)* ⇒ *r*». It is essential to define it this way if we expect the mapping from 2-cells of the underlying bicategory to arrows of spans to preserve vertical composition.

definition *tab*
where *tab* ≡ $tab_0.trnr_\eta$ (tab_1 *r*) (*rep r*)

In view of *BS2′*, the 1-cell (tab_1 *r*) ⋆ (tab_0 *r*)* has the canonical tabulation obtained via adjoint transpose of an identity. In fact, this tabulation generates the chosen tabulation of *r* in the same isomorphism class by translation along the isomorphism «*rep r* : (tab_1 *r*) ⋆ (tab_0 *r*)* ⇒ *r*». This fact is used to show that the mapping from 2-cells to arrows of spans preserves identities.

lemma *canonical-tabulation*:
shows *tabulation V H* a i *src trg*
((tab_1 *r*) ⋆ (tab_0 *r*)*) ($tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)) (tab_0 *r*) (tab_1 *r*)
proof −
have ∃ϱ. *tabulation V H* a i *src trg* ((tab_1 *r*) ⋆ (tab_0 *r*)*) ϱ (tab_0 *r*) (tab_1 *r*)
by (*simp add*: *bicategory-of-spans.BS2′ bicategory-of-spans-axioms is-ide isomorphic-reflexive*)
thus *?thesis*
using *is-ide* tab_0*.canonical-tabulation* **by** *simp*
qed

lemma *tab-def-alt*:
shows *tab* = (*rep r* ⋆ tab_0 *r*) · $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)
and (*inv* (*rep r*) ⋆ tab_0 *r*) · *tab* = $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)
proof −
have *tab* = $tab_0.trnr_\eta$ (tab_1 *r*) (*rep r* · ((tab_1 *r*) ⋆ (tab_0 *r*)*))
using *tab-def is-ide rep-in-hom* [*of r*] *comp-arr-dom* **by** *auto*
also have ... = (*rep r* ⋆ tab_0 *r*) · $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)
using *is-ide* $tab_0.trnr_\eta$*-comp* **by** *auto*
finally show *1*: *tab* = (*rep r* ⋆ tab_0 *r*) · $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*) **by** *simp*
have (*inv* (*rep r*) ⋆ tab_0 *r*) · *tab* =
((*inv* (*rep r*) ⋆ tab_0 *r*) · (*rep r* ⋆ tab_0 *r*)) · $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)
unfolding *1* **using** *comp-assoc* **by** *presburger*
also have ... = $tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*)
proof −
have *1*: (*inv* (*rep r*) ⋆ tab_0 *r*) · (*rep r* ⋆ tab_0 *r*) = ((tab_1 *r*) ⋆ (tab_0 *r*)*) ⋆ tab_0 *r*
using *whisker-right* [*of* tab_0 *r inv* (*rep r*) *rep r*] *iso-rep rep-in-hom inv-is-inverse comp-inv-arr*
by (*simp add*: *comp-inv-arr′ is-ide*)
show *?thesis*
proof −
have «$tab_0.trnr_\eta$ (tab_1 *r*) ((tab_1 *r*) ⋆ (tab_0 *r*)*) :
tab_1 *r* ⇒ (tab_1 *r* ⋆ (tab_0 *r*)*) ⋆ tab_0 *r*»

by (*meson canonical-tabulation tabulation-data.tab-in-hom*(*2*) *tabulation-def*)
hence $((tab_1\ r \star (tab_0\ r)^*) \star tab_0\ r) \cdot tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*) =$
$tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*)$
using *1 comp-cod-arr* **by** *blast*
thus *?thesis*
using *1* **by** *simp*
qed
qed
finally show $(inv\ (rep\ r) \star tab_0\ r) \cdot tab = tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*)$
by *blast*
qed

lemma *tab-is-tabulation*:
shows *tabulation V H* a i *src trg r tab* $(tab_0\ r)$ $(tab_1\ r)$
by (*metis bicategory-of-spans.iso-rep bicategory-of-spans.rep-in-hom*(*2*)
bicategory-of-spans-axioms is-ide canonical-tabulation tab-def-alt(*1*)
tabulation.is-preserved-by-base-iso)

lemma *tab-in-hom* [*intro*]:
shows «$tab : src\ (tab_0\ r) \rightarrow trg\ r$»
and «$tab : tab_1\ r \Rightarrow r \star tab_0\ r$»
proof −
interpret *tab*: *tabulation V H* a i *src trg r tab* ‹$tab_0\ r$› ‹$tab_1\ r$›
using *tab-is-tabulation* **by** *simp*
show «$tab : src\ (tab_0\ r) \rightarrow trg\ r$»
using *tab.tab-in-hom* **by** *auto*
show «$tab : tab_1\ r \Rightarrow r \star tab_0\ r$»
using *tab.tab-in-hom* **by** *auto*
qed

lemma *tab-simps* [*simp*]:
shows *arr tab*
and $src\ tab = src\ (tab_0\ r)$ **and** $trg\ tab = trg\ r$
and $dom\ tab = tab_1\ r$ **and** $cod\ tab = r \star tab_0\ r$
using *tab-in-hom* **by** *auto*

end

The following makes the chosen tabulation conveniently available whenever we are considering a particular 1-cell.

sublocale *identity-in-bicategory-of-spans* ⊆ *tabulation V H* a i *src trg r tab* ‹$tab_0\ r$› ‹$tab_1\ r$›
using *is-ide tab-is-tabulation* **by** *simp*

context *identity-in-bicategory-of-spans*
begin

interpretation tab_0: *map-in-bicategory V H* a i *src trg* ‹$tab_0\ r$›

using *is-ide rep-props* **by** *unfold-locales auto*
interpretation tab_1: *map-in-bicategory V H* a i *src trg* ‹tab_1 *r*›
using *is-ide rep-props* **by** *unfold-locales auto*

The fact that adjoint transpose is a bijection allows us to invert the definition of *tab* in terms of *rep* to express rep in terms of tab.

lemma *rep-in-terms-of-tab*:
shows *rep r* = *T0.trnr*$_\varepsilon$ *r tab*
using *is-ide T0.adjoint-transpose-right*(*3*) [*of r* tab_1 *r rep r*] *tab-def*
by *fastforce*

lemma *isomorphic-implies-same-tab*:
assumes *isomorphic r r′*
shows tab_0 *r* = tab_0 *r′* **and** tab_1 *r* = tab_1 *r′*
using *assms* tab_0*-def* tab_1*-def iso-class-eqI* **by** *auto*

"Every 1-cell has a tabulation as a span of maps." Has a nice simple ring to it, but maybe not so useful for us, since we generally really need to know that the tabulation has a specific form.

lemma *has-tabulation*:
shows ∃ *ϱ f g. is-left-adjoint f* ∧ *is-left-adjoint g* ∧ *tabulation V H* a i *src trg r ϱ f g*
using *is-ide tab-is-tabulation rep-props* **by** *blast*

end

2.3.5 Tabulations in a Bicategory of Spans

context *bicategory-of-spans*
begin

abbreviation *tab-of-ide*
where *tab-of-ide r* ≡ *identity-in-bicategory-of-spans.tab V H* a i *src trg r*

abbreviation prj_0
where prj_0 *h k* ≡ tab_0 (k^* ⋆ *h*)

abbreviation prj_1
where prj_1 *h k* ≡ tab_1 (k^* ⋆ *h*)

lemma *prj-in-hom* [*intro*]:
assumes *ide h* **and** *is-left-adjoint k* **and** *trg h* = *trg k*
shows «prj_0 *h k* : *src* (prj_0 *h k*) → *src h*»
and «prj_1 *h k* : *src* (prj_0 *h k*) → *src k*»
and «prj_0 *h k* : prj_0 *h k* ⇒ prj_0 *h k*»
and «prj_1 *h k* : prj_1 *h k* ⇒ prj_1 *h k*»
by (*intro in-hhomI*, *auto simp add*: *assms*(*1*−*3*))

lemma *prj-simps* [*simp*]:
assumes *ide h* **and** *is-left-adjoint k* **and** *trg h* = *trg k*

shows *trg* (prj_0 *h k*) = *src h*
and *src* (prj_1 *h k*) = *src* (prj_0 *h k*) **and** *trg* (prj_1 *h k*) = *src k*
and *dom* (prj_0 *h k*) = prj_0 *h k* **and** *cod* (prj_0 *h k*) = prj_0 *h k*
and *dom* (prj_1 *h k*) = prj_1 *h k* **and** *cod* (prj_1 *h k*) = prj_1 *h k*
and *is-left-adjoint* (prj_0 *h k*) **and** *is-left-adjoint* (prj_1 *h k*)
using *assms prj-in-hom* **by** *auto*

end

Many of the commutativity conditions that we would otherwise have to worry about when working with tabulations in a bicategory of spans reduce to trivialities. The following locales try to exploit this to make our life more manageable.

locale *span-of-maps* =
bicategory-of-spans +
fixes leg_0 :: $'a$
and leg_1 :: $'a$
assumes *leg0-is-map*: *is-left-adjoint* leg_0
and *leg1-is-map* : *is-left-adjoint* leg_1

The purpose of the somewhat strange-looking assumptions in this locale is to cater to the form of data that we obtain from *T1*. Under the assumption that we are in a bicategory of spans and that the legs of r and s are maps, the hypothesized 2-cells will be uniquely determined isomorphisms, and an arrow of spans w from r to s will be a map. We want to prove this once and for all under the weakest assumptions we can manage.

locale *arrow-of-spans-of-maps* =
bicategory-of-spans V H a i *src trg* +
r: *span-of-maps V H* a i *src trg* r_0 r_1 +
s: *span-of-maps V H* a i *src trg* s_0 s_1
for *V* :: $'a$ *comp* (**infixr** ‹·› *55*)
and *H* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r_0 :: $'a$
and r_1 :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$
and w :: $'a$ +
assumes *is-ide*: *ide w*
and *leg0-lax*: $\exists \vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$»
and *leg1-iso*: $\exists \nu.$ «$\nu : r_1 \Rightarrow s_1 \star w$» $\wedge$ *iso* ν
begin

notation *isomorphic* (**infix** ‹≅› *50*)

lemma *composite-leg1-is-map*:
shows *is-left-adjoint* ($s_1 \star w$)
using *r.leg1-is-map leg1-iso left-adjoint-preserved-by-iso′ isomorphic-def*

isomorphic-symmetric
by *auto*

lemma *is-map*:
shows *is-left-adjoint w*
using *is-ide composite-leg1-is-map s.leg1-is-map BS4* [*of* s_1 *w*] **by** *auto*

lemma *hseq-leg*$_0$:
shows *hseq* s_0 *w*
by (*metis ideD*(*1*) *ide-dom in-homE leg0-lax*)

lemma *composite-with-leg0-is-map*:
shows *is-left-adjoint* ($s_0 \star w$)
using *left-adjoints-compose is-map s.leg0-is-map hseq-leg*$_0$ **by** *blast*

lemma *leg0-uniquely-isomorphic*:
shows $s_0 \star w \cong r_0$
and $\exists!\vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$»
proof −
show *1*: $s_0 \star w \cong r_0$
using *leg0-lax composite-with-leg0-is-map r.leg0-is-map BS3* [*of* $s_0 \star w\ r_0$]
isomorphic-def
by *auto*
have $\exists\vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$» $\wedge$ *iso* ϑ
using *1 isomorphic-def* **by** *simp*
moreover have $\bigwedge\vartheta\ \vartheta'.$ «$\vartheta : s_0 \star w \Rightarrow r_0$» $\Longrightarrow$ «$\vartheta' : s_0 \star w \Rightarrow r_0$» $\Longrightarrow \vartheta = \vartheta'$
using *BS3 r.leg0-is-map composite-with-leg0-is-map* **by** *blast*
ultimately show $\exists!\vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$» **by** *blast*
qed

lemma *leg1-uniquely-isomorphic*:
shows $r_1 \cong s_1 \star w$
and $\exists!\nu.$ «$\nu : r_1 \Rightarrow s_1 \star w$»
proof −
show *1*: $r_1 \cong s_1 \star w$
using *leg1-iso isomorphic-def* **by** *auto*
have $\exists\nu.$ «$\nu : r_1 \Rightarrow s_1 \star w$» $\wedge$ *iso* ν
using *leg1-iso isomorphic-def isomorphic-symmetric* **by** *simp*
moreover have $\bigwedge\nu\ \nu'.$ «$\nu : r_1 \Rightarrow s_1 \star w$» $\Longrightarrow$ «$\nu' : r_1 \Rightarrow s_1 \star w$» $\Longrightarrow \nu = \nu'$
using *BS3 r.leg1-is-map composite-leg1-is-map* **by** *blast*
ultimately show $\exists!\nu.$ «$\nu : r_1 \Rightarrow s_1 \star w$» **by** *blast*
qed

definition *the-*ϑ
where *the-*$\vartheta \equiv$ *THE* $\vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$»

definition *the-*ν
where *the-*$\nu \equiv$ *THE* $\nu.$ «$\nu : r_1 \Rightarrow s_1 \star w$»

lemma *the-ϑ-props*:
shows «*the-ϑ* : $s_0 \star w \Rightarrow r_0$» **and** *iso the-ϑ*
proof −
  show «*the-ϑ* : $s_0 \star w \Rightarrow r_0$»
    unfolding *the-ϑ-def*
    using *the1I2* [*of* $\lambda\vartheta$. «ϑ : $s_0 \star w \Rightarrow r_0$» $\lambda\vartheta$. «ϑ : $s_0 \star w \Rightarrow r_0$»]
      leg0-uniquely-isomorphic
    by *simp*
  thus *iso the-ϑ*
    using *BS3 r.leg0-is-map composite-with-leg0-is-map* **by** *simp*
qed

lemma *the-ϑ-in-hom* [*intro*]:
shows «*the-ϑ* : *src* $r_0 \rightarrow$ *trg* r_0»
and «*the-ϑ* : $s_0 \star w \Rightarrow r_0$»
  using *the-ϑ-props* **apply** *auto*
by (*metis cod-trg in-hhom-def in-homE isomorphic-implies-hpar*(*3*) *leg0-uniquely-isomorphic*(*1*)
    src-dom trg.preserves-cod)

lemma *the-ϑ-simps* [*simp*]:
shows *arr the-ϑ*
and *src the-ϑ* = *src* r_0 **and** *trg the-ϑ* = *trg* r_0
and *dom the-ϑ* = $s_0 \star w$ **and** *cod the-ϑ* = r_0
  using *the-ϑ-in-hom* **by** *auto*

lemma *the-ν-props*:
shows «*the-ν* : $r_1 \Rightarrow s_1 \star w$» **and** *iso the-ν*
proof −
  show «*the-ν* : $r_1 \Rightarrow s_1 \star w$»
    unfolding *the-ν-def*
    using *the1I2* [*of* $\lambda\nu$. «ν : $r_1 \Rightarrow s_1 \star w$» $\lambda\nu$. «ν : $r_1 \Rightarrow s_1 \star w$»]
      leg1-uniquely-isomorphic
    by *simp*
  thus *iso the-ν*
    using *BS3 r.leg1-is-map composite-leg1-is-map* **by** *simp*
qed

lemma *the-ν-in-hom* [*intro*]:
shows «*the-ν* : *src* $r_1 \rightarrow$ *trg* r_1»
and «*the-ν* : $r_1 \Rightarrow s_1 \star w$»
  using *the-ν-props* **apply** *auto*
  by (*metis in-hhom-def in-homE isomorphic-implies-hpar*(*3*) *leg1-uniquely-isomorphic*(*1*)
    src-cod trg-dom)

lemma *the-ν-simps* [*simp*]:
shows *arr the-ν*
and *src the-ν* = *src* r_1 **and** *trg the-ν* = *trg* r_1
and *dom the-ν* = r_1 **and** *cod the-ν* = $s_1 \star w$
  using *the-ν-in-hom* **by** *auto*

end

locale *arrow-of-spans-of-maps-to-tabulation-data* =
bicategory-of-spans V H a i *src trg* +
arrow-of-spans-of-maps V H a i *src trg* r_0 r_1 s_0 s_1 w +
σ: *tabulation-data* V H a i *src trg* s σ s_0 s_1
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r_0 :: $'a$
and r_1 :: $'a$
and s :: $'a$
and σ :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$
and w :: $'a$

The following declaration allows us to inherit the rules and other facts defined in locale $uw\vartheta$. It is tedious to prove very much without these in place.

sublocale *arrow-of-spans-of-maps-to-tabulation-data* $\subseteq$ $uw\vartheta$ V H a i *src trg* s σ s_0 s_1 r_0 w *the-*ϑ
using σ.*tab-in-hom is-ide the-*ϑ*-props* **by** *unfold-locales auto*

locale *arrow-of-spans-of-maps-to-tabulation* =
arrow-of-spans-of-maps-to-tabulation-data +
tabulation V H a i *src trg* s σ s_0 s_1

locale *tabulation-in-maps* =
span-of-maps V H a i *src trg* s_0 s_1 +
tabulation V H a i *src trg* s σ s_0 s_1
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and s :: $'a$
and σ :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$

sublocale *tabulation-in-maps* $\subseteq$ *tabulation* V H a i *src trg* s σ s_0 s_1 **..**

sublocale *identity-in-bicategory-of-spans* ⊆
tabulation-in-maps V H a i *src trg r tab* ‹tab_0 *r*› ‹tab_1 *r*›
using *is-ide rep-props* **by** *unfold-locales auto*

locale *cospan-of-maps-in-bicategory-of-spans* =
bicategory-of-spans +
fixes *h* :: *'a*
and *k* :: *'a*
assumes *h-is-map*: *is-left-adjoint h*
and *k-is-map*: *is-left-adjoint k*
and *cospan*: *trg h* = *trg k*
begin

The following sublocale declaration is perhaps pushing the limits of sensibility, but the purpose is, given a cospan of maps (h, k), to obtain ready access to the composite $k^* \star h$ and its chosen tabulation.

sublocale *identity-in-bicategory-of-spans V H* a i *src trg* ‹$k^* \star h$›
using *h-is-map k-is-map cospan left-adjoint-is-ide*
by *unfold-locales auto*

notation *isomorphic* (**infix** ‹≅› *50*)

interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
notation *E.eval* (‹⦃-⦄›)

interpretation *h*: *map-in-bicategory V H* a i *src trg h*
using *h-is-map* **by** *unfold-locales auto*
interpretation *k*: *map-in-bicategory V H* a i *src trg k*
using *k-is-map* **by** *unfold-locales auto*

Our goal here is to reformulate the biuniversal properties of the chosen tabulation of $k^* \star h$ in terms of its transpose, which yields a 2-cell from $k \star tab_1$ ($k^* \star h$) to $h \star tab_0$ ($k^* \star h$). These results do not depend on *BS3*.

abbreviation p_0
where $p_0 \equiv prj_0\ h\ k$

abbreviation p_1
where $p_1 \equiv prj_1\ h\ k$

lemma p_0*-in-hom* [*intro*]:
shows «p_0 : *src* p_0 → *src h*»
by *auto*

lemma p_1*-in-hom* [*intro*]:
shows «p_1 : *src* p_0 → *src k*»
using *prj-in-hom cospan h.ide-left k-is-map* **by** *blast*

lemma p_0*-simps* [*simp*]:
shows *trg* p_0 = *src h*

by *simp*

lemma p_1*-simps* [*simp*]:
shows $trg\ p_1 = src\ k$
using *k.antipar*(*1*) **by** *auto*

definition φ
where $\varphi \equiv k.trnl_\varepsilon\ (h \star p_0)\ (\mathrm{a}[k^*,\ h,\ p_0] \cdot tab)$

lemma φ*-in-hom* [*intro*]:
shows «$\varphi : src\ p_0 \rightarrow trg\ h$»
and «$\varphi : k \star p_1 \Rightarrow h \star p_0$»
proof −
show *1*: «$\varphi : k \star p_1 \Rightarrow h \star p_0$»
unfolding φ*-def*
using *k.antipar cospan k.adjoint-transpose-left*(*2*) [*of* $h \star p_0\ p_1$]
by *fastforce*
show «$\varphi : src\ p_0 \rightarrow trg\ h$»
using *1 k.antipar arrI cospan vconn-implies-hpar*(*1*−*2*) **by** *force*
qed

lemma φ*-simps* [*simp*]:
shows $arr\ \varphi$
and $src\ \varphi = src\ p_0$ **and** $trg\ \varphi = trg\ h$
and $dom\ \varphi = k \star p_1$ **and** $cod\ \varphi = h \star p_0$
using φ*-in-hom* **by** *auto*

lemma *transpose-*φ:
shows $tab = \mathrm{a}^{-1}[k^*,\ h,\ p_0] \cdot k.trnl_\eta\ p_1\ \varphi$
proof −
have $\mathrm{a}^{-1}[k^*,\ h,\ p_0] \cdot k.trnl_\eta\ p_1\ \varphi = \mathrm{a}^{-1}[k^*,\ h,\ p_0] \cdot \mathrm{a}[k^*,\ h,\ p_0] \cdot tab$
unfolding φ*-def*
using *k.antipar cospan*
k.adjoint-transpose-left(*4*)
[*of* $h \star p_0\ p_1\ \mathrm{a}[k^*,\ h,\ p_0] \cdot tab$]
by *fastforce*
also have ... $= (\mathrm{a}^{-1}[k^*,\ h,\ p_0] \cdot \mathrm{a}[k^*,\ h,\ p_0]) \cdot tab$
using *comp-assoc* **by** *presburger*
also have ... $= tab$
using *k.antipar cospan comp-cod-arr comp-assoc-assoc′* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *transpose-triangle*:
assumes *ide w*
and «$\vartheta : p_0 \star w \Rightarrow u$» **and** «$\nu : v \Rightarrow p_1 \star w$»
shows $k.trnl_\varepsilon\ (h \star u)\ (\mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w) \cdot \nu) =$
$(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu)$
proof −

have u: *ide u*
  using *assms*(*2*) **by** *auto*
have v: *ide v*
  using *assms*(*3*) **by** *auto*
have *0*: *src* p_0 = *trg w*
  by (*metis assms*(*2*) *hseqE ideD*(*1*) *src.preserves-reflects-arr u vconn-implies-hpar*(*3*))
have *1*: *src h* = *trg u*
  using *assms*(*1*−*2*) *0 trg-dom trg-cod vconn-implies-hpar*(*4*) **by** *auto*
have *2*: *src k* = *trg v*
  using *assms*(*1*,*3*) *0 trg-dom trg-cod hseqI′*
 by (*metis ideD*(*1*) *leg1-simps*(*2*) *leg1-simps*(*3*) p_1*-simps trg-hcomp vconn-implies-hpar*(*4*))
have *3*: *src u* = *src v* ∧ *src u* = *src w*
  using *assms 0 k.antipar src-dom src-cod hseqI′*
  by (*metis ideD*(*1*) *leg0-simps*(*2*) *leg1-simps*(*2*) *leg1-simps*(*3*) *src-hcomp*
    vconn-implies-hpar(*3*))
have *4*: *src h* = *trg* ϑ
  using *assms 1 k.antipar* **by** *auto*
define χ **where** χ = a[k^*, h, $p_0 \star w$] · a[$k^* \star h$, p_0, w] · (*tab* ⋆ *w*)
have χ: «$\chi : p_1 \star w \Rightarrow k^* \star h \star p_0 \star w$»
  unfolding χ*-def*
  using *assms 0 k.antipar cospan* **by** (*intro comp-in-homI*, *auto*)
have $k.trnl_\varepsilon$ ($h \star u$) (a[k^*, h, u] · (($k^* \star h$) ⋆ ϑ) · a[$k^* \star h$, p_0, w] · (*tab* ⋆ *w*) · ν) =
    $k.trnl_\varepsilon$ ($h \star u$) (($k^* \star h \star \vartheta$) · χ · ν)
  unfolding χ*-def*
  using *assms 1 k.antipar cospan assoc-naturality* [*of* k^* h ϑ] *comp-assoc*
  by (*metis 4 h.ide-left ide-char in-homE k.ide-right*)
also have ... = $k.trnl_\varepsilon$ ($h \star u$) ($k^* \star h \star \vartheta$) · ($k \star \chi \cdot \nu$)
proof −
  have *ide* ($h \star u$)
    using *1 u assms h.ide-left* **by** *blast*
  moreover have *seq* ($k^* \star h \star \vartheta$) ($\chi \cdot \nu$)
    using *assms 1 k.antipar cospan* χ *seqI′*
    apply (*intro seqI*)
          apply *auto*
      apply *blast*
  proof −
    have *dom* ($k^* \star h \star \vartheta$) = $k^* \star h \star p_0 \star w$
      using *assms*
      by (*metis 4 cospan hcomp-simps*(*2*−*3*) *h.ide-left hseqI′ ide-char in-homE*
        k.antipar(*2*) *k.ide-right*)
    also have ... = *cod* χ
      using χ **by** *auto*
    finally show *dom* ($k^* \star h \star \vartheta$) = *cod* χ **by** *simp*
  qed
  moreover have *src k* = *trg* ($k^* \star h \star \vartheta$)
    using *assms k.antipar cospan calculation*(*2*) **by** *auto*
  ultimately show *?thesis*
    using $k.trnl_\varepsilon$*-comp* **by** *simp*
qed

also have ... $= k.trnl_\varepsilon\ (h \star u)\ (k^* \star h \star \vartheta) \cdot (k \star \chi) \cdot (k \star \nu)$
  using *assms u χ whisker-left*
  by (*metis k.ide-left seqI′*)
also have
  ... $= (\mathrm{l}[h \star u] \cdot (k.\varepsilon \star h \star u) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star u] \cdot (k \star k^* \star h \star \vartheta)) \cdot (k \star \chi) \cdot (k \star \nu)$
  unfolding *k.trnl$_\varepsilon$-def* **by** *simp*
also have ... $= (h \star \vartheta) \cdot$
$(\mathrm{l}[h \star p_0 \star w] \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot (k \star \chi)) \cdot$
$(k \star \nu)$
proof −
  have $\mathrm{l}[h \star u] \cdot (k.\varepsilon \star h \star u) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star u] \cdot (k \star k^* \star h \star \vartheta) =$
$\mathrm{l}[h \star u] \cdot (k.\varepsilon \star h \star u) \cdot ((k \star k^*) \star h \star \vartheta) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w]$
    using *assms 4 k.antipar cospan assoc′-naturality* [*of k k* h ⋆ ϑ*]
    by *fastforce*
  also have ... $= \mathrm{l}[h \star u] \cdot ((k.\varepsilon \star h \star u) \cdot ((k \star k^*) \star h \star \vartheta)) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w]$
    using *comp-assoc* **by** *presburger*
  also have ... $= (\mathrm{l}[h \star u] \cdot (trg\ k \star h \star \vartheta)) \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w]$
  proof −
    have $(k.\varepsilon \star h \star u) \cdot ((k \star k^*) \star h \star \vartheta) = k.\varepsilon \cdot (k \star k^*) \star (h \star u) \cdot (h \star \vartheta)$
      using *assms 1 k.antipar cospan interchange comp-arr-dom comp-cod-arr*
      by *fastforce*
    also have ... $= k.\varepsilon \star h \star \vartheta$
      using *assms k.antipar cospan comp-arr-dom comp-cod-arr k.counit-in-hom whisker-left*
      by (*metis h.ide-left in-homE*)
    also have ... $= (trg\ k \star h \star \vartheta) \cdot (k.\varepsilon \star h \star p_0 \star w)$
      using *assms 4 k.antipar cospan whisker-left comp-arr-dom comp-cod-arr interchange* [*of trg k k.ε h ⋆ ϑ h ⋆ p$_0$ ⋆ w*]
      by *auto*
    finally have $(k.\varepsilon \star h \star u) \cdot ((k \star k^*) \star h \star \vartheta) = (trg\ k \star h \star \vartheta) \cdot (k.\varepsilon \star h \star p_0 \star w)$
      by *simp*
    thus *?thesis*
      using *comp-assoc* **by** *presburger*
  qed
  also have ... $= (h \star \vartheta) \cdot \mathrm{l}[h \star p_0 \star w] \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w]$
  proof −
    have $\mathrm{l}[h \star u] \cdot (trg\ k \star h \star \vartheta) = (h \star \vartheta) \cdot \mathrm{l}[h \star p_0 \star w]$
      using *assms 1 4 k.antipar cospan lunit-naturality* [*of h ⋆ ϑ*]
      by (*metis hcomp-simps(3−4) h.ide-left hseqI′ ide-char in-homE trg-hcomp*)
    thus *?thesis*
      using *comp-assoc* **by** *presburger*
  qed
  finally have $\mathrm{l}[h \star u] \cdot (k.\varepsilon \star h \star u) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star u] \cdot (k \star k^* \star h \star \vartheta) =$
$(h \star \vartheta) \cdot \mathrm{l}[h \star p_0 \star w] \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w]$
    by *simp*
  thus *?thesis*

using *comp-assoc* **by** *presburger*
qed
also have ... $= (h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu)$
proof −
have $\mathrm{l}[h \star p_0 \star w] \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w)) =$
$\mathrm{a}[h,\ p_0,\ w] \cdot \mathrm{l}[(h \star p_0) \star w] \cdot$
$(trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w))$
proof −
have $\mathrm{l}[h \star p_0 \star w] =$
$\mathrm{a}[h,\ p_0,\ w] \cdot \mathrm{l}[(h \star p_0) \star w] \cdot (trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w])$
proof −
have $\mathrm{a}[h,\ p_0,\ w] \cdot \mathrm{l}[(h \star p_0) \star w] \cdot (trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w]) =$
$\mathrm{a}[h,\ p_0,\ w] \cdot \mathfrak{l}\ ((h \star p_0) \star w) \cdot (trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w])$
using *assms 0 k.antipar cospan comp-cod-arr* $\mathfrak{l}$*-ide-simp* **by** *simp*
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot \mathfrak{l}\ (\mathrm{a}^{-1}[h,\ p_0,\ w])$
using *assms 0 k.antipar cospan* $\mathfrak{l}$*.naturality2* [*of* $\mathrm{a}^{-1}[h,\ p_0,\ w]$] **by** *simp*
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot \mathrm{a}^{-1}[h,\ p_0,\ w] \cdot \mathfrak{l}\ (h \star p_0 \star w)$
using *assms 0 k.antipar cospan* $\mathfrak{l}$*.naturality1* [*of* $\mathrm{a}^{-1}[h,\ p_0,\ w]$] *comp-assoc*
by *simp*
also have ... $= (\mathrm{a}[h,\ p_0,\ w] \cdot \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot \mathfrak{l}\ (h \star p_0 \star w)$
using *comp-assoc* **by** *presburger*
also have ... $= \mathfrak{l}\ (h \star p_0 \star w)$
using *assms 0 k.antipar cospan comp-cod-arr comp-assoc-assoc′* **by** *simp*
also have ... $= \mathrm{l}[h \star p_0 \star w]$
using *assms 0 k.antipar cospan* $\mathfrak{l}$*-ide-simp* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$\mathrm{a}^{-1}[trg\ h,\ h \star p_0,\ w] \cdot$
$((trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot (k.\varepsilon \star h \star p_0 \star w)) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w))$
using *assms 0 k.antipar cospan lunit-hcomp comp-assoc* **by** *simp*
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$(\mathrm{a}^{-1}[trg\ h,\ h \star p_0,\ w] \cdot (k.\varepsilon \star (h \star p_0) \star w)) \cdot$
$((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w))$
proof −
have $(trg\ h \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot (k.\varepsilon \star h \star p_0 \star w) =$
$(k.\varepsilon \star (h \star p_0) \star w) \cdot ((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w])$
using *assms 0 k.antipar cospan comp-arr-dom comp-cod-arr*
interchange [*of trg h* $k.\varepsilon$ $\mathrm{a}^{-1}[h,\ p_0,\ w]$ $h \star p_0 \star w$]

interchange [*of* $k.\varepsilon\ k \star k^*\ (h \star p_0) \star w\ \mathrm{a}^{-1}[h,\ p_0,\ w]$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$((k.\varepsilon \star (h \star p_0)) \star w) \cdot \mathrm{a}^{-1}[k \star k^*,\ h \star p_0,\ w] \cdot$
$((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w))$
using *assms 0 k.antipar cospan assoc′-naturality* [*of* $k.\varepsilon\ h \star p_0\ w$] *comp-assoc*
by *simp*
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$((k.\varepsilon \star (h \star p_0)) \star w) \cdot \mathrm{a}^{-1}[k \star k^*,\ h \star p_0,\ w] \cdot$
$((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w]) \cdot (k \star \mathrm{a}[k^* \star h,\ p_0,\ w]) \cdot$
$(k \star tab \star w)$
proof −
have $k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w) =$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w]) \cdot (k \star \mathrm{a}[k^* \star h,\ p_0,\ w]) \cdot$
$(k \star tab \star w)$
proof −
have *seq* $\mathrm{a}[k^*,\ h,\ p_0 \star w]$ $(\mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w))$
using χ*-def assms 0 k.antipar cospan* χ **by** *blast*
thus *?thesis*
using *assms 0 k.antipar cospan whisker-left* **by** *auto*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$((k.\varepsilon \star (h \star p_0)) \star w) \cdot (\mathrm{a}^{-1}[k \star k^*,\ h \star p_0,\ w] \cdot$
$((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w]) \cdot (k \star \mathrm{a}[k^* \star h,\ p_0,\ w]) \cdot$
$\mathrm{a}[k,\ (k^* \star h) \star p_0,\ w]) \cdot ((k \star tab) \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
proof −
have $k \star tab \star w = \mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] \cdot \mathrm{a}^{-1}[k,\ (k^* \star h) \star p_0,\ w] \cdot (k \star tab \star w)$
proof −
have $\mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] \cdot \mathrm{a}^{-1}[k,\ (k^* \star h) \star p_0,\ w] \cdot (k \star tab \star w) =$
$(\mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] \cdot \mathrm{a}^{-1}[k,\ (k^* \star h) \star p_0,\ w]) \cdot (k \star tab \star w)$
using *comp-assoc* **by** *presburger*
also have ... $= (k \star ((k^* \star h) \star p_0) \star w) \cdot (k \star tab \star w)$
using *assms k.antipar 0 comp-assoc-assoc′* **by** *simp*
also have ... $= k \star tab \star w$
using *assms k.antipar 0 comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= \mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] \cdot ((k \star tab) \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
using *assms 0 k.antipar cospan assoc′-naturality* [*of k tab w*] **by** *simp*
finally have $k \star tab \star w = \mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] \cdot ((k \star tab) \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$

by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\mathrm{l}[h \star p_0] \star w) \cdot$
$((k.\varepsilon \star h \star p_0) \star w) \cdot$
$(\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0]) \star w) \cdot$
$((k \star tab) \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
proof −
have $\mathrm{a}^{-1}[k \star k^*,\ h \star p_0,\ w] \cdot ((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0 \star w]) \cdot$
$(k \star \mathrm{a}[k^* \star h,\ p_0,\ w]) \cdot \mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] =$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0]) \star w$
proof −
have $\mathrm{a}^{-1}[k \star k^*,\ h \star p_0,\ w] \cdot ((k \star k^*) \star \mathrm{a}^{-1}[h,\ p_0,\ w]) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0 \star w]) \cdot$
$(k \star \mathrm{a}[k^* \star h,\ p_0,\ w]) \cdot \mathrm{a}[k,\ (k^* \star h) \star p_0,\ w] =$
$\{\!|\mathbf{a}^{-1}[\langle k\rangle \star \langle k^*\rangle,\ \langle h\rangle \star \langle p_0\rangle,\ \langle w\rangle] \cdot ((\langle k\rangle \star \langle k^*\rangle) \star \mathbf{a}^{-1}[\langle h\rangle,\ \langle p_0\rangle,\ \langle w\rangle]) \cdot$
$\mathbf{a}^{-1}[\langle k\rangle,\ \langle k^*\rangle,\ \langle h\rangle \star \langle p_0\rangle \star \langle w\rangle] \cdot (\langle k\rangle \star \mathbf{a}[\langle k^*\rangle,\ \langle h\rangle,\ \langle p_0\rangle \star \langle w\rangle]) \cdot$
$(\langle k\rangle \star \mathbf{a}[\langle k^*\rangle \star \langle h\rangle,\ \langle p_0\rangle,\ \langle w\rangle]) \cdot \mathbf{a}[\langle k\rangle,\ (\langle k^*\rangle \star \langle h\rangle) \star \langle p_0\rangle,\ \langle w\rangle]|\!\}$
using *assms 0 k.antipar cospan* α*-def* a′*-def* **by** *simp*
also have ... $= \{\!|\mathbf{a}^{-1}[\langle k\rangle,\ \langle k^*\rangle,\ \langle h\rangle \star \langle p_0\rangle] \cdot$
$(\langle k\rangle \star \mathbf{a}[\langle k^*\rangle,\ \langle h\rangle,\ \langle p_0\rangle]) \star \langle w\rangle|\!\}$
using *assms 0 k.antipar cospan*
by (*intro E.eval-eqI*, *simp-all*)
also have ... $= \mathrm{a}^{-1}[k,\ k^*,\ h \star p_0] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0]) \star w$
using *assms 0 k.antipar cospan* α*-def* a′*-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot$
$(\mathrm{l}[h \star p_0] \cdot (k.\varepsilon \star h \star p_0) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0] \cdot (k \star \mathrm{a}[k^*,\ h,\ p_0]) \cdot$
$(k \star tab) \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
using *assms 0 k.antipar cospan comp-assoc whisker-right* **by** *auto*
also have ... $= \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
unfolding φ*-def* $k.trnl_\varepsilon$*-def*
using *assms 0 k.antipar cospan comp-assoc whisker-left* **by** *simp*
finally have $\mathrm{l}[h \star p_0 \star w] \cdot (k.\varepsilon \star h \star p_0 \star w) \cdot$
$\mathrm{a}^{-1}[k,\ k^*,\ h \star p_0 \star w] \cdot$
$(k \star \mathrm{a}[k^*,\ h,\ p_0 \star w] \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w)) =$
$\mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
by *blast*
thus *?thesis*
using χ*-def comp-assoc* **by** *simp*
qed
finally show *?thesis* **by** *simp*

qed

BS3 implies that φ is the unique 2-cell from $k \star p_1$ to $h \star p_0$ and is an isomorphism.

lemma *φ-uniqueness*:
shows $\bigwedge\mu.$ «$\mu : k \star p_1 \Rightarrow h \star p_0$» $\Longrightarrow \mu = \varphi$ **and** *iso* φ
proof −
  have *2*: *is-left-adjoint* $(k \star p_1)$
    using *k.antipar cospan left-adjoints-compose* **by** (*simp add*: *k-is-map*)
  have *3*: *is-left-adjoint* $(h \star p_0)$
    using *k.antipar cospan left-adjoints-compose* **by** (*simp add*: *h-is-map*)
  show $\bigwedge\mu.$ «$\mu : k \star p_1 \Rightarrow h \star p_0$» $\Longrightarrow \mu = \varphi$
    using *φ-in-hom 2 3 BS3* **by** *simp*
  show *iso* φ
    using *φ-in-hom 2 3 BS3* **by** *simp*
qed

As a consequence, the chosen tabulation of $k^* \star h$ is the unique 2-cell from p_1 to $(k^* \star h) \star p_0$, and therefore if we are given any such 2-cell we may conclude it yields a tabulation of $k^* \star h$.

lemma *tab-uniqueness*:
assumes «$\tau : p_1 \Rightarrow (k^* \star h) \star p_0$»
shows $\tau = tab$
proof −
  have «$k.trnl_\varepsilon\ (h \star p_0)\ (\mathrm{a}[k^*, h, p_0] \cdot \tau) : k \star p_1 \Rightarrow h \star p_0$»
    using *assms k.antipar cospan k.adjoint-transpose-left*(*2*) [*of* $h \star p_0\ p_1$]
      assoc-in-hom
    by *force*
  hence $tab = \mathrm{a}^{-1}[k^*, h, p_0] \cdot k.trnl_\eta\ p_1\ (k.trnl_\varepsilon\ (h \star p_0)\ (\mathrm{a}[k^*, h, p_0] \cdot \tau))$
    using *transpose-φ φ-uniqueness*(*1*) **by** *auto*
  also have ... $= \mathrm{a}^{-1}[k^*, h, p_0] \cdot \mathrm{a}[k^*, h, p_0] \cdot \tau$
    using *assms k.antipar cospan k.adjoint-transpose-left*(*4*) *assoc-in-hom* **by** *auto*
  also have ... $= (\mathrm{a}^{-1}[k^*, h, p_0] \cdot \mathrm{a}[k^*, h, p_0]) \cdot \tau$
    using *comp-assoc* **by** *presburger*
  also have ... $= \tau$
    using *assms k.antipar cospan comp-cod-arr comp-assoc-assoc′* **by** *auto*
  finally show *?thesis* **by** *simp*
qed

The following lemma reformulates the biuniversal property of the canonical tabulation of $k^* \star h$ as a biuniversal property of φ, regarded as a square that commutes up to isomorphism.

lemma *φ-biuniversal-prop*:
assumes *ide u* **and** *ide v*
shows $\bigwedge\mu.$ «$\mu : k \star v \Rightarrow h \star u$» $\Longrightarrow$
    $\exists\, w\ \vartheta\ \nu.$ *ide w* $\wedge$ «$\vartheta : p_0 \star w \Rightarrow u$» $\wedge$ «$\nu : v \Rightarrow p_1 \star w$» $\wedge$ *iso* ν $\wedge$
      $(h \star \vartheta) \cdot \mathrm{a}[h, p_0, w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k, p_1, w] \cdot (k \star \nu) = \mu$
and $\bigwedge w\ w'\ \vartheta\ \vartheta'\ \beta.$
    $[\![$ *ide w*; *ide w′*;
      «$\vartheta : p_0 \star w \Rightarrow u$»; «$\vartheta' : p_0 \star w' \Rightarrow u$»;

«$\beta : p_1 \star w \Rightarrow p_1 \star w'$»;
$(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] =$
$(h \star \vartheta') \cdot \mathrm{a}[h,\ p_0,\ w'] \cdot (\varphi \star w') \cdot \mathrm{a}^{-1}[k,\ p_1,\ w'] \cdot (k \star \beta)$ ⟧
$\Longrightarrow \exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \vartheta = \vartheta' \cdot (p_0 \star \gamma) \wedge p_1 \star \gamma = \beta$

proof −
  fix μ
  assume μ: «$\mu : k \star v \Rightarrow h \star u$»
  have *1*: *src* h = *trg* u
    using *assms* μ *ide-cod*
    by (*metis ide-def in-homE seq-if-composable*)
  have *2*: *src* k = *trg* v
    using *assms* μ *ide-dom*
    by (*metis ideD*(*1*) *in-homE not-arr-null seq-if-composable*)
  let $?\omega = \mathrm{a}^{-1}[k^*,\ h,\ u] \cdot k.trnl_\eta\ v\ \mu$
  have ω: «$?\omega : v \Rightarrow (k^* \star h) \star u$»
    using *assms* μ *1 2 k.antipar cospan k.adjoint-transpose-left*(*1*) [*of* $h \star u\ v$]
      assoc-in-hom
    by *auto*
  obtain $w\ \vartheta\ \nu$
  where $w\vartheta\nu$: *ide* $w \wedge$ «$\vartheta : p_0 \star w \Rightarrow u$» $\wedge$ «$\nu : v \Rightarrow p_1 \star w$» $\wedge$ *iso* $\nu \wedge$
      $((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w) \cdot \nu = ?\omega$
    using *assms* ω *T1* [*of* u $?\omega$] *comp-assoc* **by** (*metis in-homE*)
  have *0*: *src* p_0 = *trg* w
    using $w\vartheta\nu$ *ide-dom*
    by (*metis hseqE ideD*(*1*) *in-homE*)
  have $\mu = k.trnl_\varepsilon\ (h \star u)\ (\mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w) \cdot \nu)$
  proof −
    have $\mu = k.trnl_\varepsilon\ (h \star u)\ (\mathrm{a}[k^*,\ h,\ u] \cdot ?\omega)$
    proof −
      have $k.trnl_\varepsilon\ (h \star u)\ (\mathrm{a}[k^*,\ h,\ u] \cdot ?\omega) =$
        $k.trnl_\varepsilon\ (h \star u)\ ((\mathrm{a}[k^*,\ h,\ u] \cdot \mathrm{a}^{-1}[k^*,\ h,\ u]) \cdot k.trnl_\eta\ v\ \mu)$
        using *comp-assoc* **by** *presburger*
      also have ... $= k.trnl_\varepsilon\ (h \star u)\ (k.trnl_\eta\ v\ \mu)$
      proof −
        have $(\mathrm{a}[k^*,\ h,\ u] \cdot \mathrm{a}^{-1}[k^*,\ h,\ u]) \cdot k.trnl_\eta\ v\ \mu = (k^* \star h \star u) \cdot k.trnl_\eta\ v\ \mu$
          using *comp-assoc-assoc'*
          by (*simp add*: *1 assms*(*1*) *cospan k.antipar*(*2*))
        also have ... $= k.trnl_\eta\ v\ \mu$
          using *1* ω *assms*(*1*) *comp-ide-arr cospan k.antipar*(*2*) **by** *fastforce*
        finally show *?thesis*
          by *simp*
      qed
      also have ... $= \mu$
        using *assms* μ *k.antipar cospan 1 2 k.adjoint-transpose-left*(*3*) **by** *simp*
      finally show *?thesis* **by** *simp*
    qed
    thus *?thesis* **using** $w\vartheta\nu$ **by** *simp*
  qed
  also have ... $= (h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu)$

using *assms k.antipar cospan wϑν transpose-triangle* [*of w ϑ u ν*] **by** *auto*
finally have $(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu) = \mu$
by *simp*
thus $\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : p_0 \star w \Rightarrow u» \wedge$
$«\nu : v \Rightarrow p_1 \star w» \wedge iso\ \nu \wedge$
$(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot$
$\mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu) = \mu$
using *wϑν* **by** *blast*
next
fix $w\ w'\ \vartheta\ \vartheta'\ \beta$
assume *w*: *ide w*
assume *w′*: *ide w′*
assume *ϑ*: $«\vartheta : p_0 \star w \Rightarrow u»$
assume *ϑ′*: $«\vartheta' : p_0 \star w' \Rightarrow u»$
assume *β*: $«\beta : p_1 \star w \Rightarrow p_1 \star w'»$
assume *eq*: $(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] =$
$(h \star \vartheta') \cdot \mathrm{a}[h,\ p_0,\ w'] \cdot (\varphi \star w') \cdot \mathrm{a}^{-1}[k,\ p_1,\ w'] \cdot (k \star \beta)$
have *0*: $src\ p_0 = trg\ w$
using *ϑ ide-dom*
by (*metis ideD*(*1*) *in-homE not-arr-null seq-if-composable*)
interpret *uwϑw′ϑ′*: *uwϑw′ϑ′ V H* a i *src trg* ‹$k^* \star h$› *tab* ‹p_0› ‹p_1›
u w ϑ w′ ϑ′
using *w ϑ w′ ϑ′* **apply** (*unfold-locales*) **by** *auto*
show $\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \vartheta = \vartheta' \cdot (p_0 \star \gamma) \wedge p_1 \star \gamma = \beta$
proof −
let *?LHS* $= \mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w)$
let *?RHS* $= \mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta') \cdot \mathrm{a}[k^* \star h,\ p_0,\ w'] \cdot (tab \star w') \cdot \beta$
have *eq′*: *?LHS* = *?RHS*
proof −
have $k.trnl_\varepsilon\ (h \star u)$ *?LHS* =
$k.trnl_\varepsilon\ (h \star u)$
$(\mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w) \cdot (p_1 \star w))$
using *assms 0 w ϑ β k.antipar cospan comp-arr-dom*
by (*metis tab-simps*(*1*) *tab-simps*(*4*) *whisker-right*)
also have ... $= (h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star p_1 \star w)$
using *assms w ϑ β transpose-triangle*
by (*metis arr-dom ide-hcomp ide-in-hom*(*2*) *in-homE ide-leg1 not-arr-null seq-if-composable*)
also have ... $= (h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w]$
using *assms 0 w k.antipar cospan comp-arr-dom* **by** *simp*
also have ... $= (h \star \vartheta') \cdot \mathrm{a}[h,\ p_0,\ w'] \cdot (\varphi \star w') \cdot \mathrm{a}^{-1}[k,\ p_1,\ w'] \cdot (k \star \beta)$
using *eq* **by** *blast*
also have ... $= k.trnl_\varepsilon\ (h \star u)$ *?RHS*
using *assms w′ ϑ′ β transpose-triangle* **by** *simp*
finally have *4*: $k.trnl_\varepsilon\ (h \star u)$ *?LHS* $= k.trnl_\varepsilon\ (h \star u)$ *?RHS*
by *simp*
have $src\ k = trg\ (p_1 \star w)$
using *assms 0 w k.antipar cospan* **by** *simp*
moreover have $src\ k^* = trg\ (h \star u)$

using *assms 0 w k.antipar cospan* **by** *simp*
moreover have *ide* $(h \star u)$
using *assms 0 w k.antipar cospan* **by** *simp*
moreover have *ide* $(p_1 \star w)$
using *assms 0 w k.antipar cospan* **by** *simp*
ultimately have *inj-on* $(k.trnl_\varepsilon\ (h \star u))$ $(hom\ (p_1 \star w)\ (k^* \star h \star u))$
using *assms 0 w w′ k.antipar cospan k.adjoint-transpose-left(6) bij-betw-imp-inj-on*
by *blast*
moreover have *?LHS* $\in hom\ (p_1 \star w)\ (k^* \star h \star u)$
proof −
have «$\mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta) \cdot \mathrm{a}[k^* \star h,\ p_0,\ w] \cdot (tab \star w)$:
$p_1 \star w \Rightarrow k^* \star h \star u$»
using *k.antipar cospan*
apply (*intro comp-in-homI*) **by** *auto*
thus *?thesis* **by** *simp*
qed
moreover have *?RHS* $\in hom\ (p_1 \star w)\ (k^* \star h \star u)$
proof −
have «$\mathrm{a}[k^*,\ h,\ u] \cdot ((k^* \star h) \star \vartheta') \cdot \mathrm{a}[k^* \star h,\ p_0,\ w'] \cdot$
$(tab \star w') \cdot \beta : p_1 \star w \Rightarrow k^* \star h \star u$»
using β *k.antipar cospan*
apply (*intro comp-in-homI*) **by** *auto*
thus *?thesis* **by** *blast*
qed
ultimately show *?LHS* = *?RHS*
using *assms 4 k.antipar cospan bij-betw-imp-inj-on*
inj-on-def [*of* $k.trnl_\varepsilon\ (h \star u)$ $hom\ (p_1 \star w)\ (k^* \star h \star u)$]
by *simp*
qed
moreover have *seq* $\mathrm{a}[k^*,\ h,\ u]$ (*composite-cell* $w\ \vartheta$)
using *assms k.antipar cospan tab-in-hom* **by** *fastforce*
moreover have *seq* $\mathrm{a}[k^*,\ h,\ u]$ (*composite-cell* $w'\ \vartheta' \cdot \beta$)
using *assms* β *k.antipar cospan tab-in-hom* **by** *fastforce*
ultimately have *composite-cell* $w\ \vartheta$ = *composite-cell* $w'\ \vartheta' \cdot \beta$
using *assms 0 w w′* β *k.antipar cospan iso-assoc iso-is-section section-is-mono*
mono-cancel [*of* $\mathrm{a}[k^*,\ h,\ u]$ *composite-cell* $w\ \vartheta$ *composite-cell* $w'\ \vartheta' \cdot \beta$]
comp-assoc
by *simp*
thus *?thesis*
using $w\ w'\ \vartheta\ \vartheta'\ \beta$ *eq′ T2* [*of* $w\ w'\ \vartheta\ u\ \vartheta'\ \beta$] **by** *metis*
qed
qed

Using the uniqueness properties established for φ, we obtain yet another reformulation of the biuniversal property associated with the chosen tabulation of $k^* \star h$, this time as a kind of pseudo-pullback. We will use this to show that the category of isomorphism classes of maps has pullbacks.

lemma *has-pseudo-pullback*:
assumes *is-left-adjoint u* **and** *is-left-adjoint v* **and** *isomorphic* $(k \star v)\ (h \star u)$

shows $\exists\, w.$ *is-left-adjoint* $w \wedge$ *isomorphic* $(p_0 \star w)\ u \wedge$ *isomorphic* $v\ (p_1 \star w)$
and $\bigwedge w\ w'.\ [\![$ *is-left-adjoint* w; *is-left-adjoint* w';
$p_0 \star w \cong u$; $v \cong p_1 \star w$; $p_0 \star w' \cong u$; $v \cong p_1 \star w'\ ]\!] \Longrightarrow w \cong w'$
proof −
interpret u: *map-in-bicategory* $V\ H$ a i *src trg* u
using *assms*(*1*) **by** *unfold-locales auto*
interpret v: *map-in-bicategory* $V\ H$ a i *src trg* v
using *assms*(*2*) **by** *unfold-locales auto*
obtain μ **where** μ: «$\mu : k \star v \Rightarrow h \star u$» $\wedge$ *iso* μ
using *assms*(*3*) **by** *auto*
obtain $w\ \vartheta\ \nu$ **where** $w\vartheta\nu$: *ide* $w \wedge$ «$\vartheta : p_0 \star w \Rightarrow u$» $\wedge$
«$\nu : v \Rightarrow p_1 \star w$» $\wedge$ *iso* $\nu \wedge$
$(h \star \vartheta) \cdot \mathrm{a}[h,\ p_0,\ w] \cdot (\varphi \star w) \cdot \mathrm{a}^{-1}[k,\ p_1,\ w] \cdot (k \star \nu) = \mu$
using *assms* μ φ-*biuniversal-prop*(*1*) [*of* $u\ v\ \mu$] **by** *auto*
have *is-left-adjoint* $w \wedge$ *isomorphic* $(p_0 \star w)\ u \wedge$ *isomorphic* $v\ (p_1 \star w)$
proof (*intro conjI*)
show *1*: *is-left-adjoint* w
using *assms*(*2*) $w\vartheta\nu$ *left-adjoint-preserved-by-iso′ isomorphic-def BS4 leg1-is-map*
by *blast*
show $v \cong p_1 \star w$
using $w\vartheta\nu$ *isomorphic-def* **by** *blast*
show $p_0 \star w \cong u$
proof −
have *src* $p_0 =$ *trg* w
using $w\vartheta\nu$ *ide-dom*
by (*metis ideD*(*1*) *in-homE not-arr-null seq-if-composable*)
hence *is-left-adjoint* $(p_0 \star w)$
using *1 left-adjoints-compose* **by** *simp*
thus *?thesis*
using *assms* $w\vartheta\nu$ *1 BS3 isomorphic-def* **by** *metis*
qed
qed
thus $\exists\, w.$ *is-left-adjoint* $w \wedge p_0 \star w \cong u \wedge v \cong p_1 \star w$
by *blast*
show $\bigwedge w\ w'.\ [\![$ *is-left-adjoint* w; *is-left-adjoint* w';
$p_0 \star w \cong u$; $v \cong p_1 \star w$; $p_0 \star w' \cong u$; $v \cong p_1 \star w'\ ]\!] \Longrightarrow w \cong w'$
proof −
fix $w\ w'$
assume w: *is-left-adjoint* w **and** w': *is-left-adjoint* w'
assume *1*: $p_0 \star w \cong u$
assume *2*: $v \cong p_1 \star w$
assume *3*: $p_0 \star w' \cong u$
assume *4*: $v \cong p_1 \star w'$
obtain ϑ **where** ϑ: «$\vartheta : p_0 \star w \Rightarrow u$»
using *1* **by** *auto*
obtain ϑ' **where** ϑ': «$\vartheta' : p_0 \star w' \Rightarrow u$»
using *3* **by** *auto*
obtain ν **where** ν: «ν: $v \Rightarrow p_1 \star w$» $\wedge$ *iso* ν
using *2* **by** *blast*

```
      obtain ν' where ν': «ν': v ⇒ p₁ ⋆ w'» ∧ iso ν'
        using 4 by blast
      let ?β = ν' · inv ν
      have β: «?β : p₁ ⋆ w ⇒ p₁ ⋆ w'»
        using ν ν' by (elim conjE in-homE, auto)
      interpret uwϑ: uwϑ V H a i src trg ‹k* ⋆ h› tab ‹p₀› ‹p₁› u w ϑ
        using w ϑ left-adjoint-is-ide
        by unfold-locales auto
      interpret uw'ϑ': uwϑ V H a i src trg ‹k* ⋆ h› tab ‹p₀› ‹p₁›
                       u w' ϑ'
        using w' ϑ' left-adjoint-is-ide
        by unfold-locales auto
      interpret uwϑw'ϑ': uwϑw'ϑ' V H a i src trg ‹k* ⋆ h› tab ‹p₀› ‹p₁› u w ϑ w' ϑ'
        using w w' ϑ ϑ' left-adjoint-is-ide by unfold-locales
      have (h ⋆ ϑ) · a[h, p₀, w] · (φ ⋆ w) · a⁻¹[k, p₁, w] =
            (h ⋆ ϑ') · a[h, p₀, w'] · (φ ⋆ w') · a⁻¹[k, p₁, w'] ·
              (k ⋆ ?β)
      proof −
        let ?LHS = (h ⋆ ϑ) · a[h, p₀, w] · (φ ⋆ w) · a⁻¹[k, p₁, w]
        let ?RHS = (h ⋆ ϑ') · a[h, p₀, w'] · (φ ⋆ w') · a⁻¹[k, p₁, w'] · (k ⋆ ?β)
        have «?LHS : k ⋆ p₁ ⋆ w ⇒ h ⋆ u»
          using w k.antipar by fastforce
        moreover have «?RHS : k ⋆ p₁ ⋆ w ⇒ h ⋆ u»
          using w k.antipar β by fastforce
        moreover have is-left-adjoint (k ⋆ p₁ ⋆ w)
          using w k.is-map left-adjoints-compose by simp
        moreover have is-left-adjoint (h ⋆ u)
          using assms h.is-map left-adjoints-compose by auto
        ultimately show ?LHS = ?RHS
          using BS3 by blast
      qed
      hence ∃!γ. «γ : w ⇒ w'» ∧ ϑ = ϑ' · (p₀ ⋆ γ) ∧ p₁ ⋆ γ = ?β
        using assms left-adjoint-is-ide w w' ϑ ϑ' ν ν' β
              φ-biuniversal-prop(2) [of u v w w' ϑ ϑ' ?β]
        by presburger
      thus w ≅ w'
        using w w' BS3 isomorphic-def by metis
    qed
  qed

end
```

Tabulations in Maps

Here we focus our attention on the properties of tabulations in a bicategory of spans, in the special case in which both legs are maps.

```
context tabulation-in-maps
begin
```

The following are the conditions under which w is a 1-cell induced via *T1* by a 2-cell «ω : *dom* $\omega \Rightarrow s \star r_0$»: w is an arrow of spans and ω is obtained by composing the tabulation σ with w and the isomorphisms that witness w being an arrow of spans.

abbreviation *is-induced-by-cell*
where *is-induced-by-cell* $w\ r_0\ \omega \equiv$
  arrow-of-spans-of-maps $V\ H$ a i *src trg* r_0 (*dom* ω) $s_0\ s_1\ w \wedge$
  composite-cell w (*arrow-of-spans-of-maps.the-*ϑ $V\ H\ r_0\ s_0\ w$) $\cdot$
    (*arrow-of-spans-of-maps.the-*ν $V\ H$ (*dom* ω) $s_1\ w$) $= \omega$

lemma *induced-map-unique*:
assumes *is-induced-by-cell* $w\ r_0\ \omega$ **and** *is-induced-by-cell* $w'\ r_0\ \omega$
shows *isomorphic* $w\ w'$
proof −
  interpret w: *arrow-of-spans-of-maps* $V\ H$ a i *src trg* r_0 ‹*dom* ω› $s_0\ s_1\ w$
    using *assms*(*1*) **by** *auto*
  interpret w: *arrow-of-spans-of-maps-to-tabulation* $V\ H$ a i *src trg* r_0 ‹*dom* ω› $s\ \sigma\ s_0\ s_1\ w$
    ..
  interpret w': *arrow-of-spans-of-maps* $V\ H$ a i *src trg* r_0 ‹*dom* ω› $s_0\ s_1\ w'$
    using *assms*(*2*) **by** *auto*
  interpret w': *arrow-of-spans-of-maps-to-tabulation* $V\ H$ a i *src trg* r_0 ‹*dom* ω› $s\ \sigma\ s_0\ s_1\ w'$
    ..
  let *?*$\beta = w'$*.the-*$\nu \cdot$ *inv* w*.the-*ν
  have β: «*?*$\beta : s_1 \star w \Rightarrow s_1 \star w'$»
    using w*.the-*ν*-props* w'*.the-*ν*-props arr-iff-in-hom* **by** *auto*
  have *1*: *composite-cell* w w*.the-*ϑ $=$ *composite-cell* w' w'*.the-*$\vartheta \cdot$ (w'*.the-*$\nu \cdot$ *inv* w*.the-*ν)
  proof −
    have *composite-cell* w' w'*.the-*$\vartheta \cdot$ (w'*.the-*$\nu \cdot$ *inv* w*.the-*ν) $=$
      ((*composite-cell* w' w'*.the-*ϑ) $\cdot$ w'*.the-*ν) $\cdot$ *inv* w*.the-*ν
      using *comp-assoc* **by** *presburger*
    also have ... $= \omega \cdot$ *inv* w*.the-*ν
      using *assms*(*2*) *comp-assoc* **by** *simp*
    also have ... $=$ (*composite-cell* w w*.the-*$\vartheta \cdot$ w*.the-*ν) $\cdot$ *inv* w*.the-*ν
      using *assms*(*1*) *comp-assoc* **by** *simp*
    also have ... $=$ *composite-cell* w w*.the-*$\vartheta \cdot$ w*.the-*$\nu \cdot$ *inv* w*.the-*ν
      using *comp-assoc* **by** *presburger*
    also have ... $=$ *composite-cell* w w*.the-*ϑ
    proof −
      have w*.the-*$\nu \cdot$ *inv* w*.the-*$\nu = s_1 \star w$
        using w*.the-*ν*-props comp-arr-inv inv-is-inverse* **by** *auto*
      thus *?thesis*
        using *composite-cell-in-hom w.ide-w w.the-*ϑ*-props comp-arr-dom*
        by (*metis composite-cell-in-hom in-homE w.w-in-hom*(*1*))
    qed
    finally show *?thesis* **by** *auto*
  qed
  have $\exists\gamma$. «$\gamma : w \Rightarrow w'$»
    using *1* β *w.is-ide w'.is-ide w.the-*ϑ*-props w'.the-*ϑ*-props*
      T2 [*of* $w\ w'$ w*.the-*$\vartheta\ r_0\ w'$*.the-*ϑ *?*β]
    by *blast*

thus *?thesis*
using *BS3 w.is-map w′.is-map* **by** *blast*
qed

The object src s_0 forming the apex of the tabulation satisfies the conditions for being a map induced via *T1* by the 2-cell σ itself. This is ultimately required for the map from 2-cells to arrows of spans to be functorial with respect to vertical composition.

lemma *apex-is-induced-by-cell*:
shows *is-induced-by-cell* (*src* s_0) s_0 σ
proof −
have *1*: *arrow-of-spans-of-maps V H* a i *src trg* s_0 s_1 s_0 s_1 (*src* s_0)
using *iso-runit* [*of* s_0] *iso-runit* [*of* s_1] *tab-in-hom*
apply *unfold-locales*
apply *simp*
using *ide-leg0 isomorphic-def*
apply *blast*
using *ide-leg1 isomorphic-def leg1-simps*(*3*) *runit′-in-vhom* [*of* s_1 *src* s_0] *iso-runit′*
by *blast*
interpret *w*: *arrow-of-spans-of-maps V H* a i *src trg* s_0 ‹*dom* σ› s_0 s_1 ‹*src* s_0›
using *1 tab-in-hom* **by** *simp*
interpret *w*: *arrow-of-spans-of-maps-to-tabulation*
V H a i *src trg* s_0 ‹*dom* σ› *s* σ s_0 s_1 ‹*src* s_0›
..
show *is-induced-by-cell* (*src* s_0) s_0 σ
proof (*intro conjI*)
show *arrow-of-spans-of-maps V H* a i *src trg* s_0 (*dom* σ) s_0 s_1 (*src* s_0)
using *w.arrow-of-spans-of-maps-axioms* **by** *simp*
show *composite-cell* (*src* s_0) *w.the-*ϑ $\cdot$ *w.the-*ν $= \sigma$
proof −
have ϑ: *w.the-*ϑ $= \mathrm{r}[s_0]$
using *iso-runit* [*of* s_0] *w.leg0-uniquely-isomorphic w.the-*ϑ*-props*
the1-equality [*of* $\lambda\vartheta$. «ϑ : $s_0 \star$ *src* $s_0 \Rightarrow s_0$» $\wedge$ *iso* ϑ]
by *auto*
have ν: *w.the-*ν $= \mathrm{r}^{-1}[s_1]$
using *iso-runit′ w.leg1-uniquely-isomorphic w.the-*ν*-props leg1-simps*(*3*)
the1-equality [*of* $\lambda\nu$. «ν : $s_1 \Rightarrow s_1 \star$ *src* s_0» $\wedge$ *iso* ν] *tab-in-vhom′*
by *auto*
have *composite-cell* (*src* s_0) $\mathrm{r}[s_0] \cdot \mathrm{r}^{-1}[s_1] = \sigma$
proof −
have *composite-cell* (*src* s_0) $\mathrm{r}[s_0] \cdot \mathrm{r}^{-1}[s_1] =$
$((s \star \mathrm{r}[s_0]) \cdot \mathrm{a}[s,\ s_0,\ \mathit{src}\ s_0]) \cdot (\sigma \star \mathit{src}\ s_0) \cdot \mathrm{r}^{-1}[s_1]$
using *comp-assoc* **by** *presburger*
also have ... $= (\mathrm{r}[s \star s_0] \cdot (\sigma \star \mathit{src}\ s_0)) \cdot \mathrm{r}^{-1}[s_1]$
using *runit-hcomp comp-assoc* **by** *simp*
also have ... $= \sigma \cdot \mathrm{r}[s_1] \cdot \mathrm{r}^{-1}[s_1]$
using *runit-naturality tab-in-hom*
by (*metis tab-simps*(*1*) *tab-simps*(*2*) *tab-simps*(*4*) *tab-simps*(*5*) *comp-assoc*)
also have ... $= \sigma$
using *iso-runit tab-in-hom comp-arr-dom comp-arr-inv inv-is-inverse* **by** *simp*

```
        finally show ?thesis by simp
      qed
      thus ?thesis
        using ϑ ν comp-assoc by simp
    qed
  qed
qed

end
```

Composing Tabulations

Given tabulations $(r_0,\ \varrho,\ r_1)$ of r and $(s_0,\ \sigma,\ s_1)$ of s in a bicategory of spans, where $(r_0,\ r_1)$ and $(s_0,\ s_1)$ are spans of maps and 1-cells r and s are composable, we can construct a 2-cell that yields a tabulation of $r \star s$. The proof uses the fact that the 2-cell φ associated with the cospan $(r_0,\ s_1)$ is an isomorphism, which we have proved above (*cospan-of-maps-in-bicategory-of-spans.φ-uniqueness*) using *BS3*. However, this is the only use of *BS3* in the proof, and it seems plausible that it would be possible to establish that φ is an isomorphism in more general situations in which the subbicategory of maps is not locally essentially discrete. Alternatively, more general situations could be treated by adding the assertion that φ is an isomorphism as part of a weakening of *BS3*.

```
locale composite-tabulation-in-maps =
  bicategory-of-spans V H a i src trg +
  ϱ: tabulation-in-maps V H a i src trg r ϱ r₀ r₁ +
  σ: tabulation-in-maps V H a i src trg s σ s₀ s₁
for V :: 'a comp                (infixr ‹·› 55)
and H :: 'a ⇒ 'a ⇒ 'a           (infixr ‹⋆› 53)
and a :: 'a ⇒ 'a ⇒ 'a ⇒ 'a      (‹a[-, -, -]›)
and i :: 'a ⇒ 'a                (‹i[-]›)
and src :: 'a ⇒ 'a
and trg :: 'a ⇒ 'a
and r :: 'a
and ϱ :: 'a
and r₀ :: 'a
and r₁ :: 'a
and s :: 'a
and σ :: 'a
and s₀ :: 'a
and s₁ :: 'a +
assumes composable: src r = trg s
begin
```

Interpret $(r_0,\ s_1)$ as a *cospan-of-maps-in-bicategory-of-spans*, to obtain the isomorphism φ in the central diamond, along with the assertion that it is unique.

```
interpretation r₀s₁: cospan-of-maps-in-bicategory-of-spans V H a i src trg s₁ r₀
  using ϱ.leg0-is-map σ.leg1-is-map composable by unfold-locales auto
```

We need access to simps, etc. in the preceding interpretation, yet trying to declare it as a sublocale introduces too many conflicts at the moment. As it confusing elsewhere to figure out exactly how, in other contexts, to express the particular interpretation that is needed, to make things easier we include the following lemma. Then we can just recall the lemma to find out how to express the interpretation required in a given context.

lemma $r_0 s_1$*-is-cospan*:
shows *cospan-of-maps-in-bicategory-of-spans* V H a i *src trg* s_1 r_0
..

The following define the projections associated with the natural tabulation of $r_0{}^* \star s_1$.

abbreviation p_0
where $p_0 \equiv r_0 s_1.p_0$

abbreviation p_1
where $p_1 \equiv r_0 s_1.p_1$

$$\begin{tikzcd}
 & & \mathrm{src}\ \phi \arrow[dl, "p_1"'] \arrow[dr, "p_0"] & & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[r, phantom, "\overset{\rho}{\Rightarrow}"] & \arrow[d, phantom, "\overset{\phi}{\Rightarrow}"] & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[l, phantom, "\overset{\sigma}{\Rightarrow}"] & \\
\mathrm{trg}\ r & & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[ll, "r"] \arrow[rr, "s"'] & & \mathrm{src}\ s
\end{tikzcd}$$

Next, we define the 2-cell that is the composite of the tabulation σ, the tabulation ϱ, and the central diamond that commutes up to unique isomorphism φ.

definition *tab*
where $tab \equiv \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \cdot (r \star \sigma \star p_0) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ p_1] \cdot (\varrho \star p_1)$

lemma *tab-in-hom* [*intro*]:
shows «$tab : r_1 \star p_1 \Rightarrow (r \star s) \star s_0 \star p_0$»
using ϱ*.T0.antipar*(*1*) $r_0 s_1.\varphi$*-in-hom composable* ϱ*.leg0-in-hom*(*1*) σ*.leg1-in-hom*(*1*)
composable tab-def
by *auto*

interpretation *tabulation-data* V H a i *src trg* ‹$r \star s$› *tab* ‹$s_0 \star p_0$› ‹$r_1 \star p_1$›
using *composable tab-in-hom*
by *unfold-locales auto*

In the subsequent proof we will use coherence to shortcut a few of the calculations.

interpretation E: *self-evaluation-map* V H a i *src trg* **..**
notation E*.eval* (‹⦃-⦄›)

The following is applied twice in the proof of property *T2* for the composite tabulation. It's too long to repeat.

lemma *technical*:
assumes *ide* w
and «$\vartheta : (s_0 \star p_0) \star w \Rightarrow u$»
and $w_r = p_1 \star w$
and $\vartheta_r = (s \star \vartheta) \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
shows ϱ.*composite-cell* $w_r\ \vartheta_r = \mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w\ \vartheta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$

$$\begin{tikzcd}
 & & X \arrow[d, "w"] \arrow[ddrr, bend left, "u"] & & \\
 & & \mathrm{src}\ \phi \arrow[dl, "p_1"'] \arrow[dr, "p_0"] \arrow[r, phantom, "\overset{\theta}{\Nearrow}"] & & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[r, phantom, "\overset{\phi}{\Rightarrow}"] & & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] & \\
\mathrm{trg}\ r \arrow[rr, phantom, "\overset{\rho}{\Rightarrow}"] & & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[ll, "r"] \arrow[rr, phantom, "\overset{\sigma}{\Rightarrow}"] & & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}$$

proof −
interpret *uwϑ*: *uwϑ V H* a i *src trg* ‹$r \star s$› *tab* ‹$s_0 \star p_0$› ‹$r_1 \star p_1$› *u w ϑ*
using *assms*(*1*−*2*) *composable*
by *unfold-locales auto*
show *?thesis*
proof −
have $\mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w\ \vartheta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w] =$
$(\mathrm{a}[r,\ s,\ u] \cdot ((r \star s) \star \vartheta)) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot (tab \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *comp-assoc* **by** *presburger*
also have ... $= (r \star s \star \vartheta) \cdot \mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(tab \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *assoc-naturality* [*of r s ϑ*] *composable comp-assoc* **by** *simp*
also have ... $= (r \star s \star \vartheta) \cdot \mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$((\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0])) \cdot (r \star \sigma \star p_0) \cdot$
ϱ.*composite-cell* $p_1\ r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
unfolding *tab-def*
using *comp-assoc* **by** *presburger*
also have ... $= (r \star s \star \vartheta) \cdot ((\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w))) \cdot$
$((r \star \sigma \star p_0) \cdot \varrho$.*composite-cell* $p_1\ r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *composable* ϱ.*T0*.*antipar*(*1*) *comp-assoc whisker-right* **by** *auto*
also have ... $= (r \star s \star \vartheta) \cdot ((\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w))) \cdot$
$((r \star \sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi) \star w) \cdot$
$(\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot ((\varrho \star p_1) \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *composable* ϱ.*T0*.*antipar*(*1*) *whisker-right tab-def tab-in-hom*(*2*)
composable comp-assoc
by *force*

also have ... $= (r \star s \star \vartheta) \cdot ((\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w))) \cdot$
$((r \star \sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi) \star w) \cdot$
$((\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]) \cdot (\varrho \star p_1 \star w)$
using *assoc′-naturality* [*of* ϱ p_1 w] *ϱ.T0.antipar*(*1*) $r_0 s_1$*.base-simps*(*2*) *comp-assoc*
by *auto*
also have ... $= (r \star s \star \vartheta) \cdot ((\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w))) \cdot$
$((r \star \sigma \star p_0) \star w) \cdot (((r \star r_0 s_1.\varphi) \star w) \cdot$
$\mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w]) \cdot$ *ϱ.composite-cell* $(p_1 \star w)\ \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
proof −
have $(\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r \star r_0,\ p_1,\ w] =$
$\mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w]$
proof −
have $(\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) =$
$\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w] \cdot \mathrm{a}^{-1}[r,\ r_0,\ p_1 \star w]$
using *pentagon′ ϱ.T0.antipar*(*1*) *comp-assoc* **by** *simp*
moreover have *1*: *seq* $(\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w)(\mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]))$
using *ϱ.T0.antipar*(*1*)
by (*intro seqI hseqI*, *auto*)
ultimately
have $\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w] =$
$((\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w])) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w]$
using *ϱ.T0.antipar*(*1*) *iso-assoc*
invert-side-of-triangle(*2*)
[*of* $(\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w])$
$\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]\ \mathrm{a}^{-1}[r,\ r_0,\ p_1 \star w]$]
by *fastforce*
hence $\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w] =$
$(\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w]$
using *comp-assoc* **by** *presburger*
moreover have *seq* (*inv* $(\mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w))\ \mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]$
using *ϱ.T0.antipar*(*1*) *1* **by** *fastforce*
ultimately show *?thesis*
using *ϱ.T0.antipar*(*1*) *iso-assoc*
invert-side-of-triangle(*1*)
[*of* $\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]\ \mathrm{a}^{-1}[r,\ r_0,\ p_1] \star w$
$\mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w]$]
by *fastforce*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (r \star s \star \vartheta) \cdot ((\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w))) \cdot$
$(((r \star \sigma \star p_0) \star w) \cdot \mathrm{a}^{-1}[r,\ s_1 \star p_0,\ w]) \cdot$
$(r \star r_0 s_1.\varphi \star w) \cdot$ *ϱ.composite-cell* $(p_1 \star w)\ \mathrm{a}^{-1}[r_0,\ p_1,\ w]$

proof −
  have $((r \star r_0 s_1.\varphi) \star w) \cdot \mathrm{a}^{-1}[r,\ r_0 \star p_1,\ w] = \mathrm{a}^{-1}[r,\ s_1 \star p_0,\ w] \cdot (r \star r_0 s_1.\varphi \star w)$
    using *assoc′-naturality* [*of* $r\ r_0 s_1.\varphi\ w$] $r_0 s_1$.*cospan* **by** *auto*
  thus *?thesis*
    using *comp-assoc* **by** *presburger*
qed
also have ... $= (r \star s \star \vartheta) \cdot$
$(\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w) \cdot$
$\mathrm{a}^{-1}[r,\ (s \star s_0) \star p_0,\ w]) \cdot$
$(r \star (\sigma \star p_0) \star w) \cdot (r \star r_0 s_1.\varphi \star w) \cdot$
ϱ.*composite-cell* $(p_1 \star w)\ \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
proof −
  have $((r \star \sigma \star p_0) \star w) \cdot \mathrm{a}^{-1}[r,\ s_1 \star p_0,\ w] =$
$\mathrm{a}^{-1}[r,\ (s \star s_0) \star p_0,\ w] \cdot (r \star (\sigma \star p_0) \star w)$
    using *assoc′-naturality* [*of* $r\ \sigma \star p_0\ w$]
    by (*simp add*: *composable*)
  thus *?thesis*
    using *comp-assoc* **by** *presburger*
qed
also have ... $= (r \star s \star \vartheta) \cdot$
$(r \star (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot$
$((r \star (\sigma \star p_0) \star w) \cdot (r \star r_0 s_1.\varphi \star w) \cdot (r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w])) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot (\varrho \star p_1 \star w)$
proof −
  have $\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w) \cdot \mathrm{a}^{-1}[r,\ (s \star s_0) \star p_0,\ w] =$
$r \star (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot \mathrm{a}[s \star s_0,\ p_0,\ w]$
  proof −
    have $\mathrm{a}[r,\ s,\ (s_0 \star p_0) \star w] \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \star w) \cdot \mathrm{a}^{-1}[r,\ (s \star s_0) \star p_0,\ w] =$
$\{\!|\mathbf{a}[\langle r\rangle,\ \langle s\rangle,\ (\langle s_0\rangle \star \langle p_0\rangle) \star \langle w\rangle] \cdot \mathbf{a}[\langle r\rangle \star \langle s\rangle,\ \langle s_0\rangle \star \langle p_0\rangle,\ \langle w\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle r\rangle,\ \langle s\rangle,\ \langle s_0\rangle \star \langle p_0\rangle] \cdot (\langle r\rangle \star \mathbf{a}[\langle s\rangle,\ \langle s_0\rangle,\ \langle p_0\rangle]) \star \langle w\rangle) \cdot$
$\mathbf{a}^{-1}[\langle r\rangle,\ (\langle s\rangle \star \langle s_0\rangle) \star \langle p_0\rangle,\ \langle w\rangle]|\!\}$
      using α-*def* a′-*def composable* **by** *simp*
    also have ... $= \{\!|\langle r\rangle \star (\langle s\rangle \star \mathbf{a}^{-1}[\langle s_0\rangle,\ \langle p_0\rangle,\ \langle w\rangle]) \cdot \mathbf{a}[\langle s\rangle,\ \langle s_0\rangle,\ \langle p_0\rangle \star \langle w\rangle] \cdot$
$\mathbf{a}[\langle s\rangle \star \langle s_0\rangle,\ \langle p_0\rangle,\ \langle w\rangle]|\!\}$
      using *composable*
      by (*intro E.eval-eqI*, *simp-all*)
    also have ... $= r \star (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot \mathrm{a}[s \star s_0,\ p_0,\ w]$
      using α-*def* a′-*def composable* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
  thus *?thesis*
    using *comp-assoc* **by** *presburger*
qed
also have ... $= (r \star s \star \vartheta) \cdot$
$(r \star (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot$
ϱ.*composite-cell* $(p_1 \star w)$

$(((\sigma \star p_0) \star w) \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w])$
using *assms(3) arrI* ϱ*.T0.antipar(1) whisker-left* **by** *auto*
also have ... $= (r \star (s \star \vartheta) \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\mathrm{a}[s \star s_0,\ p_0,\ w] \cdot ((\sigma \star p_0) \star w)) \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot (\varrho \star p_1 \star w)$
using ϱ*.T0.antipar(1) comp-assoc whisker-left* **by** *auto*
also have ... $= (r \star (s \star \vartheta) \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot (\varrho \star p_1 \star w)$
using *assoc-naturality* [*of* σ p_0 w] *comp-assoc* **by** *simp*
finally show *?thesis*
using *assms(3–4)* **by** *simp*
qed
qed

lemma *composite-is-tabulation*:
shows *tabulation V H* a i *src trg* $(r \star s)$ *tab* $(s_0 \star p_0)$ $(r_1 \star p_1)$
proof
show $\bigwedge u\ \omega.\ [\![\ ide\ u;\ «\omega : dom\ \omega \Rightarrow (r \star s) \star u»\ ]\!] \Longrightarrow$
$\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : (s_0 \star p_0) \star w \Rightarrow u» \wedge$
$«\nu : dom\ \omega \Rightarrow (r_1 \star p_1) \star w» \wedge iso\ \nu \wedge$
$composite\text{-}cell\ w\ \vartheta \cdot \nu = \omega$
proof –
fix $u\ \omega$
assume u: *ide u*
assume ω: $«\omega : dom\ \omega \Rightarrow (r \star s) \star u»$
let $?v = dom\ \omega$
have *1*: $«\mathrm{a}[r,\ s,\ u] \cdot \omega : ?v \Rightarrow r \star s \star u»$
proof –

have $src\ r = trg\ s \wedge src\ s = trg\ u$
by (*metis* ω *arr-cod hseq-char in-homE hcomp-simps(1)*)
thus *?thesis*
using $u\ \omega$ **by** *fastforce*
qed

obtain $w_r\ \vartheta_r\ \nu_r$
where $w_r\vartheta_r\nu_r$: $ide\ w_r \wedge «\vartheta_r : r_0 \star w_r \Rightarrow s \star u» \wedge$
$«\nu_r : ?v \Rightarrow r_1 \star w_r» \wedge iso\ \nu_r \wedge$
$\varrho.composite\text{-}cell\ w_r\ \vartheta_r \cdot \nu_r = \mathrm{a}[r,\ s,\ u] \cdot \omega$
using $u\ \omega$ ϱ*.T1* [*of* $s \star u\ \mathrm{a}[r,\ s,\ u] \cdot \omega$]
by (*metis 1* ϱ*.ide-base* σ*.ide-base arr-cod composable ide-hcomp in-homE match-1 not-arr-null seq-if-composable*)

$$
\begin{tikzcd}
 & & X \arrow[dll, dotted, "w_r"'] \arrow[dll, bend right, "v"'] \arrow[dr, bend left, "u"] \arrow[dl, phantom, "\overset{\nu_r}{\Rightarrow}"] \arrow[d, phantom, "\overset{\theta_r}{\Rightarrow}"] & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[d, phantom, "\overset{\rho}{\Rightarrow}"] & & \\
\mathrm{trg}\ r & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[l, "r"] & & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}
$$

We need some simps, etc., otherwise the subsequent diagram chase is very painful.

have w_r: *ide* w_r
  using $w_r \vartheta_r \nu_r$ **by** *simp*
have [*simp*]: *src* w_r = *src* u
  using $w_r \vartheta_r \nu_r$ ω *1 comp-arr-dom in-homE seqE hcomp-simps*(*1*) *vseq-implies-hpar*(*1*)
  by (*metis src-hcomp*)
have [*simp*]: *trg* w_r = *src* ϱ
  using $w_r \vartheta_r \nu_r$
  by (*metis 1 arrI not-arr-null seqE seq-if-composable*)
have ϑ_r*-in-hom* [*intro*]: «$\vartheta_r : r_0 \star w_r \Rightarrow s \star u$»
  using $w_r \vartheta_r \nu_r$ **by** *simp*
have ϑ_r*-in-hhom* [*intro*]: «ϑ_r : *src* $u \rightarrow$ *trg* s»
  using ϑ_r*-in-hom src-cod* [*of* ϑ_r] *trg-cod* [*of* ϑ_r]
  by (*metis* ‹*src* w_r = *src* u› σ*.leg1-simps*(*4*) *arr-dom in-hhomI in-homE* $r_0 s_1$*.cospan*
    src-hcomp trg-hcomp vconn-implies-hpar(*1*) *vconn-implies-hpar*(*2*))
have [*simp*]: *src* ϑ_r = *src* u **using** ϑ_r*-in-hhom* **by** *auto*
have [*simp*]: *trg* ϑ_r = *trg* s **using** ϑ_r*-in-hhom* **by** *auto*
have [*simp*]: *dom* $\vartheta_r = r_0 \star w_r$ **using** ϑ_r*-in-hom* **by** *blast*
have [*simp*]: *cod* $\vartheta_r = s \star u$ **using** ϑ_r*-in-hom* **by** *blast*
have ν_r*-in-hom* [*intro*]: «ν_r : *?v* $\Rightarrow r_1 \star w_r$» **using** $w_r \vartheta_r \nu_r$ **by** *simp*
have ν_r*-in-hhom* [*intro*]: «ν_r : *src* $u \rightarrow$ *trg* r»
  using ν_r*-in-hom src-dom* [*of* ν_r] *trg-dom* [*of* ν_r]
  by (*metis* ‹*src* w_r = *src* u› ϱ*.leg1-simps*(*4*) *arr-cod in-hhomI in-homE*
    src-hcomp trg-hcomp vconn-implies-hpar(*3*) *vconn-implies-hpar*(*4*))
have [*simp*]: *src* ν_r = *src* u **using** ν_r*-in-hhom* **by** *auto*
have [*simp*]: *trg* ν_r = *trg* r **using** ν_r*-in-hhom* **by** *auto*
have [*simp*]: *dom* ν_r = *?v* **using** ν_r*-in-hom* **by** *auto*
have [*simp*]: *cod* $\nu_r = r_1 \star w_r$ **using** ν_r*-in-hom* **by** *auto*
have *iso-*ν_r: *iso* ν_r **using** $w_r \vartheta_r \nu_r$ **by** *simp*

obtain w_s ϑ_s ν_s
where $w_s \vartheta_s \nu_s$: *ide* w_s $\wedge$ «$\vartheta_s : s_0 \star w_s \Rightarrow u$» $\wedge$ «$\nu_s : r_0 \star w_r \Rightarrow s_1 \star w_s$» $\wedge$ *iso* ν_s $\wedge$
    σ*.composite-cell* w_s $\vartheta_s \cdot \nu_s = \vartheta_r$
  using u $w_r \vartheta_r \nu_r$ σ*.T1* [*of* u ϑ_r] **by** *auto*

have w_s: *ide* w_s
  using $w_s\vartheta_s\nu_s$ **by** *simp*
have [*simp*]: *src* w_s = *src* u
  using $w_s\vartheta_s\nu_s$ *src-cod*
  by (*metis* σ*.leg0-simps*(*2*) σ*.tab-simps*(*2*) ϑ_r*-in-hom arrI hseqI′ ideD*(*1*) *seqE*
    seq-if-composable src-hcomp vconn-implies-hpar(*3*))
have [*simp*]: *trg* w_s = *src* σ
  using $w_s\vartheta_s\nu_s$
  by (*metis* σ*.tab-simps*(*2*) *arr-dom in-homE not-arr-null seq-if-composable*)
have ϑ_s*-in-hom* [*intro*]: «$\vartheta_s : s_0 \star w_s \Rightarrow u$»
  using $w_s\vartheta_s\nu_s$ **by** *simp*
have ϑ_s*-in-hhom* [*intro*]: «ϑ_s : *src* $u \rightarrow$ *src* s»
  using ϑ_s*-in-hom src-cod trg-cod*
  by (*metis* ϑ_r*-in-hom arrI hseqE in-hhom-def seqE vconn-implies-hpar*(*1*)
    vconn-implies-hpar(*3*) $w_s\vartheta_s\nu_s$)
have [*simp*]: *src* ϑ_s = *src* u **using** ϑ_s*-in-hhom* **by** *auto*
have [*simp*]: *trg* ϑ_s = *src* s **using** ϑ_s*-in-hhom* **by** *auto*
have [*simp*]: *dom* $\vartheta_s = s_0 \star w_s$ **using** ϑ_s*-in-hom* **by** *blast*
have [*simp*]: *cod* $\vartheta_s = u$ **using** ϑ_s*-in-hom* **by** *blast*
have ν_s*-in-hom* [*intro*]: «$\nu_s : r_0 \star w_r \Rightarrow s_1 \star w_s$» **using** $w_s\vartheta_s\nu_s$ **by** *simp*
have ν_s*-in-hhom* [*intro*]: «ν_s : *src* $u \rightarrow$ *trg* s»
  using ν_s*-in-hom src-dom trg-cod*
    by (*metis* ‹*src* ϑ_r = *src* u› ‹*trg* ϑ_r = *trg* s› ϑ_r*-in-hom in-hhomI in-homE src-dom*
trg-dom)
have [*simp*]: *src* ν_s = *src* u **using** ν_s*-in-hhom* **by** *auto*
have [*simp*]: *trg* ν_s = *trg* s **using** ν_s*-in-hhom* **by** *auto*
have [*simp*]: *dom* $\nu_s = r_0 \star w_r$ **using** ν_s*-in-hom* **by** *auto*
have [*simp*]: *cod* $\nu_s = s_1 \star w_s$ **using** ν_s*-in-hom* **by** *auto*
have *iso-*ν_s: *iso* ν_s **using** $w_s\vartheta_s\nu_s$ **by** *simp*

obtain w ϑ_t ν_t
where $w\vartheta_t\nu_t$: *ide* $w \wedge$ «$\vartheta_t : p_0 \star w \Rightarrow w_s$» $\wedge$ «$\nu_t : w_r \Rightarrow p_1 \star w$» $\wedge$ *iso* $\nu_t \wedge$
    $(s_1 \star \vartheta_t) \cdot \mathrm{a}[s_1, p_0, w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0, p_1, w] \cdot (r_0 \star \nu_t) = \nu_s$
  using $w_r\vartheta_r\nu_r$ $w_s\vartheta_s\nu_s$ *iso-*ν_s $r_0 s_1.\varphi$*-biuniversal-prop*(*1*) [*of* w_s w_r ν_s] **by** *blast*

$$
\begin{tikzcd}
 & & X \arrow[d, dotted, "w"] \arrow[ddll, bend right, "v"'] \arrow[ddl, "w_r"'] \arrow[ddr, "w_s"] \arrow[ddrr, bend left, "u"] & & \\
 & & \mathrm{src}\ \phi \arrow[dl, "p_1"] \arrow[dr, "p_0"'] \arrow[d, phantom, "\overset{\phi}{\Rightarrow}"] & & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[r, phantom, "\overset{\rho}{\Rightarrow}"] & & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[r, phantom, "\overset{\sigma}{\Rightarrow}"] & \\
\mathrm{trg}\ r & & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[ll, "r"] \arrow[rr, "s"'] & & \mathrm{src}\ s
\end{tikzcd}
$$

Note: w is not necessarily a map.

```
have w: ide w
  using wϑtνt by simp
have [simp]: src w = src u
  using wϑtνt src-cod
by (metis νs-in-hom ‹src νs = src u› arrI seqE src-hcomp src-vcomp vseq-implies-hpar(1))
have [simp]: trg w = src p0
  using wϑtνt
  by (metis νs-in-hom arrI not-arr-null r0s1.φ-simps(2) seqE seq-if-composable)
have ϑt-in-hom [intro]: «ϑt : p0 ⋆ w ⇒ ws»
  using wϑtνt by simp
have ϑt-in-hhom [intro]: «ϑt : src u → src σ»
  using ϑt-in-hom src-cod trg-cod ‹src ws = src u› ‹trg ws = src σ› by fastforce
have [simp]: src ϑt = src u using ϑt-in-hhom by auto
have [simp]: trg ϑt = src σ using ϑt-in-hhom by auto
have [simp]: dom ϑt = p0 ⋆ w using ϑt-in-hom by blast
have  cod ϑt = ws using ϑt-in-hom by blast
have νt-in-hom [intro]: «νt : wr ⇒ p1 ⋆ w» using wϑtνt by simp
have νt-in-hhom [intro]: «νt : src u → src ϱ»
  using νt-in-hom src-dom trg-dom ‹src wr = src u› ‹trg wr = src ϱ› by fastforce
have [simp]: src νt = src u using νt-in-hhom by auto
have [simp]: trg νt = src ϱ using νt-in-hhom by auto
have  dom νt = wr using νt-in-hom by auto
have [simp]: cod νt = p1 ⋆ w using νt-in-hom by auto
have iso-νt: iso νt using wϑtνt by simp

define ϑ where ϑ = ϑs · (s0 ⋆ ϑt) · a[s0, p0, w]
have ϑ: «ϑ : (s0 ⋆ p0) ⋆ w ⇒ u»
proof (unfold ϑ-def, intro comp-in-homI)
  show «a[s0, p0, w] : (s0 ⋆ p0) ⋆ w ⇒ s0 ⋆ p0 ⋆ w»
    using wϑtνt by auto
  show «s0 ⋆ ϑt : s0 ⋆ p0 ⋆ w ⇒ s0 ⋆ ws»
    using wϑtνt by auto
  show «ϑs : s0 ⋆ ws ⇒ u»
    using wsϑsνs by simp
qed
```

define ν **where** $\nu = \mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot (r_1 \star \nu_t) \cdot \nu_r$
have ν: «ν : $?v \Rightarrow (r_1 \star p_1) \star w$»
proof (*unfold* ν*-def*, *intro comp-in-homI*)
  show «ν_r : $?v \Rightarrow r_1 \star w_r$»
    using $w_r\vartheta_r\nu_r$ **by** *blast*
  show «$r_1 \star \nu_t$: $r_1 \star w_r \Rightarrow r_1 \star p_1 \star w$»
    using $w\vartheta_t\nu_t$ **by** (*intro hcomp-in-vhom*, *auto*)
  show «$\mathrm{a}^{-1}[r_1,\ p_1,\ w]$: $r_1 \star p_1 \star w \Rightarrow (r_1 \star p_1) \star w$»
    using $w\vartheta_t\nu_t$ *assoc-in-hom* **by** (*simp add*: ϱ*.T0.antipar*(*1*))
qed
have *iso-*ν: *iso* ν
  using ν $w\vartheta_t\nu_t$ $w_r\vartheta_r\nu_r$ ϱ*.T0.antipar*(*1*)
  by (*unfold* ν*-def*, *intro isos-compose*) *auto*
have $*$: *arr* $((s \star \vartheta_s) \cdot \mathrm{a}[s,\ s_0,\ w_s] \cdot (\sigma \star w_s) \cdot (s_1 \star \vartheta_t) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
    $(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t))$
  using $w_s\vartheta_s\nu_s$ $w\vartheta_t\nu_t$ ϑ_r*-in-hom comp-assoc* **by** *auto*

have $((r \star s) \star \vartheta) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot (tab \star w) \cdot \nu = \omega$
proof −
  have *seq* $(r \star \vartheta_r)$ $(\mathrm{a}[r,\ r_0,\ w_r] \cdot (\varrho \star w_r) \cdot \nu_r)$
    using $w_r\vartheta_r\nu_r$ ϱ*.base-simps*(*2*) *composable* **by** *fastforce*
  hence $\omega = \mathrm{a}^{-1}[r,\ s,\ u] \cdot \varrho$*.composite-cell* w_r $\vartheta_r \cdot \nu_r$
    using $w_r\vartheta_r\nu_r$ *invert-side-of-triangle*(*1*) *iso-assoc*
    by (*metis 1* ϱ*.ide-base* σ*.ide-base arrI assoc′-eq-inv-assoc composable hseq-char′*
      seqE seq-if-composable u vconn-implies-hpar(*2*) *vconn-implies-hpar*(*4*) $w_s\vartheta_s\nu_s$)
  also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot \varrho$*.composite-cell* w_r $(\sigma$*.composite-cell* w_s $\vartheta_s \cdot \nu_s) \cdot \nu_r$
    using $w_s\vartheta_s\nu_s$ **by** *simp*
  also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot \mathrm{a}[s,\ s_0,\ w_s] \cdot$
      $(\sigma \star w_s) \cdot (s_1 \star \vartheta_t) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
      $\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)) \cdot \mathrm{a}[r,\ r_0,\ w_r] \cdot (\varrho \star w_r) \cdot \nu_r$
    using $w\vartheta_t\nu_t$ *comp-assoc* **by** *simp*

Rearrange to create ϑ and ν, leaving *tab* in the middle.

  also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot \mathrm{a}[s,\ s_0,\ w_s] \cdot$
      $((\sigma \star w_s) \cdot (s_1 \star \vartheta_t)) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
      $\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)) \cdot \mathrm{a}[r,\ r_0,\ w_r] \cdot (\varrho \star w_r) \cdot \nu_r$
    using *comp-assoc* **by** *presburger*
  also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (\mathrm{a}[s,\ s_0,\ w_s] \cdot$
      $((s \star s_0) \star \vartheta_t)) \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
      $\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)) \cdot \mathrm{a}[r,\ r_0,\ w_r] \cdot (\varrho \star w_r) \cdot \nu_r$
  proof −
    have $(\sigma \star w_s) \cdot (s_1 \star \vartheta_t) = \sigma \star \vartheta_t$
      using *comp-arr-dom comp-cod-arr interchange*
      by (*metis* ‹*cod* $\vartheta_t = w_s$› σ*.tab-simps*(*1*) σ*.tab-simps*(*4*) *arrI* $w\vartheta_t\nu_t$)
    also have ... $= ((s \star s_0) \star \vartheta_t) \cdot (\sigma \star p_0 \star w)$
      using *comp-arr-dom comp-cod-arr interchange* $w_s\vartheta_s\nu_s$ $w\vartheta_t\nu_t$ σ*.tab-in-hom*
      by (*metis* ‹*dom* $\vartheta_t = p_0 \star w$› σ*.tab-simps*(*5*) *arrI*)
    finally have $(\sigma \star w_s) \cdot (s_1 \star \vartheta_t) = ((s \star s_0) \star \vartheta_t) \cdot (\sigma \star p_0 \star w)$
      by *simp*

thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)) \cdot \mathrm{a}[r,\ r_0,\ w_r] \cdot (\varrho \star w_r) \cdot \nu_r$
using *assoc-naturality* [*of* $s\ s_0\ \vartheta_t$] $w\vartheta_t\nu_t$ *comp-assoc* ‹*cod* $\vartheta_t = w_s$› *arrI* **by** *force*
also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)) \cdot \mathrm{a}[r,\ r_0,\ w_r] \cdot$
$(\varrho \star w_r) \cdot \nu_r$
proof −
have $r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t) =$
$(r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t))$
proof −
have *seq* $((s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t))$
$(\mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t))$
proof −
have *seq* $(s \star \vartheta_s)$
$((s \star s_0 \star \vartheta_t) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t))$
using ‹«$\mathrm{a}[r,\ s,\ u] \cdot \omega$: *dom* $\omega \Rightarrow r \star s \star u$»› *calculation* **by** *blast*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
thus *?thesis*
using *whisker-left* [*of* r $(s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)$
$\mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot (r_0 \star \nu_t)$]
$w_s\vartheta_s\nu_s$ $w\vartheta_t\nu_t$ *comp-assoc*
by *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot ((r \star r_0 \star \nu_t) \cdot \mathrm{a}[r,\ r_0,\ w_r]) \cdot$
$(\varrho \star w_r) \cdot \nu_r$
proof −
have *seq* $(\mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w])$ $(r_0 \star \nu_t)$
using *1* $r_0 s_1.p_1$*-simps* $w\vartheta_t\nu_t$
apply (*intro seqI′ comp-in-homI*) **by** *auto*

hence $r \star (\mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot (r_0 \star \nu_t) =$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot (r \star r_0 \star \nu_t)$
using *whisker-left* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot$
$(((r \star r_0) \star \nu_t) \cdot (\varrho \star w_r)) \cdot \nu_r$
proof −
have $(r \star r_0 \star \nu_t) \cdot \mathrm{a}[r,\ r_0,\ w_r] = \mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot ((r \star r_0) \star \nu_t)$
using *assoc-naturality* [*of r* r_0 ν_t] ν_t*-in-hom* **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (\mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t))) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot$
$(\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $((r \star r_0) \star \nu_t) \cdot (\varrho \star w_r) = \varrho \star \nu_t$
using *comp-arr-dom comp-cod-arr interchange*
by (*metis* ‹*dom* $\nu_t = w_r$› *ϱ.tab-simps*(*1*) *ϱ.tab-simps*(*5*) *arrI* $w\vartheta_t\nu_t$)
also have ... $= (\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t)$
using *comp-arr-dom comp-cod-arr interchange*
by (*metis* ‹*cod* $\nu_t = p_1 \star w$› ‹*trg* ν_t = *src* ϱ› *ϱ.T0.antipar*(*1*) *ϱ.tab-simps*(*1*) *ϱ.tab-simps*(*2*) *ϱ.tab-simps*(*4*) $r_0 s_1$*.base-simps*(*2*) *trg.preserves-reflects-arr trg-hcomp*)
finally have $((r \star r_0) \star \nu_t) \cdot (\varrho \star w_r) = (\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t)$ **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot ((\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w]) \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot$
$(\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $\mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) =$
$((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w]$
proof −
have *seq* $(s \star \vartheta_s)$ $(s \star s_0 \star \vartheta_t)$
using ϑ_s*-in-hom* ϑ_s*-in-hhom* ϑ_t*-in-hom* ϑ_t*-in-hhom 1 calculation* **by** *blast*
moreover have *src r* = *trg* $(s \star \vartheta_s)$
using *composable hseqI* **by** *force*
ultimately
have $\mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star (s \star \vartheta_s) \cdot (s \star s_0 \star \vartheta_t)) =$

$(\mathrm{a}^{-1}[r,\ s,\ u] \cdot (r \star s \star \vartheta_s)) \cdot (r \star s \star s_0 \star \vartheta_t)$
using *whisker-left comp-assoc* **by** *simp*
also have ... $= ((r \star s) \star \vartheta_s) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star w_s] \cdot (r \star s \star s_0 \star \vartheta_t)$
using *assoc-naturality comp-assoc*
by (*metis* ‹*cod* $\vartheta_s = u$› ‹*dom* $\vartheta_s = s_0 \star w_s$› ‹*trg* $\vartheta_s = src\ s$›
ϱ.*base-simps*(*2*−*4*) σ.*base-simps*(*2*−*4*) *arrI assoc'-naturality composable* $w_s\vartheta_s\nu_s$)
also have ... $= (((r \star s) \star \vartheta_s) \cdot ((r \star s) \star s_0 \star \vartheta_t)) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w]$
proof −
have $\mathrm{a}^{-1}[r,\ s,\ s_0 \star w_s] \cdot (r \star s \star s_0 \star \vartheta_t) =$
$((r \star s) \star s_0 \star \vartheta_t) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w]$
using *arrI hseq-char assoc'-naturality* [*of r s* $s_0 \star \vartheta_t$] ‹*cod* $\vartheta_t = w_s$› *composable*
by *auto*
thus *?thesis*
using *comp-assoc* **by** *auto*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w]$
using ϑ-*def* ϑ *whisker-left*
by (*metis* (*full-types*) *arrI cod-comp ide-base seqE seqI*)
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot$
$((r \star \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot \mathrm{a}[s \star s_0,\ p_0,\ w] \cdot$
$((\sigma \star p_0) \star w) \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w])) \cdot$
$\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot (\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] = \mathrm{a}[s \star s_0,\ p_0,\ w] \cdot ((\sigma \star p_0) \star w)$
using *assoc-naturality* [*of* σ p_0 w] **by** (*simp add*: $w\vartheta_t\nu_t$)
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot (r \star (\sigma \star p_0) \star w) \cdot (r \star r_0 s_1.\varphi \star w) \cdot$
$((r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w]) \cdot$
$(\varrho \star p_1 \star w) \cdot (r_1 \star \nu_t) \cdot \nu_r$
using $r_0 s_1.p_1$-*simps* $w\vartheta_t\nu_t$ *whisker-left comp-assoc* **by** *force*
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot (r \star (\sigma \star p_0) \star w) \cdot (r \star r_0 s_1.\varphi \star w) \cdot$
$(\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot (\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]) \cdot$
$(\varrho \star p_1 \star w)) \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $(r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w] =$
$\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot \mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]$
proof −
have *1*: $(r \star \mathrm{a}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) =$

$\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot \mathrm{a}[r \star r_0,\ p_1,\ w]$
using *pentagon*
by (*simp add*: ϱ.*T0*.*antipar*(*1*) *w*)
moreover have *2*: *seq* $\mathrm{a}[r,\ r_0,\ p_1 \star w]\ \mathrm{a}[r \star r_0,\ p_1,\ w]$
using ϱ.*T0*.*antipar*(*1*) *w* **by** *simp*
moreover have *inv* $(r \star \mathrm{a}[r_0,\ p_1,\ w]) = r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using ϱ.*T0*.*antipar*(*1*) *w* **by** *simp*
ultimately have $\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) =$
$((r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w]) \cdot \mathrm{a}[r \star r_0,\ p_1,\ w]$
using ϱ.*T0*.*antipar*(*1*) *w comp-assoc*
invert-side-of-triangle(*1*)
[*of* $\mathrm{a}[r,\ r_0,\ p_1 \star w] \cdot \mathrm{a}[r \star r_0,\ p_1,\ w]\ r \star \mathrm{a}[r_0,\ p_1,\ w]$
$\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w)$]
by *simp*
hence $(r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w] =$
$(\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w)) \cdot \mathrm{a}^{-1}[r \star r_0,\ p_1,\ w]$
using ϱ.*T0*.*antipar*(*1*) *w*
invert-side-of-triangle(*2*)
[*of* $\mathrm{a}[r,\ r_0 \star p_1,\ w] \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w)$
$(r \star \mathrm{a}^{-1}[r_0,\ p_1,\ w]) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w]\ \mathrm{a}[r \star r_0,\ p_1,\ w]$]
using ‹*trg w* = *src* p_0› **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot (r \star (\sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}[r,\ r_0 \star p_1,\ w]) \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot ((\varrho \star p_1) \star w) \cdot$
$\mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $\mathrm{a}^{-1}[r \star r_0,\ p_1,\ w] \cdot (\varrho \star p_1 \star w) = ((\varrho \star p_1) \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *assoc′-naturality* [*of* $\varrho\ p_1\ w$] **by** (*simp add*: ϱ.*T0*.*antipar*(*1*) $w\vartheta_t\nu_t$)
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot ((r \star (\sigma \star p_0) \star w) \cdot \mathrm{a}[r,\ s_1 \star p_0,\ w]) \cdot$
$((r \star r_0 s_1.\varphi) \star w) \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot ((\varrho \star p_1) \star w) \cdot$
$\mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $(r \star r_0 s_1.\varphi \star w) \cdot \mathrm{a}[r,\ r_0 \star p_1,\ w] = \mathrm{a}[r,\ s_1 \star p_0,\ w] \cdot ((r \star r_0 s_1.\varphi) \star w)$
using *assoc-naturality* [*of* $r\ r_0 s_1.\varphi\ w$] $r_0 s_1$.*cospan* $w\vartheta_t\nu_t$ **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed

also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot \mathrm{a}[r,\ (s \star s_0) \star p_0,\ w] \cdot$
$(((r \star \sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi) \star w) \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot$
$((\varrho \star p_1) \star w)) \cdot$
$\mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $(r \star (\sigma \star p_0) \star w) \cdot \mathrm{a}[r,\ s_1 \star p_0,\ w] =$
$\mathrm{a}[r,\ (s \star s_0) \star p_0,\ w] \cdot ((r \star \sigma \star p_0) \star w)$
proof −
have *arr w* $\wedge$ *dom w* $=$ *w* $\wedge$ *cod w* $=$ *w*
using *ide-char w* **by** *blast*
then show *?thesis*
using *assoc-naturality* [*of r* $\sigma \star p_0$ *w*] *composable* **by** *auto*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot$
$(r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot \mathrm{a}[r,\ (s \star s_0) \star p_0,\ w] \cdot$
$((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \star w) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \star w)) \cdot$
$(tab \star w) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot (r_1 \star \nu_t) \cdot \nu_r$
proof −
have $((r \star \sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi) \star w) \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot ((\varrho \star p_1) \star w) =$
$(r \star \sigma \star p_0) \cdot (r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ p_1] \cdot (\varrho \star p_1) \star w$
using *w* ϱ*.T0.antipar*(*1*) *composable whisker-right* **by** *auto*
also have ... $= (((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0]) \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0])) \cdot (r \star \sigma \star p_0)) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ p_1] \cdot (\varrho \star p_1) \star w$
proof −
have $((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0]) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0])) \cdot (r \star \sigma \star p_0) =$
$r \star \sigma \star p_0$
proof −
have $((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0]) \cdot$
$(r \star \mathrm{a}[s,\ s_0,\ p_0])) \cdot (r \star \sigma \star p_0) =$
$((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot ((r \star s \star s_0 \star p_0) \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]))) \cdot (r \star \sigma \star p_0)$
using *comp-assoc-assoc′* **by** (*simp add*: *composable*)
also have ... $= ((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0])) \cdot (r \star \sigma \star p_0)$
using *comp-cod-arr* **by** (*simp add*: *composable*)
also have ... $= ((r \star (s \star s_0) \star p_0)) \cdot (r \star \sigma \star p_0)$
using *whisker-left comp-assoc-assoc′ assoc-in-hom hseqI′*
by (*metis* ϱ*.ide-base* σ*.base-simps*(*2*) σ*.ide-base* σ*.ide-leg0*
σ*.leg0-simps*(*2*−*3*) σ*.leg1-simps*(*3*) $r_0 s_1$*.ide-leg0*
$r_0 s_1$*.leg0-simps*(*2*) $r_0 s_1$*.*p_0*-simps hcomp-simps*(*1*))
also have ... $= r \star \sigma \star p_0$
using *comp-cod-arr*
by (*simp add*: *composable*)

finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= (r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot \mathrm{a}[r,\ s,\ s_0 \star p_0] \cdot$
$(\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0]) \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0]) \cdot (r \star \sigma \star p_0) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ p_1] \cdot (\varrho \star p_1) \star w$
using *comp-assoc* **by** *presburger*
also have ... $= (r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \cdot \mathrm{a}[r,\ s,\ s_0 \star p_0] \cdot tab \star w$
using *tab-def* **by** *simp*
also have ... $= ((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \star w) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \star w) \cdot (tab \star w)$
using *w ϱ.T0.antipar(1) composable comp-assoc whisker-right* **by** *auto*
finally have $((r \star \sigma \star p_0) \star w) \cdot ((r \star r_0 s_1.\varphi) \star w) \cdot (\mathrm{a}[r,\ r_0,\ p_1] \star w) \cdot$
$((\varrho \star p_1) \star w) =$
$((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \star w) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \star w) \cdot (tab \star w)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot ((r \star s) \star \mathrm{a}[s_0,\ p_0,\ w])) \cdot$
$\mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot (tab \star w) \cdot \nu$
proof −
have $\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot (r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot$
$\mathrm{a}[r,\ (s \star s_0) \star p_0,\ w] \cdot ((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \star w) \cdot (\mathrm{a}[r,\ s,\ s_0 \star p_0] \star w) =$
$((r \star s) \star \mathrm{a}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w]$
proof −
have $\mathrm{a}^{-1}[r,\ s,\ s_0 \star p_0 \star w] \cdot (r \star \mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot (r \star \mathrm{a}[s \star s_0,\ p_0,\ w]) \cdot$
$\mathrm{a}[r,\ (s \star s_0) \star p_0,\ w] \cdot ((r \star \mathrm{a}^{-1}[s,\ s_0,\ p_0]) \star w) \cdot$
$(\mathrm{a}[r,\ s,\ s_0 \star p_0] \star w) =$
$\{\!|\mathbf{a}^{-1}[\langle r\rangle,\ \langle s\rangle,\ \langle s_0\rangle \star \langle p_0\rangle \star \langle w\rangle] \cdot (\langle r\rangle \star \mathbf{a}[\langle s\rangle,\ \langle s_0\rangle,\ \langle p_0\rangle \star \langle w\rangle]) \cdot$
$(\langle r\rangle \star \mathbf{a}[\langle s\rangle \star \langle s_0\rangle,\ \langle p_0\rangle,\ \langle w\rangle]) \cdot \mathbf{a}[\langle r\rangle,\ (\langle s\rangle \star \langle s_0\rangle) \star \langle p_0\rangle,\ \langle w\rangle] \cdot$
$((\langle r\rangle \star \mathbf{a}^{-1}[\langle s\rangle,\ \langle s_0\rangle,\ \langle p_0\rangle]) \star \langle w\rangle) \cdot (\mathbf{a}[\langle r\rangle,\ \langle s\rangle,\ \langle s_0\rangle \star \langle p_0\rangle] \star \langle w\rangle)|\!\}$
using *w comp-assoc* a′*-def α-def composable* **by** *simp*
also have ... $= \{\!|((\langle r\rangle \star \langle s\rangle) \star \mathbf{a}[\langle s_0\rangle,\ \langle p_0\rangle,\ \langle w\rangle]) \cdot \mathbf{a}[\langle r\rangle \star \langle s\rangle,\ \langle s_0\rangle \star \langle p_0\rangle,\ \langle w\rangle]|\!\}$
using *w composable*
apply (*intro E.eval-eqI*) **by** *simp-all*
also have ... $= ((r \star s) \star \mathrm{a}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w]$
using *w comp-assoc* a′*-def α-def composable* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *ν-def comp-assoc* **by** *simp*
qed
also have ... $= ((r \star s) \star \vartheta) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot (tab \star w) \cdot \nu$
proof −
have $((r \star s) \star \vartheta_s \cdot (s_0 \star \vartheta_t)) \cdot ((r \star s) \star \mathrm{a}[s_0,\ p_0,\ w]) = (r \star s) \star \vartheta$
using *ϑ-def w whisker-left composable*
by (*metis ϑ arrI ide-base comp-assoc*)
thus *?thesis*

using *comp-assoc* **by** *presburger*
qed
finally show $((r \star s) \star \vartheta) \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w] \cdot (tab \star w) \cdot \nu = \omega$
by *simp*
qed
thus $\exists\, w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : (s_0 \star p_0) \star w \Rightarrow u» \wedge$
$«\nu : dom\ \omega \Rightarrow (r_1 \star p_1) \star w» \wedge iso\ \nu \wedge$
$composite\text{-}cell\ w\ \vartheta \cdot \nu = \omega$
using $w\vartheta_t\nu_t$ ϑ ν *iso-*ν *comp-assoc* **by** *metis*
qed

show $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.$
$[\![\ ide\ w;\ ide\ w';\ «\vartheta : (s_0 \star p_0) \star w \Rightarrow u»;\ «\vartheta' : (s_0 \star p_0) \star w' \Rightarrow u»;$
$«\beta : (r_1 \star p_1) \star w \Rightarrow (r_1 \star p_1) \star w'»;$
$composite\text{-}cell\ w\ \vartheta = composite\text{-}cell\ w'\ \vartheta' \cdot \beta\ ]\!] \Longrightarrow$
$\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = (r_1 \star p_1) \star \gamma \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma)$
proof −
fix $u\ w\ w'\ \vartheta\ \vartheta'\ \beta$
assume w: $ide\ w$
assume w': $ide\ w'$
assume ϑ: $«\vartheta : (s_0 \star p_0) \star w \Rightarrow u»$
assume ϑ': $«\vartheta' : (s_0 \star p_0) \star w' \Rightarrow u»$
assume β: $«\beta : (r_1 \star p_1) \star w \Rightarrow (r_1 \star p_1) \star w'»$
assume *eq*: $composite\text{-}cell\ w\ \vartheta = composite\text{-}cell\ w'\ \vartheta' \cdot \beta$
interpret $uw\vartheta w'\vartheta'\beta$: $uw\vartheta w'\vartheta'\beta\ \ V\ H$ a i $src\ trg$
$‹r \star s›\ tab\ ‹s_0 \star p_0›\ ‹r_1 \star p_1›\ u\ w\ \vartheta\ w'\ \vartheta'\ \beta$
using $w\ w'\ \vartheta\ \vartheta'\ \beta$ *eq composable tab-in-hom comp-assoc*
by *unfold-locales auto*

$$
\begin{tikzcd}
 & & X \arrow[d, "w'"] \arrow[dll, bend right, "w"'] \arrow[ddrr, bend left, "u"] & & \\
 & & \mathrm{src}\ \phi \arrow[dl, "p_1"'] \arrow[dr, "p_0"] \arrow[d, phantom, "\Downarrow\beta"] \arrow[dr, phantom, "\theta'\Nearrow"] & & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[rr, phantom, "\overset{\phi}{\Rightarrow}"] & & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] & \\
\mathrm{trg}\ r & \arrow[l, "r"] \arrow[rr, phantom, "\overset{\rho}{\Rightarrow}"] & \mathrm{src}\ r = \mathrm{trg}\ s & \arrow[l, "s"] \arrow[rr, phantom, "\overset{\sigma}{\Rightarrow}"] & \mathrm{src}\ s
\end{tikzcd}
$$

$$=$$

$$
\begin{tikzcd}
 & & X \arrow[d, "w"] \arrow[ddrr, bend left, "u"] & & \\
 & & \mathrm{src}\ \phi \arrow[dl, "p_1"'] \arrow[dr, "p_0"] \arrow[dr, phantom, "\theta\Nearrow"] & & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[rr, phantom, "\overset{\phi}{\Rightarrow}"] & & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] & \\
\mathrm{trg}\ r & \arrow[l, "r"] \arrow[rr, phantom, "\overset{\rho}{\Rightarrow}"] & \mathrm{src}\ r = \mathrm{trg}\ s & \arrow[l, "s"] \arrow[rr, phantom, "\overset{\sigma}{\Rightarrow}"] & \mathrm{src}\ s
\end{tikzcd}
$$

First apply property $\varrho.T2$ using «$\beta_r : r_1 \star p_1 \star w \Rightarrow r_1 \star p_1 \star w'$» (obtained by composing β with associativities) and "everything to the right" as ϑ_r and $\vartheta_r{}'$. This yields «$\gamma_r : p_1 \star w \Rightarrow p_1 \star w'$». Next, apply property $\varrho.T2$ to obtain «$\gamma_s : p_0 \star w \Rightarrow p_0 \star w'$». For this use «$\vartheta_s : s_0 \star p_0 \star w \Rightarrow u$» and «$\vartheta_s{}' : s_0 \star p_0 \star w'$» obtained by composing ϑ and ϑ' with associativities. We also need «$\beta_s : s_1 \star p_0 \star w \Rightarrow s_1 \star p_0 \star w'$». To get this, transport $r_0 \star \gamma_r$ across φ; we need φ to be an isomorphism in order to do this. Finally, apply the biuniversal property of φ to get «$\gamma : w \Rightarrow w'$» and verify the required equation.

let $?w_r = p_1 \star w$
have w_r: *ide* $?w_r$ **by** *simp*
let $?w_r{}' = p_1 \star w'$
have $w_r{}'$: *ide* $?w_r{}'$ **by** *simp*
define ϑ_r **where** $\vartheta_r = (s \star \vartheta) \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot$
$(\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
have ϑ_r: «$\vartheta_r : r_0 \star ?w_r \Rightarrow s \star u$»
unfolding ϑ_r*-def*
using $\varrho.T0.antipar(1)$ **by** *fastforce*
define $\vartheta_r{}'$ **where** $\vartheta_r{}' = (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$(\sigma \star p_0 \star w') \cdot \mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w']$
have $\vartheta_r{}'$: «$\vartheta_r{}' : r_0 \star ?w_r{}' \Rightarrow s \star u$»
unfolding $\vartheta_r{}'$*-def*
using $\varrho.T0.antipar(1)$ **by** *fastforce*
define β_r **where** $\beta_r = \mathrm{a}[r_1,\ p_1,\ w'] \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$

have β_r: «$\beta_r : r_1 \star ?w_r \Rightarrow r_1 \star ?w_r{}'$»
  unfolding β_r*-def*
  using ϱ*.T0.antipar(1)* **by** *force*
have eq_r: ϱ*.composite-cell* $?w_r\ \vartheta_r = \varrho$*.composite-cell* $?w_r{}'\ \vartheta_r{}' \cdot \beta_r$

$$\begin{tikzcd}
 & & X \arrow[dl, "w_r"'] \arrow[dl, "w'_r"] \arrow[dr, bend left, "u"] \arrow[dd, phantom, "\Downarrow\beta_r"] \arrow[dd, phantom, "\theta'_r \Rightarrow"] & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[d, phantom, "\overset{\rho}{\Rightarrow}"] & & \\
\mathrm{trg}\ r & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[l, "r"] & & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}
=
\begin{tikzcd}
 & X \arrow[dl, "w_r"'] \arrow[dr, bend left, "u"] \arrow[d, phantom, "\theta_r \Rightarrow"] & \\
 & \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[d, phantom, "\overset{\rho}{\Rightarrow}"] & \\
\mathrm{trg}\ r & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[l, "r"] & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}$$

proof −
  have ϱ*.composite-cell* $?w_r\ \vartheta_r = \mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w\ \vartheta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
    using ϑ_r*-def technical* $uw\vartheta w'\vartheta'\beta.uw\vartheta.uw\vartheta$ **by** *blast*
  also have ... $= \mathrm{a}[r,\ s,\ u] \cdot (((r \star s) \star \vartheta') \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w'] \cdot$
      $(tab \star w') \cdot \beta) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
    using *eq comp-assoc* **by** *simp*
  also have ... $= (r \star \vartheta_r{}') \cdot \mathrm{a}[r,\ r_0,\ ?w_r{}'] \cdot (\varrho \star ?w_r{}') \cdot \beta_r$
  proof −
    have $\mathrm{a}[r,\ s,\ u] \cdot ($*composite-cell* $w'\ \vartheta' \cdot \beta) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w] =$
      $\mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w'\ \vartheta' \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
      using *comp-assoc* **by** *presburger*
    also have ... $= (r \star (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
      $(\sigma \star p_0 \star w') \cdot \mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot$
      $\mathrm{a}^{-1}[r_0,\ p_1,\ w']) \cdot$
      $\mathrm{a}[r,\ r_0,\ p_1 \star w'] \cdot (\varrho \star p_1 \star w') \cdot \mathrm{a}[r_1,\ p_1,\ w'] \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
    proof −
      have $\mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w'\ \vartheta' \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w] =$
        $\mathrm{a}[r,\ s,\ u] \cdot$ *composite-cell* $w'\ \vartheta' \cdot$
        $((\mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \mathrm{a}[r_1,\ p_1,\ w']) \cdot \beta) \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
      proof −
        have $(\mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \mathrm{a}[r_1,\ p_1,\ w']) \cdot \beta = \beta$
          using *comp-cod-arr* ϱ*.T0.antipar(1)* β *comp-assoc-assoc'* **by** *simp*

thus *?thesis* **by** *argo*
qed
also have ... $= (\mathrm{a}[r,\ s,\ u] \cdot ((r \star s) \star \vartheta') \cdot \mathrm{a}[r \star s,\ s_0 \star p_0,\ w'] \cdot (tab \star w') \cdot$
$\mathrm{a}^{-1}[r_1,\ p_1,\ w']) \cdot \mathrm{a}[r_1,\ p_1,\ w'] \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using *comp-assoc* **by** *presburger*
also have ... $= ((r \star (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$(\sigma \star p_0 \star w') \cdot \mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w']) \cdot \mathrm{a}[r,\ r_0,\ p_1 \star w'] \cdot (\varrho \star p_1 \star w')) \cdot$
$\mathrm{a}[r_1,\ p_1,\ w'] \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using $\vartheta_r{}'$*-def technical* [*of* w' ϑ' u $?w_r{}'$ $\vartheta_r{}'$] *comp-assoc* **by** *fastforce*
finally show *?thesis*
using *comp-assoc* **by** *simp*
qed
finally show *?thesis*
using $\vartheta_r{}'$*-def* β_r*-def comp-assoc* **by** *auto*
qed
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
have *1*: $\exists!\gamma.\ «\gamma : ?w_r \Rightarrow ?w_r{}'» \wedge \vartheta_r = \vartheta_r{}' \cdot (r_0 \star \gamma) \wedge \beta_r = r_1 \star \gamma$
using eq_r $\varrho.T2$ [*of* $?w_r$ $?w_r{}'$ ϑ_r $s \star u$ $\vartheta_r{}'$ β_r] w_r $w_r{}'$ ϑ_r $\vartheta_r{}'$ β_r **by** *blast*
obtain γ_r **where** γ_r: $«\gamma_r : ?w_r \Rightarrow ?w_r{}'» \wedge \vartheta_r = \vartheta_r{}' \cdot (r_0 \star \gamma_r) \wedge \beta_r = r_1 \star \gamma_r$
using *1* **by** *blast*

let $?w_s = p_0 \star w$
have w_s: *ide* $?w_s$ **by** *simp*
let $?w_s{}' = p_0 \star w'$
have $w_s{}'$: *ide* $?w_s{}'$ **by** *simp*
define ϑ_s **where** $\vartheta_s = \vartheta \cdot \mathrm{a}^{-1}[s_0,\ p_0,\ w]$
have ϑ_s: $«\vartheta_s : s_0 \star p_0 \star w \Rightarrow u»$
using ϑ_s*-def* **by** *auto*
define $\vartheta_s{}'$ **where** $\vartheta_s{}' = \vartheta' \cdot \mathrm{a}^{-1}[s_0,\ p_0,\ w']$
have $\vartheta_s{}'$: $«\vartheta_s{}' : s_0 \star p_0 \star w' \Rightarrow u»$
using $\vartheta_s{}'$*-def* **by** *auto*
define β_s **where** $\beta_s = \mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star \gamma_r) \cdot$
$\mathrm{a}[r_0,\ p_1,\ w] \cdot (inv\ r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
have β_s: $«\beta_s : s_1 \star ?w_s \Rightarrow s_1 \star ?w_s{}'»$
unfolding β_s*-def*
using γ_r $r_0 s_1.\varphi$*-in-hom*(*2*) $r_0 s_1.\varphi$*-uniqueness*(*2*) $\varrho.T0.antipar$(*1*)
apply (*intro comp-in-homI*)
apply *auto*
by *auto*
have eq_s: $\sigma.composite\text{-}cell\ (p_0 \star w)\ (\vartheta \cdot \mathrm{a}^{-1}[s_0,\ p_0,\ w]) =$
$\sigma.composite\text{-}cell\ (p_0 \star w')\ (\vartheta' \cdot \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \beta_s$

$$
\begin{tikzcd}
 & X \arrow[dl, bend right, "w"'] \arrow[d, "w'"] \arrow[ddr, bend left, "u"] & \\
\mathrm{src}\ \phi \arrow[d, "p_0"'] \arrow[dr, "p_1"] \arrow[r, phantom, "\Swarrow \gamma_r"] & \mathrm{src}\ \phi \arrow[d, "p_1"] \arrow[dr, "p_0"] \arrow[r, phantom, "\theta'_s \Nearrow"] & \\
\mathrm{src}\ \sigma \arrow[dr, "s_1"'] \arrow[r, phantom, "\Swarrow \phi^{-1}"] & \mathrm{src}\ \rho \arrow[d, "r_0"] \arrow[r, phantom, "\phi \Nearrow"] & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[r, phantom, "\overset{\sigma}{\Rightarrow}"] & \\
 & \mathrm{src}\ r = \mathrm{trg}\ s & & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}
$$

$$=$$

$$
\begin{tikzcd}
X \arrow[dr, bend right, "w_s"'] \arrow[drr, bend left, "u"] \arrow[r, phantom, "\theta_s \Nearrow"] & & \\
 & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[r, phantom, "\overset{\sigma}{\Rightarrow}"] & \\
\mathrm{src}\ r = \mathrm{trg}\ s & & \mathrm{src}\ s \arrow[ll, "s"]
\end{tikzcd}
$$

proof −
  have σ.*composite-cell* $(p_0 \star w')$ $(\vartheta' \cdot \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \beta_s =$
    $(\vartheta_r{}' \cdot (r_0 \star \gamma_r)) \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
    using β_s*-def* $\vartheta_r{}'$*-def whisker-left comp-assoc* **by** *simp*
  also have ... $= \vartheta_r \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
    using γ_r **by** *simp*
  also have ... $= ((s \star \vartheta) \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w])) \cdot \mathrm{a}[s,\ s_0,\ ?w_s] \cdot (\sigma \star ?w_s) \cdot$
    $\mathrm{a}[s_1,\ p_0,\ w] \cdot ((r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot$
    $\mathrm{a}[r_0,\ p_1,\ w]) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
    using ϑ_r*-def comp-assoc* **by** *simp*
  also have ... $= (s \star \vartheta) \cdot \sigma$.*composite-cell* $(p_0 \star w)\ \mathrm{a}^{-1}[s_0,\ p_0,\ w]$
  proof −
    have $(\sigma \star p_0 \star w) \cdot$
      $\mathrm{a}[s_1,\ p_0,\ w] \cdot ((r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot$
      $\mathrm{a}[r_0,\ p_1,\ w]) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w] =$
      $\sigma \star p_0 \star w$
    proof −
      have $\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot \mathrm{a}[r_0,\ p_1,\ w] = \mathit{cod}\ (\mathit{inv}\ r_0 s_1.\varphi \star w)$
        using $r_0 s_1.\varphi$*-uniqueness(2)* ϱ*.T0.antipar(1) comp-assoc-assoc′* **by** *simp*

Here the fact that φ is a retraction is used.

      moreover have $(r_0 s_1.\varphi \star w) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) = \mathit{cod}\ \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
        using $r_0 s_1.\varphi$*-uniqueness(2) comp-arr-inv′ whisker-right* [*of* $w\ r_0 s_1.\varphi$ *inv* $r_0 s_1.\varphi$]
        by *simp*
      moreover have $\mathrm{a}[s_1,\ p_0,\ w] \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w] = \mathit{dom}\ (\sigma \star p_0 \star w)$
        using $r_0 s_1$*.base-simps(2) hseq-char comp-assoc-assoc′* **by** *auto*

moreover have *hseq* (*inv* $r_0 s_1.\varphi$) w
using $r_0 s_1.\varphi$*-uniqueness*(*2*)
by (*intro hseqI*, *auto*)
moreover have *hseq* σ ($p_0 \star w$)
by (*intro hseqI*, *auto*)
ultimately show *?thesis*
using *comp-arr-dom comp-cod-arr* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... = σ.*composite-cell* ($p_0 \star w$) ($\vartheta \cdot \mathrm{a}^{-1}[s_0, p_0, w]$)
using ϑ_s*-def whisker-left*
by (*metis* σ.*ide-base* ϑ_s *arrI comp-assoc*)
finally show *?thesis* **by** *simp*
qed
hence *2*: $\exists!\gamma$. «γ : *?*$w_s \Rightarrow$ *?*$w_s{}'$» $\wedge\ \vartheta_s = \vartheta_s{}' \cdot (s_0 \star \gamma) \wedge \beta_s = s_1 \star \gamma$
using σ.*T2* [*of ?*w_s *?*$w_s{}'$ ϑ_s *u* $\vartheta_s{}'$ β_s] w_s $w_s{}'$ ϑ_s $\vartheta_s{}'$ β_s
by (*metis* $\vartheta_s{}'$*-def* ϑ_s*-def*)
obtain γ_s **where** γ_s: «γ_s : *?*$w_s \Rightarrow$ *?*$w_s{}'$» $\wedge\ \vartheta_s = \vartheta_s{}' \cdot (s_0 \star \gamma_s) \wedge \beta_s = s_1 \star \gamma_s$
using *2* **by** *blast*

have *eq*$_t$: $(s_1 \star \gamma_s) \cdot \mathrm{a}[s_1, p_0, w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0, p_1, w] =$
$(s_1 \star$ *?*$w_s{}') \cdot \mathrm{a}[s_1, p_0, w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0, p_1, w'] \cdot (r_0 \star \gamma_r)$

$$\begin{tikzcd}
 & X \arrow[d, "w'"] \arrow[dl, bend right, "w_r"'] \arrow[dl, phantom, "\Downarrow\gamma_r"] & \\
\mathrm{src}\ \rho \arrow[dr, "r_0"'] \arrow[rr, phantom, "\overset{\phi}{\Rightarrow}"] & \mathrm{src}\ \phi \arrow[dl, "p_1"] \arrow[dr, "p_0"] & \mathrm{src}\ \sigma \arrow[dl, "s_1"] \\
 & \mathrm{src}\ r = \mathrm{trg}\ s &
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
 & X \arrow[d, "w"] \arrow[dr, bend left, "w'_s"] \arrow[dr, phantom, "\gamma_s\Nearrow"] & \\
\mathrm{src}\ \rho \arrow[dr, "r_0"'] \arrow[rr, phantom, "\overset{\phi}{\Rightarrow}"] & \mathrm{src}\ \phi \arrow[dl, "p_1"] \arrow[dr, "p_0"] & \mathrm{src}\ \sigma \arrow[dl, "s_1"] \\
 & \mathrm{src}\ r = \mathrm{trg}\ s &
\end{tikzcd}$$

proof −
have $(s_1 \star$ *?*$w_s{}') \cdot \mathrm{a}[s_1, p_0, w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0, p_1, w'] \cdot (r_0 \star \gamma_r) =$
$\beta_s \cdot \mathrm{a}[s_1, p_0, w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0, p_1, w]$
proof −
have $\beta_s \cdot \mathrm{a}[s_1, p_0, w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0, p_1, w] =$
$(\mathrm{a}[s_1, p_0, w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0, p_1, w']) \cdot (r_0 \star \gamma_r) \cdot$
$\mathrm{a}[r_0, p_1, w] \cdot ((\mathit{inv}\ r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[s_1, p_0, w] \cdot$
$\mathrm{a}[s_1, p_0, w]) \cdot (r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[r_0, p_1, w]$
using β_s*-def comp-assoc* **by** *metis*
also have ... = $(\mathrm{a}[s_1, p_0, w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0, p_1, w']) \cdot (r_0 \star \gamma_r)$
proof −

have $(r_0 \star \gamma_r) \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot ((inv\ r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[s_1,\ p_0,\ w] \cdot$
$\mathrm{a}[s_1,\ p_0,\ w]) \cdot (r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] =$
$r_0 \star \gamma_r$
proof −
have $(r_0 \star \gamma_r) \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot ((inv\ r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[s_1,\ p_0,\ w] \cdot$
$\mathrm{a}[s_1,\ p_0,\ w]) \cdot (r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w] =$
$(r_0 \star \gamma_r) \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot ((inv\ r_0 s_1.\varphi \star w) \cdot (r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using *$r_0 s_1.\varphi$-uniqueness(2) comp-assoc-assoc′ comp-cod-arr* **by** *simp*

also have ... $= (r_0 \star \gamma_r) \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using *$r_0 s_1.\varphi$-uniqueness(2) comp-inv-arr′ ϱ.T0.antipar(1)*
whisker-right [of w inv $r_0 s_1.\varphi$ $r_0 s_1.\varphi$] comp-cod-arr
by *simp*
also have ... $= r_0 \star \gamma_r$
proof −
have *hseq r_0 γ_r*
using *β_s β_s-def* **by** *blast*
thus *?thesis*
using *comp-assoc-assoc′ comp-arr-dom*
by (*metis* (*no-types*) *γ_r ϱ.ide-leg0 comp-assoc-assoc′(1) hcomp-simps(3)*
hseq-char ide-char in-homE $r_0 s_1$.ide-leg1 $r_0 s_1.p_1$-simps w w_r)
qed
finally show *?thesis* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... $= \mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star \gamma_r)$
using *comp-assoc* **by** *presburger*
also have ... $= (s_1 \star ?w_s{}') \cdot$
$\mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star \gamma_r)$
proof −
have $(s_1 \star ?w_s{}') \cdot \mathrm{a}[s_1,\ p_0,\ w'] = \mathrm{a}[s_1,\ p_0,\ w']$
using *comp-cod-arr* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
finally show *?thesis* **by** *simp*
qed
also have ... $= (s_1 \star \gamma_s) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using γ_s **by** *simp*
finally show *?thesis* **by** *simp*
qed
have *3*: $\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \gamma_s = (p_0 \star w') \cdot (p_0 \star \gamma) \wedge p_1 \star \gamma = \gamma_r$
using *w w′ $w_s{}'$ w_r γ_r γ_s eq_t*
$r_0 s_1.\varphi$-biuniversal-prop(2) [of $?w_s{}'$ $?w_r$ w w′ γ_s $p_0 \star w'$ γ_r]
by *blast*
obtain γ **where** γ: «$\gamma : w \Rightarrow w'$» $\wedge\ \gamma_s = (p_0 \star w') \cdot (p_0 \star \gamma) \wedge p_1 \star \gamma = \gamma_r$
using *3* **by** *blast*

show $\exists!\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = (r_1 \star p_1) \star \gamma \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma)$
proof −
  have $\exists\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = (r_1 \star p_1) \star \gamma \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma)$
  proof −
    have $\vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma)$
    proof −
      have $\vartheta' \cdot ((s_0 \star p_0) \star \gamma) = (\vartheta_s{}' \cdot \mathrm{a}[s_0,\ p_0,\ w']) \cdot ((s_0 \star p_0) \star \gamma)$
        using *$\vartheta_s{}'$-def comp-arr-dom comp-assoc comp-assoc-assoc′(2)* **by** *auto*
      also have ... $= (\vartheta_s{}' \cdot (s_0 \star p_0 \star \gamma)) \cdot \mathrm{a}[s_0,\ p_0,\ w]$
        using *assoc-naturality* [*of* s_0 p_0 γ] *comp-assoc*
        by (*metis* γ γ_r *σ.leg0-simps(4−5)* r_0s_1*.leg0-simps(4−5)*
          r_0s_1*.leg1-simps(3) hseqE in-homE leg0-simps(2)*)
      also have ... $= \vartheta_s \cdot \mathrm{a}[s_0,\ p_0,\ w]$
        by (*metis* γ γ_s *arrI comp-ide-arr* $w_s{}'$)
      also have ... $= \vartheta$
        using *ϑ_s-def comp-assoc comp-arr-dom comp-assoc-assoc′* **by** *simp*
      finally show *?thesis* **by** *simp*
    qed
    moreover have $\beta = (r_1 \star p_1) \star \gamma$
    proof −
      have $\beta = \mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \beta_r \cdot \mathrm{a}[r_1,\ p_1,\ w]$
      proof −
        have $\mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \beta_r \cdot \mathrm{a}[r_1,\ p_1,\ w] =$
          $(\mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \mathrm{a}[r_1,\ p_1,\ w']) \cdot \beta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w] \cdot \mathrm{a}[r_1,\ p_1,\ w]$
          using *β_r-def comp-assoc* **by** *simp*
        also have ... $= \beta$
          using *comp-arr-dom comp-cod-arr*
          by (*metis* ϱ*.ide-leg1* r_0s_1*.ide-leg1 comp-assoc-assoc′(2) hseqE ideD(1)*
            uwϑw′ϑ′β.β-simps(1) uwϑw′ϑ′β.β-simps(4−5) leg1-simps(2) w w' w_r $w_r{}'$)
        finally show *?thesis* **by** *simp*
      qed
      also have ... $= \mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot (r_1 \star \gamma_r) \cdot \mathrm{a}[r_1,\ p_1,\ w]$
        using γ_r **by** *simp*
      also have ... $= \mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \mathrm{a}[r_1,\ p_1,\ w'] \cdot ((r_1 \star p_1) \star \gamma)$
        using *assoc-naturality* [*of* r_1 p_1 γ]
        by (*metis* γ γ_r *ϱ.ide-leg1* r_0s_1*.leg1-simps(5−6) hseqE*
          ide-char in-homE leg1-simps(2))
      also have ... $= (\mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot \mathrm{a}[r_1,\ p_1,\ w']) \cdot ((r_1 \star p_1) \star \gamma)$
        using *comp-assoc* **by** *presburger*
      also have ... $= (r_1 \star p_1) \star \gamma$
        using *comp-cod-arr*
        by (*metis* ϱ*.ide-leg1* r_0s_1*.ide-leg1 calculation comp-assoc-assoc′(2) comp-ide-arr*
          hseqE ideD(1) ide-cod local.uwϑw′ϑ′β.β-simps(1) local.uwϑw′ϑ′β.β-simps(5)
          w' $w_r{}'$)
      finally show *?thesis* **by** *simp*
    qed
    ultimately show $\exists\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \beta = (r_1 \star p_1) \star \gamma \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma)$
      using γ **by** *blast*
  qed

moreover have $\bigwedge\gamma'.$ «$\gamma' : w \Rightarrow w'$» $\wedge\ \beta = (r_1 \star p_1) \star \gamma' \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma')$
$\Longrightarrow \gamma' = \gamma$
proof −
fix γ'
assume γ': «$\gamma' : w \Rightarrow w'$» $\wedge\ \beta = (r_1 \star p_1) \star \gamma' \wedge \vartheta = \vartheta' \cdot ((s_0 \star p_0) \star \gamma')$
show $\gamma' = \gamma$
proof −
let $?P_r = \lambda\gamma.$ «$\gamma : ?w_r \Rightarrow ?w_r{}'$» $\wedge\ \vartheta_r = \vartheta_r{}' \cdot (r_0 \star \gamma) \wedge \beta_r = r_1 \star \gamma$
let $?P_s = \lambda\gamma.$ «$\gamma : ?w_s \Rightarrow ?w_s{}'$» $\wedge\ \vartheta_s = \vartheta_s{}' \cdot (s_0 \star \gamma) \wedge \beta_s = s_1 \star \gamma$
let $?\gamma_r{}' = p_1 \star \gamma'$
let $?\gamma_s{}' = p_0 \star \gamma'$
let $?P_t = \lambda\gamma.$ «$\gamma : w \Rightarrow w'$» $\wedge\ \gamma_s = (p_0 \star w') \cdot (p_0 \star \gamma) \wedge p_1 \star \gamma = \gamma_r$
have *hseq* $p_0\ \gamma'$
proof (*intro hseqI*)
show «$\gamma' : src\ \vartheta \rightarrow src\ p_0$»
using γ'
by (*metis hseqE hseqI′ in-hhom-def uwϑw′ϑ′β.β-simps*(*1*) *src-hcomp*
src-vcomp leg0-simps(*2*) *leg1-simps*(*3*)
uwϑw′ϑ′β.uwϑ.ϑ-simps(*1*) *vseq-implies-hpar*(*1*))
show «$p_0 : src\ p_0 \rightarrow src\ s_0$» **by** *simp*
qed
hence *hseq* $p_1\ \gamma'$
using *hseq-char* **by** *simp*
have «$?\gamma_r{}' : ?w_r \Rightarrow ?w_r{}'$»
using γ' **by** *auto*
moreover have $\vartheta_r = \vartheta_r{}' \cdot (r_0 \star ?\gamma_r{}')$
proof −

Note that ϑ_r is the composite of "everything to the right" of $\varrho \star p_1 \star w$, and similarly for $\vartheta_r{}'$. We can factor ϑ_r as $(s \star \vartheta) \cdot X\ w$, where $X\ w$ is a composite of σ and φ. We can similarly factor $\vartheta_r{}'$ as $(s \star \vartheta') \cdot X\ w'$. Then $\vartheta_r{}' \cdot (r_0 \star p_1 \star \gamma') = (s \star \vartheta') \cdot X\ w' \cdot (r_0 \star p_1 \star \gamma')$, which equals $(s \star \vartheta') \cdot (s \star (s_0 \star p_0) \star p_1 \star \gamma') \cdot X\ w = \vartheta_r$.

let $?X = \lambda w.\ (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w]) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w] \cdot (\sigma \star p_0 \star w) \cdot$
$\mathrm{a}[s_1,\ p_0,\ w] \cdot (r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
have $\vartheta_r{}' \cdot (r_0 \star ?\gamma_r{}') = (s \star \vartheta') \cdot ?X\ w' \cdot (r_0 \star ?\gamma_r{}')$
using $\vartheta_r{}'$*-def comp-assoc* **by** *simp*
also have ... $= (s \star \vartheta') \cdot (s \star (s_0 \star p_0) \star \gamma') \cdot ?X\ w$
proof −
have $(s \star \vartheta') \cdot ((s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot (\sigma \star p_0 \star w') \cdot$
$\mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w']) \cdot (r_0 \star p_1 \star \gamma') =$
$(s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot (\sigma \star p_0 \star w') \cdot$
$\mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0 s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star p_1 \star \gamma')$
using *comp-assoc* **by** *presburger*
also have ... $= (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$(\sigma \star p_0 \star w') \cdot \mathrm{a}[s_1,\ p_0,\ w'] \cdot ((r_0 s_1.\varphi \star w') \cdot$
$((r_0 \star p_1) \star \gamma')) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using *assoc′-naturality* [*of* $r_0\ p_1\ \gamma'$] *comp-assoc*
by (*metis* γ' ‹«$p_1 \star \gamma' : p_1 \star w \Rightarrow p_1 \star w'$»› *ϱ.T0.antipar*(*1*)
ϱ.leg0-in-hom(*2*) $r_0 s_1$*.leg1-simps*(*4*−*6*)

$r_0 s_1$.*base-simps*(*2*) *hcomp-in-vhomE in-homE trg-hcomp*)
also have ... $= (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$(\sigma \star p_0 \star w') \cdot (\mathrm{a}[s_1,\ p_0,\ w'] \cdot ((s_1 \star p_0) \star \gamma')) \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
proof −
have $(r_0 s_1.\varphi \star w') \cdot ((r_0 \star p_1) \star \gamma') = r_0 s_1.\varphi \star \gamma'$
using γ' *interchange* [*of* $r_0 s_1.\varphi\ r_0 \star p_1\ w'\ \gamma'$] *comp-arr-dom comp-cod-arr*
by *auto*
also have ... $= ((s_1 \star p_0) \star \gamma') \cdot (r_0 s_1.\varphi \star w)$
using γ' *interchange* ‹*hseq* $p_0\ \gamma'$› *comp-arr-dom comp-cod-arr*
by (*metis comp-arr-ide* $r_0 s_1.\varphi$*-simps*(*1*,*5*) *seqI'*
$uw\vartheta w'\vartheta'\beta.uw\vartheta.w$*-in-hom*(*2*) *w*)
finally have $(r_0 s_1.\varphi \star w') \cdot ((r_0 \star p_1) \star \gamma') =$
$((s_1 \star p_0) \star \gamma') \cdot (r_0 s_1.\varphi \star w)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot \mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$((\sigma \star p_0 \star w') \cdot (s_1 \star p_0 \star \gamma')) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using γ' *assoc-naturality* [*of* $s_1\ p_0\ \gamma'$] *comp-assoc*
by (*metis* σ*.leg1-simps*(*2*) σ*.leg1-simps*(*3*,*5*−*6*) $r_0 s_1$*.leg0-simps*(*4*−*5*)
hcomp-in-vhomE hseqE in-homE $uw\vartheta w'\vartheta'\beta.\beta$*-simps*(*1*)
leg0-in-hom(*2*) *leg1-simps*(*3*))
also have ... $= (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot (\mathrm{a}[s,\ s_0,\ p_0 \star w'] \cdot$
$((s \star s_0) \star p_0 \star \gamma')) \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
proof −
have $(\sigma \star p_0 \star w') \cdot (s_1 \star p_0 \star \gamma') = \sigma \star p_0 \star \gamma'$
using γ' *interchange* [*of* $\sigma\ s_1\ p_0 \star w'\ p_0 \star \gamma'$]
whisker-left ‹*hseq* $p_0\ \gamma'$›*comp-arr-dom comp-cod-arr*
by (*metis* σ*.tab-simps*(*1*) σ*.tab-simps*(*4*) *hcomp-simps*(*4*) *in-homE*
$r_0 s_1$*.leg0-simps*(*5*))
also have ... $= ((s \star s_0) \star p_0 \star \gamma') \cdot (\sigma \star p_0 \star w)$
using γ' *interchange* [*of* $s \star s_0\ \sigma\ p_0 \star \gamma'\ p_0 \star w$]
whisker-left comp-arr-dom comp-cod-arr ‹*hseq* $p_0\ \gamma'$›
by (*metis* σ*.tab-simps*(*1*) σ*.tab-simps*(*5*) *hcomp-simps*(*3*) *in-homE*
$r_0 s_1$*.leg0-simps*(*4*))
finally have $(\sigma \star p_0 \star w') \cdot (s_1 \star p_0 \star \gamma') =$
$((s \star s_0) \star p_0 \star \gamma') \cdot (\sigma \star p_0 \star w)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (s \star \vartheta') \cdot (s \star \mathrm{a}^{-1}[s_0,\ p_0,\ w']) \cdot ((s \star s_0 \star p_0 \star \gamma') \cdot$
$\mathrm{a}[s,\ s_0,\ p_0 \star w]) \cdot (\sigma \star p_0 \star w) \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot$
$(r_0 s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using γ' *assoc-naturality* [*of* $s\ s_0\ p_0 \star \gamma'$] ‹*hseq* $p_0\ \gamma'$› **by** *force*

```
  also have ... = (s ⋆ ϑ′) · ((s ⋆ a⁻¹[s₀, p₀, w′]) · (s ⋆ s₀ ⋆ p₀ ⋆ γ′)) ·
                  a[s, s₀, p₀ ⋆ w] · (σ ⋆ p₀ ⋆ w) · a[s₁, p₀, w] ·
                  (r₀s₁.φ ⋆ w) · a⁻¹[r₀, p₁, w]
    using comp-assoc by presburger
  also have ... = (s ⋆ ϑ′) · ((s ⋆ (s₀ ⋆ p₀) ⋆ γ′) · (s ⋆ a⁻¹[s₀, p₀, w])) ·
                  a[s, s₀, p₀ ⋆ w] · (σ ⋆ p₀ ⋆ w) · a[s₁, p₀, w] ·
                  (r₀s₁.φ ⋆ w) · a⁻¹[r₀, p₁, w]
  proof −
    have (s ⋆ a⁻¹[s₀, p₀, w′]) · (s ⋆ s₀ ⋆ p₀ ⋆ γ′) =
          (s ⋆ (s₀ ⋆ p₀) ⋆ γ′) · (s ⋆ a⁻¹[s₀, p₀, w])
    proof −
      have (s ⋆ a⁻¹[s₀, p₀, w′]) · (s ⋆ s₀ ⋆ p₀ ⋆ γ′) =
            s ⋆ a⁻¹[s₀, p₀, w′] · (s₀ ⋆ p₀ ⋆ γ′)
      proof −
        have seq a⁻¹[s₀, p₀, w′] (s₀ ⋆ p₀ ⋆ γ′)
        proof

          show «s₀ ⋆ p₀ ⋆ γ′ : s₀ ⋆ p₀ ⋆ w ⇒ s₀ ⋆ p₀ ⋆ w′»
            using γ′
            by (intro hcomp-in-vhom, auto)
          show «a⁻¹[s₀, p₀, w′] : s₀ ⋆ p₀ ⋆ w′ ⇒ (s₀ ⋆ p₀) ⋆ w′»
            by auto
        qed
        thus ?thesis
          using w w′ γ′ whisker-left by simp
      qed
      also have ... = s ⋆ ((s₀ ⋆ p₀) ⋆ γ′) · a⁻¹[s₀, p₀, w]
        using γ′ ‹hseq p₀ γ′› assoc′-naturality [of s₀ p₀ γ′] by fastforce
      also have ... = (s ⋆ (s₀ ⋆ p₀) ⋆ γ′) · (s ⋆ a⁻¹[s₀, p₀, w])
      proof −
        have seq ((s₀ ⋆ p₀) ⋆ γ′) a⁻¹[s₀, p₀, w]
        proof

          show «a⁻¹[s₀, p₀, w] : s₀ ⋆ p₀ ⋆ w ⇒ (s₀ ⋆ p₀) ⋆ w»
            by auto
          show «(s₀ ⋆ p₀) ⋆ γ′ : (s₀ ⋆ p₀) ⋆ w ⇒ (s₀ ⋆ p₀) ⋆ w′»
            using γ′ by (intro hcomp-in-vhom, auto)
        qed
        thus ?thesis
          using w w′ γ′ whisker-left by simp
      qed
      finally show ?thesis by blast
    qed
    thus ?thesis by simp
  qed
  also have ... = (s ⋆ ϑ′) · (s ⋆ (s₀ ⋆ p₀) ⋆ γ′) · ?X w
    using comp-assoc by presburger
  finally show ?thesis by simp
qed
```

also have ... $= \vartheta_r$
using ϑ_r*-def* γ' *uw*ϑ*w*$'\vartheta'\beta$*.uw*ϑ*.*ϑ*-simps*(*1*) *whisker-left* σ*.ide-base comp-assoc*
by *simp*
finally show *?thesis* **by** *simp*
qed
moreover have $\beta_r = r_1 \star ?\gamma_r{}'$
proof −
have $\beta_r = \mathrm{a}[r_1,\ p_1,\ w'] \cdot ((r_1 \star p_1) \star \gamma') \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$
using β_r*-def* γ' **by** *simp*
also have ... $= \mathrm{a}[r_1,\ p_1,\ w'] \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w'] \cdot (r_1 \star p_1 \star \gamma')$
using γ' *assoc'-naturality*
by (*metis* ϱ*.leg1-simps*(*5*−*6*) r_0s_1*.leg1-simps*(*5*−*6*)
hcomp-in-vhomE hseqE in-homE uwϑ*w*$'\vartheta'\beta$*.*β*-simps*(*1*) *leg1-in-hom*(*2*))
also have ... $= (\mathrm{a}[r_1,\ p_1,\ w'] \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w']) \cdot (r_1 \star p_1 \star \gamma')$
using *comp-assoc* **by** *presburger*
also have ... $= r_1 \star p_1 \star \gamma'$
using *comp-cod-arr*
by (*metis* (*no-types*, *lifting*) β_r ϱ*.ide-leg1* r_0s_1*.ide-leg1 arrI calculation*
comp-assoc-assoc'(*1*) *comp-ide-arr ide-hcomp hseq-char'*
ideD(*1*) *seq-if-composable hcomp-simps*(*2*) *leg1-simps*(*2*) w' $w_r{}'$)
finally show *?thesis* **by** *simp*
qed
ultimately have $P_r{}'$: $?P_r$ $?\gamma_r{}'$
by *simp*
have eq_r: $\gamma_r = ?\gamma_r{}'$
using *1* γ_r $P_r{}'$ **by** *blast*
have «$?\gamma_s{}'$: $?w_s \Rightarrow ?w_s{}'$»
using γ' **by** *auto*
moreover have $\vartheta_s = \vartheta_s{}' \cdot (s_0 \star ?\gamma_s{}')$
using γ' ‹*hseq* p_0 γ'› σ*.leg0-simps*(*2*,*4*−*5*) σ*.leg1-simps*(*3*) $\vartheta_s{}'$*-def* ϑ_s*-def*
assoc'-naturality hseqE in-homE comp-assoc r_0s_1*.leg0-simps*(*4*−*5*)
r_0s_1*.*p_0*-simps*
by *metis*
moreover have $\beta_s = s_1 \star ?\gamma_s{}'$
proof −
have $\mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0s_1.\varphi \star w') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star \gamma_r) \cdot$
$\mathrm{a}[r_0,\ p_1,\ w] \cdot (inv\ r_0s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w] =$
$\mathrm{a}[s_1,\ p_0,\ w'] \cdot (r_0s_1.\varphi \star w') \cdot (\mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star p_1 \star \gamma')) \cdot$
$\mathrm{a}[r_0,\ p_1,\ w] \cdot (inv\ r_0s_1.\varphi \star w) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
using eq_r *comp-assoc* **by** *simp*
also have ... $= \mathrm{a}[s_1,\ p_0,\ w'] \cdot ((r_0s_1.\varphi \star w') \cdot ((r_0 \star p_1) \star \gamma')) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot (inv\ r_0s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[s_1,\ p_0,\ w]$
proof −
have $\mathrm{a}^{-1}[r_0,\ p_1,\ w'] \cdot (r_0 \star p_1 \star \gamma') = ((r_0 \star p_1) \star \gamma') \cdot \mathrm{a}^{-1}[r_0,\ p_1,\ w]$
using γ' *assoc'-naturality* ‹*hseq* p_1 γ'›
by (*metis* ϱ*.leg0-simps*(*2*,*4*−*5*) ϱ*.leg1-simps*(*3*)
r_0s_1*.leg1-simps*(*5*−*6*) *hseqE in-homE leg1-simps*(*2*))
thus *?thesis*

using *comp-assoc* **by** *presburger*
qed
also have ... $= (\mathrm{a}[s_1,\ p_0,\ w'] \cdot ((s_1 \star p_0) \star \gamma')) \cdot (r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot \mathrm{a}[r_0,\ p_1,\ w] \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) \cdot$
$\mathrm{a}^{-1}[s_1,\ p_0,\ w]$
proof −
have $(r_0 s_1.\varphi \star w') \cdot ((r_0 \star p_1) \star \gamma') = r_0 s_1.\varphi \star \gamma'$
using γ' *interchange* [*of* $r_0 s_1.\varphi\ r_0 \star p_1\ w'\ \gamma'$]
comp-arr-dom comp-cod-arr
by *auto*
also have ... $= ((s_1 \star p_0) \star \gamma') \cdot (r_0 s_1.\varphi \star w)$
using γ' *interchange* ‹*hseq* $p_0\ \gamma'$› *comp-arr-dom comp-cod-arr*
by (*metis in-homE* $r_0 s_1.\varphi$*-simps*(*1*,*5*))
finally have $(r_0 s_1.\varphi \star w') \cdot ((r_0 \star p_1) \star \gamma') =$
$((s_1 \star p_0) \star \gamma') \cdot (r_0 s_1.\varphi \star w)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (s_1 \star ?\gamma_s{}') \cdot \mathrm{a}[s_1,\ p_0,\ w] \cdot ((r_0 s_1.\varphi \star w) \cdot (\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot$
$\mathrm{a}[r_0,\ p_1,\ w]) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w)) \cdot \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
proof −
have $\mathrm{a}[s_1,\ p_0,\ w'] \cdot ((s_1 \star p_0) \star \gamma') = (s_1 \star ?\gamma_s{}') \cdot \mathrm{a}[s_1,\ p_0,\ w]$
using γ' *assoc-naturality* [*of* $s_1\ p_0\ \gamma'$] ‹*hseq* $p_0\ \gamma'$› **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= s_1 \star ?\gamma_s{}'$
proof −
have $\mathrm{a}^{-1}[r_0,\ p_1,\ w] \cdot \mathrm{a}[r_0,\ p_1,\ w] = \mathit{cod}\ (\mathit{inv}\ r_0 s_1.\varphi \star w)$
using $r_0 s_1.\varphi$*-uniqueness*(*2*) *ϱ.T0.antipar*(*1*) *comp-assoc-assoc′*
by *simp*

Here the fact that φ is a retraction is used.

moreover have $(r_0 s_1.\varphi \star w) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) = \mathit{cod}\ \mathrm{a}^{-1}[s_1,\ p_0,\ w]$
using $r_0 s_1.\varphi$*-uniqueness*(*2*) *comp-arr-inv′*
whisker-right [*of* $w\ r_0 s_1.\varphi\ \mathit{inv}\ r_0 s_1.\varphi$]
by *simp*
moreover have $\mathit{cod}\ (\mathit{inv}\ r_0 s_1.\varphi \star w) \cdot (\mathit{inv}\ r_0 s_1.\varphi \star w) = \mathit{inv}\ r_0 s_1.\varphi \star w$
using β_s*-def* β_s
by (*meson arrI comp-cod-arr seqE*)
ultimately show *?thesis*
using γ' ‹*hseq* $p_0\ \gamma'$› *comp-arr-dom comp-cod-arr comp-assoc-assoc′*
whisker-left [*of* $s_1\ p_0 \star \gamma'\ p_0 \star w$] *whisker-left* [*of* p_0]
by (*metis σ.ide-leg1 assoc′-simps*(*1*) *hseqE ideD*(*1*) *in-homE* $r_0 s_1.$*ide-leg0*
$r_0 s_1.p_0$*-simps w* w_s)
qed
finally show *?thesis*
using β_s*-def* **by** *simp*

```
          qed
          ultimately have P_s': ?P_s ?γ_s'
            by simp
          have eq_s: γ_s = ?γ_s'
            using 2 γ_s P_s' by blast
          have ?P_t γ'
            using γ' comp-cod-arr ‹«p_0 ⋆ γ' : p_0 ⋆ w ⇒ p_0 ⋆ w'»› eq_r eq_s by auto
          thus γ' = γ
            using 3 γ by blast
        qed
      qed
      ultimately show ?thesis by blast
    qed
  qed
qed

end

sublocale composite-tabulation-in-maps ⊆
          tabulation V H a i src trg ‹r ⋆ s› tab ‹s_0 ⋆ p_0› ‹r_1 ⋆ p_1›
  using composite-is-tabulation by simp

sublocale composite-tabulation-in-maps ⊆
          tabulation-in-maps V H a i src trg ‹r ⋆ s› tab ‹s_0 ⋆ p_0› ‹r_1 ⋆ p_1›
  using T0.is-map ϱ.leg1-is-map ϱ.T0.antipar(2) composable ϱ.leg1-is-map ϱ.T0.antipar
  apply unfold-locales
   apply simp
  apply (intro left-adjoints-compose)
  by auto
```

2.3.6 The Classifying Category of Maps

We intend to show that if B is a bicategory of spans, then B is biequivalent to $Span(Maps(B))$, for a specific category $Maps(B)$ derived from B. The category $Maps(B)$ is constructed in this section as the "classifying category" of maps of B, which has the same objects as B and which has as 1-cells the isomorphism classes of maps of B. We show that, if B is a bicategory of spans, then $Maps(B)$ has pullbacks.

```
locale maps-category =
  B: bicategory-of-spans
begin

  no-notation B.in-hhom (‹«- : - → -»›)
  no-notation B.in-hom (‹«- : - →_B -»›)
  notation B.in-hhom (‹«- : - →_B -»›)
  notation B.in-hom (‹«- : - ⇒_B -»›)
  notation B.isomorphic (infix ‹≅_B› 50)
  notation B.iso-class (‹⟦-⟧_B›)
```

I attempted to modularize the construction here, by refactoring "classifying category"

out as a separate locale, but it ended up causing extra work because to apply it we first need to obtain the full sub-bicategory of 2-cells between maps, then construct its classifying category, and then we have to re-prove everything about it, to get rid of any mention of the sub-bicategory construction. So the construction is being done here as a "one-off" special case construction, with the necessary properties proved just once.

The "hom-categories" of $Maps(C)$ have as arrows the isomorphism classes of maps of B.

abbreviation *Hom*
where $Hom\ a\ b \equiv \{F.\ \exists f.\ «f : a \rightarrow_B b» \wedge B.is\text{-}left\text{-}adjoint\ f \wedge F = [\![f]\!]_B\}$

lemma *in-HomD*:
assumes $F \in Hom\ a\ b$
shows $F \neq \{\}$
and $B.is\text{-}iso\text{-}class\ F$
and $f \in F \Longrightarrow B.ide\ f$
and $f \in F \Longrightarrow «f : a \rightarrow_B b»$
and $f \in F \Longrightarrow B.is\text{-}left\text{-}adjoint\ f$
and $f \in F \Longrightarrow F = [\![f]\!]_B$
proof −
  show $F \neq \{\}$
    using *assms B.ide-in-iso-class B.left-adjoint-is-ide B.iso-class-is-nonempty* **by** *auto*
  show *1*: $B.is\text{-}iso\text{-}class\ F$
    using *assms B.is-iso-classI B.left-adjoint-is-ide* **by** *fastforce*
  show $f \in F \Longrightarrow B.ide\ f$
    using *assms 1 B.iso-class-memb-is-ide* **by** *blast*
  obtain f' **where** f': $«f' : a \rightarrow_B b» \wedge B.is\text{-}left\text{-}adjoint\ f' \wedge F = [\![f']\!]_B$
    using *assms* **by** *auto*
  show $f \in F \Longrightarrow «f : a \rightarrow_B b»$
    using *assms f′ B.iso-class-def B.isomorphic-implies-hpar* **by** *auto*
  show $f \in F \Longrightarrow B.is\text{-}left\text{-}adjoint\ f$
    using *assms f′ B.iso-class-def B.left-adjoint-preserved-by-iso* [*of f′*] **by** *auto*
  show $f \in F \Longrightarrow F = [\![f]\!]_B$
    by (*metis B.adjoint-pair-antipar*(*1*) *f′ B.ide-in-iso-class B.is-iso-classI*
      B.iso-class-elems-isomorphic B.iso-class-eqI)
qed

definition *Comp*
where $Comp\ G\ F \equiv \{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge g \in G \wedge g \star f \cong_B h)\}$

lemma *in-CompI* [*intro*]:
assumes $B.is\text{-}iso\text{-}class\ F$ **and** $B.is\text{-}iso\text{-}class\ G$
and $f \in F$ **and** $g \in G$ **and** $g \star f \cong_B h$
shows $h \in Comp\ G\ F$
  unfolding *Comp-def*
  using *assms* **by** *auto*

lemma *in-CompE* [*elim*]:

assumes $h \in$ *Comp G F*
and $\bigwedge f\ g.\ [\![\ B.\textit{is-iso-class}\ F;\ B.\textit{is-iso-class}\ G;\ f \in F;\ g \in G;\ g \star f \cong_B h\ ]\!] \Longrightarrow T$
shows T
  using *assms Comp-def* **by** *auto*

lemma *is-iso-class-Comp*:
assumes *Comp G F* $\neq \{\}$
shows *B.is-iso-class* (*Comp G F*)
proof −
  obtain h **where** h: $h \in$ *Comp G F*
    using *assms* **by** *auto*
  have *ide-h*: *B.ide h*
    using *h Comp-def B.isomorphic-implies-hpar*(*2*) **by** *auto*
  obtain *f g* **where** *fg*: *B.is-iso-class F* $\wedge$ *B.is-iso-class G* $\wedge$ $f \in F \wedge g \in G \wedge g \star f \cong_B h$
    using *h Comp-def* **by** *auto*
  have *Comp G F* $= [\![g \star f]\!]_B \wedge$ *B.ide* $(g \star f)$
  proof (*intro conjI*)
    show *B.ide* $(g \star f)$
      using *fg B.iso-class-memb-is-ide B.isomorphic-implies-ide*(*1*) **by** *auto*
    show *Comp G F* $= [\![g \star f]\!]_B$
    proof
      show $[\![g \star f]\!]_B \subseteq$ *Comp G F*
        unfolding *Comp-def B.iso-class-def*
        using *fg* **by** *auto*
      show *Comp G F* $\subseteq [\![g \star f]\!]_B$
      proof
        fix h'
        assume h': $h' \in$ *Comp G F*
        obtain $f'\ g'$ **where** $f'g'$: $f' \in F \wedge g' \in G \wedge g' \star f' \cong_B h'$
          using h' *Comp-def* **by** *auto*
        have *1*: $f' \cong_B f \wedge g' \cong_B g$
          using $f'g'$ *fg B.iso-class-elems-isomorphic* **by** *auto*
        moreover have *B.ide f* $\wedge$ *B.ide* $f' \wedge$ *B.ide g* $\wedge$ *B.ide* g'
          using *1 B.isomorphic-implies-hpar* **by** *auto*
        ultimately have $g' \star f' \cong_B g \star f$
          using $f'g'$ *fg B.hcomp-isomorphic-ide B.hcomp-ide-isomorphic*
            B.isomorphic-transitive B.isomorphic-implies-hpar
          by (*meson B.hseqE B.ideD*(*1*))
        hence $h' \cong_B g \star f$
          using $f'g'$ *B.isomorphic-symmetric B.isomorphic-transitive* **by** *blast*
        thus $h' \in$ *B.iso-class* $(g \star f)$
          using *B.iso-class-def B.isomorphic-symmetric* **by** *simp*
      qed
    qed
  qed
  thus *?thesis*
    using *assms B.is-iso-class-def B.ide-in-iso-class* **by** *auto*
qed

lemma *Comp-extensionality*:
assumes $Comp\ G\ F \neq \{\}$
shows *B.is-iso-class F* **and** *B.is-iso-class G*
and $F \neq \{\}$ **and** $G \neq \{\}$
  using *assms Comp-def* **by** *auto*

lemma *Comp-eqI* [*intro*]:
assumes $h \in Comp\ G\ F$ **and** $h' \in Comp\ G'\ F'$ **and** $h \cong_B h'$
shows $Comp\ G\ F = Comp\ G'\ F'$
proof −
  obtain $f\ g$ **where** *fg*: $f \in F \wedge g \in G \wedge g \star f \cong_B h$
    using *assms comp-def* **by** *auto*
  obtain $f'\ g'$ **where** $f'g'$: $f' \in F' \wedge\ \ g' \in G' \wedge g' \star f' \cong_B h'$
    using *assms* **by** *auto*
  have $h \in Comp\ G\ F \cap Comp\ G'\ F'$
  by (*meson IntI assms in-CompE in-CompI B.isomorphic-symmetric B.isomorphic-transitive*)
  hence $Comp\ G\ F \cap Comp\ G'\ F' \neq \{\}$
    by *auto*
  thus *?thesis*
    using *assms is-iso-class-Comp*
    by (*metis empty-iff B.iso-class-eq*)
qed

lemma *Comp-eq-iso-class-memb*:
assumes $h \in Comp\ G\ F$
shows $Comp\ G\ F = [\![h]\!]_B$
proof
  show $Comp\ G\ F \subseteq [\![h]\!]_B$
  proof
    fix h'
    assume h': $h' \in Comp\ G\ F$
    obtain $f\ g$ **where** *fg*: $f \in F \wedge g \in G \wedge g \star f \cong_B h$
      using *assms* **by** *auto*
    obtain $f'\ g'$ **where** $f'g'$: $f' \in F \wedge g' \in G \wedge g' \star f' \cong_B h'$
      using h' **by** *auto*
    have $f \cong_B f' \wedge g \cong_B g'$
      using *assms fg f'g' in-HomD*(*6*) *B.iso-class-elems-isomorphic* **by** *auto*
    moreover have $B.ide\ f \wedge B.ide\ f' \wedge B.ide\ g \wedge B.ide\ g'$
      using *assms fg f'g' in-HomD* [*of F*] *in-HomD* [*of G*]
      by (*meson calculation B.isomorphic-implies-ide*(*1*) *B.isomorphic-implies-ide*(*2*))
    moreover have $src\ g = trg\ f \wedge src\ g = trg\ f' \wedge src\ g' = trg\ f \wedge src\ g' = trg\ f'$
      using *fg f'g'*
      by (*metis B.seq-if-composable calculation*(*1*) *B.ideD*(*1*)
        B.isomorphic-implies-hpar(*1*) *B.isomorphic-implies-hpar*(*3*) *B.not-arr-null*)
    ultimately have $g \star f \cong_B g' \star f'$
      using *fg f'g' B.hcomp-ide-isomorphic B.hcomp-isomorphic-ide B.isomorphic-transitive*
      by *metis*
    thus $h' \in [\![h]\!]_B$
      using *fg f'g' B.isomorphic-symmetric B.isomorphic-transitive B.iso-class-def* [*of h*]

by *blast*
qed
show $[\![h]\!]_B \subseteq$ *Comp G F*
proof (*unfold B.iso-class-def Comp-def*)
obtain $f\ g$ **where** *1*: $f \in F \wedge g \in G \wedge g \star f \cong_B h$
using *assms in-HomD Comp-def*
by (*meson in-CompE*)
show $\{h'.\ B.isomorphic\ h\ h'\} \subseteq$
$\{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge (\exists f\ g.\ f \in F \wedge g \in G \wedge g \star f \cong_B h)\}$
using *assms 1 B.isomorphic-transitive* **by** *blast*
qed
qed

interpretation *concrete-category* ‹*Collect B.obj*› *Hom B.iso-class* ‹λ- - -. *Comp*›
proof
show $\bigwedge a.\ a \in Collect\ B.obj \Longrightarrow [\![a]\!]_B \in Hom\ a\ a$
by (*metis* (*mono-tags*, *lifting*) *B.ide-in-hom*(*1*) *mem-Collect-eq B.objE*
B.obj-is-self-adjoint(*1*))
show $\bigwedge a\ b\ c\ F\ G.\ [\![\ a \in Collect\ B.obj;\ b \in Collect\ B.obj;\ c \in Collect\ B.obj;$
$F \in Hom\ a\ b;\ G \in Hom\ b\ c\ ]\!] \Longrightarrow Comp\ G\ F \in Hom\ a\ c$
proof −
fix $a\ b\ c\ F\ G$
assume *a*: $a \in Collect\ B.obj$ **and** *b*: $b \in Collect\ B.obj$ **and** *c*: $c \in Collect\ B.obj$
and *F*: $F \in Hom\ a\ b$ **and** *G*: $G \in Hom\ b\ c$
obtain f **where** *f*: «$f : a \rightarrow_B b$» $\wedge$ *B.is-left-adjoint* $f \wedge F = [\![f]\!]_B$
using *F* **by** *blast*
obtain g **where** *g*: «$g : b \rightarrow_B c$» $\wedge$ *B.is-left-adjoint* $g \wedge G = [\![g]\!]_B$
using *G* **by** *blast*
have $\{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B h)\} =$
$[\![g \star f]\!]_B$
proof
show $\{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B h)\}$
$\subseteq [\![g \star f]\!]_B$
proof
fix h
assume $h \in \{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B h)\}$
hence *h*: $B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B h)$
by *simp*
show $h \in [\![g \star f]\!]_B$
proof −
obtain $f'\ g'$ **where** $f'g'$: $g' \in G \wedge f' \in F \wedge g' \star f' \cong_B h$
using *h* **by** *auto*
obtain φ **where** φ: «$\varphi : f \Rightarrow_B f'$» $\wedge$ *B.iso* φ
using $f\ f'g'$ *F B.iso-class-def* **by** *auto*
obtain ψ **where** ψ: «$\psi : g \Rightarrow_B g'$» $\wedge$ *B.iso* ψ

using $g\ f'g'\ G\ B.iso\text{-}class\text{-}def$ **by** *auto*
have *1*: «$\psi \star \varphi : g \star f \Rightarrow_B g' \star f'$»
using $f\ g\ \varphi\ \psi$ **by** *auto*
moreover have $B.iso\ (\psi \star \varphi)$
using $f\ g\ \varphi\ \psi$ *1* $B.iso\text{-}hcomp$ **by** *auto*
ultimately show *?thesis*
using $f'g'\ B.iso\text{-}class\text{-}def\ B.isomorphic\text{-}def$ **by** *auto*
qed
qed
show $[\![g \star f]\!]_B \subseteq \{h.\ B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B h)\}$
using $f\ g\ B.iso\text{-}class\text{-}def\ B.isomorphic\text{-}reflexive\ B.left\text{-}adjoint\text{-}is\text{-}ide\ B.is\text{-}iso\text{-}classI$
by *blast*
qed
hence *1*: $\bigwedge gf.\ gf \in B.iso\text{-}class\ (g \star f) \Longrightarrow$
$B.is\text{-}iso\text{-}class\ F \wedge B.is\text{-}iso\text{-}class\ G \wedge$
$(\exists f\ g.\ f \in F \wedge$ «$f : a \rightarrow_B b$» $\wedge\ g \in G \wedge$ «$g : b \rightarrow_B c$» $\wedge\ g \star f \cong_B gf)$
by *blast*
show $Comp\ G\ F \in Hom\ a\ c$
proof −
have gf: $B.is\text{-}left\text{-}adjoint\ (g \star f)$
by (*meson* $f\ g\ B.hseqE\ B.hseqI\ B.left\text{-}adjoints\text{-}compose$)
obtain gf' **where** gf': $B.adjoint\text{-}pair\ (g \star f)\ gf'$
using gf **by** *blast*
hence $[\![g \star f]\!]_B = Comp\ G\ F$
using *1* $Comp\text{-}eq\text{-}iso\text{-}class\text{-}memb\ B.ide\text{-}in\text{-}iso\text{-}class\ B.left\text{-}adjoint\text{-}is\text{-}ide$ **by** *blast*
thus *?thesis*
using $f\ g\ gf'$ **by** *blast*
qed
qed
show $\bigwedge a\ b\ F.\ [\![\ a \in Collect\ B.obj;\ F \in Hom\ a\ b\ ]\!] \Longrightarrow Comp\ F\ [\![a]\!]_B = F$
proof −
fix $a\ b\ F$
assume a: $a \in Collect\ B.obj$
assume F: $F \in Hom\ a\ b$
obtain f **where** f: «$f : a \rightarrow_B b$» $\wedge\ B.is\text{-}left\text{-}adjoint\ f \wedge F = [\![f]\!]_B$
using F **by** *auto*
have $*$: $\bigwedge f'.\ f' \in F \Longrightarrow$ «$f' : a \rightarrow_B b$» $\wedge\ B.ide\ f' \wedge f \cong_B f'$
using $f\ B.iso\text{-}class\text{-}def$ **by** *force*
show $Comp\ F\ [\![a]\!]_B = F$
proof
show $Comp\ F\ [\![a]\!]_B \subseteq F$
proof
fix h
assume $h \in Comp\ F\ [\![a]\!]_B$
hence h: $\exists f'\ a'.\ f' \in F \wedge a' \in [\![a]\!]_B \wedge f' \star a' \cong_B h$
unfolding $Comp\text{-}def$ **by** *auto*
obtain $f'\ a'$ **where** $f'a'$: $f' \in F \wedge a' \in [\![a]\!]_B \wedge f' \star a' \cong_B h$
using h **by** *auto*

```
      have B.isomorphic f h
      proof −
        have B.isomorphic f (f ⋆ a)
          using f B.iso-runit' [of f] B.isomorphic-def B.left-adjoint-is-ide
          by blast
        also have f ⋆ a ≅B f' ⋆ a
          using f f'a' B.iso-class-def B.hcomp-isomorphic-ide
          apply (elim conjE B.in-hhomE) by auto
        also have f' ⋆ a ≅B f' ⋆ a'
          using f'a' ∗ [of f'] B.iso-class-def B.hcomp-ide-isomorphic by auto
        also have f' ⋆ a' ≅B h
          using f'a' by simp
        finally show ?thesis by blast
      qed
      thus h ∈ F
        using f B.iso-class-def by simp
    qed
    show F ⊆ Comp F ⟦a⟧B
    proof
      fix h
      assume h: h ∈ F
      have f ∈ F
        using f B.iso-class-def B.right-adjoint-determines-left-up-to-iso by auto
      moreover have a ∈ B.iso-class a
        using a B.iso-class-def B.isomorphic-reflexive by auto
      moreover have f ⋆ a ≅B h
      proof −
        have f ⋆ a ≅B f
          using f B.iso-runit [of f] B.isomorphic-def B.left-adjoint-is-ide by blast
        also have f ≅B h
          using h ∗ by simp
        finally show ?thesis by blast
      qed
      ultimately show h ∈ Comp F ⟦a⟧B
        unfolding Comp-def
        using a f F B.is-iso-classI B.left-adjoint-is-ide by auto
    qed
  qed
qed
show ⋀a b F. ⟦ b ∈ Collect B.obj; F ∈ Hom a b ⟧ ⟹ Comp ⟦b⟧B F = F
proof −
  fix a b F
  assume b: b ∈ Collect B.obj
  assume F: F ∈ Hom a b
  obtain f where f: «f : a →B b» ∧ B.is-left-adjoint f ∧ F = ⟦f⟧B
    using F by auto
  have ∗: ⋀f'. f' ∈ F ⟹ «f' : a →B b» ∧ B.ide f' ∧ f ≅B f'
    using f B.iso-class-def by force
  show Comp ⟦b⟧B F = F
```

```
  proof
    show Comp ⟦b⟧_B F ⊆ F
    proof
      fix h
      assume h ∈ Comp ⟦b⟧_B F
      hence h: ∃ b' f'. b' ∈ ⟦b⟧_B ∧ f' ∈ F ∧ b' ⋆ f' ≅_B h
        unfolding Comp-def by auto
      obtain b' f' where b'f': b' ∈ ⟦b⟧_B ∧ f' ∈ F ∧ b' ⋆ f' ≅_B h
        using h by auto
      have f ≅_B h
      proof −
        have f ≅_B b ⋆ f
          using f B.iso-lunit' [of f] B.isomorphic-def B.left-adjoint-is-ide
          by blast
        also have ... ≅_B b ⋆ f'
          using f b'f' B.iso-class-def B.hcomp-ide-isomorphic
          by (elim conjE B.in-hhomE, auto)
        also have ... ≅_B b' ⋆ f'
          using b'f' ∗ [of f'] B.iso-class-def B.hcomp-isomorphic-ide by auto
        also have ... ≅_B h
          using b'f' by simp
        finally show ?thesis by blast
      qed
      thus h ∈ F
        using f B.iso-class-def by simp
    qed
    show F ⊆ Comp ⟦b⟧_B F
    proof
      fix h
      assume h: h ∈ F
      have f ∈ F
        using f B.iso-class-def B.right-adjoint-determines-left-up-to-iso by auto
      moreover have b ∈ B.iso-class b
        using b B.iso-class-def B.isomorphic-reflexive by auto
      moreover have b ⋆ f ≅_B h
      proof −
        have b ⋆ f ≅_B f
          using f B.iso-lunit [of f] B.isomorphic-def B.left-adjoint-is-ide
          by blast
        also have f ≅_B h
          using h ∗ by simp
        finally show ?thesis by blast
      qed
      ultimately show h ∈ Comp ⟦b⟧_B F
        unfolding Comp-def
        using b f F B.is-iso-classI B.left-adjoint-is-ide by auto
    qed
  qed
qed
```

```
show ⋀a b c d F G H.
      ⟦ a ∈ Collect B.obj; b ∈ Collect B.obj; c ∈ Collect B.obj; d ∈ Collect B.obj;
        F ∈ Hom a b; G ∈ Hom b c; H ∈ Hom c d ⟧ ⟹
      Comp H (Comp G F) = Comp (Comp H G) F
proof −
  fix a b c d F G H
  assume F: F ∈ Hom a b and G: G ∈ Hom b c and H: H ∈ Hom c d
  show Comp H (Comp G F) = Comp (Comp H G) F
  proof
    show Comp H (Comp G F) ⊆ Comp (Comp H G) F
    proof
      fix x
      assume x: x ∈ Comp H (Comp G F)
      obtain f g h gf
      where 1: B.is-iso-class F ∧ B.is-iso-class G ∧ B.is-iso-class H ∧
               h ∈ H ∧ g ∈ G ∧ f ∈ F ∧ gf ∈ Comp G F ∧ g ⋆ f ≅B gf ∧ h ⋆ gf ≅B x
        using x unfolding Comp-def by blast
      have hgf: B.ide f ∧ B.ide g ∧ B.ide h
        using 1 F G H B.isomorphic-implies-hpar in-HomD B.left-adjoint-is-ide
        by (metis (mono-tags, lifting))
      have h ⋆ g ⋆ f ≅B x
        by (metis 1 B.hcomp-ide-isomorphic B.hseqE B.ide-char'
            B.isomorphic-implies-hpar(4) B.isomorphic-implies-ide(1)
            B.isomorphic-transitive hgf)
      moreover have (h ⋆ g) ⋆ f ≅B h ⋆ g ⋆ f
        using 1 hgf B.iso-assoc B.assoc-in-hom B.isomorphic-def
        by (metis B.hseq-char B.ideD(1−3) B.isomorphic-implies-hpar(1)
            B.trg-hcomp calculation)
      moreover have hg: «h ⋆ g : b →B d»
        using G H 1 in-HomD(4) by blast
      moreover have h ⋆ g ∈ Comp H G
        unfolding Comp-def
        using 1 hgf F G H in-HomD [of F a b] in-HomD [of G b c] in-HomD [of H c d]
              B.isomorphic-reflexive B.hcomp-ide-isomorphic B.hseqI'
        by (metis (no-types, lifting) B.hseqE B.hseqI mem-Collect-eq)
      ultimately show x ∈ Comp (Comp H G) F
        by (metis 1 B.isomorphic-transitive emptyE in-CompI is-iso-class-Comp)
    qed
    show Comp (Comp H G) F ⊆ Comp H (Comp G F)
    proof
      fix x
      assume x: x ∈ Comp (Comp H G) F
      obtain f g h hg
      where 1: B.is-iso-class F ∧ B.is-iso-class G ∧ B.is-iso-class H ∧
               h ∈ H ∧ g ∈ G ∧ f ∈ F ∧ hg ∈ Comp H G ∧ h ⋆ g ≅B hg ∧ hg ⋆ f ≅B x
        using x unfolding Comp-def by blast
      have hgf: B.ide f ∧ B.ide g ∧ B.ide h ∧ src h = trg g ∧ src g = trg f
        using 1 F G H in-HomD B.left-adjoint-is-ide
        by (metis (no-types, lifting) B.hseq-char' B.ideD(1) B.ide-dom
```

B.in-homE B.isomorphic-def B.isomorphic-symmetric B.seqI′
B.seq-if-composable B.src-dom B.src-hcomp B.vseq-implies-hpar(1))
have *2*: $(h \star g) \star f \cong_B x$
by (*meson 1 B.hcomp-isomorphic-ide B.hseqE B.ideD(1) B.isomorphic-implies-ide(1)*
B.isomorphic-symmetric B.isomorphic-transitive hgf)
moreover have $(h \star g) \star f \cong_B h \star g \star f$
using *hgf B.iso-assoc B.assoc-in-hom B.isomorphic-def* **by** *auto*
moreover have $g \star f \in$ *Comp G F*
using *1 F G hgf B.isomorphic-reflexive* **by** *blast*
ultimately show $x \in$ *Comp H* (*Comp G F*)
using *1 hgf is-iso-class-Comp* [*of G F*] *B.isomorphic-reflexive* [*of* $g \star f$]
apply (*intro in-CompI*)
apply *auto*[*6*]
apply *simp*
apply *auto*[*1*]
by (*meson B.isomorphic-symmetric B.isomorphic-transitive*)
qed
qed
qed
qed

lemma *is-concrete-category*:
shows *concrete-category* (*Collect B.obj*) *Hom B.iso-class* (λ- - -. *Comp*)
..

sublocale *concrete-category* ‹*Collect B.obj*› *Hom B.iso-class* ‹λ- - -. *Comp*›
using *is-concrete-category* **by** *simp*

abbreviation *comp* (**infixr** ‹⊙› *55*)
where *comp* ≡ *COMP*
notation *in-hom* (‹«- : - → -»›)
no-notation *B.in-hom* (‹«- : - $\rightarrow_B$ -»›)

lemma *Map-memb-in-hhom*:
assumes «$F : A \rightarrow B$» **and** $f \in$ *Map F*
shows «f : *Dom A* $\rightarrow_B$ *Dom B*»
proof −
have «f : *Dom F* $\rightarrow_B$ *Cod F*»
using *assms arr-char* [*of F*] *in-HomD* [*of Map F Dom F Cod F*] **by** *blast*
moreover have *Dom F* = *Dom A*
using *assms* **by** *auto*
moreover have *Cod F* = *Dom B*
using *assms* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma *MkArr-in-hom′*:
assumes *B.is-left-adjoint f* **and** «$f : a \rightarrow_B b$»
shows «*MkArr a b* $[\![f]\!]_B$: *MkIde a* → *MkIde b*»

using *assms MkArr-in-hom* **by** *blast*

The isomorphisms in $Maps(B)$ are the isomorphism classes of equivalence maps in B.

lemma *iso-char*:
shows $iso\ F \longleftrightarrow arr\ F \wedge (\forall f.\ f \in Map\ F \longrightarrow B.equivalence\text{-}map\ f)$
proof
  assume F: $iso\ F$
  have $\bigwedge f.\ f \in Map\ F \Longrightarrow B.equivalence\text{-}map\ f$
  proof −
    fix f
    assume f: $f \in Map\ F$
    obtain G **where** G: *inverse-arrows* $F\ G$
      using F **by** *auto*
    obtain g **where** g: $g \in Map\ G$
      using G *arr-char* [*of* G] *in-HomD*(*1*) **by** *blast*
    have f: $f \in Map\ F \wedge «f : Dom\ F \rightarrow_B Cod\ F» \wedge B.ide\ f \wedge B.is\text{-}left\text{-}adjoint\ f$
      by (*metis* (*mono-tags*, *lifting*) F *iso-is-arr is-concrete-category*
        concrete-category.arr-char f *in-HomD*(*2*,*4*−*5*) *B.iso-class-memb-is-ide*)
    have g: $g \in Map\ G \wedge «g : Cod\ F \rightarrow_B Dom\ F» \wedge B.ide\ g \wedge B.is\text{-}left\text{-}adjoint\ g$
      by (*metis* (*no-types*, *lifting*) F G *Cod-cod Cod-dom arr-inv cod-inv dom-inv*
        inverse-unique iso-is-arr is-concrete-category concrete-category.Map-in-Hom
        g *in-HomD*(*2*,*4*−*5*) *B.iso-class-memb-is-ide*)
    have $src\ (g \star f) \cong_B g \star f$
    proof −
      have $g \star f \in Map\ (G \odot F)$
      proof −
        have *1*: $f \in Map\ F \wedge g \in Map\ G \wedge g \star f \cong_B g \star f$
          using F G f g *B.isomorphic-reflexive* **by** *force*
        have *2*: $Dom\ G = Cod\ F \wedge Cod\ G = Dom\ F$
          using F G *arr-char*
          by (*metis* (*no-types*, *lifting*) *Cod.simps*(*1*) *Cod-dom arr-inv*
            cod-char comp-inv-arr dom-inv inverse-unique
            iso-is-arr is-concrete-category concrete-category.Cod-comp)
        have $g \star f \in Comp\ (Map\ G)\ (Map\ F)$
          using *1* F *iso-is-arr Map-in-Hom* [*of* F] *in-HomD*(*2*)
          apply (*intro in-CompI*, *auto*)
        proof −
          show *B.is-iso-class* (*Map* G)
            using G *iso-is-arr Map-in-Hom* [*of* G] *in-HomD*(*2*) [*of Map* G] **by** *blast*
        qed
        thus *?thesis*
          using F G f g *comp-char* [*of* G F] **by** *auto*
      qed
      moreover have $Dom\ F \in Map\ (G \odot F)$
        by (*metis* (*no-types*, *lifting*) F G *Map-dom comp-inv-arr iso-is-arr*
          g *B.ide-in-iso-class B.in-hhomE B.objE*)
      moreover have $Map\ (G \odot F) = [\![Dom\ F]\!]_B$
        by (*simp add*: F G *comp-inv-arr iso-is-arr*)
      moreover have $Dom\ F = src\ (g \star f)$

```
      using f g by fastforce
    ultimately show ?thesis
      using f g B.iso-class-elems-isomorphic B.is-iso-classI
      by (metis B.hseqI B.ide-src)
  qed
  moreover have f ⋆ g ≅B trg f
  proof −
    have f ⋆ g ∈ Map (F ⊙ G)
    proof −
      have 1: f ∈ Map F ∧ g ∈ Map G ∧ f ⋆ g ≅B f ⋆ g
        using F G f g B.isomorphic-reflexive by force
      have 2: Dom G = Cod F ∧ Cod G = Dom F
        using F G arr-char
        by (metis (no-types, lifting) Cod.simps(1) Cod-dom arr-inv
            cod-char comp-inv-arr dom-inv inverse-unique
            iso-is-arr is-concrete-category concrete-category.Cod-comp)
      hence f ⋆ g ∈ Comp (Map F) (Map G)
        using 1 F iso-is-arr Map-in-Hom [of F] in-HomD(2)
        apply (intro in-CompI, auto)
      proof −
        show B.is-iso-class (Map G)
          using G iso-is-arr Map-in-Hom [of G] in-HomD(2) [of Map G] by blast
      qed
      thus ?thesis
        using F G f g comp-char [of F G] by auto
    qed
    moreover have Cod F ∈ Map (F ⊙ G)
      by (metis (no-types, lifting) F G Map-cod comp-arr-inv dom-inv
          inverse-unique iso-is-arr g B.ide-in-iso-class B.in-hhomE B.src.preserves-ide)
    moreover have Map (F ⊙ G) = ⟦Cod F⟧B
      by (metis (no-types, lifting) F G Map-cod comp-arr-inv dom-inv
          inverse-unique iso-is-arr)
    moreover have Cod F = trg (f ⋆ g)
      using f g by fastforce
    ultimately show ?thesis
      using B.iso-class-elems-isomorphic
      by (metis f g B.is-iso-classI B.in-hhomE B.src.preserves-ide)
  qed
  ultimately show B.equivalence-map f
    using f g B.equivalence-mapI B.quasi-inversesI [of f g] by fastforce
qed
thus arr F ∧ (∀f. f ∈ Map F ⟶ B.equivalence-map f)
  using F by blast
next
assume F: arr F ∧ (∀f. f ∈ Map F ⟶ B.equivalence-map f)
show iso F
proof −
  obtain f where f: f ∈ Map F
    using F arr-char in-HomD(1) by blast
```

have *f-in-hhom*: «f : *Dom F* $\rightarrow_B$ *Cod F*»
  using *f F arr-char in-HomD*(*4*) [*of Map F Dom F Cod F f*] **by** *simp*
have *Map F* = *B.iso-class f*
  using *f F arr-char in-HomD*(*6*) [*of Map F Dom F Cod F f*] **by** *simp*
obtain g η ε' **where** ε': *equivalence-in-bicategory V H* a i *src trg f g* η ε'
  using *f F B.equivalence-map-def* **by** *auto*
interpret ε': *equivalence-in-bicategory V H* a i *src trg f g* η ε'
  using ε' **by** *auto*
obtain ε **where** ε: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* η ε
  using *f F* ε'*.ide-right* ε'*.antipar* ε'*.unit-in-hom* ε'*.unit-is-iso B.equivalence-map-def*
    B.equivalence-refines-to-adjoint-equivalence [*of f g* η]
  by *auto*
interpret ε: *adjoint-equivalence-in-bicategory V H* a i *src trg f g* η ε
  using ε **by** *simp*
have *g-in-hhom*: «g : *Cod F* $\rightarrow_B$ *Dom F*»
  using ε*.ide-right* ε*.antipar f-in-hhom* **by** *auto*
have *fg*: *B.quasi-inverses f g*
  using *B.quasi-inverses-def* ε*.equivalence-in-bicategory-axioms* **by** *auto*
have *g*: «g : *Cod F* $\rightarrow_B$ *Dom F*» $\wedge$ *B.is-left-adjoint g* $\wedge$ $[\![g]\!]_B = [\![g]\!]_B$
  using ε'*.dual-equivalence B.equivalence-is-left-adjoint B.equivalence-map-def*
    g-in-hhom
  by *blast*
have *F-as-MkArr*: *F* = *MkArr* (*Dom F*) (*Cod F*) $[\![f]\!]_B$
  using *F MkArr-Map* ‹*Map F* = *B.iso-class f*› **by** *fastforce*
have *F-in-hom*: *in-hom F* (*MkIde* (*Dom F*)) (*MkIde* (*Cod F*))
  using *F arr-char dom-char cod-char*
  by (*intro in-homI*, *auto*)
let *?G* = *MkArr* (*Cod F*) (*Dom F*) $[\![g]\!]_B$
have *arr ?G*
  using *MkArr-in-hom′ g* **by** *blast*
hence *G-in-hom*: «*?G* : *MkIde* (*Cod F*) $\rightarrow$ *MkIde* (*Dom F*)»
  using *arr-char MkArr-in-hom* **by** *simp*
have *inverse-arrows F ?G*
proof
  show *ide* (*?G* $\odot$ *F*)
  proof −
    have *?G* $\odot$ *F* = *dom F*
    proof (*intro arr-eqI*)
      show *1*: *seq ?G F*
        using *F-in-hom G-in-hom* **by** *blast*
      show *arr* (*dom F*)
        using *F-in-hom ide-dom* **by** *fastforce*
      show *Dom* (*?G* $\odot$ *F*) = *Dom* (*dom F*)
        using *F-in-hom G-in-hom 1 comp-char* **by** *auto*
      show *Cod* (*?G* $\odot$ *F*) = *Cod* (*dom F*)
        using *F-in-hom G-in-hom 1 comp-char* **by** *auto*
      show *Map* (*?G* $\odot$ *F*) = *Map* (*dom F*)
      proof −
        have *Map* (*?G* $\odot$ *F*) = *Comp* $[\![g]\!]_B$ $[\![f]\!]_B$

```
        using 1 comp-char [of ?G F] ‹Map F = B.iso-class f› by simp
      also have ... = ⟦g ⋆ f⟧B
      proof −
        have g ⋆ f ∈ Comp ⟦g⟧B ⟦f⟧B
          by (metis ε.ide-left ε.ide-right ε.unit-in-vhom ε.unit-simps(3) B.arrI
              B.ide-cod B.ide-in-iso-class in-CompI B.is-iso-classI
              B.isomorphic-reflexive)
        thus ?thesis
          using Comp-eq-iso-class-memb F-in-hom G-in-hom arr-char arr-char
                ‹Map F = B.iso-class f›
          by auto
      qed
      also have ... = ⟦src f⟧B
        using ε.unit-in-hom ε.unit-is-iso B.isomorphic-def B.iso-class-def
              B.isomorphic-symmetric
        by (intro B.iso-class-eqI, blast)
      also have ... = ⟦Dom F⟧B
        using ε.ide-left f-in-hhom B.ide-in-iso-class B.in-hhomE B.src.preserves-ide
              B.isomorphic-reflexive
        by (intro B.iso-class-eqI, fastforce)
      also have ... = Map (dom F)
        using F-in-hom dom-char by auto
      finally show ?thesis by blast
    qed
  qed
  thus ?thesis
    using F by simp
qed
show ide (F ⊙ ?G)
proof −
  have F ⊙ ?G = cod F
  proof (intro arr-eqI)
    show 1: seq F ?G
      using F-in-hom G-in-hom by blast
    show arr (cod F)
      using F-in-hom ide-cod by fastforce
    show Dom (F ⊙ ?G) = Dom (cod F)
      using F-in-hom G-in-hom 1 comp-char by auto
    show Cod (F ⊙ ?G) = Cod (cod F)
      using F-in-hom G-in-hom 1 comp-char by auto
    show Map (F ⊙ ?G) = Map (cod F)
    proof −
      have Map (F ⊙ ?G) = Comp ⟦f⟧B ⟦g⟧B
        using 1 comp-char [of F ?G] ‹Map F = ⟦f⟧B› by simp
      also have ... = ⟦f ⋆ g⟧B
      proof −
        have f ⋆ g ∈ Comp ⟦f⟧B ⟦g⟧B
          by (metis ε.antipar(1) ε.ide-left ε.ide-right B.ide-hcomp
              B.ide-in-iso-class in-CompI B.is-iso-classI B.isomorphic-reflexive)
```

```
            thus ?thesis
              using Comp-eq-iso-class-memb F-in-hom G-in-hom arr-char arr-char
                    ‹Map F = ⟦f⟧_B›
              by auto
          qed
          also have ... = ⟦trg f⟧_B
          proof −
            have trg f ∈ ⟦f ⋆ g⟧_B
              using ε.counit-in-hom ε.counit-is-iso B.isomorphic-def B.iso-class-def
                    B.isomorphic-symmetric
              by blast
            thus ?thesis
              using B.iso-class-eqI
              by (metis ε.antipar(1) ε.ide-left ε.ide-right B.ide-hcomp
                  B.ide-in-iso-class B.is-iso-classI B.iso-class-elems-isomorphic)
          qed
          also have ... = ⟦Cod F⟧_B
            using f-in-hhom by auto
          also have ... = Map (cod F)
            using F-in-hom dom-char by auto
          finally show ?thesis by blast
        qed
      qed
      thus ?thesis
        using F by simp
    qed
  qed
  thus ?thesis by auto
 qed
qed

lemma iso-char′:
shows iso F ⟷ arr F ∧ (∃f. f ∈ Map F ∧ B.equivalence-map f)
proof −
 have arr F ⟹ (∀f. f ∈ Map F ⟶ B.equivalence-map f) ⟷
              (∃f. f ∈ Map F ∧ B.equivalence-map f)
 proof
   assume F: arr F
   show (∀f. f ∈ Map F ⟶ B.equivalence-map f) ⟹
        (∃f. f ∈ Map F ∧ B.equivalence-map f)
     using F arr-char in-HomD(1) by blast
   show (∃f. f ∈ Map F ∧ B.equivalence-map f) ⟹
        (∀f. f ∈ Map F ⟶ B.equivalence-map f)
     by (metis (no-types, lifting) F is-concrete-category concrete-category.arr-char
         B.equivalence-map-preserved-by-iso in-HomD(2) B.iso-class-elems-isomorphic)
 qed
 thus ?thesis
   using iso-char by blast
qed
```

The following mapping takes a map in B to the corresponding arrow of $Maps(B)$.

abbreviation *CLS* (‹⟦⟦-⟧⟧›)
where $[\![[\![f]\!]]\!] \equiv MkArr\ (src\ f)\ (trg\ f)\ [\![f]\!]_B$

lemma *ide-CLS-obj*:
assumes *B.obj a*
shows *ide* $[\![[\![a]\!]]\!]$
  by (*simp add*: *assms B.obj-simps*)

lemma *CLS-in-hom*:
assumes *B.is-left-adjoint f*
shows «$[\![[\![f]\!]]\!] : [\![[\![src\ f]\!]]\!] \rightarrow [\![[\![trg\ f]\!]]\!]$»
  using *assms B.left-adjoint-is-ide MkArr-in-hom MkArr-in-hom′* **by** *simp*

lemma *CLS-eqI*:
assumes *B.ide f*
shows $[\![[\![f]\!]]\!] = [\![[\![g]\!]]\!] \longleftrightarrow f \cong_B g$
  by (*metis arr.inject assms B.ide-in-iso-class B.iso-class-def B.iso-class-eqI B.isomorphic-implies-hpar*(*3*) *B.isomorphic-implies-hpar*(*4*) *B.isomorphic-symmetric mem-Collect-eq*)

lemma *CLS-hcomp*:
assumes *B.ide f* **and** *B.ide g* **and** $src\ f = trg\ g$
shows $[\![[\![f \star g]\!]]\!] = MkArr\ (src\ g)\ (trg\ f)\ (Comp\ [\![f]\!]_B\ [\![g]\!]_B)$
proof −
  have $[\![[\![f \star g]\!]]\!] = MkArr\ (src\ g)\ (trg\ f)\ [\![f \star g]\!]_B$
    using *assms B.left-adjoint-is-ide* **by** *simp*
  moreover have $[\![f \star g]\!]_B = Comp\ [\![f]\!]_B\ [\![g]\!]_B$
  proof
    show $[\![f \star g]\!]_B \subseteq Comp\ [\![f]\!]_B\ [\![g]\!]_B$
      by (*metis assms*(*1*−*2*) *B.ide-in-iso-class in-CompI B.is-iso-classI B.iso-class-def mem-Collect-eq subsetI*)
    show $Comp\ [\![f]\!]_B\ [\![g]\!]_B \subseteq [\![f \star g]\!]_B$
      by (*metis Comp-eq-iso-class-memb* ‹$[\![f \star g]\!]_B \subseteq Comp\ [\![f]\!]_B\ [\![g]\!]_B$› *assms*(*1*−*3*) *B.ide-hcomp B.ide-in-iso-class subset-iff*)
  qed
  ultimately show *?thesis* **by** *simp*
qed

lemma *comp-CLS*:
assumes *B.is-left-adjoint f* **and** *B.is-left-adjoint g* **and** $f \star g \cong_B h$
shows $[\![[\![f]\!]]\!] \odot [\![[\![g]\!]]\!] = [\![[\![h]\!]]\!]$
proof −
  have *hseq-fg*: *B.hseq f g*
    using *assms B.isomorphic-implies-hpar*(*1*) **by** *simp*
  have *seq* $[\![[\![f]\!]]\!]\ [\![[\![g]\!]]\!]$
    using *assms hseq-fg CLS-in-hom* [*of f*] *CLS-in-hom* [*of g*]
    apply (*elim B.hseqE*) **by** *auto*
  hence $[\![[\![f]\!]]\!] \odot [\![[\![g]\!]]\!] = MkArr\ (src\ g)\ (trg\ f)\ (Comp\ [\![f]\!]_B\ [\![g]\!]_B)$

```
    using comp-char [of CLS f CLS g] by simp
  also have ... = ⟦⟦f ⋆ g⟧⟧
    using assms hseq-fg CLS-hcomp
    apply (elim B.hseqE)
    using B.adjoint-pair-antipar(1) by auto
  also have ... = ⟦⟦h⟧⟧
    using assms B.isomorphic-symmetric
    by (simp add: assms(3) B.iso-class-eqI B.isomorphic-implies-hpar(3)
        B.isomorphic-implies-hpar(4))
  finally show ?thesis by simp
qed
```

The following mapping that takes an arrow of $Maps(B)$ and chooses some representative map in B.

```
definition REP
where REP F ≡ if arr F then SOME f. f ∈ Map F else B.null

lemma REP-in-Map:
assumes arr A
shows REP A ∈ Map A
proof −
  have Map A ≠ {}
    using assms arr-char in-HomD(1) by blast
  thus ?thesis
    using assms REP-def someI-ex [of λf. f ∈ Map A] by auto
qed

lemma REP-in-hhom [intro]:
assumes in-hom F A B
shows «REP F : src (REP A) →B src (REP B)»
and B.is-left-adjoint (REP F)
proof −
  have Map F ≠ {}
    using assms in-hom-char arr-char null-char in-HomD(1) [of Map F Dom F Cod F]
    by blast
  hence 1: REP F ∈ Map F
    using assms REP-def someI-ex [of λf. f ∈ Map F] by auto
  hence 2: B.arr (REP F)
   using assms 1 in-hom-char [of F A B] B.iso-class-def Map-memb-in-hhom B.in-hhom-def
    by blast
  hence «REP F : Dom A →B Dom B»
    using assms 1 in-hom-char [of F A B] Map-memb-in-hhom by auto
  thus «REP F : src (REP A) →B src (REP B)»
    using assms
    by (metis (no-types, lifting) Map-memb-in-hhom ideD(1)
        in-homI in-hom-char REP-in-Map B.in-hhom-def)
  have REP F ∈ ⟦REP F⟧B
    using assms 1 2 arr-char [of F] in-HomD(6) B.ide-in-iso-class B.iso-class-memb-is-ide
          in-hom-char
```

by *blast*
thus *B.is-left-adjoint* (*REP F*)
using *assms 1 2 arr-char in-HomD*(*5*) [*of Map F Dom F Cod F REP F*]
by *auto*
qed

lemma *ide-REP*:
assumes *arr A*
shows *B.ide* (*REP A*)
using *assms REP-in-hhom*(*2*) *B.adjoint-pair-antipar*(*1*) **by** *blast*

lemma *REP-simps* [*simp*]:
assumes *arr A*
shows *B.ide* (*REP A*)
and *src* (*REP A*) = *Dom A* **and** *trg* (*REP A*) = *Cod A*
and *B.dom* (*REP A*) = *REP A* **and** *B.cod* (*REP A*) = *REP A*
proof −
show *B.ide* (*REP A*)
using *assms ide-REP* **by** *blast*
show *src* (*REP A*) = *Dom A*
using *assms REP-in-hhom*
by (*metis* (*no-types*, *lifting*) *Map-memb-in-hhom Dom-dom in-homI*
REP-in-Map B.in-hhom-def)
show *trg* (*REP A*) = *Cod A*
using *assms REP-in-hhom*
by (*metis* (*no-types*, *lifting*) *Map-memb-in-hhom Dom-cod in-homI*
REP-in-Map B.in-hhom-def)
show *B.dom* (*REP A*) = *REP A*
using *assms in-homI REP-in-hhom*(*2*) *B.adjoint-pair-antipar*(*1*) *B.ideD*(*2*)
by *blast*
show *B.cod* (*REP A*) = *REP A*
using *assms in-homI REP-in-hhom*(*2*) *B.adjoint-pair-antipar*(*1*) *B.ideD*(*3*)
by *blast*
qed

lemma *isomorphic-REP-src*:
assumes *ide A*
shows *REP A* $\cong_B$ *src* (*REP A*)
using *assms*
by (*metis* (*no-types*, *lifting*) *ideD*(*1*) *ide-char*$_{CC}$ *REP-in-Map ide-REP*
REP-simps(*2*) *B.is-iso-classI B.ide-in-iso-class B.iso-class-elems-isomorphic*
B.src.preserves-ide)

lemma *isomorphic-REP-trg*:
assumes *ide A*
shows *REP A* $\cong_B$ *trg* (*REP A*)
using *assms ide-char*$_{CC}$ *isomorphic-REP-src* **by** *auto*

lemma *CLS-REP*:

assumes *arr F*
shows $[\![[\![REP\ F]\!]]\!] = F$
  by (*metis* (*mono-tags*, *lifting*) *MkArr-Map*
    is-concrete-category REP-in-Map REP-simps(*2*) *REP-simps*(*3*) *assms*
    concrete-category.Map-in-Hom in-HomD(*6*))

lemma *REP-CLS*:
assumes *B.is-left-adjoint f*
shows $REP\ [\![[\![f]\!]]\!] \cong_B f$
  by (*metis* (*mono-tags*, *lifting*) *CLS-in-hom Map.simps*(*1*) *in-homE REP-in-Map*
    assms B.iso-class-def B.isomorphic-symmetric mem-Collect-eq)

lemma *isomorphic-hcomp-REP*:
assumes *seq F G*
shows $REP\ F \star REP\ G \cong_B REP\ (F \odot G)$
proof −
  have *1*: *Dom F* = *Cod G*
    using *assms seq-char* **by** *simp*
  have *2*: *Map F* = *B.iso-class* (*REP F*)
    using *assms seq-char arr-char REP-in-Map in-HomD*(*6*) **by** *meson*
  have *3*: *Map G* = *B.iso-class* (*REP G*)
    using *assms seq-char arr-char REP-in-Map*
      in-HomD(*6*) [*of Map G Dom G Cod G REP G*]
    by *auto*
  have $Map\ (F \odot G) = Comp\ [\![REP\ F]\!]_B\ [\![REP\ G]\!]_B$
    using *assms seq-char null-char*
    by (*metis* (*no-types*, *lifting*) *CLS-REP Map.simps*(*1*) *Map-comp*)
  have $Comp\ [\![REP\ F]\!]_B\ [\![REP\ G]\!]_B = [\![REP\ F \star REP\ G]\!]_B$
  proof −
    have $REP\ F \star REP\ G \in Comp\ [\![REP\ F]\!]_B\ [\![REP\ G]\!]_B$
    proof −
      have $REP\ F \in Map\ F \wedge REP\ G \in Map\ G$
        using *assms seq-char REP-in-Map* **by** *auto*
      moreover have $REP\ F \star REP\ G \cong_B REP\ F \star REP\ G$
        using *assms seq-char 2 B.isomorphic-reflexive* **by** *auto*
      ultimately show *?thesis*
        using *assms 1 2 3 B.iso-class-def B.is-iso-class-def*
        by (*intro in-CompI*, *auto*)
    qed
    moreover have $[\![REP\ F]\!]_B \in Hom\ (Cod\ G)\ (Cod\ F)$
      using *assms 1 2 arr-char* [*of F*] **by** *auto*
    moreover have $[\![REP\ G]\!]_B \in Hom\ (Dom\ G)\ (Cod\ G)$
      using *assms 1 3 arr-char* [*of G*] **by** *auto*
    ultimately show *?thesis*
      using *Comp-eq-iso-class-memb assms arr-char arr-char REP-in-Map*
      by (*simp add*: *Comp-def*)
  qed
  moreover have $REP\ (F \odot G) \in Comp\ [\![REP\ F]\!]_B\ [\![REP\ G]\!]_B$
  proof −

have *Map* $(F \odot G)$ = *Comp* (*Map F*) (*Map G*)
using *assms 1 comp-char* [*of F G*] **by** *simp*
thus *?thesis*
using *assms 1 2 3 REP-in-Map* [*of* $F \odot G$] **by** *simp*
qed
ultimately show *?thesis*
using *B.iso-class-def* **by** *simp*
qed

We now show that *Maps*(*B*) has pullbacks. For this we need to exhibit functions PRJ_0 and PRJ_1 that produce the legs of the pullback of a cospan (H, K) and verify that they have the required universal property. These are obtained by choosing representatives h and k of H and K, respectively, and then taking $PRJ_0 = CLS\ (tab_0\ (k^* \star h))$ and $PRJ_1 = CLS\ (tab_1\ (k^* \star h))$. That these constitute a pullback in *Maps*(*B*) follows from the fact that $tab_0\ (k^* \star h)$ and $tab_1\ (k^* \star h)$ form a pseudo-pullback of (h, k) in the underlying bicategory.

For our purposes here, it is not sufficient simply to show that *Maps*(*B*) has pullbacks and then to treat it as an abstract "category with pullbacks" where the pullbacks are chosen arbitrarily. Instead, we have to retain the connection between a pullback in Maps and the tabulation of $k^* \star h$ that is ultimately used to obtain it. If we don't do this, then it becomes problematic to define the compositor of a pseudofunctor from the underlying bicategory B to *Span*(*Maps*(*B*)), because the components will go from the apex of a composite span (obtained by pullback) to the apex of a tabulation (chosen arbitrarily using *BS2*) and these need not be in agreement with each other.

definition PRJ_0
where $PRJ_0 \equiv \lambda K\ H.$ *if cospan K H then* $[\![B.tab_0\ ((REP\ K)^* \star (REP\ H))]\!]$ *else null*
definition PRJ_1
where $PRJ_1 \equiv \lambda K\ H.$ *if cospan K H then* $[\![B.tab_1\ ((REP\ K)^* \star (REP\ H))]\!]$ *else null*

interpretation *elementary-category-with-pullbacks comp* PRJ_0 PRJ_1
proof
show $\bigwedge H\ K.\ \neg$ *cospan K H* $\Longrightarrow PRJ_0\ K\ H = null$
unfolding PRJ_0*-def* **by** *auto*
show $\bigwedge H\ K.\ \neg$ *cospan K H* $\Longrightarrow PRJ_1\ K\ H = null$
unfolding PRJ_1*-def* **by** *auto*
show $\bigwedge H\ K.$ *cospan K H* $\Longrightarrow$ *commutative-square K H* $(PRJ_1\ K\ H)\ (PRJ_0\ K\ H)$
proof −
fix *H K*
assume *HK*: *cospan K H*
define *h* **where** *h* = *REP H*
define *k* **where** *k* = *REP K*
have *h*: $h \in$ *Map H*
using *h-def HK REP-in-Map* **by** *blast*
have *k*: $k \in$ *Map K*
using *k-def HK REP-in-Map* **by** *blast*
have *1*: *B.is-left-adjoint h* ∧ *B.is-left-adjoint k* ∧ *B.ide h* ∧ *B.ide k* ∧ *trg h* = *trg k*
using *h k h-def k-def HK arr-char cod-char B.in-hhom-def B.left-adjoint-is-ide in-HomD*(*5*) [*of Map H Dom H Cod H h*]

in-HomD(*5*) [*of Map K Dom K Cod K k*]
apply *auto*
by (*metis* (*no-types*, *lifting*) *HK Dom-cod*)
interpret *h*: *map-in-bicategory* ‹(·)› ‹(⋆)› a i *src trg h*
using *1* **by** *unfold-locales auto*
interpret *k*: *map-in-bicategory* ‹(·)› ‹(⋆)› a i *src trg k*
using *1* **by** *unfold-locales auto*
interpret *hk*: *cospan-of-maps-in-bicategory-of-spans* ‹(·)› ‹(⋆)› a i *src trg h k*
using *1* **by** *unfold-locales auto*
let *?f* = $B.tab_0$ (k^* ⋆ *h*)
let *?g* = $B.tab_1$ (k^* ⋆ *h*)
have *span*: *span* ⟦⟦*?f*⟧⟧ ⟦⟦*?g*⟧⟧
using *dom-char CLS-in-hom* [*of ?f*] *CLS-in-hom* [*of ?g*] **by** *auto*
have *seq*: *seq H* ⟦⟦*?f*⟧⟧
using *HK seq-char hk.leg0-is-map CLS-in-hom h-def* $hk.p_0$*-simps hk.satisfies-T0*
by *fastforce*
have *seq′*: *seq K* ⟦⟦*?g*⟧⟧
using *HK k arr-char dom-char cod-char in-HomD*(*5*) *hk.leg1-is-map CLS-in-hom*
by (*metis* (*no-types*, *lifting*) *Cod.simps*(*1*) *seq-char REP-simps*(*2*)
$hk.p_1$*-simps k-def span*)
show *commutative-square K H* (PRJ_1 *K H*) (PRJ_0 *K H*)
proof
show *cospan K H*
using *HK* **by** *simp*
show *dom K* = *cod* (PRJ_1 *K H*)
using *seq′* PRJ_1*-def HK h-def k-def* **by** *auto*
show *span* (PRJ_1 *K H*) (PRJ_0 *K H*)
unfolding PRJ_0*-def* PRJ_1*-def* **using** *HK span h-def k-def* **by** *simp*
show *K* ⊙ PRJ_1 *K H* = *H* ⊙ PRJ_0 *K H*
proof −
have *iso*: *h* ⋆ *?f* $\cong_B$ *k* ⋆ *?g*
using *hk.φ-uniqueness B.isomorphic-symmetric B.isomorphic-def* **by** *blast*
have ∗: *Comp* (*Map H*) $[\![?f]\!]_B$ = *Comp* (*Map K*) $[\![?g]\!]_B$
proof (*intro Comp-eqI*)
show *h* ⋆ *?f* ∈ *Comp* (*Map H*) $[\![B.tab_0\ (k^* \star h)]\!]_B$
proof (*unfold Comp-def*)
have *B.is-iso-class* $[\![?f]\!]_B$
by (*simp add*: *B.is-iso-classI*)
moreover have *B.is-iso-class* (*Map H*)
using *CLS-REP HK Map.simps*(*1*) *B.is-iso-classI h.ide-left h-def*
by (*metis* (*no-types*, *lifting*))
moreover have *?f* ∈ $[\![B.tab_0\ (k^* \star h)]\!]_B$
by (*simp add*: *B.ide-in-iso-class*(*1*))
moreover have «*?f* : *src* ($B.tab_0$ (k^* ⋆ *h*)) $\rightarrow_B$ *Dom H*»
using *seq seq-char* **by** *simp*
moreover have *h* ∈ *Map H*
by *fact*
moreover have «*h* : *Dom H* $\rightarrow_B$ *Cod H*»
by (*simp add*: *HK h-def*)

moreover have $h \star ?f \cong_B h \star ?f$
using *B.isomorphic-reflexive* **by** *auto*
ultimately show $h \star B.tab_0\ (k^* \star h)$
$\in \{h'.\ B.is\text{-}iso\text{-}class\ [\![B.tab_0\ (k^* \star h)]\!]_B \wedge$
$B.is\text{-}iso\text{-}class\ (Map\ H) \wedge$
$(\exists f\ g.\ f \in [\![B.tab_0\ (k^* \star h)]\!]_B \wedge$
$g \in Map\ H \wedge g \star f \cong_B h')\}$
by *auto*
qed
show $k \star ?g \in Comp\ (Map\ K)\ [\![B.tab_1\ (k^* \star h)]\!]_B$
proof (*unfold Comp-def*)
have $B.is\text{-}iso\text{-}class\ [\![?g]\!]_B$
by (*simp add*: *B.is-iso-classI*)
moreover have $B.is\text{-}iso\text{-}class\ (Map\ K)$
by (*metis* (*no-types*, *lifting*) *CLS-REP HK Map.simps*(*1*)
B.is-iso-classI k.ide-left k-def)
moreover have $?g \in [\![B.tab_1\ (k^* \star h)]\!]_B$
by (*simp add*: *B.ide-in-iso-class*(*1*))
moreover have «$?g : src\ (B.tab_1\ (k^* \star h)) \rightarrow_B Dom\ K$»
using *seq seq-char B.in-hhom-def seq'* **by** *auto*
moreover have $k \in Map\ K$
by *fact*
moreover have «$k : Dom\ K \rightarrow_B Cod\ K$»
by (*simp add*: *HK k-def*)
moreover have $k \star ?g \cong_B k \star ?g$
using *B.isomorphic-reflexive iso B.isomorphic-implies-hpar*(*2*) **by** *auto*
ultimately show $k \star B.tab_1\ (k^* \star h)$
$\in \{h'.\ B.is\text{-}iso\text{-}class\ [\![B.tab_1\ (k^* \star h)]\!]_B \wedge$
$B.is\text{-}iso\text{-}class\ (Map\ K) \wedge$
$(\exists f\ g.\ f \in [\![B.tab_1\ (k^* \star h)]\!]_B \wedge$
$g \in Map\ K \wedge g \star f \cong_B h')\}$
by *auto*
qed
show $h \star ?f \cong_B k \star ?g$
using *iso* **by** *simp*
qed
have $K \odot PRJ_1\ K\ H = K \odot [\![[\![?g]\!]]\!]$
unfolding PRJ_1*-def* **using** *HK h-def k-def* **by** *simp*
also have ... $= MkArr\ (src\ ?g)\ (Cod\ K)\ (Comp\ (Map\ K)\ [\![?g]\!]_B)$
using *seq' comp-char* [*of K* $[\![[\![?g]\!]]\!]$] **by** *simp*
also have ... $= MkArr\ (src\ ?f)\ (Cod\ H)\ (Comp\ (Map\ H)\ [\![?f]\!]_B)$
using $*$ *HK cod-char* **by** *auto*
also have ... $= comp\ H\ [\![[\![?f]\!]]\!]$
using *seq comp-char* [*of H* $[\![[\![?f]\!]]\!]$] **by** *simp*
also have ... $= comp\ H\ (PRJ_0\ K\ H)$
unfolding PRJ_0*-def* **using** *HK h-def k-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

qed
show $\bigwedge H\ K\ U\ V.$ *commutative-square* $K\ H\ V\ U \Longrightarrow$
$\exists! E.$ *comp* $(PRJ_1\ K\ H)\ E = V \wedge$ *comp* $(PRJ_0\ K\ H)\ E = U$
proof −
fix $H\ K\ U\ V$
assume *cs*: *commutative-square* $K\ H\ V\ U$
have *HK*: *cospan* $K\ H$
using *cs* **by** *auto*

define h **where** $h = REP\ H$
define k **where** $k = REP\ K$
have h: $h \in Map\ H$
using *h-def HK REP-in-Map* **by** *blast*
have k: $k \in Map\ K$
using *k-def HK REP-in-Map* **by** *blast*
have *1*: *B.is-left-adjoint* $h \wedge$ *B.is-left-adjoint* $k \wedge$ *B.ide* $h \wedge$ *B.ide* $k \wedge trg\ h = trg\ k$
using *h k h-def k-def HK arr-char cod-char B.in-hhom-def B.left-adjoint-is-ide*
in-HomD(*5*) [*of Map H Dom H Cod H h*]
in-HomD(*5*) [*of Map K Dom K Cod K k*]
apply *auto*
by (*metis* (*no-types*, *lifting*) *HK Dom-cod*)
interpret h: *map-in-bicategory* ‹(·)› ‹(⋆)› a i *src trg h*
using *1* **by** *unfold-locales auto*
interpret k: *map-in-bicategory* ‹(·)› ‹(⋆)› a i *src trg k*
using *1* **by** *unfold-locales auto*
interpret *hk*: *cospan-of-maps-in-bicategory-of-spans* ‹(·)› ‹(⋆)› a i *src trg h k*
using *1* **by** *unfold-locales auto*
let *?f* $= B.tab_0\ (k^* \star h)$
let *?g* $= B.tab_1\ (k^* \star h)$
have *seq-HU*: *seq H U*
using *cs* **by** *auto*
have *seq-KV*: *seq K V*
using *cs* **by** *auto*
let *?u* $= REP\ U$
let *?v* $= REP\ V$
have u: *B.ide ?u*
using *ide-REP seq-HU* **by** *auto*
have v: *B.ide ?v*
using *ide-REP seq-KV* **by** *auto*
have *u-is-map*: *B.is-left-adjoint ?u*
using *u seq-HU REP-in-Map arr-char* [*of U*]
in-HomD(*5*) [*of Map U Dom U Cod U ?u*]
by *auto*
have *v-is-map*: *B.is-left-adjoint ?v*
using *v seq-KV REP-in-Map arr-char* [*of V*]
in-HomD(*5*) [*of Map V Dom V Cod V ?v*]
by *auto*
have $*$: $h \star ?u \cong_B k \star ?v$
proof −

```
  have h ⋆ ?u ≅B REP (H ⊙ U)
  proof −
    have h ⋆ ?u ≅B REP H ⋆ ?u
    proof −
      have h ≅B REP H
        using h h-def HK arr-char REP-in-Map B.iso-class-elems-isomorphic
              in-HomD(5) [of Map H Dom H Cod H h] B.isomorphic-reflexive
        by auto
      thus ?thesis
        using h-def seq-HU B.isomorphic-implies-hpar(1) B.isomorphic-reflexive
        by (simp add: seq-char)
    qed
    also have ... ≅B REP (H ⊙ U)
      using seq-HU isomorphic-hcomp-REP isomorphic-def by blast
    finally show ?thesis by blast
  qed
  also have ... ≅B REP (K ⊙ V)
    using seq-HU cs B.isomorphic-reflexive by auto
  also have ... ≅B (k ⋆ ?v)
  proof −
    have ... ≅B REP K ⋆ ?v
      using seq-KV isomorphic-hcomp-REP B.isomorphic-def B.isomorphic-symmetric
      by blast
    also have ... ≅B k ⋆ ?v
    proof −
      have k ≅B REP K
        using k k-def HK arr-char REP-in-Map B.iso-class-elems-isomorphic
              in-HomD(5) [of Map K Dom K Cod K k] B.isomorphic-reflexive
        by auto
      thus ?thesis
        using k-def seq-KV B.isomorphic-implies-hpar(1) B.isomorphic-reflexive
        by (simp add: seq-char)
    qed
    finally show ?thesis by blast
  qed
  finally show ?thesis by blast
qed
have hseq-hu: src h = trg ?u
  using ∗ B.isomorphic-implies-hpar
  by (meson B.hseqE B.ideD(1))
have hseq-kv: src k = trg ?v
  using ∗ B.isomorphic-implies-hpar
  by (meson B.hseqE B.ideD(1))

obtain w where w: B.is-left-adjoint w ∧ ?f ⋆ w ≅B ?u ∧ ?v ≅B (?g ⋆ w)
  using ∗ u-is-map v-is-map hk.has-pseudo-pullback [of ?u ?v] B.isomorphic-symmetric
  by blast
have w-in-hom: «w : src ?u →B src ?f» ∧ B.ide w
  using w B.left-adjoint-is-ide B.src-cod B.trg-cod B.isomorphic-def
```

by (*metis B.hseqE B.ideD*(*1*) *B.in-hhom-def B.isomorphic-implies-hpar*(*3*)
B.isomorphic-implies-ide(*1*) *B.src-hcomp*)
let *?W* = *CLS w*
have *W*: «*?W* : *dom U* → *dom* ⟦⟦*?f*⟧⟧»
proof (*intro in-homI*)
show *arr ?W*
using *w CLS-in-hom* **by** *blast*
thus *dom ?W* = *dom U*
using *w-in-hom dom-char REP-in-hhom*(*1*) *CLS-in-hom*
by (*metis* (*no-types*, *lifting*) *Dom.simps*(*1*) *commutative-squareE*
dom-char REP-simps(*2*) *cs B.in-hhomE*)
show *cod ?W* = *dom* ⟦⟦*?f*⟧⟧
proof −
have *src ?f* = *trg w*
by (*metis* (*lifting*) *B.in-hhomE w-in-hom*)
thus *?thesis*
using *CLS-in-hom* [*of ?f*] *CLS-in-hom* [*of w*] *hk.satisfies-T0 w* **by** *fastforce*
qed
qed
show ∃!*E*. PRJ_1 *K H* ⊙ *E* = *V* ∧ PRJ_0 *K H* ⊙ *E* = *U*
proof −
have PRJ_1 *K H* ⊙ *?W* = *V* ∧ PRJ_0 *K H* ⊙ *?W* = *U*
proof −
have ⟦⟦*?f*⟧⟧ ⊙ *?W* = *U*
using *w w-in-hom u CLS-in-hom comp-CLS*
B.isomorphic-symmetric CLS-REP hk.leg0-is-map
by (*metis* (*mono-tags*, *lifting*) *commutative-square-def cs*)
moreover have ⟦⟦*?g*⟧⟧ ⊙ *?W* = *V*
using *w w-in-hom v CLS-in-hom comp-CLS*
B.isomorphic-symmetric CLS-REP hk.leg1-is-map
by (*metis* (*mono-tags*, *lifting*) *commutative-square-def cs*)
ultimately show *?thesis*
using *HK h-def k-def* PRJ_0*-def* PRJ_1*-def* **by** *auto*
qed
moreover have
⋀*W′*. PRJ_1 *K H* ⊙ *W′* = *V* ∧ PRJ_0 *K H* ⊙ *W′* = *U* ⟹ *W′* = *?W*
proof −
fix *W′*
assume PRJ_1 *K H* ⊙ *W′* = *V* ∧ PRJ_0 *K H* ⊙ *W′* = *U*
hence *W′*: «*W′* : *dom U* → *dom* ⟦⟦*?f*⟧⟧» ∧ ⟦⟦*?f*⟧⟧ ⊙ *W′* = *U* ∧ ⟦⟦*?g*⟧⟧ ⊙ *W′* = *V*
using PRJ_0*-def* PRJ_1*-def HK h-def k-def* **apply** *simp*
using *cs arr-iff-in-hom* **by** *blast*
let *?w′* = *REP W′*
have *w′*: *B.ide ?w′*
using *W′ ide-REP* **by** *auto*

have *fw′-iso-u*: *?f* ⋆ *?w′* $\cong_B$ *?u*
proof −
have *?f* ⋆ *?w′* $\cong_B$ *REP* ⟦⟦*?f*⟧⟧ ⋆ *?w′*

```
        by (metis (no-types, lifting) Cod.simps(1) in-hom-char
            REP-CLS REP-simps(3) W W' B.hcomp-isomorphic-ide hk.satisfies-T0
            B.in-hhomE B.isomorphic-symmetric w' w-in-hom)
      also have REP ⟦⟦?f⟧⟧ ⋆ ?w' ≅B ?u
         using W' isomorphic-hcomp-REP cs by blast
      finally show ?thesis by blast
    qed

    have gw'-iso-v: ?g ⋆ ?w' ≅B ?v
    proof −
      have ?g ⋆ ?w' ≅B REP ⟦⟦?g⟧⟧ ⋆ ?w'
      proof −
        have ?g ≅B REP ⟦⟦?g⟧⟧
          using REP-CLS B.isomorphic-symmetric hk.leg1-is-map by blast
        moreover have B.ide (REP W')
          using W' by auto
        moreover have src ?f = trg ?w'
          using w-in-hom W W' in-hom-char arr-char B.in-hhom-def
          by (meson fw'-iso-u B.hseqE B.ideD(1) B.isomorphic-implies-ide(1))
        ultimately show ?thesis
          using B.hcomp-isomorphic-ide by simp
      qed
      also have ... ≅B ?v
         using W' isomorphic-hcomp-REP cs by blast
      finally show ?thesis by blast
    qed

    show W' = ?W
    proof −
      have W' = ⟦⟦?w'⟧⟧
        using w' W' CLS-REP by auto
      also have ... = ?W
      proof −
        have ?w' ≅B w
          using ∗ w W' hk.has-pseudo-pullback(2) u-is-map v-is-map
                B.isomorphic-symmetric fw'-iso-u gw'-iso-v
          by blast
        thus ?thesis
          using CLS-eqI B.iso-class-eqI w' by blast
      qed
      finally show ?thesis by blast
    qed
  qed
  ultimately show ?thesis by auto
 qed
qed
qed

lemma is-elementary-category-with-pullbacks:
```

shows *elementary-category-with-pullbacks comp* PRJ_0 PRJ_1
..

lemma *is-category-with-pullbacks*:
shows *category-with-pullbacks comp*
..

sublocale *elementary-category-with-pullbacks comp* PRJ_0 PRJ_1
using *is-elementary-category-with-pullbacks* **by** *simp*

end

Here we relate the projections of the chosen pullbacks in $Maps(B)$ to the projections associated with the chosen tabulations in B.

context *composite-tabulation-in-maps*
begin

interpretation *Maps*: *maps-category V H* a i *src trg*
..

interpretation $r_0 s_1$: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* s_1 r_0
using *ϱ.leg0-is-map σ.leg1-is-map composable* **by** *unfold-locales auto*

lemma *prj-char*:
shows $Maps.PRJ_0$ $[\![r_0]\!]$ $[\![s_1]\!]$ = $[\![prj_0\ s_1\ r_0]\!]$
and $Maps.PRJ_1$ $[\![r_0]\!]$ $[\![s_1]\!]$ = $[\![prj_1\ s_1\ r_0]\!]$
proof −
have *Maps.arr* (*Maps.MkArr* (*src* s_0) (*trg s*) $[\![s_1]\!]$)
using *σ.leg1-in-hom Maps.CLS-in-hom σ.leg1-is-map Maps.arr-char* **by** *auto*
moreover have *Maps.arr* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$)
using *Maps.CLS-in-hom composable* $r_0 s_1$*.k-is-map* **by** *fastforce*
moreover have *Maps.cod* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$) =
Maps.cod (*Maps.MkArr* (*src* s_0) (*trg s*) $[\![s_1]\!]$)
unfolding *Maps.arr-char*
using *σ.leg1-in-hom ϱ.leg0-in-hom*
by (*simp add*: *Maps.cod-char calculation*(*1*) *calculation*(*2*))
ultimately have $Maps.PRJ_0$ $[\![r_0]\!]$ $[\![s_1]\!]$ =
$[\![tab_0$ ((*Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$))* $\star$
Maps.REP (*Maps.MkArr* (*src* s_0) (*trg s*) $[\![s_1]\!]$))$]\!]$ $\wedge$
$Maps.PRJ_1$ (*Maps.CLS* r_0) (*Maps.CLS* s_1) =
$[\![tab_1$ ((*Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$))* $\star$
Maps.REP (*Maps.MkArr* (*src* s_0) (*trg s*) $[\![s_1]\!]$))$]\!]$
unfolding $Maps.PRJ_0$*-def* $Maps.PRJ_1$*-def*
using *Maps.CLS-in-hom σ.leg1-is-map ϱ.leg0-is-map composable* **by** *simp*
moreover have $r_0{}^*$ $\star$ s_1 $\cong$ (*Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$))* $\star$
Maps.REP (*Maps.MkArr* (*src* s_0) (*trg s*) $[\![s_1]\!]$)
proof −
have r_0 $\cong$ *Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg s*) $[\![r_0]\!]$)
using *Maps.REP-CLS composable isomorphic-symmetric* $r_0 s_1$*.k-is-map* **by** *fastforce*

hence *3*: *isomorphic* $r_0{}^*$ (*Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg* s) $[\![r_0]\!]$))*
using *ϱ.leg0-is-map*
by (*simp add*: *isomorphic-to-left-adjoint-implies-isomorphic-right-adjoint*)
moreover have *4*: $s_1 \cong$ *Maps.REP* (*Maps.MkArr* (*src* s_0) (*trg* s) $[\![s_1]\!]$)
using *Maps.REP-CLS isomorphic-symmetric* $r_0 s_1$*.h-is-map* **by** *fastforce*
ultimately show *?thesis*
proof −
have *1*: *src* $r_0{}^* =$ *trg* s_1
using *ϱ.T0.antipar*(*2*) $r_0 s_1$*.cospan* **by** *argo*
have *2*: *ide* s_1
by *simp*
have *src* (*Maps.REP* (*Maps.MkArr* (*src* r_0) (*trg* s) $[\![r_0]\!]$))* = *trg* s_1
by (*metis 3 ϱ.T0.antipar*(*2*) *isomorphic-implies-hpar*(*3*) $r_0 s_1$*.cospan*)
thus *?thesis*
using *1 2*
by (*meson 3 4 hcomp-ide-isomorphic hcomp-isomorphic-ide isomorphic-implies-ide*(*2*) *isomorphic-transitive*)
qed
qed
ultimately have *1*: *Maps.PRJ*$_0$ $[\![[\![r_0]\!]]\!]$ $[\![[\![s_1]\!]]\!]$ = $[\![[\![prj_0\ s_1\ r_0]\!]]\!]$ ∧
Maps.PRJ$_1$ $[\![[\![r_0]\!]]\!]$ $[\![[\![s_1]\!]]\!]$ = $[\![[\![prj_1\ s_1\ r_0]\!]]\!]$
using $r_0 s_1$*.isomorphic-implies-same-tab* **by** *simp*
show *Maps.PRJ*$_0$ $[\![[\![r_0]\!]]\!]$ $[\![[\![s_1]\!]]\!]$ = $[\![[\![prj_0\ s_1\ r_0]\!]]\!]$
using *1* **by** *simp*
show *Maps.PRJ*$_1$ $[\![[\![r_0]\!]]\!]$ $[\![[\![s_1]\!]]\!]$ = $[\![[\![prj_1\ s_1\ r_0]\!]]\!]$
using *1* **by** *simp*
qed

end

context *identity-in-bicategory-of-spans*
begin

interpretation *Maps*: *maps-category V H* a i *src trg* **..**
interpretation *Span*: *span-bicategory Maps.comp Maps.PRJ*$_0$ *Maps.PRJ*$_1$ **..**

notation *isomorphic* (**infix** ‹≅› *50*)

A 1-cell r in a bicategory of spans is a map if and only if the "input leg" $tab_0\ r$ of the chosen tabulation of r is an equivalence map. Since a tabulation of r is unique up to equivalence, and equivalence maps compose, the result actually holds if "chosen tabulation" is replaced by "any tabulation".

lemma *is-map-iff-tab*$_0$*-is-equivalence*:
shows *is-left-adjoint* $r \longleftrightarrow$ *equivalence-map* ($tab_0\ r$)
proof
assume *1*: *equivalence-map* ($tab_0\ r$)
have *2*: *quasi-inverses* ($tab_0\ r$) ($tab_0\ r$)*
proof −
obtain $g'\ \eta'\ \varepsilon'$ **where** $\eta'\varepsilon'$: *equivalence-in-bicategory V H* a i *src trg* ($tab_0\ r$) $g'\ \eta'\ \varepsilon'$

using *1 equivalence-map-def* **by** *auto*
have *adjoint-pair* $(tab_0\ r)\ g'$
using $\eta'\varepsilon'$ *quasi-inverses-def quasi-inverses-are-adjoint-pair* **by** *blast*
moreover have *adjoint-pair* $(tab_0\ r)\ (tab_0\ r)^*$
using *T0.adjunction-in-bicategory-axioms adjoint-pair-def* **by** *auto*
ultimately have $g' \cong (tab_0\ r)^*$
using *left-adjoint-determines-right-up-to-iso* **by** *simp*
thus *?thesis*
using $\eta'\varepsilon'$ *quasi-inverses-def quasi-inverses-isomorphic-right* **by** *blast*
qed
obtain $\eta'\ \varepsilon'$ **where** $\eta'\varepsilon'$: *equivalence-in-bicategory V H* a i *src trg* $(tab_0\ r)\ (tab_0\ r)^*\ \eta'\ \varepsilon'$
using *2 quasi-inverses-def* **by** *auto*
interpret $\eta'\varepsilon'$: *equivalence-in-bicategory V H* a i *src trg* ‹$tab_0\ r$› ‹$(tab_0\ r)^*$› $\eta'\ \varepsilon'$
using $\eta'\varepsilon'$ **by** *auto*
have *is-left-adjoint* $(tab_0\ r)^*$
using *2 quasi-inverses-are-adjoint-pair quasi-inverses-symmetric* **by** *blast*
hence *is-left-adjoint* $(tab_1\ r \star (tab_0\ r)^*)$
using *left-adjoints-compose* **by** *simp*
thus *is-left-adjoint r*
using *yields-isomorphic-representation isomorphic-def left-adjoint-preserved-by-iso′*
by *meson*
next
assume *1*: *is-left-adjoint r*
have *2*: *is-left-adjoint* $(tab_1\ r \star (tab_0\ r)^*)$
using *1 yields-isomorphic-representation left-adjoint-preserved-by-iso′*
isomorphic-symmetric isomorphic-def
by *meson*
hence *is-left-adjoint* $(tab_0\ r)^*$
using *is-ide BS4* [*of* $tab_1\ r\ (tab_0\ r)^*$] **by** *auto*
hence *is-left-adjoint* $((tab_0\ r)^* \star tab_0\ r) \wedge$ *is-left-adjoint* $(tab_0\ r \star (tab_0\ r)^*)$
using *left-adjoints-compose T0.antipar* **by** *simp*
hence *3*: *iso* $\eta \wedge$ *iso* ε
using *BS3* [*of src* $(tab_0\ r)\ (tab_0\ r)^* \star tab_0\ r\ \eta\ \eta$]
BS3 [*of* $tab_0\ r \star (tab_0\ r)^*$ *trg* $(tab_0\ r)\ \varepsilon\ \varepsilon$]
T0.unit-in-hom T0.counit-in-hom obj-is-self-adjoint
by *auto*
hence *equivalence-in-bicategory V H* a i *src trg* $(tab_0\ r)\ (tab_0\ r)^*\ \eta\ \varepsilon$
apply *unfold-locales* **by** *auto*
thus *equivalence-map* $(tab_0\ r)$
using *equivalence-map-def* **by** *blast*
qed

The chosen tabulation (and indeed, any other tabulation, which is equivalent) of an object is symmetric in the sense that its two legs are isomorphic.

lemma *obj-has-symmetric-tab*:
assumes *obj r*
shows $tab_0\ r \cong tab_1\ r$
proof −
have $tab_0\ r \cong r \star tab_0\ r$

proof −
have *trg* $(tab_0\ r) = r$
using *assms* **by** *auto*
moreover have «$\mathrm{l}^{-1}[tab_0\ r] : tab_0\ r \Rightarrow trg\ (tab_0\ r) \star tab_0\ r$» $\wedge$ *iso* $\mathrm{l}^{-1}[tab_0\ r]$
using *assms* **by** *simp*
ultimately show *?thesis*
unfolding *isomorphic-def* **by** *metis*
qed
also have ... $\cong tab_1\ r$
proof −
have «$tab : tab_1\ r \Rightarrow r \star tab_0\ r$»
using *tab-in-hom* **by** *simp*
moreover have *is-left-adjoint* $(r \star tab_0\ r)$
using *assms left-adjoints-compose obj-is-self-adjoint* **by** *simp*
ultimately show *?thesis*
using *BS3* [*of* $tab_1\ r\ r \star tab_0\ r\ tab\ tab$] *isomorphic-symmetric isomorphic-def*
by *auto*
qed
finally show *?thesis* **by** *simp*
qed

The chosen tabulation of r determines a span in $Maps(B)$.

lemma *determines-span*:
assumes *ide r*
shows *span-in-category Maps.comp* (|*Leg0* = $[\![[\![tab_0\ r]\!]]\!]$, *Leg1* = $[\![[\![tab_1\ r]\!]]\!]$|)
using *assms Maps.CLS-in-hom* [*of* $tab_0\ r$] *Maps.CLS-in-hom* [*of* $tab_1\ r$]
tab_0*-in-hom* tab_1*-in-hom*
apply *unfold-locales* **by** *fastforce*

end

2.3.7 Arrows of Tabulations in Maps

Here we consider the situation of two tabulations: a tabulation ϱ of r and a tabulation σ of s, both "legs" of each tabulation being maps, together with an arbitrary 2-cell «$\mu : r \Rightarrow s$». The 2-cell μ at the base composes with the tabulation ϱ to yield a 2-cell $\Delta = (\mu \star r_0) \cdot \varrho$ "over" s. By property *T1* of tabulation σ, this induces a map from the apex of ϱ to the apex of σ, which together with the other data forms a triangular prism whose sides commute up to (unique) isomorphism.

$$
\begin{tikzcd}
 & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[d, phantom, "\overset{\sigma}{\Rightarrow}"] & \\
\mathrm{trg}\ s \arrow[rr, "s"'] \arrow[dd, equal] \arrow[dr, phantom, "\mu\Uparrow"] & & \mathrm{src}\ s \arrow[dd, equal] \\
 & \mathrm{src}\ \rho \arrow[uul, dotted, "\mathrm{chine}"] \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[d, phantom, "\overset{\rho}{\Rightarrow}"] & \\
\mathrm{trg}\ r \arrow[rr, "r"'] & & \mathrm{src}\ r
\end{tikzcd}
$$

locale *arrow-of-tabulations-in-maps* =
  bicategory-of-spans V H a i *src trg* +
  ϱ: *tabulation-in-maps* V H a i *src trg* r ϱ r_0 r_1 +
  σ: *tabulation-in-maps* V H a i *src trg* s σ s_0 s_1
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r :: $'a$
and ϱ :: $'a$
and r_0 :: $'a$
and r_1 :: $'a$
and s :: $'a$
and σ :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$
and μ :: $'a$ +
assumes *in-hom*: «$\mu : r \Rightarrow s$»
begin

  abbreviation (*input*) Δ
  where $\Delta \equiv (\mu \star r_0) \cdot \varrho$

  lemma Δ-*in-hom* [*intro*]:
  shows «Δ : *src* $\varrho \to$ *trg* σ»
  and «$\Delta : r_1 \Rightarrow s \star r_0$»
  proof −
    show «$\Delta : r_1 \Rightarrow s \star r_0$»
      using *in-hom* ϱ.*leg0-in-hom*(*2*) ϱ.*tab-in-vhom'* **by** *auto*
    thus «Δ : *src* $\varrho \to$ *trg* σ»
      by (*metis* ϱ.*tab-simps*(*3*) ϱ.*base-in-hom*(*2*) σ.*tab-simps*(*3*) σ.*base-in-hom*(*2*) *arrI in-hom seqI' vcomp-in-hhom vseq-implies-hpar*(*1*−*2*))

qed

lemma Δ-*simps* [*simp*]:
shows *arr* Δ
and *src* Δ = *src* ϱ **and** *trg* Δ = *trg* σ
and *dom* $\Delta = r_1$ **and** *cod* $\Delta = s \star r_0$
  using Δ-*in-hom* **by** *auto*

abbreviation *is-induced-map*
where *is-induced-map* $w \equiv \sigma$.*is-induced-by-cell* $w\ r_0\ \Delta$

The following is an equivalent restatement, in elementary terms, of the conditions for being an induced map.

abbreviation (*input*) *is-induced-map′*
where *is-induced-map′* $w \equiv$
    ide $w \wedge$
    $(\exists \nu\ \vartheta.$ «$\vartheta : s_0 \star w \Rightarrow r_0$» $\wedge$ «$\nu : r_1 \Rightarrow s_1 \star w$» $\wedge$ *iso* $\nu \wedge$
      $\Delta = (s \star \vartheta) \cdot \mathrm{a}[s, s_0, w] \cdot (\sigma \star w) \cdot \nu)$

lemma *is-induced-map-iff*:
shows *is-induced-map* $w \longleftrightarrow$ *is-induced-map′* w
proof
  assume w: *is-induced-map′* w
  show *is-induced-map* w
  proof
    have *1*: *dom* $\Delta = r_1$
      by *auto*
    interpret w: *arrow-of-spans-of-maps-to-tabulation* $V\ H$ a i *src trg*
        r_0 ‹*dom* Δ› $s\ \sigma\ s_0\ s_1\ w$
    proof −
      have *arrow-of-spans-of-maps-to-tabulation* $V\ H$ a i *src trg* $r_0\ r_1\ s\ \sigma\ s_0\ s_1\ w$
        using w **apply** *unfold-locales* **by** *auto*
      thus *arrow-of-spans-of-maps-to-tabulation* $V\ H$ a i *src trg* r_0 (*dom* Δ) $s\ \sigma\ s_0\ s_1\ w$
        using *1* **by** *simp*
    qed
    show *arrow-of-spans-of-maps* $V\ H$ a i *src trg* r_0 (*dom* Δ) $s_0\ s_1\ w$
      using w.*arrow-of-spans-of-maps-axioms* **by** *auto*
    show σ.*composite-cell* w w.*the-*$\vartheta \cdot w$.*the-*$\nu = \Delta$
    proof −
      obtain $\vartheta\ \nu$
      where $\vartheta\nu$: «$\vartheta : s_0 \star w \Rightarrow r_0$» $\wedge$ «$\nu : r_1 \Rightarrow s_1 \star w$» $\wedge$ *iso* $\nu \wedge$
        $\Delta = (s \star \vartheta) \cdot \mathrm{a}[s, s_0, w] \cdot (\sigma \star w) \cdot \nu$
        using w w.*the-*ϑ*-props*(*1*) **by** *auto*
      have $(s \star \vartheta) \cdot \mathrm{a}[s, s_0, w] \cdot (\sigma \star w) \cdot \nu = \Delta$
        using $\vartheta\nu$ **by** *argo*
      moreover have $\vartheta = w$.*the-*$\vartheta \wedge \nu = w$.*the-*ν
        using $\vartheta\nu$ *1* w.*the-*ν*-props*(*1*) w.*leg0-uniquely-isomorphic* w.*leg1-uniquely-isomorphic*
        by *auto*
      ultimately show *?thesis*

```
      using comp-assoc by simp
    qed
  qed
  next
  assume w: is-induced-map w
  show is-induced-map′ w
  proof −
    interpret w: arrow-of-spans-of-maps V H a i src trg r₀ r₁ s₀ s₁ w
      using w in-hom by auto
    interpret w: arrow-of-spans-of-maps-to-tabulation V H a i src trg r₀ r₁ s σ s₀ s₁ w
      ..
    have dom Δ = r₁ by auto
    thus ?thesis
      using w comp-assoc w.the-ν-props(1) w.the-ν-props(2) w.uwϑ by metis
  qed
qed

lemma exists-induced-map:
shows ∃ w. is-induced-map w
proof −
  obtain w ϑ ν
  where wϑν: ide w ∧ «ϑ : s₀ ⋆ w ⇒ r₀» ∧ «ν : r₁ ⇒ s₁ ⋆ w» ∧ iso ν ∧
             Δ = (s ⋆ ϑ) · a[s, s₀, w] · (σ ⋆ w) · ν
    using Δ-in-hom ϱ.ide-leg0 σ.T1 comp-assoc
    by (metis in-homE)
  thus ?thesis
    using is-induced-map-iff by blast
qed

lemma induced-map-unique:
assumes is-induced-map w and is-induced-map w′
shows w ≅ w′
  using assms σ.induced-map-unique by blast

definition chine
where chine ≡ SOME w. is-induced-map w

lemma chine-is-induced-map:
shows is-induced-map chine
  unfolding chine-def
  using exists-induced-map someI-ex [of is-induced-map] by simp

lemma chine-in-hom [intro]:
shows «chine : src r₀ → src s₀»
and «chine: chine ⇒ chine»
proof −
  show «chine : src r₀ → src s₀»
    using chine-is-induced-map
    by (metis Δ-simps(1) Δ-simps(4) ϱ.leg1-simps(3) σ.ide-base σ.ide-leg0 σ.leg0-simps(3)
```

σ.tab-simps(*2*) *arrow-of-spans-of-maps.is-ide arrow-of-spans-of-maps.the-ν-simps*(*2*)
assoc-simps(*2*) *hseqE in-hhom-def seqE src-vcomp vseq-implies-hpar*(*1*))
show «*chine*: *chine* ⇒ *chine*»
using *chine-is-induced-map*
by (*meson arrow-of-spans-of-maps.is-ide ide-in-hom*(*2*))
qed

lemma *chine-simps* [*simp*]:
shows *arr chine* **and** *ide chine*
and *src chine* = *src* r_0 **and** *trg chine* = *src* s_0
and *dom chine* = *chine* **and** *cod chine* = *chine*
using *chine-in-hom* **apply** *auto*
by (*meson arrow-of-spans-of-maps.is-ide chine-is-induced-map*)

end

sublocale *arrow-of-tabulations-in-maps* ⊆
arrow-of-spans-of-maps V H a i *src trg* r_0 r_1 s_0 s_1 *chine*
using *chine-is-induced-map is-induced-map-iff*
by *unfold-locales auto*

sublocale *arrow-of-tabulations-in-maps* ⊆
arrow-of-spans-of-maps-to-tabulation V H a i *src trg* r_0 r_1 *s σ* s_0 s_1 *chine*
..

context *arrow-of-tabulations-in-maps*
begin

The two factorizations of the composite 2-cell Δ amount to a naturality condition.

lemma *Δ-naturality*:
shows $(\mu \star r_0) \cdot \varrho = (s \star \textit{the-}\vartheta) \cdot \mathrm{a}[s, s_0, \textit{chine}] \cdot (\sigma \star \textit{chine}) \cdot \textit{the-}\nu$
using *chine-is-induced-map is-induced-map-iff*
by (*metis leg0-uniquely-isomorphic*(*2*) *leg1-uniquely-isomorphic*(*2*) *the-ν-props*(*1*) *uwϑ*)

lemma *induced-map-preserved-by-iso*:
assumes *is-induced-map w* **and** *isomorphic w w′*
shows *is-induced-map w′*
proof −
interpret *w*: *arrow-of-spans-of-maps V H* a i *src trg* r_0 r_1 s_0 s_1 *w*
using *assms in-hom* **by** *auto*
interpret *w*: *arrow-of-spans-of-maps-to-tabulation V H* a i *src trg* r_0 r_1 *s σ* s_0 s_1 *w*
..
obtain *φ* **where** *φ*: «*φ* : *w* ⇒ *w′*» ∧ *iso φ*
using *assms*(*2*) *isomorphic-def* **by** *auto*
show *?thesis*
proof
interpret *w′*: *arrow-of-spans-of-maps V H* a i *src trg* r_0 ‹*dom* Δ› s_0 s_1 *w′*
proof
show *is-left-adjoint* r_0

```
    by (simp add: ϱ.satisfies-T0)
  show is-left-adjoint (dom Δ)
    by (simp add: ϱ.leg1-is-map)
  show ide w' using assms by force
  show ∃ϑ. «ϑ : s₀ ⋆ w' ⇒ r₀»
    using φ w.the-ϑ-props σ.leg0-in-hom(2) assms(2) comp-in-homI hcomp-in-vhom
          inv-in-hom isomorphic-implies-hpar(4) w.the-ϑ-simps(4) w.w-simps(4)
    by metis
  have «(s₁ ⋆ φ) · w.the-ν : r₁ ⇒ s₁ ⋆ w'» ∧ iso ((s₁ ⋆ φ) · w.the-ν)
  proof (intro conjI)
    show «(s₁ ⋆ φ) · w.the-ν : r₁ ⇒ s₁ ⋆ w'»
      using φ w.the-ν-props
      by (intro comp-in-homI, auto)
    thus iso ((s₁ ⋆ φ) · w.the-ν)
      using φ w.the-ν-props
      by (meson σ.ide-leg1 arrI iso-hcomp hseqE ide-is-iso isos-compose seqE)
  qed
  hence ∃ν. «ν : r₁ ⇒ s₁ ⋆ w'» ∧ iso ν
    by auto
  thus ∃ν. «ν : dom Δ ⇒ s₁ ⋆ w'» ∧ iso ν
    by auto
qed
interpret w': arrow-of-spans-of-maps-to-tabulation V H a i src trg
              r₀ ‹dom Δ› s σ s₀ s₁ w'
  ..
show arrow-of-spans-of-maps V H a i src trg r₀ (dom Δ) s₀ s₁ w'
  using w'.arrow-of-spans-of-maps-axioms by auto
show σ.composite-cell w' w'.the-ϑ · w'.the-ν = Δ
proof −
  have 1: w'.the-ϑ = w.the-ϑ · (s₀ ⋆ inv φ)
  proof −
    have «w.the-ϑ · (s₀ ⋆ inv φ) : s₀ ⋆ w' ⇒ r₀»
      using w.the-ϑ-props φ
      by (intro comp-in-homI, auto)
    moreover have «w'.the-ϑ : s₀ ⋆ w' ⇒ r₀»
      using w'.the-ϑ-props by simp
    ultimately show ?thesis
      using w'.leg0-uniquely-isomorphic(2) by blast
  qed
  moreover have w'.the-ν = (s₁ ⋆ φ) · w.the-ν
  proof −
    have «(s₁ ⋆ φ) · w.the-ν : dom Δ ⇒ s₁ ⋆ w'»
      using w.the-ν-props φ
      by (intro comp-in-homI, auto)
    moreover have iso ((s₁ ⋆ φ) · w.the-ν)
      using w.the-ν-props φ iso-hcomp
      by (meson σ.ide-leg1 arrI calculation hseqE ide-is-iso isos-compose seqE)
    ultimately show ?thesis
      using w'.the-ν-props w'.leg1-uniquely-isomorphic(2) by blast
```

```
qed
ultimately have σ.composite-cell w′ w′.the-ϑ · w′.the-ν =
                σ.composite-cell w′ (w.the-ϑ · (s₀ ⋆ inv φ)) · (s₁ ⋆ φ) · w.the-ν
  by simp
also have ... = (s ⋆ w.the-ϑ · (s₀ ⋆ inv φ)) · a[s, s₀, w′] ·
                (σ ⋆ w′) · (s₁ ⋆ φ) · w.the-ν
  using comp-assoc by presburger
also have ... = (s ⋆ w.the-ϑ) · ((s ⋆ s₀ ⋆ inv φ) · a[s, s₀, w′] ·
                (σ ⋆ w′) · (s₁ ⋆ φ)) · w.the-ν
  using 1 comp-assoc w′.the-ϑ-simps(1) whisker-left
  by auto
also have ... = (s ⋆ w.the-ϑ) · (a[s, s₀, w] · (σ ⋆ w)) · w.the-ν
proof −
  have (s ⋆ s₀ ⋆ inv φ) · a[s, s₀, w′] · (σ ⋆ w′) · (s₁ ⋆ φ) =
       a[s, s₀, w] · (σ ⋆ w)
  proof −
    have (s ⋆ s₀ ⋆ inv φ) · a[s, s₀, w′] · (σ ⋆ w′) · (s₁ ⋆ φ) =
         a[s, s₀, w] · ((s ⋆ s₀) ⋆ inv φ) · (σ ⋆ w′) · (s₁ ⋆ φ)
    proof −
      have (s ⋆ s₀ ⋆ inv φ) · a[s, s₀, w′] = a[s, s₀, w] · ((s ⋆ s₀) ⋆ inv φ)
        using assms φ assoc-naturality [of s s₀ inv φ] w.w-simps(4)
        by (metis σ.leg0-simps(2−5) σ.base-simps(2−4) arr-inv cod-inv dom-inv
            in-homE trg-cod)
      thus ?thesis using comp-assoc by metis
    qed
    also have ... = a[s, s₀, w] · (σ ⋆ w) · (s₁ ⋆ inv φ) · (s₁ ⋆ φ)
    proof −
      have ((s ⋆ s₀) ⋆ inv φ) · (σ ⋆ w′) = (σ ⋆ w) · (s₁ ⋆ inv φ)
        using φ comp-arr-dom comp-cod-arr in-hhom-def
              interchange [of s ⋆ s₀ σ inv φ w′]
              interchange [of σ s₁ w inv φ]
        by auto
     thus ?thesis
        using comp-assoc by metis
    qed
    also have ... = a[s, s₀, w] · (σ ⋆ w)
    proof −
      have (σ ⋆ w) · (s₁ ⋆ inv φ) · (s₁ ⋆ φ) = σ ⋆ w
      proof −
        have (σ ⋆ w) · (s₁ ⋆ inv φ) · (s₁ ⋆ φ) = (σ ⋆ w) · (s₁ ⋆ inv φ · φ)
          using φ whisker-left in-hhom-def by auto
        also have ... = (σ ⋆ w) · (s₁ ⋆ w)
          using φ comp-inv-arr′ by auto
        also have ... = σ ⋆ w
          using whisker-right [of w σ s₁] comp-arr-dom in-hhom-def by auto
        finally show ?thesis by blast
      qed
      thus ?thesis by simp
    qed
```

```
        finally show ?thesis by simp
      qed
      thus ?thesis by simp
    qed
    also have ... = Δ
      using assms(1) comp-assoc w.is-ide w.the-ν-props(1) w.the-ϑ-props(1) by simp
    finally show ?thesis
      using comp-assoc by auto
  qed
 qed
qed

end
```

In the special case that μ is an identity 2-cell, the induced map from the apex of ϱ to the apex of σ is an equivalence map.

```
locale identity-arrow-of-tabulations-in-maps =
  arrow-of-tabulations-in-maps +
assumes is-ide: ide μ
begin

  lemma r-eq-s:
  shows r = s
    using is-ide by (metis ide-char in-hom in-homE)

  lemma Δ-eq-ϱ:
  shows Δ = ϱ
    by (meson Δ-simps(1) comp-ide-arr ide-hcomp hseq-char' ide-u is-ide seqE
        seq-if-composable)

  lemma chine-is-equivalence:
  shows equivalence-map chine
  proof −
    obtain w w' φ ψ ϑ ν ϑ' ν'
      where e: equivalence-in-bicategory (·) (⋆) a i src trg w' w ψ φ ∧
               «w : src s0 → src r0» ∧ «w' : src r0 → src s0» ∧
               «ϑ : r0 ⋆ w ⇒ s0» ∧ «ν : s1 ⇒ r1 ⋆ w» ∧ iso ν ∧
               σ = (s ⋆ ϑ) · a[s, r0, w] · (ϱ ⋆ w) · ν ∧
               «ϑ' : s0 ⋆ w' ⇒ r0» ∧ «ν' : r1 ⇒ s1 ⋆ w'» ∧ iso ν' ∧
               ϱ = (s ⋆ ϑ') · a[s, s0, w'] · (σ ⋆ w') · ν'
      using r-eq-s σ.apex-unique-up-to-equivalence [of ϱ r0 r1] ϱ.tabulation-axioms by blast
    have w': equivalence-map w'
      using e equivalence-map-def by auto
    hence is-induced-map w'
      using e r-eq-s Δ-eq-ϱ is-induced-map-iff comp-assoc equivalence-map-is-ide by metis
    hence isomorphic chine w'
      using induced-map-unique chine-is-induced-map by simp
    thus ?thesis
      using w' equivalence-map-preserved-by-iso isomorphic-symmetric by blast
```

qed

end

The following gives an interpretation of *arrow-of-tabulations-in-maps* in the special case that the tabulations are those that we have chosen for the domain and codomain of the underlying 2-cell «$\mu : r \Rightarrow s$». In this case, we can recover μ from Δ via adjoint transpose.

locale *arrow-in-bicategory-of-spans* =
  bicategory-of-spans V H a i *src trg* +
  r: *identity-in-bicategory-of-spans* V H a i *src trg* r +
  s: *identity-in-bicategory-of-spans* V H a i *src trg* s
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r :: $'a$
and s :: $'a$
and μ :: $'a$ +
assumes *in-hom*: «$\mu : r \Rightarrow s$»
begin

  abbreviation (*input*) r_0 **where** $r_0 \equiv tab_0\ r$
  abbreviation (*input*) r_1 **where** $r_1 \equiv tab_1\ r$
  abbreviation (*input*) s_0 **where** $s_0 \equiv tab_0\ s$
  abbreviation (*input*) s_1 **where** $s_1 \equiv tab_1\ s$

  lemma *is-arrow-of-tabulations-in-maps*:
  shows *arrow-of-tabulations-in-maps* V H a i *src trg* r $r.tab$ r_0 r_1 s $s.tab$ s_0 s_1 μ
    using *in-hom* **by** *unfold-locales auto*

end

sublocale *identity-in-bicategory-of-spans* $\subseteq$ *arrow-in-bicategory-of-spans* V H a i *src trg* r r r
  apply *unfold-locales* **using** *is-ide* **by** *auto*

context *arrow-in-bicategory-of-spans*
begin

  interpretation *arrow-of-tabulations-in-maps* V H a i *src trg* r $r.tab$ r_0 r_1 s $s.tab$ s_0 s_1 μ
    using *is-arrow-of-tabulations-in-maps* **by** *simp*
  interpretation r: *arrow-of-tabulations-in-maps* V H a i *src trg* r $r.tab$ r_0 r_1 r $r.tab$ r_0 r_1 r
    using r.*is-arrow-of-tabulations-in-maps* **by** *simp*

  lemma μ-*in-terms-of*-Δ:
  shows $\mu = r.T0.trnr_\varepsilon\ (cod\ \mu)\ \Delta \cdot inv\ (r.T0.trnr_\varepsilon\ r\ r.tab)$
  proof −

```
  have μ: arr μ
    using in-hom by auto
  have μ · r.T0.trnr_ε r r.tab = r.T0.trnr_ε s Δ
  proof −
    have μ · r.T0.trnr_ε r r.tab =
          (μ · r[r]) · (r ⋆ r.ε) · a[r, tab_0 r, (tab_0 r)*] · (r.tab ⋆ (tab_0 r)*)
      unfolding r.T0.trnr_ε-def using comp-assoc by presburger
    also have ... = r[s] · ((μ ⋆ src μ) · (r ⋆ r.ε)) ·
                    a[r, tab_0 r, (tab_0 r)*] · (r.tab ⋆ (tab_0 r)*)
      using μ runit-naturality comp-assoc
      by (metis in-hom in-homE)
    also have ... = r[s] · (s ⋆ r.ε) · ((μ ⋆ tab_0 r ⋆ (tab_0 r)*) ·
                    a[r, tab_0 r, (tab_0 r)*]) · (r.tab ⋆ (tab_0 r)*)
    proof −
      have (μ ⋆ src μ) · (r ⋆ r.ε) = μ ⋆ r.ε
        using μ interchange comp-arr-dom comp-cod-arr
        by (metis in-hom in-homE r.T0.counit-simps(1) r.T0.counit-simps(3) r.u-simps(3)
            src-dom)
      also have ... = (s ⋆ r.ε) · (μ ⋆ tab_0 r ⋆ (tab_0 r)*)
        using in-hom interchange [of s μ r.ε tab_0 r ⋆ (tab_0 r)*]
              comp-arr-dom comp-cod-arr r.T0.counit-simps(1) r.T0.counit-simps(2)
        by auto
      finally have (μ ⋆ src μ) · (r ⋆ r.ε) = (s ⋆ r.ε) · (μ ⋆ tab_0 r ⋆ (tab_0 r)*)
        by blast
      thus ?thesis
        using comp-assoc by presburger
    qed
    also have ... = r[s] · (s ⋆ r.ε) · a[s, tab_0 r, (tab_0 r)*] ·
                    ((μ ⋆ tab_0 r) ⋆ (tab_0 r)*) · (r.tab ⋆ (tab_0 r)*)
    proof −
      have (μ ⋆ tab_0 r ⋆ (tab_0 r)*) · a[r, tab_0 r, (tab_0 r)*] =
            a[s, tab_0 r, (tab_0 r)*] · ((μ ⋆ tab_0 r) ⋆ (tab_0 r)*)
        using μ assoc-naturality [of μ tab_0 r (tab_0 r)*]
        by (metis ide-char in-hom in-homE r.T0.antipar(1) r.T0.ide-right r.u-simps(3)
            src-dom u-simps(2) u-simps(4−5))
      thus ?thesis
        using comp-assoc by presburger
    qed
    also have ... = r[s] · (s ⋆ r.ε) · a[s, tab_0 r, (tab_0 r)*] ·
                    ((μ ⋆ tab_0 r) · r.tab ⋆ (tab_0 r)*)
      using μ whisker-right Δ-simps(1) by auto
    also have ... = r.T0.trnr_ε s Δ
      unfolding r.T0.trnr_ε-def by simp
    finally show ?thesis by blast
  qed
  thus ?thesis
    using μ r.yields-isomorphic-representation invert-side-of-triangle(2)
    by (metis in-hom in-homE seqI')
qed
```

end

Vertical Composite

locale *vertical-composite-of-arrows-of-tabulations-in-maps* =
  bicategory-of-spans V H a i *src trg* +
  ϱ: *tabulation-in-maps* V H a i *src trg* r ϱ r_0 r_1 +
  σ: *tabulation-in-maps* V H a i *src trg* s σ s_0 s_1 +
  τ: *tabulation-in-maps* V H a i *src trg* t τ t_0 t_1 +
  μ: *arrow-of-tabulations-in-maps* V H a i *src trg* r ϱ r_0 r_1 s σ s_0 s_1 μ +
  π: *arrow-of-tabulations-in-maps* V H a i *src trg* s σ s_0 s_1 t τ t_0 t_1 π
for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r :: $'a$
and ϱ :: $'a$
and r_0 :: $'a$
and r_1 :: $'a$
and s :: $'a$
and σ :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$
and t :: $'a$
and τ :: $'a$
and t_0 :: $'a$
and t_1 :: $'a$
and μ :: $'a$
and π :: $'a$
begin

notation *isomorphic* (**infix** ‹≅› *50*)

interpretation *arrow-of-tabulations-in-maps V H* a i *src trg r ϱ* r_0 r_1 *t τ* t_0 t_1 ‹π · μ›
  using *μ.in-hom π.in-hom* **by** (*unfold-locales*, *blast*)

lemma *is-arrow-of-tabulations-in-maps*:
shows *arrow-of-tabulations-in-maps V H* a i *src trg r ϱ* r_0 r_1 *t τ* t_0 t_1 (π · μ)
  ..

lemma *chine-char*:
shows *chine* ≅ *π.chine* ⋆ *μ.chine*
proof −
  have *is-induced-map* (*π.chine* ⋆ *μ.chine*)
  proof −
    let *?f* = *μ.chine*
    have *f*: «*?f* : *src ϱ* → *src σ*» ∧ *is-left-adjoint ?f* ∧ *ide ?f* ∧ *μ.is-induced-map ?f*
      using *μ.chine-is-induced-map μ.is-map* **by** *auto*
    let *?g* = *π.chine*
    have *g*: «*?g* : *src σ* → *src τ*» ∧ *is-left-adjoint ?g* ∧ *ide ?g* ∧ *π.is-induced-map ?g*
      using *π.chine-is-induced-map π.is-map* **by** *auto*
    let *?ϑ* = *μ.the-ϑ* · (*π.the-ϑ* ⋆ *?f*) · $a^{-1}[t_0, ?g, ?f]$
    let *?ν* = $a[t_1, ?g, ?f]$ · (*π.the-ν* ⋆ *?f*) · *μ.the-ν*
    have *ϑ*: «*?ϑ* : t_0 ⋆ *?g* ⋆ *?f* ⇒ r_0»
      using *f g π.the-ϑ-props μ.the-ϑ-props*
      by (*intro comp-in-homI hcomp-in-vhom*, *auto*+)
    have *ν*: «*?ν* : r_1 ⇒ t_1 ⋆ *?g* ⋆ *?f*»

using *f g* π*.the-*ϑ*-props* μ*.the-*ϑ*-props*
by (*intro comp-in-homI hcomp-in-vhom*, *auto*)
interpret *gf*: *arrow-of-spans-of-maps V H* a i *src trg* r_0 r_1 t_0 t_1 ‹*?g* $\star$ *?f*›
proof
show *ide* (*?g* $\star$ *?f*) **by** *simp*
show $\exists\,\vartheta$. «ϑ : t_0 $\star$ *?g* $\star$ *?f* $\Rightarrow$ r_0»
using ϑ **by** *blast*
show $\exists\,\nu$. «ν : r_1 $\Rightarrow$ t_1 $\star$ *?g* $\star$ *?f*» $\wedge$ *iso* ν
using ν μ*.the-*ν*-props* μ*.the-*ϑ*-props* π*.the-*ν*-props* π*.the-*ϑ*-props*
isos-compose [*of* μ*.the-*ν π*.the-*ν] μ*.is-ide* ν ‹*ide* (π*.chine* $\star$ μ*.chine*)›
π*.uw*ϑ π*.w-simps*(*4*) τ*.ide-leg1* τ*.leg1-simps*(*3*) *arrI hseq-char ideD*(*1*)
ide-is-iso iso-assoc iso-hcomp isos-compose seqE
by *metis*
qed
show *?thesis*
proof (*intro conjI*)
have ϑ*-eq*: *?*ϑ = *gf.the-*ϑ
using ϑ *gf.the-*ϑ*-props gf.leg0-uniquely-isomorphic* **by** *auto*
have ν*-eq*: *?*ν = *gf.the-*ν
using ν *gf.the-*ν*-props gf.leg1-uniquely-isomorphic* **by** *auto*
have *A*: *src ?g* = *trg ?f*
using *f g* **by** *fastforce*
show *arrow-of-spans-of-maps V H* a i *src trg* r_0 (*dom* Δ) t_0 t_1 (*?g* $\star$ *?f*)
using *gf.arrow-of-spans-of-maps-axioms* **by** *simp*
have ((t $\star$ *gf.the-*ϑ) $\cdot$ a[t, t_0, *?g* $\star$ *?f*] $\cdot$ (τ $\star$ *?g* $\star$ *?f*)) $\cdot$ *gf.the-*ν = Δ
proof −
have Δ = (π $\star$ r_0) $\cdot$ (μ $\star$ r_0) $\cdot$ ϱ
using *whisker-right comp-assoc*
by (*metis* Δ*-simps*(*1*) *hseqE ide-u seqE*)
also have ... = ((π $\star$ r_0) $\cdot$ (s $\star$ μ*.the-*ϑ)) $\cdot$ a[s, s_0, *?f*] $\cdot$ (σ $\star$ *?f*) $\cdot$ μ*.the-*ν
using μ.Δ*-naturality comp-assoc* **by** *simp*
also have ... = (t $\star$ μ*.the-*ϑ) $\cdot$ ((π $\star$ s_0 $\star$ *?f*) $\cdot$ a[s, s_0, *?f*]) $\cdot$ (σ $\star$ *?f*) $\cdot$ μ*.the-*ν
proof −
have (π $\star$ r_0) $\cdot$ (s $\star$ μ*.the-*ϑ) = π $\star$ μ*.the-*ϑ
using *f comp-arr-dom comp-cod-arr* μ*.the-*ϑ*-props* π*.in-hom*
interchange [*of* π s r_0 μ*.the-*ϑ]
by (*metis in-homE*)
also have ... = (t $\star$ μ*.the-*ϑ) $\cdot$ (π $\star$ s_0 $\star$ *?f*)
using *f comp-arr-dom comp-cod-arr* μ*.the-*ϑ*-props* π*.in-hom*
interchange [*of* t π μ*.the-*ϑ s_0 $\star$ *?f*]
by (*metis in-homE*)
finally have (π $\star$ r_0) $\cdot$ (s $\star$ μ*.the-*ϑ) = (t $\star$ μ*.the-*ϑ) $\cdot$ (π $\star$ s_0 $\star$ *?f*)
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... = (t $\star$ μ*.the-*ϑ) $\cdot$ a[t, s_0, *?f*] $\cdot$ (((π $\star$ s_0) $\star$ *?f*) $\cdot$ (σ $\star$ *?f*)) $\cdot$ μ*.the-*ν
proof −
have (π $\star$ s_0 $\star$ *?f*) $\cdot$ a[s, s_0, *?f*] = a[t, s_0, *?f*] $\cdot$ ((π $\star$ s_0) $\star$ *?f*)

using *f assoc-naturality* [*of* π s_0 *?f*] π*.in-hom* **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ s_0,\ ?f] \cdot (\pi.\Delta \star ?f) \cdot \mu.\textit{the-}\nu$
using *whisker-right comp-assoc* **by** *simp*
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ s_0,\ ?f] \cdot$
$((t \star \pi.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ t_0,\ ?g] \cdot (\tau \star ?g) \cdot \pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
using $\pi.\Delta$*-naturality* **by** *simp*
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ s_0,\ ?f] \cdot$
$(((t \star \pi.\textit{the-}\vartheta) \star ?f) \cdot (\mathrm{a}[t,\ t_0,\ ?g] \star ?f) \cdot ((\tau \star ?g) \star ?f) \cdot$
$(\pi.\textit{the-}\nu \star ?f)) \cdot \mu.\textit{the-}\nu$
using *f g* π*.the-*ϑ*-props* π*.the-*ν*-props whisker-right*
by (*metis* $\pi.\Delta$*-simps*(*1*) $\pi.\Delta$*-naturality seqE*)
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot (\mathrm{a}[t,\ s_0,\ ?f] \cdot$
$((t \star \pi.\textit{the-}\vartheta) \star ?f)) \cdot (\mathrm{a}[t,\ t_0,\ ?g] \star ?f) \cdot ((\tau \star ?g) \star ?f) \cdot$
$(\pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
using *comp-assoc* **by** *presburger*
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot (t \star \pi.\textit{the-}\vartheta \star ?f) \cdot$
$(\mathrm{a}[t,\ t_0 \star ?g,\ ?f] \cdot (\mathrm{a}[t,\ t_0,\ ?g] \star ?f)) \cdot$
$((\tau \star ?g) \star ?f) \cdot (\pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
using *f* π*.the-*ϑ*-props assoc-naturality* [*of t* π*.the-*ϑ *?f*] $\pi.\vartheta$*-simps*(*3*) *comp-assoc*
by *auto*
also have ... $= (t \star \mu.\textit{the-}\vartheta) \cdot (t \star \pi.\textit{the-}\vartheta \star ?f) \cdot$
$(t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]) \cdot \mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot (\mathrm{a}[t \star t_0,\ ?g,\ ?f] \cdot$
$((\tau \star ?g) \star ?f)) \cdot (\pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
proof −
have *seq* $\mathrm{a}[t,\ t_0,\ ?g \star ?f]\ \mathrm{a}[t \star t_0,\ ?g,\ ?f]$
using *f g* **by** *fastforce*
moreover have *inv* $(t \star \mathrm{a}[t_0,\ ?g,\ ?f]) = t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]$
using *f g* **by** *simp*
moreover have *iso* $(t \star \mathrm{a}[t_0,\ ?g,\ ?f])$
using *f g* **by** *simp*
have $\mathrm{a}[t,\ t_0 \star ?g,\ ?f] \cdot (\mathrm{a}[t,\ t_0,\ ?g] \star ?f) =$
$(t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]) \cdot \mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot \mathrm{a}[t \star t_0,\ ?g,\ ?f]$
proof −
have *seq* $\mathrm{a}[t,\ t_0,\ ?g \star ?f]\ \mathrm{a}[t \star t_0,\ ?g,\ ?f]$
using *f g* **by** *fastforce*
moreover have *inv* $(t \star \mathrm{a}[t_0,\ ?g,\ ?f]) = t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]$
using *f g* **by** *simp*
moreover have *iso* $(t \star \mathrm{a}[t_0,\ ?g,\ ?f])$
using *f g* **by** *simp*
ultimately show *?thesis*
using *A f g pentagon invert-side-of-triangle*(*1*)
by (*metis* π*.w-simps*(*4*) τ*.ide-base* τ*.ide-leg0* τ*.leg0-simps*(*3*))
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed

also have ... $= ((t \star \mu.\textit{the-}\vartheta) \cdot (t \star \pi.\textit{the-}\vartheta \star ?f)) \cdot$
$(t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]) \cdot \mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot (\tau \star ?g \star ?f) \cdot$
$\mathrm{a}[t_1,\ ?g,\ ?f] \cdot (\pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
using *f g assoc-naturality* [*of* τ *?g ?f*] *comp-assoc* **by** *simp*
also have ... $= (t \star \mu.\textit{the-}\vartheta \cdot (\pi.\textit{the-}\vartheta \star ?f) \cdot \mathrm{a}^{-1}[t_0,\ ?g,\ ?f]) \cdot$
$\mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot$
$(\tau \star ?g \star ?f) \cdot \mathrm{a}[t_1,\ ?g,\ ?f] \cdot (\pi.\textit{the-}\nu \star ?f) \cdot \mu.\textit{the-}\nu$
proof −
have *1*: *seq* $\mu.\textit{the-}\vartheta$ $((\pi.\textit{the-}\vartheta \star ?f) \cdot \mathrm{a}^{-1}[t_0,\ ?g,\ ?f])$
using ϑ*-eq* **by** *auto*
hence $t \star (\pi.\textit{the-}\vartheta \star ?f) \cdot \mathrm{a}^{-1}[t_0,\ ?g,\ ?f] =$
$(t \star \pi.\textit{the-}\vartheta \star ?f) \cdot (t \star \mathrm{a}^{-1}[t_0,\ ?g,\ ?f])$
using *whisker-left* τ*.ide-base* **by** *blast*
thus *?thesis*
using *1 whisker-left* τ*.ide-base comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star gf.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot (\tau \star ?g \star ?f)) \cdot gf.\textit{the-}\nu$
using ϑ*-eq* ν*-eq* **by** (*simp add*: *comp-assoc*)
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
thus $((t \star gf.\textit{the-}\vartheta) \cdot \mathrm{a}[t,\ t_0,\ ?g \star ?f] \cdot (\tau \star ?g \star ?f)) \cdot$
arrow-of-spans-of-maps.the-ν $(\cdot)$ $(\star)$ $(\textit{dom}\ ((\pi \cdot \mu \star r_0) \cdot \varrho))\ t_1\ (?g \star ?f) =$
Δ
by *simp*
qed
qed
thus *?thesis*
using *chine-is-induced-map induced-map-unique* **by** *simp*
qed

end

sublocale *vertical-composite-of-arrows-of-tabulations-in-maps* $\subseteq$
arrow-of-tabulations-in-maps $V\ H$ a i *src trg* $r\ \varrho\ r_0\ r_1\ t\ \tau\ t_0\ t_1\ \pi \cdot \mu$
using *is-arrow-of-tabulations-in-maps* **by** *simp*

Horizontal Composite

locale *horizontal-composite-of-arrows-of-tabulations-in-maps* =
bicategory-of-spans $V\ H$ a i *src trg* +
ϱ: *tabulation-in-maps* $V\ H$ a i *src trg* $r\ \varrho\ r_0\ r_1$ +
σ: *tabulation-in-maps* $V\ H$ a i *src trg* $s\ \sigma\ s_0\ s_1$ +
τ: *tabulation-in-maps* $V\ H$ a i *src trg* $t\ \tau\ t_0\ t_1$ +
μ: *tabulation-in-maps* $V\ H$ a i *src trg* $u\ \mu\ u_0\ u_1$ +
$\varrho\sigma$: *composite-tabulation-in-maps* $V\ H$ a i *src trg* $r\ \varrho\ r_0\ r_1\ s\ \sigma\ s_0\ s_1$ +
$\tau\mu$: *composite-tabulation-in-maps* $V\ H$ a i *src trg* $t\ \tau\ t_0\ t_1\ u\ \mu\ u_0\ u_1$ +
ω: *arrow-of-tabulations-in-maps* $V\ H$ a i *src trg* $r\ \varrho\ r_0\ r_1\ t\ \tau\ t_0\ t_1\ \omega$ +
χ: *arrow-of-tabulations-in-maps* $V\ H$ a i *src trg* $s\ \sigma\ s_0\ s_1\ u\ \mu\ u_0\ u_1\ \chi$

for V :: $'a$ *comp* (**infixr** ‹·› *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⋆› *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a[-, -, -]›)
and i :: $'a \Rightarrow 'a$ (‹i[-]›)
and *src* :: $'a \Rightarrow 'a$
and *trg* :: $'a \Rightarrow 'a$
and r :: $'a$
and ϱ :: $'a$
and r_0 :: $'a$
and r_1 :: $'a$
and s :: $'a$
and σ :: $'a$
and s_0 :: $'a$
and s_1 :: $'a$
and t :: $'a$
and τ :: $'a$
and t_0 :: $'a$
and t_1 :: $'a$
and u :: $'a$
and μ :: $'a$
and u_0 :: $'a$
and u_1 :: $'a$
and ω :: $'a$
and χ :: $'a$
begin

$$
\begin{tikzcd}
& & & \mathrm{src}\ t_0u_1.\phi \arrow[dl, "\tau\mu.p_1"'] \arrow[dr, "\tau\mu.p_0"] \arrow[dd, phantom, "\overset{t_0u_1.\phi}{\Rightarrow}"] & & \\
& & \mathrm{src}\ \tau \arrow[dl, "t_1"'] \arrow[dr, "t_0"] \arrow[dd, phantom, "\overset{\tau}{\Rightarrow}"] & & \mathrm{src}\ \mu \arrow[dl, "u_1"'] \arrow[dr, "u_0"] \arrow[dd, phantom, "\overset{\mu}{\Rightarrow}"] & \\
\mathrm{trg}\ t \arrow[rr, "t"'] & & \mathrm{src}\ t = \mathrm{trg}\ u \arrow[rr, "u"'] & & \mathrm{src}\ u & \\
& \mathrm{src}\ r_0s_1.\phi \arrow[uul, "\omega.\mathrm{chine}"] \arrow[uur, "\mathrm{chine}"] \arrow[dl, "\rho\sigma.p_1"'] \arrow[dr, "\rho\sigma.p_0"] \arrow[dd, phantom, "\overset{r_0s_1.\phi}{\Rightarrow}"] & & \chi.\mathrm{chine} & & \\
& \mathrm{src}\ \rho \arrow[dl, "r_1"'] \arrow[dr, "r_0"] \arrow[dd, phantom, "\overset{\rho}{\Rightarrow}"] & & \mathrm{src}\ \sigma \arrow[dl, "s_1"'] \arrow[dr, "s_0"] \arrow[dd, phantom, "\overset{\sigma}{\Rightarrow}"] & & \\
\mathrm{trg}\ r \arrow[rr, "r"'] & & \mathrm{src}\ r = \mathrm{trg}\ s \arrow[rr, "s"'] & & \mathrm{src}\ s & \\
\omega\Uparrow & & & \chi\Uparrow & &
\end{tikzcd}
$$

notation *isomorphic* (**infix** ‹≅› *50*)

interpretation *arrow-of-tabulations-in-maps* V H a i *src trg*
‹$r \star s$› $\varrho\sigma.tab$ ‹$s_0 \star \varrho\sigma.p_0$› ‹$r_1 \star \varrho\sigma.p_1$›

$\langle t \star u\rangle$ $\tau\mu.tab$ $\langle u_0 \star \tau\mu.p_0\rangle$ $\langle t_1 \star \tau\mu.p_1\rangle$ $\langle \omega \star \chi\rangle$
using $\varrho\sigma.composable$ $\omega.in\text{-}hom$ $\chi.in\text{-}hom$
by *unfold-locales auto*

lemma *is-arrow-of-tabulations-in-maps*:
shows *arrow-of-tabulations-in-maps* V H a i *src trg*
$(r \star s)$ $\varrho\sigma.tab$ $(s_0 \star \varrho\sigma.p_0)$ $(r_1 \star \varrho\sigma.p_1)$
$(t \star u)$ $\tau\mu.tab$ $(u_0 \star \tau\mu.p_0)$ $(t_1 \star \tau\mu.p_1)$ $(\omega \star \chi)$
..

sublocale *arrow-of-tabulations-in-maps* V H a i *src trg*
$\langle r \star s\rangle$ $\varrho\sigma.tab$ $\langle s_0 \star \varrho\sigma.p_0\rangle$ $\langle r_1 \star \varrho\sigma.p_1\rangle$
$\langle t \star u\rangle$ $\tau\mu.tab$ $\langle u_0 \star \tau\mu.p_0\rangle$ $\langle t_1 \star \tau\mu.p_1\rangle$ $\langle \omega \star \chi\rangle$
using *is-arrow-of-tabulations-in-maps* **by** *simp*

interpretation *Maps*: *maps-category* V H a i *src trg* **..**
notation *Maps.comp* (**infixr** $\langle\odot\rangle$ *55*)

interpretation r_0s_1: *cospan-of-maps-in-bicategory-of-spans* $\langle(\cdot)\rangle$ $\langle(\star)\rangle$ a i *src trg* s_1 r_0
using $\varrho.leg0\text{-}is\text{-}map$ $\sigma.leg1\text{-}is\text{-}map$ $\varrho\sigma.composable$ **apply** *unfold-locales* **by** *auto*
interpretation r_0s_1: *arrow-of-tabulations-in-maps* $\langle(\cdot)\rangle$ $\langle(\star)\rangle$ a i *src trg*
$\langle {r_0}^* \star s_1\rangle$ $r_0s_1.tab$ $r_0s_1.p_0$ $r_0s_1.p_1$
$\langle {r_0}^* \star s_1\rangle$ $r_0s_1.tab$ $r_0s_1.p_0$ $r_0s_1.p_1$
$\langle {r_0}^* \star s_1\rangle$
using $r_0s_1.is\text{-}arrow\text{-}of\text{-}tabulations\text{-}in\text{-}maps$ **by** *simp*
interpretation t_0u_1: *cospan-of-maps-in-bicategory-of-spans* $\langle(\cdot)\rangle$ $\langle(\star)\rangle$ a i *src trg* u_1 t_0
using $\tau.leg0\text{-}is\text{-}map$ $\mu.leg1\text{-}is\text{-}map$ $\tau\mu.composable$ **apply** *unfold-locales* **by** *auto*
interpretation t_0u_1: *arrow-of-tabulations-in-maps* $\langle(\cdot)\rangle$ $\langle(\star)\rangle$ a i *src trg*
$\langle {t_0}^* \star u_1\rangle$ $t_0u_1.tab$ $t_0u_1.p_0$ $t_0u_1.p_1$
$\langle {t_0}^* \star u_1\rangle$ $t_0u_1.tab$ $t_0u_1.p_0$ $t_0u_1.p_1$
$\langle {t_0}^* \star u_1\rangle$
using $t_0u_1.is\text{-}arrow\text{-}of\text{-}tabulations\text{-}in\text{-}maps$ **by** *simp*

interpretation E: *self-evaluation-map* V H a i *src trg* **..**
notation $E.eval$ ($\langle\{\!|\text{-}|\!\}\rangle$)

no-notation *in-hom* ($\langle\langle\text{- : - } \rightarrow \text{ -}\rangle\rangle$)

The following lemma states that the rectangular faces of the "top prism" commute up to isomorphism. This was not already proved in *composite-tabulation-in-maps*, because there we did not consider any composite structure of the "source" 2-cell. There are common elements, though to the proof that the composite of tabulations is a tabulation and the present lemma. The proof idea is to use property *T2* of the "base" tabulations to establish the existence of the desired isomorphisms. The proofs have to be carried out in sequence, starting from the "output" side, because the arrow β required in the hypotheses of *T2* depends, for the "input" tabulation, on the isomorphism constructed for the "output" tabulation.

lemma *prj-chine*:

shows $\tau\mu.p_0 \star chine \cong \chi.chine \star \varrho\sigma.p_0$
and $\tau\mu.p_1 \star chine \cong \omega.chine \star \varrho\sigma.p_1$
proof −
  have *1*: *arrow-of-spans-of-maps* V H a i *src trg*
    $(s_0 \star \varrho\sigma.p_0)$ $(r_1 \star \varrho\sigma.p_1)$ $(u_0 \star \tau\mu.p_0)$ $(t_1 \star \tau\mu.p_1)$ $chine \wedge$
    $(((t \star u) \star the\text{-}\vartheta) \cdot \mathrm{a}[t \star u,\ u_0 \star \tau\mu.p_0,\ chine] \cdot (\tau\mu.tab \star chine)) \cdot the\text{-}\nu =$
    $((\omega \star \chi) \star s_0 \star \varrho\sigma.p_0) \cdot \varrho\sigma.tab$
    using *chine-is-induced-map* **by** *simp*
  let $?u_\tau = u \star s_0 \star \varrho\sigma.p_0$
  let $?w_\tau = \omega.chine \star \varrho\sigma.p_1$
  let $?w_\tau{}' = \tau\mu.p_1 \star chine$
  have u_τ: *ide* $?u_\tau$
    using χ.*u-simps*(*3*) **by** *auto*
  have w_τ: *ide* $?w_\tau \wedge$ *is-left-adjoint* $?w_\tau$
    by (*simp add*: ω.*is-map* ϱ.*T0.antipar*(*1*) *left-adjoints-compose*)
  have $w_\tau{}'$: *ide* $?w_\tau{}' \wedge$ *is-left-adjoint* $?w_\tau{}'$
    by (*simp add*: *is-map left-adjoints-compose*)
  let $?\vartheta_\tau = \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot (\omega.the\text{-}\vartheta \star \varrho\sigma.p_1) \cdot$
      $\mathrm{a}^{-1}[t_0,\ \omega.chine,\ \varrho\sigma.p_1]$
  let $?\vartheta_\tau{}' = (u \star the\text{-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ chine] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star chine) \cdot$
      $((\mu \star \tau\mu.p_0) \star chine) \cdot (t_0 u_1.\varphi \star chine) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ chine]$
  let $?\beta_\tau = \mathrm{a}[t_1,\ \tau\mu.p_1,\ chine] \cdot the\text{-}\nu \cdot (inv\ \omega.the\text{-}\nu \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_1,\ \omega.chine,\ \varrho\sigma.p_1]$
  have ϑ_τ: «$?\vartheta_\tau : t_0 \star ?w_\tau \Rightarrow ?u_\tau$»
    using ϱ.*T0.antipar*(*1*) ω.*the-ϑ-in-hom* χ.*u-simps*(*3*)
    by (*intro comp-in-homI*, *auto*)
  have $\vartheta_\tau{}'$: «$?\vartheta_\tau{}' : t_0 \star ?w_\tau{}' \Rightarrow ?u_\tau$»
  proof (*intro comp-in-homI*)
    show «$\mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ chine] : t_0 \star \tau\mu.p_1 \star chine \Rightarrow (t_0 \star \tau\mu.p_1) \star chine$»
      using $t_0 u_1.p_1$*-simps assoc′-in-hom* **by** *simp*
    show «$t_0 u_1.\varphi \star chine : (t_0 \star \tau\mu.p_1) \star chine \Rightarrow (u_1 \star \tau\mu.p_0) \star chine$»
      using τ.*T0.antipar*(*1*)
      by (*intro hcomp-in-vhom*, *auto*)
    show «$(\mu \star \tau\mu.p_0) \star chine : (u_1 \star \tau\mu.p_0) \star chine \Rightarrow ((u \star u_0) \star \tau\mu.p_0) \star chine$»
      by (*intro hcomp-in-vhom*, *auto*)
    show «$\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star chine : ((u \star u_0) \star \tau\mu.p_0) \star chine \Rightarrow (u \star u_0 \star \tau\mu.p_0) \star chine$»
      using *assoc-in-hom* **by** *auto*
    show «$\mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ chine] : (u \star u_0 \star \tau\mu.p_0) \star chine \Rightarrow u \star (u_0 \star \tau\mu.p_0) \star chine$»
      by *auto*
    show «$u \star the\text{-}\vartheta : u \star (u_0 \star \tau\mu.p_0) \star chine \Rightarrow ?u_\tau$»
      by (*intro hcomp-in-vhom*, *auto*)
  qed
  have β_τ: «$?\beta_\tau : t_1 \star ?w_\tau \Rightarrow t_1 \star ?w_\tau{}'$»
  proof (*intro comp-in-homI*)
    show «$\mathrm{a}^{-1}[t_1,\ \omega.chine,\ \varrho\sigma.p_1] : t_1 \star ?w_\tau \Rightarrow (t_1 \star \omega.chine) \star \varrho\sigma.p_1$»
      using ϱ.*T0.antipar*(*1*) **by** *auto*
    show «$inv\ \omega.the\text{-}\nu \star \varrho\sigma.p_1 : (t_1 \star \omega.chine) \star \varrho\sigma.p_1 \Rightarrow r_1 \star \varrho\sigma.p_1$»
      using ω.*the-ν-props* ϱ.*T0.antipar*(*1*)
      by (*intro hcomp-in-vhom*, *auto*)
    show «$the\text{-}\nu : r_1 \star \varrho\sigma.p_1 \Rightarrow (t_1 \star \tau\mu.p_1) \star chine$»

using *the-ν-in-hom*(*2*) **by** *simp*
show «a[t_1, $\tau\mu.p_1$, *chine*] : (t_1 ⋆ $\tau\mu.p_1$) ⋆ *chine* ⇒ t_1 ⋆ *?*w_τ'»
using $t_0u_1.p_1$*-simps assoc-in-hom* **by** *simp*
qed
define *LHS* **where** *LHS* = (t ⋆ *?*ϑ_τ) · a[t, t_0, *?*w_τ] · (τ ⋆ *?*w_τ)
have *LHS*: «*LHS* : t_1 ⋆ *?*w_τ ⇒ t ⋆ *?*u_τ»
proof (*unfold LHS-def*, *intro comp-in-homI*)
show «τ ⋆ *?*w_τ : t_1 ⋆ ω.*chine* ⋆ $\varrho\sigma.p_1$ ⇒ (t ⋆ t_0) ⋆ *?*w_τ»
using ϱ.*T0.antipar*(*1*)
by (*intro hcomp-in-vhom*, *auto*)
show «a[t, t_0, *?*w_τ] : (t ⋆ t_0) ⋆ *?*w_τ ⇒ t ⋆ t_0 ⋆ *?*w_τ»
using ϱ.*T0.antipar*(*1*) **by** *auto*
show «t ⋆ *?*ϑ_τ : t ⋆ t_0 ⋆ *?*w_τ ⇒ t ⋆ *?*u_τ»
proof −
have *src* t = *trg* (t_0 ⋆ ω.*chine* ⋆ $r_0s_1.p_1$)
by (*metis* χ.*u-simps*(*3*) μ.*ide-base* σ.*ide-leg0* σ.*leg1-simps*(*3*) $\tau\mu$.*composable* ϑ_τ *arrI assoc-simps*(*3*) r_0s_1.*ide-u* $r_0s_1.p_0$*-simps trg-vcomp vconn-implies-hpar*(*2*))
thus *?thesis*
using ϑ_τ **by** *blast*
qed
qed
define *RHS* **where** *RHS* = ((t ⋆ *?*ϑ_τ') · a[t, t_0, *?*w_τ'] · (τ ⋆ *?*w_τ')) · *?*β_τ
have *RHS*: «*RHS* : t_1 ⋆ *?*w_τ ⇒ t ⋆ *?*u_τ»
unfolding *RHS-def*
proof
show «*?*β_τ : t_1 ⋆ *?*w_τ ⇒ t_1 ⋆ *?*w_τ'»
using β_τ **by** *simp*
show «(t ⋆ *?*ϑ_τ') · a[t, t_0, *?*w_τ'] · (τ ⋆ *?*w_τ') : t_1 ⋆ *?*w_τ' ⇒ t ⋆ *?*u_τ»
proof
show «a[t, t_0, *?*w_τ'] · (τ ⋆ *?*w_τ') : t_1 ⋆ *?*w_τ' ⇒ t ⋆ t_0 ⋆ *?*w_τ'»
using τ.*T0.antipar*(*1*) **by** *fastforce*
show «t ⋆ *?*ϑ_τ' : t ⋆ t_0 ⋆ *?*w_τ' ⇒ t ⋆ *?*u_τ»
using w_τ' ϑ_τ' τ.*leg0-simps*(*2*) τ.*leg0-simps*(*3*) *hseqI'* *ideD*(*1*) $t_0u_1.p_1$*-simps trg-hcomp* τ.*base-in-hom*(*2*) *hcomp-in-vhom*
by *presburger*
qed
qed
have *eq*: *LHS* = *RHS*
proof −
have a^{-1}[t, u, s_0 ⋆ $\varrho\sigma.p_0$] · *LHS* · a[t_1, ω.*chine*, $\varrho\sigma.p_1$] · (ω.*the-ν* ⋆ $\varrho\sigma.p_1$) = Δ
proof −

Here we use $\omega.\Delta$*-naturality* to replace ω.*chine* in favor of ω. We have to bring ω.*the-ν*, τ, and ω.*the-ϑ* together, with $r_0s_1.p_1$ on the right.

have a^{-1}[t, u, s_0 ⋆ $\varrho\sigma.p_0$] · *LHS* · a[t_1, ω.*chine*, $\varrho\sigma.p_1$] · (ω.*the-ν* ⋆ $\varrho\sigma.p_1$) =
a^{-1}[t, u, s_0 ⋆ $\varrho\sigma.p_0$] ·
(t ⋆ a[u, s_0, $\varrho\sigma.p_0$] · ((χ ⋆ s_0) · σ ⋆ $\varrho\sigma.p_0$) · $r_0s_1.\varphi$ ·
(ω.*the-ϑ* ⋆ $\varrho\sigma.p_1$) · a^{-1}[t_0, ω.*chine*, $\varrho\sigma.p_1$]) ·
a[t, t_0, ω.*chine* ⋆ $\varrho\sigma.p_1$] · (τ ⋆ ω.*chine* ⋆ $\varrho\sigma.p_1$) · a[t_1, ω.*chine*, $\varrho\sigma.p_1$] ·

$(\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
unfolding *LHS-def*
using *comp-assoc* **by** *presburger*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$(t \star \omega.\textit{the-}\vartheta \star \varrho\sigma.p_1) \cdot (t \star \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot$
$\mathrm{a}[t,\ t_0,\ \omega.\textit{chine} \star \varrho\sigma.p_1] \cdot ((\tau \star \omega.\textit{chine} \star \varrho\sigma.p_1) \cdot$
$\mathrm{a}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
proof −
have $t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\textit{the-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1] =$
$(t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$(t \star \omega.\textit{the-}\vartheta \star \varrho\sigma.p_1) \cdot (t \star \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1])$
using *whisker-left* τ*.ide-base* ϑ_τ *arrI seqE*
by (*metis* (*full-types*))
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$(t \star \omega.\textit{the-}\vartheta \star \varrho\sigma.p_1) \cdot ((t \star \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot$
$\mathrm{a}[t,\ t_0,\ \omega.\textit{chine} \star \varrho\sigma.p_1] \cdot \mathrm{a}[t \star t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot$
$((\tau \star \omega.\textit{chine}) \star \varrho\sigma.p_1) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
proof −
have $(\tau \star \omega.\textit{chine} \star \varrho\sigma.p_1) \cdot \mathrm{a}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1] =$
$\mathrm{a}[t \star t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1] \cdot ((\tau \star \omega.\textit{chine}) \star \varrho\sigma.p_1)$
using *assoc-naturality*
by (*metis* ω*.w-simps*(*2*−*6*) ϱ*.leg1-simps*(*3*) $\varrho\sigma$*.leg1-simps*(*2*) τ*.tab-simps*(*1*)
τ*.tab-simps*(*2*,*4*−*5*) *hseqE* $r_0 s_1$*.leg1-simps*(*5*) $r_0 s_1$*.leg1-simps*(*6*))
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$((t \star \omega.\textit{the-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}[t,\ t_0 \star \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot$
$(\mathrm{a}[t,\ t_0,\ \omega.\textit{chine}] \star \varrho\sigma.p_1) \cdot ((\tau \star \omega.\textit{chine}) \star \varrho\sigma.p_1) \cdot$
$(\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
proof −
have $(t \star \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot \mathrm{a}[t,\ t_0,\ \omega.\textit{chine} \star \varrho\sigma.p_1] \cdot$
$\mathrm{a}[t \star t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1] =$
$\mathrm{a}[t,\ t_0 \star \omega.\textit{chine},\ \varrho\sigma.p_1] \cdot (\mathrm{a}[t,\ t_0,\ \omega.\textit{chine}] \star \varrho\sigma.p_1)$
proof −
have *seq* $\mathrm{a}[t,\ t_0,\ \omega.\textit{chine} \star \varrho\sigma.p_1]\ \mathrm{a}[t \star t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]$
by (*simp add*: ϱ*.T0.antipar*(*1*))
moreover have *inv* $(t \star \mathrm{a}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) = t \star \mathrm{a}^{-1}[t_0,\ \omega.\textit{chine},\ \varrho\sigma.p_1]$
using ϱ*.T0.antipar*(*1*) **by** *simp*
ultimately show *?thesis*
using *pentagon* ϱ*.T0.antipar*(*1*) *iso-hcomp*
invert-side-of-triangle(*1*)

$[of\ \mathrm{a}[t,\ t_0,\ \omega.chine \star \varrho\sigma.p_1] \cdot \mathrm{a}[t \star t_0,\ \omega.chine,\ \varrho\sigma.p_1]$
$t \star \mathrm{a}[t_0,\ \omega.chine,\ \varrho\sigma.p_1]$
$\mathrm{a}[t,\ t_0 \star \omega.chine,\ \varrho\sigma.p_1] \cdot (\mathrm{a}[t,\ t_0,\ \omega.chine] \star \varrho\sigma.p_1)]$
by *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot (((t \star \omega.the\text{-}\vartheta) \star \varrho\sigma.p_1) \cdot$
$(\mathrm{a}[t,\ t_0,\ \omega.chine] \star \varrho\sigma.p_1) \cdot ((\tau \star \omega.chine) \star \varrho\sigma.p_1)) \cdot$
$(\omega.the\text{-}\nu \star \varrho\sigma.p_1)$
proof −
have $(t \star \omega.the\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}[t,\ t_0 \star \omega.chine,\ \varrho\sigma.p_1] =$
$\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot ((t \star \omega.the\text{-}\vartheta) \star \varrho\sigma.p_1)$
using *assoc-naturality* [*of t* $\omega.the\text{-}\vartheta$ $\varrho\sigma.p_1$] $\omega.\vartheta$*-simps*(*3*) $\varrho\sigma.leg1$*-simps*(*2*) *hseq-char*
by *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot ((\omega \star r_0) \cdot \varrho \star \varrho\sigma.p_1)$
using *whisker-right* $\varrho.T0.antipar(1)$ $\omega.\Delta$*-simps*(*1*) $\omega.\Delta$*-naturality comp-assoc*
by *fastforce*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot ((t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot$
$(t \star (\chi \star s_0) \star \varrho\sigma.p_0)) \cdot (t \star \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot ((\omega \star r_0) \cdot \varrho \star \varrho\sigma.p_1)$
proof −
have $t \star (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0 = (t \star (\chi \star s_0) \star \varrho\sigma.p_0) \cdot (t \star \sigma \star \varrho\sigma.p_0)$
using *whisker-left whisker-right* $\varrho.T0.antipar(1)$
by (*metis* (*full-types*) $\chi.\Delta$*-simps*(*1*) τ*.ide-base* ϑ_τ *arrI* $r_0 s_1$*.ide-u seqE*)
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot$
$(t \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot ((\omega \star r_0) \cdot \varrho \star \varrho\sigma.p_1)$
proof −
have $(t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star (\chi \star s_0) \star \varrho\sigma.p_0) =$
$t \star \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((\chi \star s_0) \star \varrho\sigma.p_0)$
using χ*.in-hom whisker-left* **by** *auto*
also have ... $= t \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]$
using *assoc-naturality* [*of* χ s_0 $\varrho\sigma.p_0$] χ*.in-hom* **by** *auto*
also have ... $= (t \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot (t \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0])$
proof −
have *seq* $(\chi \star s_0 \star \varrho\sigma.p_0)$ $\mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]$
using χ*.in-hom*

apply (*intro seqI hseqI*)
apply *auto*
proof −
show «χ : *src u* → *trg* χ»
by (*metis* χ.Δ-*simps*(*1*) χ.*u-simps*(*3*) *hseqE in-hhom-def seqE*)
show *dom* $(\chi \star s_0 \star \varrho\sigma.p_0) = s \star s_0 \star \varrho\sigma.p_0$
by (*metis* Δ-*simps*(*1*) χ.*in-hom hcomp-simps*(*1*,*3*) *hseq-char in-homE seqE u-simps*(*4*))
qed
thus *?thesis*
using *whisker-left* **by** *simp*
qed
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot$
$(t \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star \sigma \star \varrho\sigma.p_0) \cdot (t \star r_0 s_1.\varphi) \cdot$
$(\mathrm{a}[t,\ r_0,\ \varrho\sigma.p_1] \cdot ((\omega \star r_0) \star \varrho\sigma.p_1)) \cdot (\varrho \star \varrho\sigma.p_1)$
using *whisker-right comp-assoc* **by** *simp*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (t \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot$
$(t \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star \sigma \star \varrho\sigma.p_0) \cdot ((t \star r_0 s_1.\varphi) \cdot$
$(\omega \star r_0 \star \varrho\sigma.p_1)) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
using *assoc-naturality* [*of* ω r_0 $\varrho\sigma.p_1$] ω.*in-hom* ϱ.*T0*.*antipar*(*1*) *comp-assoc*
by *fastforce*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot ((t \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot$
$(t \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0]) \cdot (t \star \sigma \star \varrho\sigma.p_0)) \cdot (\omega \star s_1 \star \varrho\sigma.p_0) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
proof −
have $(t \star r_0 s_1.\varphi) \cdot (\omega \star r_0 \star \varrho\sigma.p_1) = \omega \star r_0 s_1.\varphi$
using *comp-cod-arr comp-arr-dom* ω.*in-hom interchange comp-ide-arr*
by (*metis* τ.*base-in-hom*(*2*) τ.*ide-base* $r_0 s_1.\varphi$-*simps*(*1*) $r_0 s_1.\varphi$-*simps*(*4*) *seqI'*)
also have ... $= (\omega \star s_1 \star \varrho\sigma.p_0) \cdot (r \star r_0 s_1.\varphi)$
using $r_0 s_1.\varphi$-*in-hom comp-cod-arr comp-arr-dom* ω.*in-hom interchange*
by (*metis in-homE*)
finally have $(t \star r_0 s_1.\varphi) \cdot (\omega \star r_0 \star \varrho\sigma.p_1) = (\omega \star s_1 \star \varrho\sigma.p_0) \cdot (r \star r_0 s_1.\varphi)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$
$((t \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)) \cdot$
$(\omega \star s_1 \star \varrho\sigma.p_0)) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
using *whisker-left* ϱ.*T0*.*antipar*(*1*) $\varrho\sigma$.*composable* χ.*in-hom comp-assoc* **by** *auto*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$
$(\omega \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
proof −
have $(t \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)) \cdot (\omega \star s_1 \star \varrho\sigma.p_0) =$

$\omega \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)$
proof −
have «$(\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0) : s_1 \star \varrho\sigma.p_0 \Rightarrow u \star s_0 \star \varrho\sigma.p_0$»
using *ω.in-hom χ.in-hom* **by** *force*
thus *?thesis*
by (*metis* (*no-types*) *ω.in-hom comp-arr-dom comp-cod-arr in-homE interchange*)
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$
$(\omega \star \chi \star s_0 \star \varrho\sigma.p_0)) \cdot (r \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
proof −
have $\omega \star (\chi \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0) =$
$(\omega \star \chi \star s_0 \star \varrho\sigma.p_0) \cdot (r \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0))$
proof −
have *seq* $(\chi \star s_0 \star \varrho\sigma.p_0)$ $(\mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0))$
using *χ.in-hom* **by** *force*
thus *?thesis*
using *comp-arr-dom comp-cod-arr ω.in-hom χ.in-hom interchange*
by (*metis in-homE*)
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((\omega \star \chi) \star s_0 \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}^{-1}[r,\ s,\ s_0 \star \varrho\sigma.p_0] \cdot (r \star \mathrm{a}[s,\ s_0,\ \varrho\sigma.p_0] \cdot (\sigma \star \varrho\sigma.p_0)) \cdot$
$(r \star r_0 s_1.\varphi) \cdot \mathrm{a}[r,\ r_0,\ \varrho\sigma.p_1] \cdot (\varrho \star \varrho\sigma.p_1)$
proof −
have $\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot (\omega \star \chi \star s_0 \star \varrho\sigma.p_0) =$
$((\omega \star \chi) \star s_0 \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[r,\ s,\ s_0 \star \varrho\sigma.p_0]$
using *assoc-naturality ω.in-hom χ.in-hom*
by (*metis ρσ.leg0-simps*(*3*) *assoc′-naturality hcomp-in-vhomE in-hom in-homE u-simps*(*2*) *u-simps*(*4*) *u-simps*(*5*))
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \Delta$
using *whisker-left ρσ.tab-def comp-assoc* **by** *simp*
finally show *?thesis* **by** *auto*
qed
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot RHS \cdot \mathrm{a}[t_1,\ \omega.chine,\ \varrho\sigma.p_1] \cdot (\omega.the\text{-}\nu \star \varrho\sigma.p_1)$
proof −

Now cancel *ω.the-ν* and its inverse.

have $\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot RHS \cdot \mathrm{a}[t_1,\ \omega.chine,\ \varrho\sigma.p_1] \cdot (\omega.the\text{-}\nu \star \varrho\sigma.p_1) =$
$\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$

$(t \star (u \star \textit{the-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \textit{chine}] \cdot$
$(\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot ((\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot$
$(t_0 u_1.\varphi \star \textit{chine}) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}]) \cdot$
$\mathrm{a}[t,\ t_0,\ \tau\mu.p_1 \star \textit{chine}] \cdot (\tau \star \tau\mu.p_1 \star \textit{chine}) \cdot$
$\mathrm{a}[t_1,\ \tau\mu.p_1,\ \textit{chine}] \cdot \textit{the-}\nu \cdot (\textit{inv}\ \omega.\textit{the-}\nu \star \varrho\sigma.p_1) \cdot$
$((\mathrm{a}^{-1}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot \mathrm{a}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
unfolding *RHS-def*
using *comp-assoc* **by** *presburger*
also have ... $= \mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$
$(t \star (u \star \textit{the-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \textit{chine}] \cdot$
$(\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot ((\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot$
$(t_0 u_1.\varphi \star \textit{chine}) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}]) \cdot$
$\mathrm{a}[t,\ t_0,\ \tau\mu.p_1 \star \textit{chine}] \cdot (\tau \star \tau\mu.p_1 \star \textit{chine}) \cdot$
$\mathrm{a}[t_1,\ \tau\mu.p_1,\ \textit{chine}] \cdot \textit{the-}\nu$
proof −
have $\textit{the-}\nu \cdot (\textit{inv}\ \omega.\textit{the-}\nu \star \varrho\sigma.p_1) \cdot$
$((\mathrm{a}^{-1}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot \mathrm{a}[t_1,\ \omega.\textit{chine},\ \varrho\sigma.p_1]) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1) =$
$\textit{the-}\nu \cdot (\textit{inv}\ \omega.\textit{the-}\nu \star \varrho\sigma.p_1) \cdot$
$((t_1 \star \omega.\textit{chine}) \star \varrho\sigma.p_1) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
using *comp-inv-arr* $\varrho.T0.\textit{antipar}(1)$ *comp-assoc-assoc′* **by** *simp*
also have ... $= \textit{the-}\nu \cdot (\textit{inv}\ \omega.\textit{the-}\nu \star \varrho\sigma.p_1) \cdot (\omega.\textit{the-}\nu \star \varrho\sigma.p_1)$
using *comp-cod-arr* $\varrho.T0.\textit{antipar}(1)$ **by** *simp*
also have ... $= \textit{the-}\nu \cdot (r_1 \star \varrho\sigma.p_1)$
using *whisker-right* [*of* $\varrho\sigma.p_1$] $r_0 s_1.\textit{ide-leg1}$ $\omega.\textit{the-}\nu\textit{-props}(2)$ $\omega.\textit{the-}\nu\textit{-simps}(4)$
$\varrho.\textit{leg1-simps}(2)$ *comp-inv-arr′*
by *metis*
also have ... $= \textit{the-}\nu$
using *comp-arr-dom* **by** *simp*
finally show *?thesis*
using *comp-assoc* **by** *simp*
qed

Now reassociate to move *the-*ϑ to the left and get other terms composed with *chine*, where they can be reduced to $\tau\mu.\textit{tab}$.

also have ... $= (\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot$
$(t \star u \star \textit{the-}\vartheta)) \cdot (t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \textit{chine}]) \cdot$
$(t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot (t \star (\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot$
$(t \star t_0 u_1.\varphi \star \textit{chine}) \cdot (t \star \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}]) \cdot$
$\mathrm{a}[t,\ t_0,\ \tau\mu.p_1 \star \textit{chine}] \cdot (\tau \star \tau\mu.p_1 \star \textit{chine}) \cdot$
$\mathrm{a}[t_1,\ \tau\mu.p_1,\ \textit{chine}] \cdot \textit{the-}\nu$
proof −
have $\textit{arr}\ ((u \star \textit{the-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \textit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot (t_0 u_1.\varphi \star \textit{chine}) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}])$
using $\vartheta_\tau{}'$ **by** *blast*
moreover have $\textit{arr}\ (\mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \textit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot (t_0 u_1.\varphi \star \textit{chine}) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}])$
using *calculation* **by** *blast*
moreover have $\textit{arr}\ ((\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \textit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \textit{chine}) \cdot (t_0 u_1.\varphi \star \textit{chine}) \cdot \mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \textit{chine}])$

using *calculation* **by** *blast*
moreover have *arr* $(((\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot (t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}])$
using *calculation* **by** *blast*
moreover have *arr* $((t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}])$
using *calculation* **by** *blast*
ultimately
have $t \star (u \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}[u, u_0 \star \tau\mu.p_0, \mathit{chine}] \cdot (\mathrm{a}[u, u_0, \tau\mu.p_0] \star \mathit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot (t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}] =$
$(t \star u \star \mathit{the}\text{-}\vartheta) \cdot (t \star \mathrm{a}[u, u_0 \star \tau\mu.p_0, \mathit{chine}]) \cdot$
$(t \star \mathrm{a}[u, u_0, \tau\mu.p_0] \star \mathit{chine}) \cdot (t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot$
$(t \star t_0u_1.\varphi \star \mathit{chine}) \cdot (t \star \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}])$
using *whisker-left* ϱ*.T0.antipar*(*1*) τ*.ide-base* **by** *presburger*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t, u, (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u, u_0 \star \tau\mu.p_0, \mathit{chine}]) \cdot (t \star \mathrm{a}[u, u_0, \tau\mu.p_0] \star \mathit{chine}) \cdot$
$(t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot$
$(t \star t_0u_1.\varphi \star \mathit{chine}) \cdot$
$(t \star \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}]) \cdot \mathrm{a}[t, t_0, \tau\mu.p_1 \star \mathit{chine}] \cdot$
$((\tau \star \tau\mu.p_1 \star \mathit{chine}) \cdot \mathrm{a}[t_1, \tau\mu.p_1, \mathit{chine}]) \cdot \mathit{the}\text{-}\nu$
using *assoc′-naturality* [*of t u the-*ϑ] $\tau\mu$*.composable* ϑ*-simps*(*3*) *comp-assoc* **by** *auto*
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t, u, (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u, u_0 \star \tau\mu.p_0, \mathit{chine}]) \cdot (t \star \mathrm{a}[u, u_0, \tau\mu.p_0] \star \mathit{chine}) \cdot$
$(t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot$
$(t \star t_0u_1.\varphi \star \mathit{chine}) \cdot$
$((t \star \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}]) \cdot \mathrm{a}[t, t_0, \tau\mu.p_1 \star \mathit{chine}] \cdot$
$\mathrm{a}[t \star t_0, \tau\mu.p_1, \mathit{chine}]) \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine}) \cdot \mathit{the}\text{-}\nu$
proof −
have $(\tau \star \tau\mu.p_1 \star \mathit{chine}) \cdot \mathrm{a}[t_1, \tau\mu.p_1, \mathit{chine}] =$
$\mathrm{a}[t \star t_0, \tau\mu.p_1, \mathit{chine}] \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine})$
using *assoc-naturality*
by (*metis* τ*.leg1-simps*(*3*) τ*.tab-simps*(*1*,*2*,*4*) τ*.tab-simps*(*5*) $\tau\mu$*.leg0-simps*(*2*)
$\tau\mu$*.leg1-simps*(*2*) *hseqE src-hcomp* t_0u_1*.leg1-simps*(*3*,*5*−*6*) *w-simps*(*2*)
w-simps(*4*−*6*))
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t, u, (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u, u_0 \star \tau\mu.p_0, \mathit{chine}]) \cdot (t \star \mathrm{a}[u, u_0, \tau\mu.p_0] \star \mathit{chine}) \cdot$
$(t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot$
$((t \star t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}[t, t_0 \star \tau\mu.p_1, \mathit{chine}]) \cdot$
$(\mathrm{a}[t, t_0, \tau\mu.p_1] \star \mathit{chine}) \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine}) \cdot \mathit{the}\text{-}\nu$
proof −
have $(t \star \mathrm{a}^{-1}[t_0, \tau\mu.p_1, \mathit{chine}]) \cdot \mathrm{a}[t, t_0, \tau\mu.p_1 \star \mathit{chine}] \cdot$
$\mathrm{a}[t \star t_0, \tau\mu.p_1, \mathit{chine}] =$
$\mathrm{a}[t, t_0 \star \tau\mu.p_1, \mathit{chine}] \cdot (\mathrm{a}[t, t_0, \tau\mu.p_1] \star \mathit{chine})$
using *pentagon* $t_0u_1.p_1$*-simps uw*ϑ τ*.T0.antipar*(*1*) *iso-hcomp*
comp-assoc-assoc′

invert-side-of-triangle(*1*)
[*of* $\mathrm{a}[t,\ t_0,\ \tau\mu.p_1 \star \mathit{chine}] \cdot \mathrm{a}[t \star t_0,\ \tau\mu.p_1,\ \mathit{chine}]$
$t \star \mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]$
$\mathrm{a}[t,\ t_0 \star \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star \mathit{chine})$]
by *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$((t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot \mathrm{a}[t,\ u_1 \star \tau\mu.p_0,\ \mathit{chine}]) \cdot$
$((t \star t_0u_1.\varphi) \star \mathit{chine}) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star \mathit{chine}) \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine}) \cdot \mathit{the}\text{-}\nu$
proof −
have $(t \star t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}[t,\ t_0 \star \tau\mu.p_1,\ \mathit{chine}] =$
$\mathrm{a}[t,\ u_1 \star \tau\mu.p_0,\ \mathit{chine}] \cdot ((t \star t_0u_1.\varphi) \star \mathit{chine})$
using *assoc-naturality* [*of* $t\ t_0u_1.\varphi\ \mathit{chine}$] $t_0u_1.\mathit{cospan}$ **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ \mathit{chine}] \cdot$
$((t \star \mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot ((t \star t_0u_1.\varphi) \star \mathit{chine}) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star \mathit{chine}) \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine}) \cdot \mathit{the}\text{-}\nu$
proof −
have $(t \star (\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot \mathrm{a}[t,\ u_1 \star \tau\mu.p_0,\ \mathit{chine}] =$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ \mathit{chine}] \cdot ((t \star (\mu \star \tau\mu.p_0)) \star \mathit{chine})$
using *assoc-naturality* [*of* $t\ \mu \star \tau\mu.p_0\ \mathit{chine}$]
by (*simp add*: $\tau\mu.\mathit{composable}$)
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ \mathit{chine}] \cdot$
$((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine}) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine}) \cdot$
$((t \star \mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot ((t \star t_0u_1.\varphi) \star \mathit{chine}) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star \mathit{chine}) \cdot ((\tau \star \tau\mu.p_1) \star \mathit{chine}) \cdot \mathit{the}\text{-}\nu$
proof −
have $(((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine}) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine})) \cdot$
$((t \star \mu \star \tau\mu.p_0) \star \mathit{chine}) =$
$((t \star ((u \star u_0) \star \tau\mu.p_0)) \star \mathit{chine}) \cdot ((t \star \mu \star \tau\mu.p_0) \star \mathit{chine})$
using *whisker-right whisker-left* [*of* $t\ \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]\ \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]$]
$\tau\mu.\mathit{composable}$ *comp-assoc-assoc′*
by *simp*
also have ... $= (t \star \mu \star \tau\mu.p_0) \star \mathit{chine}$
using *comp-cod-arr* $\tau\mu.\mathit{composable}$ **by** *simp*
finally have $(((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine}) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine})) \cdot$

$((t \star \mu \star \tau\mu.p_0) \star chine) =$
$(t \star \mu \star \tau\mu.p_0) \star chine$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *metis*
qed
also have ... $= ((t \star u) \star the\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star chine] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ chine]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star chine) \cdot$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ chine] \cdot$
$((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star chine) \cdot (((\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine) \cdot$
$(\mathrm{a}^{-1}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine)) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star chine)) \cdot$
$((t \star \mu \star \tau\mu.p_0) \star chine) \cdot ((t \star t_0u_1.\varphi) \star chine) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star chine) \cdot ((\tau \star \tau\mu.p_1) \star chine) \cdot the\text{-}\nu$
proof −
have $((\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine) \cdot (\mathrm{a}^{-1}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine)) \cdot$
$((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star chine) =$
$((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star chine)$
using *comp-inv-arr′ comp-cod-arr* $\tau\mu$*.composable comp-assoc-assoc′*
whisker-right [*of chine* $\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0]$ $\mathrm{a}^{-1}[t,\ u,\ u_0 \star \tau\mu.p_0]$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= ((t \star u) \star the\text{-}\vartheta) \cdot \mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star chine] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ chine]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star chine) \cdot$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ chine] \cdot$
$((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star chine) \cdot (\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine) \cdot$
$((\mathrm{a}^{-1}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star chine) \cdot$
$((t \star \mu \star \tau\mu.p_0) \star chine) \cdot ((t \star t_0u_1.\varphi) \star chine) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star chine) \cdot ((\tau \star \tau\mu.p_1) \star chine)) \cdot the\text{-}\nu$
using *comp-assoc* **by** *presburger*
also have ... $= ((t \star u) \star the\text{-}\vartheta) \cdot$
$(\mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star chine] \cdot$
$(t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ chine]) \cdot (t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star chine) \cdot$
$\mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ chine] \cdot$
$((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star chine) \cdot (\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine)) \cdot$
$(\tau\mu.tab \star chine) \cdot the\text{-}\nu$
proof −
have $(\mathrm{a}^{-1}[t,\ u,\ u_0 \star \tau\mu.p_0] \star chine) \cdot ((t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0]) \star chine) \cdot$
$((t \star \mu \star \tau\mu.p_0) \star chine) \cdot ((t \star t_0u_1.\varphi) \star chine) \cdot$
$(\mathrm{a}[t,\ t_0,\ \tau\mu.p_1] \star chine) \cdot ((\tau \star \tau\mu.p_1) \star chine) =$
$\tau\mu.tab \star chine$
using *uw*ϑ *whisker-right* [*of chine*]
by (*metis* $\tau\mu$*.tab-def* $\tau\mu$*.tab-in-vhom′ arrI seqE*)
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((t \star u) \star the\text{-}\vartheta) \cdot \mathrm{a}[t \star u,\ u_0 \star \tau\mu.p_0,\ chine] \cdot (\tau\mu.tab \star chine) \cdot the\text{-}\nu$
proof −

have $\mathrm{a}^{-1}[t,\ u,\ (u_0 \star \tau\mu.p_0) \star \mathit{chine}] \cdot (t \star \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}]) \cdot$
$(t \star \mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot \mathrm{a}[t,\ (u \star u_0) \star \tau\mu.p_0,\ \mathit{chine}] \cdot$
$((t \star \mathrm{a}^{-1}[u,\ u_0,\ \tau\mu.p_0]) \star \mathit{chine}) \cdot (\mathrm{a}[t,\ u,\ u_0 \star \tau\mu.p_0] \star \mathit{chine}) =$
$\{\!|\mathbf{a}^{-1}[\langle t\rangle,\ \langle u\rangle,\ (\langle u_0\rangle \star \langle\tau\mu.p_0\rangle) \star \langle \mathit{chine}\rangle] \cdot (\langle t\rangle \star \mathbf{a}[\langle u\rangle,\ \langle u_0\rangle \star \langle\tau\mu.p_0\rangle,\ \langle \mathit{chine}\rangle])$
.
$(\langle t\rangle \star \mathbf{a}[\langle u\rangle,\ \langle u_0\rangle,\ \langle\tau\mu.p_0\rangle] \star \langle \mathit{chine}\rangle) \cdot \mathbf{a}[\langle t\rangle,\ (\langle u\rangle \star \langle u_0\rangle) \star \langle\tau\mu.p_0\rangle,\ \langle \mathit{chine}\rangle] \cdot$
$((\langle t\rangle \star \mathbf{a}^{-1}[\langle u\rangle,\ \langle u_0\rangle,\ \langle\tau\mu.p_0\rangle]) \star \langle \mathit{chine}\rangle) \cdot$
$(\mathbf{a}[\langle t\rangle,\ \langle u\rangle,\ \langle u_0\rangle \star \langle\tau\mu.p_0\rangle] \star \langle \mathit{chine}\rangle)|\!\}$
using a'*-def* α*-def* $\tau\mu$*.composable* **by** *simp*
also have ... $= \{\!|\mathbf{a}[\langle t\rangle \star \langle u\rangle,\ \langle u_0\rangle \star \langle\tau\mu.p_0\rangle,\ \langle \mathit{chine}\rangle]|\!\}$
using $\tau\mu$*.composable*
apply (*intro E.eval-eqI*) **by** *simp-all*
also have ... $= \mathrm{a}[t \star u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}]$
using a'*-def* α*-def* $\tau\mu$*.composable* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... $= \Delta$
using Δ*-naturality* **by** *simp*
finally show *?thesis* **by** *simp*
qed
finally have $\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot \mathit{LHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (\omega.\mathit{the}\text{-}\nu \star \varrho\sigma.p_1) =$
$\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0] \cdot \mathit{RHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (\omega.\mathit{the}\text{-}\nu \star \varrho\sigma.p_1)$
by *blast*

hence $(\mathit{LHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]) \cdot (\omega.\mathit{the}\text{-}\nu \star \varrho\sigma.p_1) =$
$(\mathit{RHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]) \cdot (\omega.\mathit{the}\text{-}\nu \star \varrho\sigma.p_1)$
using u_τ r_0s_1*.ide-u LHS RHS iso-is-section* [*of* $\mathrm{a}^{-1}[t,\ u,\ s_0 \star \varrho\sigma.p_0]$] *section-is-mono mono-cancel* $\tau\mu$*.composable comp-assoc*
by (*metis* (*no-types*, *lifting*) Δ*-simps*(*1*) μ*.ide-base*
‹$\mathrm{a}^{-1}[t,\ u,\ s_0 \star r_0s_1.p_0] \cdot \mathit{LHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ r_0s_1.p_1] \cdot (\omega.\mathit{the}\text{-}\nu \star r_0s_1.p_1) =$
$((\omega \star \chi) \star s_0 \star r_0s_1.p_0) \cdot \varrho\sigma.\mathit{tab}$›
τ*.ide-base hseq-char ideD*(*1*) *ide-u iso-assoc*$'$)
hence *1*: $\mathit{LHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] = \mathit{RHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]$
using *epi-cancel LHS RHS iso-is-retraction retraction-is-epi* $\tau\mu$*.composable* ω*.the*-ν*-props iso-hcomp*
by (*metis* Δ*-simps*(*1*) ω*.the*-ν*-simps*(*2*)
‹$((\omega \star \chi) \star s_0 \star r_0s_1.p_0) \cdot \varrho\sigma.\mathit{tab} =$
$\mathrm{a}^{-1}[t,\ u,\ s_0 \star r_0s_1.p_0] \cdot \mathit{RHS} \cdot \mathrm{a}[t_1,\ \omega.\mathit{chine},\ r_0s_1.p_1] \cdot (\omega.\mathit{the}\text{-}\nu \star r_0s_1.p_1)$›
ϱ*.leg1-simps*(*3*) *ide-is-iso local.comp-assoc* r_0s_1*.ide-leg1* $r_0s_1.p_1$*-simps seqE*)
show $\mathit{LHS} = \mathit{RHS}$
proof −
have *epi* $\mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]$
using *iso-is-retraction retraction-is-epi* ϱ*.T0.antipar*(*1*) **by** *simp*
moreover have *seq LHS* $\mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]$
using *LHS* ϱ*.T0.antipar*(*1*) **by** *auto*
moreover have *seq RHS* $\mathrm{a}[t_1,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]$
using *RHS* ϱ*.T0.antipar*(*1*) **by** *auto*
ultimately show *?thesis*
using *epi-cancel 1* **by** *blast*

qed
qed
have *1*: $\exists!\gamma.$ «γ : *?*w_τ $\Rightarrow$ *?*w_τ'» $\wedge$ *?*$\beta_\tau = t_1 \star \gamma \wedge$ *?*$\vartheta_\tau =$ *?*$\vartheta_\tau' \cdot (t_0 \star \gamma)$
using *LHS-def RHS-def* u_τ w_τ w_τ' β_τ ϑ_τ ϑ_τ' *eq* τ*.T2* [*of* *?*w_τ *?*w_τ' *?*ϑ_τ *?*u_τ *?*ϑ_τ' *?*β_τ]
by *fastforce*
obtain γ_τ **where** γ_τ: «γ_τ : *?*w_τ $\Rightarrow$ *?*w_τ'» $\wedge$ *?*$\beta_\tau = t_1 \star \gamma_\tau \wedge$ *?*$\vartheta_\tau =$ *?*$\vartheta_\tau' \cdot (t_0 \star \gamma_\tau)$
using *1* **by** *auto*

At this point we could show that γ_τ is invertible using *BS3*, but we want to avoid using *BS3* if possible and we also want to establish a characterization of *inv* γ_τ. So we show the invertibility of γ_τ directly, using a few more applications of *T2*.

have *iso-*β_τ: *iso ?*β_τ
using *uw*ϑ β_τ *the-*ν*-props* ω*.the-*ν*-props hseqI' iso-assoc'* ω*.hseq-leg*$_0$
apply (*intro isos-compose*)
apply (*metis* ω*.is-ide* $\varrho\sigma$*.leg1-simps*(*2*) τ*.ide-leg1* τ*.leg1-simps*(*2*)
τ*.leg1-simps*(*3*) *hseqE* r_0s_1*.ide-leg1 hcomp-simps*(*1*) *vconn-implies-hpar*(*3*))
apply (*metis* $\varrho\sigma$*.leg1-simps*(*2*) *hseqE ide-is-iso* r_0s_1*.ide-leg1 src-inv iso-inv-iso*
iso-hcomp vconn-implies-hpar(*1*))
apply *blast*
apply *blast*
apply *blast*
apply (*metis* τ*.ide-leg1* τ*.leg1-simps*(*3*) *hseqE ide-char iso-assoc* t_0u_1*.ide-leg1*
t_0u_1*.*p_1*-simps* w_τ')
by *blast*
hence *eq'*: $((t \star$ *?*$\vartheta_\tau') \cdot$ a$[t,\ t_0,$ *?*$w_\tau'] \cdot (\tau \star$ *?*$w_\tau')) =$
$((t \star$ *?*$\vartheta_\tau) \cdot$ a$[t,\ t_0,$ *?*$w_\tau] \cdot (\tau \star$ *?*$w_\tau)) \cdot$ *inv ?*β_τ
proof −
have *seq* $((t \star$ *?*$\vartheta_\tau') \cdot$ a$[t,\ t_0,$ *?*$w_\tau'] \cdot (\tau \star$ *?*$w_\tau'))$ *?*β_τ
using *LHS RHS-def eq* **by** *blast*
hence $(t \star$ *?*$\vartheta_\tau') \cdot$ a$[t,\ t_0,$ *?*$w_\tau'] \cdot (\tau \star$ *?*$w_\tau') =$
$(((t \star$ *?*$\vartheta_\tau') \cdot$ a$[t,\ t_0,$ *?*$w_\tau'] \cdot (\tau \star$ *?*$w_\tau')) \cdot$ *?*$\beta_\tau) \cdot$ *inv ?*β_τ
by (*meson invert-side-of-triangle*(*2*) *iso-*β_τ)
thus *?thesis*
using *LHS-def RHS-def eq* **by** *argo*
qed
have *2*: $\exists!\gamma.$ «γ : *?*w_τ' $\Rightarrow$ *?*w_τ» $\wedge$ *inv ?*$\beta_\tau = t_1 \star \gamma \wedge$ *?*$\vartheta_\tau' =$ *?*$\vartheta_\tau \cdot (t_0 \star \gamma)$
using u_τ w_τ w_τ' β_τ ϑ_τ ϑ_τ' *eq'* τ*.T2* [*of* *?*w_τ' *?*w_τ *?*ϑ_τ'*?*u_τ *?*ϑ_τ *inv ?*β_τ] *iso-*β_τ *comp-assoc*
by *blast*
obtain γ_τ' **where**
γ_τ': «γ_τ' : *?*w_τ' $\Rightarrow$ *?*w_τ» $\wedge$ *inv ?*$\beta_\tau = t_1 \star \gamma_\tau' \wedge$ *?*$\vartheta_\tau' =$ *?*$\vartheta_\tau \cdot (t_0 \star \gamma_\tau')$
using *2* **by** *auto*
have *inverse-arrows* γ_τ γ_τ'
proof
have «$\gamma_\tau' \cdot \gamma_\tau$: *?*w_τ $\Rightarrow$ *?*w_τ»
using γ_τ γ_τ' **by** *auto*
moreover have $t_1 \star \gamma_\tau' \cdot \gamma_\tau = t_1 \star$ *?*w_τ
using γ_τ γ_τ' *whisker-left* β_τ *iso-*β_τ *comp-inv-arr'*
by (*metis* (*no-types*, *lifting*) τ*.ide-leg1 calculation in-homE*)
moreover have *?*$\vartheta_\tau =$ *?*$\vartheta_\tau \cdot (t_0 \star \gamma_\tau' \cdot \gamma_\tau)$

proof −
  have $?\vartheta_\tau = ?\vartheta_\tau{}' \cdot (t_0 \star \gamma_\tau)$
    using γ_τ **by** *simp*
  also have ... $= ?\vartheta_\tau \cdot (t_0 \star \gamma_\tau{}') \cdot (t_0 \star \gamma_\tau)$
    using $\gamma_\tau{}'$ *comp-assoc* **by** *simp*
  also have ... $= ?\vartheta_\tau \cdot (t_0 \star \gamma_\tau{}' \cdot \gamma_\tau)$
    using γ_τ $\gamma_\tau{}'$ *whisker-left*
    by (*metis* (*full-types*) τ*.ide-leg0 seqI'*)
  finally show *?thesis* **by** *simp*
qed
moreover have
  $\bigwedge\gamma.$ «$\gamma : ?w_\tau \Rightarrow ?w_\tau$» $\wedge\ t_1 \star \gamma = t_1 \star ?w_\tau \wedge ?\vartheta_\tau = ?\vartheta_\tau \cdot (t_0 \star \gamma) \Longrightarrow \gamma = ?w_\tau$
proof −
  have $\exists!\gamma.$ «$\gamma : ?w_\tau \Rightarrow ?w_\tau$» $\wedge\ t_1 \star ?w_\tau = t_1 \star \gamma \wedge ?\vartheta_\tau = ?\vartheta_\tau \cdot (t_0 \star \gamma)$
  proof −
    have $(t \star ?\vartheta_\tau) \cdot \mathrm{a}[t,\ t_0,\ ?w_\tau] \cdot (\tau \star ?w_\tau) =$
      $((t \star ?\vartheta_\tau) \cdot \mathrm{a}[t,\ t_0,\ ?w_\tau] \cdot (\tau \star ?w_\tau)) \cdot (t_1 \star ?w_\tau)$
      by (*metis LHS LHS-def comp-arr-dom in-homE*)
    thus *?thesis*
      using w_τ ϑ_τ *ω.w-simps*(*4*) τ*.leg1-in-hom*(*2*) τ*.leg1-simps*(*3*) *hcomp-in-vhom ideD*(*1*) *trg-hcomp ide-in-hom*(*2*) τ*.T2*
      by *presburger*
  qed
  thus $\bigwedge\gamma.$ «$\gamma : ?w_\tau \Rightarrow ?w_\tau$» $\wedge\ t_1 \star \gamma = t_1 \star ?w_\tau \wedge ?\vartheta_\tau = ?\vartheta_\tau \cdot (t_0 \star \gamma) \Longrightarrow \gamma = ?w_\tau$
    by (*metis* ϑ_τ *comp-arr-dom ide-in-hom*(*2*) *in-homE* w_τ)
qed
ultimately have $\gamma_\tau{}' \cdot \gamma_\tau = ?w_\tau$
  by *simp*
thus *ide* $(\gamma_\tau{}' \cdot \gamma_\tau)$
  using w_τ **by** *simp*
have «$\gamma_\tau \cdot \gamma_\tau{}' : ?w_\tau{}' \Rightarrow ?w_\tau{}'$»
  using γ_τ $\gamma_\tau{}'$ **by** *auto*
moreover have $t_1 \star \gamma_\tau \cdot \gamma_\tau{}' = t_1 \star ?w_\tau{}'$
  by (*metis* β_τ γ_τ $\gamma_\tau{}'$ τ*.ide-leg1 calculation comp-arr-inv' in-homE iso-*β_τ *whisker-left*)
moreover have $?\vartheta_\tau{}' = ?\vartheta_\tau{}' \cdot (t_0 \star \gamma_\tau \cdot \gamma_\tau{}')$
proof −
  have $?\vartheta_\tau{}' = ?\vartheta_\tau \cdot (t_0 \star \gamma_\tau{}')$
    using $\gamma_\tau{}'$ **by** *simp*
  also have ... $= ?\vartheta_\tau{}' \cdot (t_0 \star \gamma_\tau) \cdot (t_0 \star \gamma_\tau{}')$
    using γ_τ *comp-assoc* **by** *simp*
  also have ... $= ?\vartheta_\tau{}' \cdot (t_0 \star \gamma_\tau \cdot \gamma_\tau{}')$
    using γ_τ $\gamma_\tau{}'$ *whisker-left* τ*.ide-leg0* **by** *fastforce*
  finally show *?thesis* **by** *simp*
qed
moreover have $\bigwedge\gamma.$ «$\gamma : ?w_\tau{}' \Rightarrow ?w_\tau{}'$» $\wedge\ t_1 \star \gamma = t_1 \star ?w_\tau{}' \wedge ?\vartheta_\tau{}' = ?\vartheta_\tau{}' \cdot (t_0 \star \gamma)$
    $\Longrightarrow \gamma = ?w_\tau{}'$
proof −
  have $\exists!\gamma.$ «$\gamma : ?w_\tau{}' \Rightarrow ?w_\tau{}'$» $\wedge\ t_1 \star ?w_\tau{}' = t_1 \star \gamma \wedge ?\vartheta_\tau{}' = ?\vartheta_\tau{}' \cdot (t_0 \star \gamma)$
  proof −

have $(t \star ?\vartheta_\tau{}') \cdot \mathrm{a}[t,\ t_0,\ ?w_\tau{}'] \cdot (\tau \star ?w_\tau{}') =$
$((t \star ?\vartheta_\tau{}') \cdot \mathrm{a}[t,\ t_0,\ ?w_\tau{}'] \cdot (\tau \star ?w_\tau{}')) \cdot (t_1 \star ?w_\tau{}')$
proof −
have *1*: $t_1 \star \gamma_\tau \cdot \gamma_\tau{}' = (t_1 \star \gamma_\tau) \cdot (t_1 \star \gamma_\tau{}')$
by (*meson* γ_τ $\gamma_\tau{}'$ τ*.ide-leg1 seqI′ whisker-left*)
have $((LHS \cdot inv\ ?\beta_\tau) \cdot (t_1 \star \gamma_\tau)) \cdot (t_1 \star \gamma_\tau{}') = LHS \cdot inv\ ?\beta_\tau$
using *LHS-def RHS-def* γ_τ $\gamma_\tau{}'$ *eq eq′* **by** *argo*
thus *?thesis*
unfolding *LHS-def*
using *1* **by** (*simp add*: *calculation*(*2*) *eq′ comp-assoc*)
qed
thus *?thesis*
using $w_\tau{}'$ $\vartheta_\tau{}'$ ω*.w-simps*(*4*) τ*.leg1-in-hom*(*2*) τ*.leg1-simps*(*3*) *hcomp-in-vhom ideD*(*1*) *trg-hcomp ide-in-hom*(*2*) τ*.T2* τ*.T0.antipar*(*1*) t_0u_1*.base-simps*(*2*) t_0u_1*.leg1-simps*(*4*)
by *presburger*
qed
thus $\bigwedge\gamma.$ «γ : $?w_\tau{}' \Rightarrow ?w_\tau{}'$» $\wedge$ $t_1 \star \gamma = t_1 \star ?w_\tau{}' \wedge ?\vartheta_\tau{}' = ?\vartheta_\tau{}' \cdot (t_0 \star \gamma)$
$\Longrightarrow \gamma = ?w_\tau{}'$
by (*metis* $\vartheta_\tau{}'$ *comp-arr-dom ide-in-hom*(*2*) *in-homE* $w_\tau{}'$)
qed
ultimately have $\gamma_\tau \cdot \gamma_\tau{}' = ?w_\tau{}'$
by *simp*
thus *ide* $(\gamma_\tau \cdot \gamma_\tau{}')$
using $w_\tau{}'$ **by** *simp*
qed
thus $\tau\mu.p_1 \star chine \cong \omega.chine \star \varrho\sigma.p_1$
using w_τ $w_\tau{}'$ γ_τ *isomorphic-symmetric isomorphic-def* **by** *blast*
have *iso-*γ_τ: *iso* γ_τ
using ‹*inverse-arrows* γ_τ $\gamma_\tau{}'$› **by** *auto*
have $\gamma_\tau{}'$*-eq*: $\gamma_\tau{}' = inv\ \gamma_\tau$
using ‹*inverse-arrows* γ_τ $\gamma_\tau{}'$› *inverse-unique* **by** *blast*

let $?w_\mu = \tau\mu.p_0 \star chine$
let $?w_\mu{}' = \chi.chine \star \varrho\sigma.p_0$
let $?u_\mu = s_0 \star \varrho\sigma.p_0$
let $?\vartheta_\mu = the\text{-}\vartheta \cdot \mathrm{a}^{-1}[u_0,\ \tau\mu.p_0,\ chine]$
let $?\vartheta_\mu{}' = (\chi.the\text{-}\vartheta \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[u_0,\ \chi.chine,\ \varrho\sigma.p_0]$
let $?\beta_\mu = \mathrm{a}[u_1,\ \chi.chine,\ \varrho\sigma.p_0] \cdot (\chi.the\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0s_1.\varphi \cdot (\omega.the\text{-}\vartheta \star \varrho\sigma.p_1) \cdot$
$\mathrm{a}^{-1}[t_0,\ \omega.chine,\ \varrho\sigma.p_1] \cdot (t_0 \star inv\ \gamma_\tau) \cdot \mathrm{a}[t_0,\ \tau\mu.p_1,\ chine] \cdot$
$(inv\ t_0u_1.\varphi \star chine) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ chine]$
have w_μ: *ide* $?w_\mu \wedge$ *is-left-adjoint* $?w_\mu$
using *is-map left-adjoints-compose* **by** *simp*
have $w_\mu{}'$: *ide* $?w_\mu{}' \wedge$ *is-left-adjoint* $?w_\mu{}'$
using χ*.is-map left-adjoints-compose*
by (*simp add*: *is-map left-adjoints-compose*)
have *1*: $\exists!\gamma.$ «γ : $?w_\mu \Rightarrow ?w_\mu{}'$» $\wedge$ $?\beta_\mu = u_1 \star \gamma \wedge ?\vartheta_\mu = ?\vartheta_\mu{}' \cdot (u_0 \star \gamma)$
proof −
have ϑ_μ: «$?\vartheta_\mu$: $u_0 \star ?w_\mu \Rightarrow ?u_\mu$»

by *auto*
have $\vartheta_\mu{}'$: «$?\vartheta_\mu{}' : u_0 \star ?w_\mu{}' \Rightarrow ?u_\mu$»
by *fastforce*
have β_μ: «$?\beta_\mu : u_1 \star ?w_\mu \Rightarrow u_1 \star ?w_\mu{}'$»
proof (*intro comp-in-homI*)
show «$\mathrm{a}^{-1}[u_1, \tau\mu.p_0, chine] : u_1 \star \tau\mu.p_0 \star chine \Rightarrow (u_1 \star \tau\mu.p_0) \star chine$»
by *auto*
show «$inv\ t_0u_1.\varphi \star chine : (u_1 \star \tau\mu.p_0) \star chine \Rightarrow (t_0 \star \tau\mu.p_1) \star chine$»
using $t_0u_1.\varphi$*-uniqueness*(*2*) *hcomp-in-vhom*
by (*simp add*: $t_0u_1.\varphi$*-in-hom*(*2*) *w-in-hom*(*2*))
show «$\mathrm{a}[t_0, \tau\mu.p_1, chine] : (t_0 \star \tau\mu.p_1) \star chine \Rightarrow t_0 \star \tau\mu.p_1 \star chine$»
using τ*.T0.antipar*(*1*) **by** *auto*
show «$t_0 \star inv\ \gamma_\tau : t_0 \star \tau\mu.p_1 \star chine \Rightarrow t_0 \star \omega.chine \star \varrho\sigma.p_1$»
using γ_τ *iso-*γ_τ **using** τ*.T0.antipar*(*1*) **by** *auto*
show «$\mathrm{a}^{-1}[t_0, \omega.chine, \varrho\sigma.p_1] : t_0 \star \omega.chine \star \varrho\sigma.p_1 \Rightarrow (t_0 \star \omega.chine) \star \varrho\sigma.p_1$»
using ϱ*.T0.antipar*(*1*) **by** *auto*
show «$\omega.the\text{-}\vartheta \star \varrho\sigma.p_1 : (t_0 \star \omega.chine) \star \varrho\sigma.p_1 \Rightarrow r_0 \star \varrho\sigma.p_1$»
using ϱ*.T0.antipar*(*1*) **by** *auto*
show «$r_0s_1.\varphi : r_0 \star \varrho\sigma.p_1 \Rightarrow s_1 \star \varrho\sigma.p_0$»
by *auto*
show «$\chi.the\text{-}\nu \star \varrho\sigma.p_0 : s_1 \star \varrho\sigma.p_0 \Rightarrow (u_1 \star \chi.chine) \star \varrho\sigma.p_0$»
by *auto*
show «$\mathrm{a}[u_1, \chi.chine, \varrho\sigma.p_0] : (u_1 \star \chi.chine) \star \varrho\sigma.p_0 \Rightarrow u_1 \star \chi.chine \star \varrho\sigma.p_0$»
by *auto*
qed

The proof of the equation below needs to make use of the equation $\vartheta_\tau{}' = \vartheta_\tau \cdot (t_0 \star \gamma_\tau{}')$ from the previous section. So the overall strategy is to work toward an expression of the form $\vartheta_\tau \cdot (t_0 \star \gamma_\tau{}')$ and perform the replacement to eliminate $\gamma_\tau{}'$.

have eq_μ: $(u \star ?\vartheta_\mu) \cdot \mathrm{a}[u, u_0, ?w_\mu] \cdot (\mu \star ?w_\mu) =$
$((u \star ?\vartheta_\mu{}') \cdot \mathrm{a}[u, u_0, ?w_\mu{}'] \cdot (\mu \star ?w_\mu{}')) \cdot ?\beta_\mu$
proof −
let $?LHS = (u \star ?\vartheta_\mu) \cdot \mathrm{a}[u, u_0, ?w_\mu] \cdot (\mu \star ?w_\mu)$
let $?RHS = ((u \star ?\vartheta_\mu{}') \cdot \mathrm{a}[u, u_0, ?w_\mu{}'] \cdot (\mu \star ?w_\mu{}')) \cdot ?\beta_\mu$
have $?RHS = (u \star (\chi.the\text{-}\vartheta \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[u_0, \chi.chine, \varrho\sigma.p_0]) \cdot$
$\mathrm{a}[u, u_0, \chi.chine \star \varrho\sigma.p_0] \cdot (\mu \star \chi.chine \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}[u_1, \chi.chine, \varrho\sigma.p_0] \cdot (\chi.the\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0s_1.\varphi \cdot$
$(\omega.the\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0, \omega.chine, \varrho\sigma.p_1] \cdot (t_0 \star inv\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0, \tau\mu.p_1, chine] \cdot (inv\ t_0u_1.\varphi \star chine) \cdot \mathrm{a}^{-1}[u_1, \tau\mu.p_0, chine]$
using *comp-assoc* **by** *presburger*
also have ... $= (u \star \chi.the\text{-}\vartheta \star \varrho\sigma.p_0) \cdot ((u \star \mathrm{a}^{-1}[u_0, \chi.chine, \varrho\sigma.p_0]) \cdot$
$\mathrm{a}[u, u_0, \chi.chine \star \varrho\sigma.p_0]) \cdot (\mu \star \chi.chine \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}[u_1, \chi.chine, \varrho\sigma.p_0] \cdot (\chi.the\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0s_1.\varphi \cdot$
$(\omega.the\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0, \omega.chine, \varrho\sigma.p_1] \cdot (t_0 \star inv\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0, \tau\mu.p_1, chine] \cdot (inv\ t_0u_1.\varphi \star chine) \cdot$
$\mathrm{a}^{-1}[u_1, \tau\mu.p_0, chine]$
proof −
have $u \star (\chi.the\text{-}\vartheta \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[u_0, \chi.chine, \varrho\sigma.p_0] =$
$(u \star \chi.the\text{-}\vartheta \star \varrho\sigma.p_0) \cdot (u \star \mathrm{a}^{-1}[u_0, \chi.chine, \varrho\sigma.p_0])$

using *whisker-left* μ*.ide-base* $\vartheta_\mu{}'$ **by** *blast*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... = $((u \star \chi.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_0) \cdot \mathrm{a}[u,\ u_0 \star \chi.\mathit{chine},\ \varrho\sigma.p_0]) \cdot$
$(\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot$
$(\mu \star \chi.\mathit{chine} \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have *seq* $(u \star \mathrm{a}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0])$
$(\mathrm{a}[u,\ u_0 \star \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0))$
by *auto*
moreover have *src* $u =$ *trg* $\mathrm{a}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
by *simp*
moreover have *inv* $(u \star \mathrm{a}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]) = u \star \mathrm{a}^{-1}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
by *simp*
moreover have *iso* $(u \star \mathrm{a}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0])$
by *simp*
moreover have *iso* $\mathrm{a}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
by *simp*
ultimately have $(u \star \mathrm{a}^{-1}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]) \cdot \mathrm{a}[u,\ u_0,\ \chi.\mathit{chine} \star \varrho\sigma.p_0] =$
$\mathrm{a}[u,\ u_0 \star \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}^{-1}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
using *pentagon hseqI′ comp-assoc*
invert-opposite-sides-of-square
[*of* $u \star \mathrm{a}[u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
$\mathrm{a}[u,\ u_0 \star \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0)$
$\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine} \star \varrho\sigma.p_0]\ \mathrm{a}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$]
inv-hcomp χ*.is-ide* χ*.w-simps*(*3*) χ*.w-simps*(*4*) μ*.base-simps*(*2*) μ*.ide-base*
μ*.ide-leg0* μ*.leg0-simps*(*2*) μ*.leg0-simps*(*3*) σ*.leg1-simps*(*3*)
assoc′-eq-inv-assoc ide-hcomp $r_0 s_1$*.ide-u* $r_0 s_1.p_0$*-simps hcomp-simps*(*1*)
by *presburger*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... = $\mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot$
$(\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot (\mathrm{a}^{-1}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot$
$(\mu \star \chi.\mathit{chine} \star \varrho\sigma.p_0)) \cdot$
$\mathrm{a}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $(u \star \chi.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_0) \cdot \mathrm{a}[u,\ u_0 \star \chi.\mathit{chine},\ \varrho\sigma.p_0] =$
$\mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0)$
using *assoc-naturality* [*of* u χ*.the-*ϑ $\varrho\sigma.p_0$] χ*.*ϑ*-simps*(*3*) **by** *auto*

thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot$
$(\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot ((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot$
$\mathrm{a}^{-1}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot \mathrm{a}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot$
$(\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $\mathrm{a}^{-1}[u \star u_0,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot (\mu \star \chi.\mathit{chine} \star \varrho\sigma.p_0) =$
$((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot \mathrm{a}^{-1}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]$
using *assoc′-naturality* [*of* $\mu\ \chi.\mathit{chine}\ \varrho\sigma.p_0$] **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot$
$(\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot ((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot$
$((\mathrm{a}^{-1}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot \mathrm{a}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]) \cdot$
$(\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0)) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-assoc* **by** *metis*
also have ... $= \mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot (((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot$
$(\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot ((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot$
$(\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0)) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1] \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $(\mathrm{a}^{-1}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0] \cdot \mathrm{a}[u_1,\ \chi.\mathit{chine},\ \varrho\sigma.p_0]) \cdot (\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) =$
$\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0$
using *comp-inv-arr′ comp-cod-arr* **by** *auto*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (\mathrm{a}[u,\ s_0,\ \varrho\sigma.p_0] \cdot ((\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0) \cdot r_0 s_1.\varphi \cdot$
$(\omega.\mathit{the}\text{-}\vartheta \star \varrho\sigma.p_1) \cdot \mathrm{a}^{-1}[t_0,\ \omega.\mathit{chine},\ \varrho\sigma.p_1]) \cdot (t_0 \star \mathit{inv}\ \gamma_\tau) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0 u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $\mathit{arr}\ ((u \star \chi.\mathit{the}\text{-}\vartheta) \cdot \mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \cdot (\mu \star \chi.\mathit{chine}) \cdot \chi.\mathit{the}\text{-}\nu)$
using $\chi.\vartheta\text{-}\mathit{simps}(3)$ **by** *simp*
hence $((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot (\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot$
$((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot (\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) =$
$(u \star \chi.\mathit{the}\text{-}\vartheta) \cdot \mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \cdot (\mu \star \chi.\mathit{chine}) \cdot \chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0$
using *whisker-right* **by** *simp*

also have ... $= (\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0$
using $\chi.\Delta$-*naturality* **by** *simp*
finally have $((u \star \chi.\mathit{the}\text{-}\vartheta) \star \varrho\sigma.p_0) \cdot (\mathrm{a}[u,\ u_0,\ \chi.\mathit{chine}] \star \varrho\sigma.p_0) \cdot$
$((\mu \star \chi.\mathit{chine}) \star \varrho\sigma.p_0) \cdot (\chi.\mathit{the}\text{-}\nu \star \varrho\sigma.p_0) =$
$(\chi \star s_0) \cdot \sigma \star \varrho\sigma.p_0$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (?\vartheta_\tau \cdot (t_0 \star \mathit{inv}\ \gamma_\tau)) \cdot$
$\mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-assoc* **by** *presburger*
also have ... $= ?\vartheta_\tau{}' \cdot \mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}] \cdot (\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using $\gamma_\tau{}'\ \gamma_\tau{}'$-*eq* **by** *simp*
also have ... $= (u \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot ((t_0u_1.\varphi \star \mathit{chine}) \cdot$
$((\mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot \mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot$
$(\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine})) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-assoc* **by** *presburger*
also have ... $= (u \star \mathit{the}\text{-}\vartheta) \cdot \mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$((\mu \star \tau\mu.p_0) \star \mathit{chine}) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $((t_0u_1.\varphi \star \mathit{chine}) \cdot ((\mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot \mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot$
$(\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine})) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}] =$
$\mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
proof −
have $((t_0u_1.\varphi \star \mathit{chine}) \cdot ((\mathrm{a}^{-1}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot \mathrm{a}[t_0,\ \tau\mu.p_1,\ \mathit{chine}]) \cdot$
$(\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine})) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}] =$
$((t_0u_1.\varphi \star \mathit{chine}) \cdot ((t_0 \star \tau\mu.p_1) \star \mathit{chine}) \cdot$
$(\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine})) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-inv-arr'* τ.*T0.antipar*(*1*) **by** *auto*
also have ... $= ((t_0u_1.\varphi \star \mathit{chine}) \cdot (\mathit{inv}\ t_0u_1.\varphi \star \mathit{chine})) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-cod-arr* $t_0u_1.\varphi$-*uniqueness* **by** *simp*
also have ... $= (t_0u_1.\varphi \cdot \mathit{inv}\ t_0u_1.\varphi \star \mathit{chine}) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *whisker-right* $t_0u_1.\varphi$-*uniqueness* **by** *simp*
also have ... $= ((u_1 \star \tau\mu.p_0) \star \mathit{chine}) \cdot \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-arr-inv'* τ.*T0.antipar*(*1*) $t_0u_1.\varphi$-*uniqueness* **by** *simp*
also have ... $= \mathrm{a}^{-1}[u_1,\ \tau\mu.p_0,\ \mathit{chine}]$
using *comp-cod-arr* τ.*T0.antipar*(*1*) **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (u \star \mathit{the}\text{-}\vartheta) \cdot (\mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine}) \cdot$
$\mathrm{a}^{-1}[u \star u_0,\ \tau\mu.p_0,\ \mathit{chine}]) \cdot (\mu \star ?w_\mu)$
using *assoc'-naturality* [*of* μ $\tau\mu.p_0$ *chine*] *comp-assoc* **by** *auto*

also have ... = $((u \star \mathit{the}\text{-}\vartheta) \cdot (u \star \mathrm{a}^{-1}[u_0,\ \tau\mu.p_0,\ \mathit{chine}])) \cdot \mathrm{a}[u,\ u_0,\ ?w_\mu] \cdot (\mu \star ?w_\mu)$
using *uwϑ pentagon comp-assoc*
invert-opposite-sides-of-square
[*of* $u \star \mathrm{a}[u_0,\ \tau\mu.p_0,\ \mathit{chine}]$
$\mathrm{a}[u,\ u_0 \star \tau\mu.p_0,\ \mathit{chine}] \cdot (\mathrm{a}[u,\ u_0,\ \tau\mu.p_0] \star \mathit{chine})\ \mathrm{a}[u,\ u_0,\ ?w_\mu]$
$\mathrm{a}[u \star u_0,\ \tau\mu.p_0,\ \mathit{chine}]$]
μ.base-simps(*2*) *μ.ide-base μ.ide-leg0 μ.leg0-simps*(*2*) *assoc′-eq-inv-assoc*
ide-hcomp hcomp-simps(*1*) t_0u_1*.ide-u*
by *force*
also have ... = $(u \star \mathit{the}\text{-}\vartheta \cdot \mathrm{a}^{-1}[u_0,\ \tau\mu.p_0,\ \mathit{chine}]) \cdot \mathrm{a}[u,\ u_0,\ ?w_\mu] \cdot (\mu \star ?w_\mu)$
using *whisker-left comp-assoc* **by** *simp*
finally show *?thesis* **by** *simp*
qed
show $\exists!\gamma.\ «\gamma : ?w_\mu \Rightarrow ?w_\mu{}'» \wedge ?\beta_\mu = u_1 \star \gamma \wedge ?\vartheta_\mu = ?\vartheta_\mu{}' \cdot (u_0 \star \gamma)$
using $w_\mu\ w_\mu{}'\ \vartheta_\mu\ \vartheta_\mu{}'\ \beta_\mu\ eq_\mu\ \mu.T2$ [*of* $?w_\mu\ ?w_\mu{}'\ ?\vartheta_\mu\ ?u_\mu\ ?\vartheta_\mu{}'\ ?\beta_\mu$] **by** *fast*
qed
obtain γ_μ **where** γ_μ: $«\gamma_\mu : ?w_\mu \Rightarrow ?w_\mu{}'» \wedge ?\beta_\mu = u_1 \star \gamma_\mu \wedge ?\vartheta_\mu = ?\vartheta_\mu{}' \cdot (u_0 \star \gamma_\mu)$
using *1* **by** *auto*
show $?w_\mu \cong ?w_\mu{}'$
using $w_\mu\ w_\mu{}'\ \gamma_\mu$ *BS3* [*of* $?w_\mu\ ?w_\mu{}'\ \gamma_\mu\ \gamma_\mu$] *isomorphic-def* **by** *auto*
qed

lemma *comp-L*:
shows *Maps.seq* $[\![[\![t_0]\!]]\!]\ [\![[\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]]\!]$
and $[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]]\!] =$
Maps.MkArr (*src* $(\omega.\mathit{chine} \star \varrho\sigma.p_1)$) (*src t*) (*Maps.Comp* $[\![t_0]\!]\ [\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]$)
proof −
show *Maps.seq* $[\![[\![t_0]\!]]\!]\ [\![[\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]]\!]$
proof −
have *is-left-adjoint* $(\omega.\mathit{chine} \star \varrho\sigma.p_1)$
using *ω.is-map* r_0s_1*.leg1-is-map left-adjoints-compose* $r_0s_1.p_1$*-simps* **by** *auto*
thus *?thesis*
using *Maps.CLS-in-hom* r_0s_1*.leg1-is-map*
apply (*intro Maps.seqI′*)
apply *blast*
using *Maps.CLS-in-hom* [*of* t_0] *τ.leg0-is-map ϱσ.leg1-in-hom* **by** *auto*
qed
thus $[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]]\!] =$
Maps.MkArr (*src* $(\omega.\mathit{chine} \star \varrho\sigma.p_1)$) (*src t*) (*Maps.Comp* $[\![t_0]\!]\ [\![\omega.\mathit{chine} \star \varrho\sigma.p_1]\!]$)
using *Maps.comp-char* **by** *auto*
qed

lemma *comp-R*:
shows *Maps.seq* $[\![[\![u_1]\!]]\!]\ [\![[\![\chi.\mathit{chine} \star \varrho\sigma.p_0]\!]]\!]$
and $[\![[\![u_1]\!]]\!] \odot [\![[\![\chi.\mathit{chine} \star \varrho\sigma.p_0]\!]]\!] =$
Maps.MkArr (*src* $r_0s_1.p_0$) (*trg u*) (*Maps.Comp* $[\![u_1]\!]\ [\![\chi.\mathit{chine} \star r_0s_1.p_0]\!]$)
proof −
show *Maps.seq* $[\![[\![u_1]\!]]\!]\ [\![[\![\chi.\mathit{chine} \star \varrho\sigma.p_0]\!]]\!]$
proof −

have *is-left-adjoint* (χ.*chine* $\star$ $\varrho\sigma$.p_0)
using χ.*is-map* r_0s_1.*leg0-is-map left-adjoints-compose* [*of* χ.*chine* $\varrho\sigma$.p_0] **by** *simp*
thus *?thesis*
using *Maps.CLS-in-hom* μ.*leg1-is-map*
apply (*intro Maps.seqI'*)
apply *blast*
using *Maps.CLS-in-hom* [*of* u_1] μ.*leg1-is-map* **by** *simp*
qed
thus $[\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!] =$
Maps.MkArr (*src* $r_0s_1.p_0$) (*trg* u) (*Maps.Comp* $[\![u_1]\!]$ $[\![\chi.chine \star r_0s_1.p_0]\!]$)
using *Maps.comp-char* **by** *auto*
qed

lemma *comp-L-eq-comp-R*:
shows $[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!] = [\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
proof (*intro Maps.arr-eqI*)
show *Maps.seq* $[\![[\![t_0]\!]]\!]$ $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$
using *comp-L*(*1*) **by** *simp*
show *Maps.seq* $[\![[\![u_1]\!]]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
using *comp-R*(*1*) **by** *simp*
show *Maps.Dom* ($[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$) = *Maps.Dom* ($[\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$)
by (*metis* (*no-types*, *lifting*) *Maps.Dom.simps*(*1*) ω.*w-simps*(*2*) ω.*w-simps*(*3*)
ϱ.*leg1-simps*(*3*) $\varrho\sigma$.*leg1-in-hom*(*2*) *comp-L*(*2*) *comp-R*(*2*) *hcomp-in-vhomE hseqI'*
r_0s_1.*leg1-simps*(*3*) *hcomp-simps*(*1*))
show *Maps.Cod* ($[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$) = *Maps.Cod* ($[\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$)
by (*metis Maps.Cod.simps*(*1*) $\tau\mu$.*composable comp-L*(*2*) *comp-R*(*2*))
have *A*: *Maps.Map* ($[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$) = *Maps.Comp* $[\![t_0]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$
using *comp-L*(*1*) *Maps.comp-char* **by** *auto*
have *B*: *Maps.Map* ($[\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$) = *Maps.Comp* $[\![u_1]\!]$ $[\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *comp-R*(*1*) *Maps.comp-char* **by** *auto*
have *C*: *Maps.Comp* $[\![t_0]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$ = *Maps.Comp* $[\![u_1]\!]$ $[\![\chi.chine \star \varrho\sigma.p_0]\!]$
proof (*intro Maps.Comp-eqI*)
show $t_0 \star \omega.chine \star \varrho\sigma.p_1 \in$ *Maps.Comp* $[\![t_0]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$
proof (*intro Maps.in-CompI*)
show *is-iso-class* $[\![\omega.chine \star \varrho\sigma.p_1]\!]$
using *prj-chine*(*2*) *is-iso-classI isomorphic-implies-hpar*(*2*) **by** *blast*
show *is-iso-class* $[\![t_0]\!]$
using *is-iso-classI* **by** *auto*
show $\omega.chine \star \varrho\sigma.p_1 \in [\![\omega.chine \star \varrho\sigma.p_1]\!]$
using *ide-in-iso-class prj-chine*(*2*) *isomorphic-implies-hpar*(*2*) **by** *blast*
show $t_0 \in [\![t_0]\!]$
using *ide-in-iso-class* **by** *simp*
show $t_0 \star \omega.chine \star \varrho\sigma.p_1 \cong t_0 \star \omega.chine \star \varrho\sigma.p_1$
using *isomorphic-reflexive prj-chine*(*2*) *isomorphic-implies-hpar*(*2*) **by** *auto*
qed
show $u_1 \star \chi.chine \star \varrho\sigma.p_0 \in$ *Maps.Comp* $[\![u_1]\!]$ $[\![\chi.chine \star \varrho\sigma.p_0]\!]$
proof (*intro Maps.in-CompI*)
show *is-iso-class* $[\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *is-iso-classI* **by** *simp*

show *is-iso-class* $[\![u_1]\!]$
using *is-iso-classI* **by** *simp*
show $\chi.chine \star \varrho\sigma.p_0 \in [\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *ide-in-iso-class* **by** *simp*
show $u_1 \in$ *iso-class* u_1
using *ide-in-iso-class* **by** *simp*
show $u_1 \star \chi.chine \star \varrho\sigma.p_0 \cong u_1 \star \chi.chine \star \varrho\sigma.p_0$
using *isomorphic-reflexive isomorphic-implies-hpar*(*2*) **by** *auto*
qed
show $t_0 \star \omega.chine \star \varrho\sigma.p_1 \cong u_1 \star \chi.chine \star \varrho\sigma.p_0$
proof −
have $t_0 \star \omega.chine \star \varrho\sigma.p_1 \cong (t_0 \star \omega.chine) \star \varrho\sigma.p_1$
using *assoc′-in-hom* [*of* t_0 $\omega.chine$ $\varrho\sigma.p_1$] *iso-assoc′ isomorphic-def* $r_0 s_1.p_1$*-simps*
by *auto*
also have ... $\cong r_0 \star \varrho\sigma.p_1$
using ω*.leg0-uniquely-isomorphic hcomp-isomorphic-ide*
by (*simp add*: ϱ*.T0.antipar*(*1*))
also have ... $\cong s_1 \star \varrho\sigma.p_0$
using *isomorphic-def* $r_0 s_1.\varphi$*-uniqueness*(*2*) **by** *blast*
also have ... $\cong (u_1 \star \chi.chine) \star \varrho\sigma.p_0$
using χ*.leg1-uniquely-isomorphic hcomp-isomorphic-ide* **by** *auto*
also have ... $\cong u_1 \star \chi.chine \star \varrho\sigma.p_0$
using *assoc-in-hom* [*of* u_1 $\chi.chine$ $\varrho\sigma.p_0$] *iso-assoc isomorphic-def* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed
show *Maps.Map* $([\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]) =$ *Maps.Map* $([\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!])$
using *A B C* **by** *simp*
qed

lemma *csq*:
shows *Maps.commutative-square* $[\![[\![t_0]\!]]\!]$ $[\![[\![u_1]\!]]\!]$ $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
proof
show *Maps.cospan* $[\![[\![t_0]\!]]\!]$ $[\![[\![u_1]\!]]\!]$
using *comp-L*(*1*) *comp-R*(*1*) *comp-L-eq-comp-R*
by (*metis* (*no-types*, *lifting*) *Maps.cod-comp Maps.seq-char*)
show *Maps.span* $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
using *comp-L*(*1*) *comp-R*(*1*) *comp-L-eq-comp-R*
by (*metis* (*no-types*, *lifting*) *Maps.dom-comp Maps.seq-char*)
show *Maps.dom* $[\![[\![t_0]\!]]\!] =$ *Maps.cod* $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$
using *comp-L*(*1*) **by** *auto*
show $[\![[\![t_0]\!]]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!] = [\![[\![u_1]\!]]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
using *comp-L-eq-comp-R* **by** *simp*
qed

lemma *CLS-chine*:
shows $[\![[\![chine]\!]]\!] =$ *Maps.tuple* $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$ $[\![[\![t_0]\!]]\!]$ $[\![[\![u_1]\!]]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
proof −
let *?T* = *Maps.tuple* $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$ $[\![[\![t_0]\!]]\!]$ $[\![[\![u_1]\!]]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$

have $\exists!l.\ [\![t_0 u_1.p_1]\!] \odot l = [\![\omega.chine \star \varrho\sigma.p_1]\!] \wedge [\![t_0 u_1.p_0]\!] \odot l = [\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *csq* $\tau\mu$*.prj-char*
Maps.universal [*of* $[\![t_0]\!]$ $[\![u_1]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$ $[\![\chi.chine \star \varrho\sigma.p_0]\!]$]
by *simp*
moreover have $[\![\tau\mu.p_1]\!] \odot ?T = [\![\omega.chine \star \varrho\sigma.p_1]\!] \wedge$
$[\![\tau\mu.p_0]\!] \odot ?T = [\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *csq* $\tau\mu$*.prj-char*
Maps.prj-tuple [*of* $[\![t_0]\!]$ $[\![u_1]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$ $[\![\chi.chine \star \varrho\sigma.p_0]\!]$]
by *simp*
moreover have $[\![t_0 u_1.p_1]\!] \odot [\![chine]\!] = [\![\omega.chine \star \varrho\sigma.p_1]\!] \wedge$
$[\![t_0 u_1.p_0]\!] \odot [\![chine]\!] = [\![\chi.chine \star \varrho\sigma.p_0]\!]$
using *prj-chine* $\tau\mu$*.leg0-is-map* $\tau\mu$*.leg1-is-map is-map* $t_0 u_1$*.leg1-is-map*
$t_0 u_1$*.satisfies-T0 Maps.comp-CLS*
by *blast*
ultimately show $[\![chine]\!] = ?T$ **by** *auto*
qed

end

2.3.8 Equivalence of B and Span(Maps(B))

The Functor SPN

We now define a function *SPN* on arrows and will ultimately show that it extends to a biequivalence from the underlying bicategory B to $Span(Maps(B))$. The idea is that *SPN* takes «$\mu : r \Rightarrow s$» to the isomorphism class of an induced arrow of spans from the chosen tabulation of r to the chosen tabulation of s. To obtain this, we first use isomorphisms $r.tab_1 \star r.tab_0{}^* \cong r$ and $s.tab_1 \star s.tab_0{}^* \cong s$ to transform μ to «$\mu' : r.tab_1 \star r.tab_0{}^* \Rightarrow s.tab_1 \star s.tab_0{}^*$». We then take the adjoint transpose of μ' to obtain «$\omega : r.tab_1 \Rightarrow (s.tab_1 \star s.tab_0{}^*) \star r.tab_0$». The 2-cell ω induces a map w which is an arrow of spans from $(r.tab_0,\ r.tab_1)$ to $(s.tab_0,\ s.tab_1)$. We take the arrow of $Span(Maps(B))$ defined by w as the value of *SPN* μ.

Ensuring that *SPN* is functorial is a somewhat delicate point, which requires that all the underlying definitions that have been set up are "just so", with no extra choices other than those that are forced, and with the tabulation assigned to each 1-cell r in the proper relationship with the canonical tabulation assigned to its chosen factorization $r = g \star f^*$.

context *bicategory-of-spans*
begin

interpretation *Maps*: *maps-category V H* a i *src trg* **..**
interpretation *Span*: *span-bicategory Maps.comp Maps.PRJ*$_0$ *Maps.PRJ*$_1$ **..**

no-notation *Fun.comp* (**infixl** ‹∘› *55*)
notation *Span.vcomp* (**infixr** ‹•› *55*)
notation *Span.hcomp* (**infixr** ‹∘› *53*)
notation *Maps.comp* (**infixr** ‹⊙› *55*)
notation *isomorphic* (**infix** ‹≅› *50*)

definition *spn*
where *spn* $\mu \equiv$
arrow-of-tabulations-in-maps.chine V H a i *src trg*
(*tab-of-ide* (*dom* μ)) (tab_0 (*dom* μ)) (*cod* μ)
(*tab-of-ide* (*cod* μ)) (tab_0 (*cod* μ)) (tab_1 (*cod* μ)) μ

lemma *is-induced-map-spn*:
assumes *arr* μ
shows *arrow-of-tabulations-in-maps.is-induced-map* V H a i *src trg*
(*tab-of-ide* (*dom* μ)) (tab_0 (*dom* μ)) (*cod* μ)
(*tab-of-ide* (*cod* μ)) (tab_0 (*cod* μ)) (tab_1 (*cod* μ))
μ (*spn* μ)
proof −
interpret μ: *arrow-in-bicategory-of-spans* V H a i *src trg* ‹*dom* μ› ‹*cod* μ› μ
using *assms* **by** *unfold-locales auto*
interpret μ: *arrow-of-tabulations-in-maps* V H a i *src trg*
‹*dom* μ› μ.*r.tab* ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
‹*cod* μ› μ.*s.tab* ‹tab_0 (*cod* μ)› ‹tab_1 (*cod* μ)›
μ
using μ.*is-arrow-of-tabulations-in-maps* **by** *simp*
show *?thesis*
unfolding *spn-def*
using μ.*chine-is-induced-map* **by** *blast*
qed

lemma *spn-props*:
assumes *arr* μ
shows «*spn* μ : *src* (tab_0 (*dom* μ)) $\rightarrow$ *src* (tab_0 (*cod* μ))»
and *is-left-adjoint* (*spn* μ)
and tab_0 (*cod* μ) $\star$ *spn* μ $\cong$ tab_0 (*dom* μ)
and tab_1 (*cod* μ) $\star$ *spn* μ $\cong$ tab_1 (*dom* μ)
proof −
interpret μ: *arrow-in-bicategory-of-spans* V H a i *src trg* ‹*dom* μ› ‹*cod* μ› μ
using *assms* **by** *unfold-locales auto*
interpret μ: *arrow-of-tabulations-in-maps* V H a i *src trg*
‹*dom* μ› μ.*r.tab* ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
‹*cod* μ› μ.*s.tab* ‹tab_0 (*cod* μ)› ‹tab_1 (*cod* μ)›
μ
using μ.*is-arrow-of-tabulations-in-maps* **by** *simp*
show «*spn* μ : *src* (tab_0 (*dom* μ)) $\rightarrow$ *src* (tab_0 (*cod* μ))»
using *spn-def* **by** *simp*
show *is-left-adjoint* (*spn* μ)
using *spn-def* **by** (*simp add*: μ.*is-map*)
show tab_0 (*cod* μ) $\star$ *spn* μ $\cong$ tab_0 (*dom* μ)
using *spn-def isomorphic-def* μ.*leg0-uniquely-isomorphic*(*1*) **by** *auto*
show tab_1 (*cod* μ) $\star$ *spn* μ $\cong$ tab_1 (*dom* μ)
using *spn-def isomorphic-def isomorphic-symmetric*
μ.*leg1-uniquely-isomorphic*(*1*)

by *auto*
qed

lemma *spn-in-hom* [*intro*]:
assumes *arr* μ
shows «*spn* μ : *src* (tab_0 (*dom* μ)) $\rightarrow$ *src* (tab_0 (*cod* μ))»
and «*spn* μ : *spn* $\mu \Rightarrow$ *spn* μ»
using *assms spn-props left-adjoint-is-ide* **by** *auto*

lemma *spn-simps* [*simp*]:
assumes *arr* μ
shows *is-left-adjoint* (*spn* μ)
and *ide* (*spn* μ)
and *src* (*spn* μ) = *src* (tab_0 (*dom* μ))
and *trg* (*spn* μ) = *src* (tab_0 (*cod* μ))
using *assms spn-props left-adjoint-is-ide* **by** *auto*

We need the next result to show that *SPN* is functorial; in particular, that it takes «r : $r \Rightarrow r$» in the underlying bicategory to a 1-cell in *Span*(*Maps*(B)). The 1-cells in *Span*(*Maps*(B)) have objects of *Maps*(B) as their chines, and objects of *Maps*(B) are isomorphism classes of objects in the underlying bicategory B. So we need the induced map associated with r to be isomorphic to an object.

lemma *spn-ide*:
assumes *ide r*
shows *spn r* $\cong$ *src* (tab_0 *r*)
proof −
interpret *r*: *identity-in-bicategory-of-spans V H* a i *src trg r*
using *assms* **by** *unfold-locales auto*
interpret *r*: *arrow-of-tabulations-in-maps V H* a i *src trg*
r r.tab ‹tab_0 *r*› ‹tab_1 *r*› *r r.tab* ‹tab_0 *r*› ‹tab_1 *r*› *r*
using *r.is-arrow-of-tabulations-in-maps* **by** *simp*
interpret *tab*: *tabulation V H* a i *src trg r* ‹*r.tab*› ‹tab_0 *r*› ‹*dom r.tab*›
using *assms r.tab-is-tabulation* **by** *simp*
interpret *tab*: *tabulation-in-maps V H* a i *src trg r* ‹*r.tab*› ‹tab_0 *r*› ‹*dom r.tab*›
by (*unfold-locales*, *simp-all*)
have *tab.is-induced-by-cell* (*spn r*) (tab_0 *r*) *r.tab*
using *spn-def comp-ide-arr r.chine-is-induced-map* **by** *auto*
thus *?thesis*
using *tab.induced-map-unique* [*of* tab_0 *r r.tab spn r src* $r.s_0$]
tab.apex-is-induced-by-cell
by (*simp add*: *comp-assoc*)
qed

The other key result we need to show that *SPN* is functorial is to show that the induced map of a composite is isomorphic to the composite of induced maps.

lemma *spn-hcomp*:
assumes *seq* τ μ **and** $g \cong$ *spn* τ **and** $f \cong$ *spn* μ
shows *spn* ($\tau \cdot \mu$) $\cong g \star f$
proof −

interpret τ: *arrow-in-bicategory-of-spans* V H a i *src trg* ‹*dom* τ› ‹*cod* τ› τ
  using *assms* **by** *unfold-locales auto*
interpret τ: *arrow-of-tabulations-in-maps* V H a i *src trg*
  ‹*dom* τ› τ.*r.tab* ‹tab_0 (*dom* τ)› ‹tab_1 (*dom* τ)›
  ‹*cod* τ› τ.*s.tab* ‹tab_0 (*cod* τ)› ‹tab_1 (*cod* τ)›
  τ
  using τ.*is-arrow-of-tabulations-in-maps* **by** *simp*
interpret μ: *arrow-in-bicategory-of-spans* V H a i *src trg* ‹*dom* μ› ‹*dom* τ› μ
  using *assms* **apply** *unfold-locales*
  apply *auto*[*1*]
  by (*elim seqE*, *auto*)
interpret μ: *arrow-of-tabulations-in-maps* V H a i *src trg*
  ‹*dom* μ› μ.*r.tab* ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
  ‹*dom* τ› τ.*r.tab* ‹tab_0 (*dom* τ)› ‹tab_1 (*dom* τ)›
  μ
  using μ.*is-arrow-of-tabulations-in-maps* **by** *simp*
interpret $\tau\mu$: *vertical-composite-of-arrows-of-tabulations-in-maps* V H a i *src trg*
  ‹*dom* μ› μ.*r.tab* ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
  ‹*dom* τ› τ.*r.tab* ‹tab_0 (*dom* τ)› ‹tab_1 (*dom* τ)›
  ‹*cod* τ› τ.*s.tab* ‹tab_0 (*cod* τ)› ‹tab_1 (*cod* τ)›
  μ τ
  ..
have $g \cong \tau$.*chine*
  using *assms*(*2*) *spn-def* **by** *auto*
moreover have $f \cong \mu$.*chine*
  using *assms*(*1*) *assms*(*3*) *spn-def* **by** *auto*
moreover have *src* g = *trg* f
  using *calculation*(*1*−*2*) *isomorphic-implies-hpar*(*3*−*4*) **by** *auto*
moreover have *src* g = *trg* μ.*chine*
  using *calculation*(*1*) *isomorphic-implies-hpar*(*3*) **by** *auto*
ultimately have $g \star f \cong \tau$.*chine* $\star$ μ.*chine*
  using *hcomp-ide-isomorphic hcomp-isomorphic-ide isomorphic-transitive*
  by (*meson* μ.*is-ide isomorphic-implies-ide*(*1*))
also have ... $\cong$ *spn* ($\tau \cdot \mu$)
  using *spn-def* $\tau\mu$.*chine-char isomorphic-symmetric*
  by (*metis* $\tau\mu$.*in-hom in-homE*)
finally show *?thesis*
  using *isomorphic-symmetric* **by** *simp*
qed

abbreviation (*input*) SPN_0
where SPN_0 $r \equiv$ *Span.mkIde* $[\![[\![tab_0\ r]\!]]\!]$ $[\![[\![tab_1\ r]\!]]\!]$

definition *SPN*
where *SPN* $\mu \equiv$ *if arr* μ *then*
  $(\!|$*Chn* = $[\![[\![spn\ \mu]\!]]\!]$,
  Dom = $(\!|$*Leg0* = $[\![[\![tab_0\ (dom\ \mu)]\!]]\!]$, *Leg1* = $[\![[\![tab_1\ (dom\ \mu)]\!]]\!]$$|\!)$,
  Cod = $(\!|$*Leg0* = $[\![[\![tab_0\ (cod\ \mu)]\!]]\!]$, *Leg1* = $[\![[\![tab_1\ (cod\ \mu)]\!]]\!]$$|\!)$$|\!)$
  else Span.null

lemma *Dom-SPN* [*simp*]:
assumes *arr* μ
shows *Dom* (*SPN* μ) = ⦇*Leg0* = ⟦⟦tab_0 (*dom* μ)⟧⟧, *Leg1* = ⟦⟦tab_1 (*dom* μ)⟧⟧⦈
  using *assms SPN-def* **by** *simp*

lemma *Cod-SPN* [*simp*]:
assumes *arr* μ
shows *Cod* (*SPN* μ) = ⦇*Leg0* = ⟦⟦tab_0 (*cod* μ)⟧⟧, *Leg1* = ⟦⟦tab_1 (*cod* μ)⟧⟧⦈
  using *assms SPN-def* **by** *simp*

Now we have to show this does the right thing for us.

lemma *SPN-in-hom*:
assumes *arr* μ
shows *Span.in-hom* (*SPN* μ) (SPN_0 (*dom* μ)) (SPN_0 (*cod* μ))
proof
  interpret *Dom*: *span-in-category Maps.comp* ‹*Dom* (*SPN* μ)›
  proof
    interpret *r*: *identity-in-bicategory-of-spans V H* a i *src trg* ‹*dom* μ›
      using *assms* **by** *unfold-locales auto*
    show *Maps.span* (*Leg0* (*Dom* (*SPN* μ))) (*Leg1* (*Dom* (*SPN* μ)))
      using *assms Maps.CLS-in-hom SPN-def*
      by (*metis* (*no-types*, *lifting*) *Maps.in-homE bicategory-of-spans.Dom-SPN*
        bicategory-of-spans-axioms r.leg1-is-map r.leg1-simps(*3*) *r.satisfies-T0*
        span-data.simps(*1*) *span-data.simps*(*2*))
  qed
  interpret *Dom′*: *span-in-category Maps.comp*
        ‹⦇*Leg0* = ⟦⟦tab_0 (*dom* μ)⟧⟧, *Leg1* = ⟦⟦tab_1 (*dom* μ)⟧⟧⦈›
    using *assms Dom.span-in-category-axioms SPN-def* **by** *simp*
  interpret *Cod*: *span-in-category Maps.comp* ‹*Cod* (*SPN* μ)›
  proof
    interpret *s*: *identity-in-bicategory-of-spans V H* a i *src trg* ‹*cod* μ›
      using *assms* **by** *unfold-locales auto*
    show *Maps.span* (*Leg0* (*Cod* (*SPN* μ))) (*Leg1* (*Cod* (*SPN* μ)))
      using *assms Maps.CLS-in-hom SPN-def*
      by (*metis* (*no-types*, *lifting*) *bicategory-of-spans.Cod-SPN bicategory-of-spans-axioms*
        ide-dom s.base-simps(*2*) *s.base-simps*(*3*) *s.determines-span span-in-category.is-span*)
  qed
  interpret *Cod′*: *span-in-category Maps.comp*
        ‹⦇*Leg0* = ⟦⟦tab_0 (*cod* μ)⟧⟧, *Leg1* = ⟦⟦tab_1 (*cod* μ)⟧⟧⦈›
    using *assms Cod.span-in-category-axioms SPN-def* **by** *simp*
  show *1*: *Span.arr* (*SPN* μ)
  proof (*unfold Span.arr-char*)
    show *arrow-of-spans Maps.comp* (*SPN* μ)
    proof (*unfold-locales*)
      show *Maps.in-hom* (*Chn* (*SPN* μ)) *Dom.apex Cod.apex*
        unfolding *SPN-def Maps.in-hom-char*
        using *assms Dom′.apex-def Cod′.apex-def Dom′.is-span Cod′.is-span Maps.arr-char*
        by *auto*

```
      show Cod.leg0 ⊙ Chn (SPN μ) = Dom.leg0
        unfolding SPN-def
        using assms spn-props [of μ] Maps.comp-CLS [of tab₀ (cod μ) spn μ] by simp
      show Cod.leg1 ⊙ Chn (SPN μ) = Dom.leg1
        unfolding SPN-def
        using assms spn-props [of μ] Maps.comp-CLS [of tab₁ (cod μ) spn μ] by simp
    qed
  qed
  show Span.dom (SPN μ) = SPN₀ (dom μ)
    using assms 1 Span.dom-char Dom'.apex-def SPN-def by simp
  show Span.cod (SPN μ) = SPN₀ (cod μ)
    using assms 1 Span.cod-char Cod'.apex-def SPN-def by simp
qed

interpretation SPN: functor V Span.vcomp SPN
proof
  show ⋀μ. ¬ arr μ ⟹ SPN μ = Span.null
    unfolding SPN-def by simp
  show 1: ⋀μ. arr μ ⟹ Span.arr (SPN μ)
    using SPN-in-hom by auto
  show ⋀μ. arr μ ⟹ Span.dom (SPN μ) = SPN (dom μ)
  proof −
    fix μ
    assume μ: arr μ
    have 1: Maps.arr (Maps.MkArr (src (tab₀ (dom μ))) (src μ) ⟦tab₀ (dom μ)⟧)
    proof −
      have src (tab₀ (dom μ)) ∈ Collect obj
        using μ by simp
      moreover have src μ ∈ Collect obj
        using μ by simp
      moreover have ⟦tab₀ (dom μ)⟧ ∈ Maps.Hom (src (tab₀ (local.dom μ))) (src μ)
      proof −
        have «tab₀ (dom μ) : src (tab₀ (dom μ)) → src μ»
          using μ by simp
        moreover have is-left-adjoint (tab₀ (dom μ))
          using μ tab₀-simps [of dom μ] by auto
        ultimately show ?thesis by auto
      qed
      ultimately show ?thesis by simp
    qed
    have ⟦spn (dom μ)⟧ = ⟦src (tab₀ (dom μ))⟧
      using μ spn-ide iso-class-eqI by auto
    hence SPN (dom μ) = SPN₀ (dom μ)
      unfolding SPN-def
      using μ 1 Maps.dom-char by simp
    thus Span.dom (SPN μ) = SPN (dom μ)
      using μ SPN-in-hom by auto
  qed
  show ⋀μ. arr μ ⟹ Span.cod (SPN μ) = SPN (cod μ)
```

```
proof −
  fix μ
  assume μ: arr μ
  have 1: Maps.arr (Maps.MkArr (src (tab₀ (cod μ))) (src μ) ⟦tab₀ (cod μ)⟧)
  proof −
    have src (tab₀ (cod μ)) ∈ Collect obj
      using μ by simp
    moreover have src μ ∈ Collect obj
      using μ by simp
    moreover have ⟦tab₀ (cod μ)⟧ ∈ Maps.Hom (src (tab₀ (cod μ))) (src μ)
    proof −
      have «tab₀ (cod μ) : src (tab₀ (cod μ)) → src μ»
        using μ by simp
      moreover have is-left-adjoint (tab₀ (cod μ))
        using μ by simp
      ultimately show ?thesis by auto
    qed
    ultimately show ?thesis by simp
  qed
  have ⟦spn (cod μ)⟧ = ⟦src (tab₀ (cod μ))⟧
    using μ spn-ide iso-class-eqI by auto
  hence SPN (cod μ) = SPN₀ (cod μ)
    unfolding SPN-def
    using μ 1 Maps.dom-char by simp
  thus Span.cod (SPN μ) = SPN (cod μ)
    using μ SPN-in-hom by auto
qed
show ⋀ν μ. seq ν μ ⟹ SPN (ν · μ) = SPN ν ⋅ SPN μ
proof −
  fix μ ν
  assume seq: seq ν μ
  have Dom (SPN (ν · μ)) = Dom (SPN ν ⋅ SPN μ)
    using seq 1 Span.vcomp-def Span.arr-char
    by (elim seqE, simp)
  moreover have Cod (SPN (ν · μ)) = Cod (SPN ν ⋅ SPN μ)
    using seq 1 Span.vcomp-def Span.arr-char
    by (elim seqE, simp)
  moreover have Chn (SPN (ν · μ)) = Chn (SPN ν ⋅ SPN μ)
  proof −
    have ∗: ⟦spn (ν · μ)⟧ = Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧
    proof
      show ⟦spn (ν · μ)⟧ ⊆ Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧
      proof
        fix h
        assume h: h ∈ ⟦spn (ν · μ)⟧
        show h ∈ Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧
        proof −
          have 1: spn ν ∈ ⟦spn ν⟧
            using seq ide-in-iso-class by auto
```

```
      moreover have 2: spn μ ∈ ⟦spn μ⟧
        using seq ide-in-iso-class by auto
      moreover have spn ν ⋆ spn μ ≅ h
      proof −
        have spn ν ⋆ spn μ ≅ spn (ν · μ)
          using seq spn-hcomp 1 2 iso-class-def isomorphic-reflexive
                isomorphic-symmetric
          by simp
        thus ?thesis
          using h isomorphic-transitive iso-class-def by simp
      qed
      ultimately show ?thesis
        unfolding Maps.Comp-def
        by (metis (mono-tags, lifting) is-iso-classI spn-simps(2)
            mem-Collect-eq seq seqE)
    qed
  qed
  show Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧ ⊆ ⟦spn (ν · μ)⟧
  proof
    fix h
    assume h: h ∈ Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧
    show h ∈ ⟦spn (ν · μ)⟧
    proof −
      obtain f g where 1: g ∈ ⟦spn ν⟧ ∧ f ∈ ⟦spn μ⟧ ∧ g ⋆ f ≅ h
        using h Maps.Comp-def [of iso-class (spn ν) iso-class (spn μ)]
              iso-class-def iso-class-elems-isomorphic
        by blast
      have fg: g ≅ spn ν ∧ f ≅ spn μ ∧ g ⋆ f ≅ h
      proof −
        have spn ν ∈ ⟦spn ν⟧ ∧ spn μ ∈ ⟦spn μ⟧
          using seq ide-in-iso-class by auto
        thus ?thesis
          using 1 iso-class-elems-isomorphic isomorphic-symmetric is-iso-classI
          by (meson spn-simps(2) seq seqE)
      qed
      have g ⋆ f ∈ ⟦spn (ν · μ)⟧
        using seq fg 1 spn-hcomp iso-class-def isomorphic-symmetric by simp
      thus ?thesis
        using fg isomorphic-transitive iso-class-def by blast
    qed
  qed
qed
have Chn (SPN ν · SPN μ) =
      Maps.MkArr (src (tab₀ (cod μ))) (src (tab₀ (cod ν))) ⟦spn ν⟧ ⊙
        Maps.MkArr (src (tab₀ (dom μ))) (src (tab₀ (cod μ))) ⟦spn μ⟧
  using 1 seq SPN-def Span.vcomp-def Span.arr-char
  apply (elim seqE)
  apply simp
  by (metis (no-types, lifting) seq vseq-implies-hpar(1) vseq-implies-hpar(2))
```

```
  also have ... = Maps.MkArr (src (tab₀ (dom μ))) (src (tab₀ (cod ν)))
                          (Maps.Comp ⟦spn ν⟧ ⟦spn μ⟧)
  proof −
    have Maps.seq (Maps.MkArr (src (tab₀ (cod μ))) (src (tab₀ (cod ν))) ⟦spn ν⟧)
                (Maps.MkArr (src (tab₀ (dom μ))) (src (tab₀ (cod μ))) ⟦spn μ⟧)
    proof
      show Maps.in-hom (Maps.MkArr (src (tab₀ (local.dom μ))) (src (tab₀ (cod μ)))
                              ⟦spn μ⟧)
                      (Maps.MkIde (src (tab₀ (dom μ))))
                      (Maps.MkIde (src (tab₀ (cod μ))))
      proof −
        have src (tab₀ (dom μ)) ∈ Collect obj
          using in-hhom-def seq by auto
        moreover have src (tab₀ (cod μ)) ∈ Collect obj
          using seq by auto
        moreover have ⟦spn μ⟧ ∈ Maps.Hom (src (tab₀ (dom μ))) (src (tab₀ (cod μ)))
          using spn-props
          by (metis (mono-tags, lifting) mem-Collect-eq seq seqE)
        ultimately show ?thesis
          using Maps.MkArr-in-hom by simp
      qed
      show Maps.in-hom (Maps.MkArr (src (tab₀ (cod μ))) (src (tab₀ (cod ν))) ⟦spn ν⟧)
                      (Maps.MkIde (src (tab₀ (cod μ))))
                      (Maps.MkIde (src (tab₀ (cod ν))))
      proof −
        have src (tab₀ (cod μ)) ∈ Collect obj
          using in-hhom-def seq by auto
        moreover have src (tab₀ (cod ν)) ∈ Collect obj
          using seq by auto
        moreover have ⟦spn ν⟧ ∈ Maps.Hom (src (tab₀ (cod μ))) (src (tab₀ (cod ν)))
          using spn-props
          by (metis (mono-tags, lifting) mem-Collect-eq seq seqE)
        ultimately show ?thesis
          using Maps.MkArr-in-hom by simp
      qed
    qed
    thus ?thesis
      using Maps.comp-char
            [of Maps.MkArr (src (tab₀ (cod μ))) (src (tab₀ (cod ν))) ⟦spn ν⟧
                Maps.MkArr (src (tab₀ (dom μ))) (src (tab₀ (cod μ))) ⟦spn μ⟧]
      by simp
  qed
  also have ... = Maps.MkArr (src (tab₀ (dom μ))) (src (tab₀ (cod ν))) ⟦spn (ν · μ)⟧
    using ∗ by simp
  also have ... = Chn (SPN (ν · μ))
    using seq SPN-def Span.vcomp-def
    by (elim seqE, simp)
  finally show ?thesis by simp
qed
```

```
    ultimately show SPN (ν · μ) = SPN ν · SPN μ by simp
  qed
qed

lemma SPN-is-functor:
shows functor V Span.vcomp SPN
  ..

interpretation SPN: weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN
proof
  show ⋀μ. arr μ ⟹ Span.isomorphic (SPN (src μ)) (Span.src (SPN μ))
  proof −
    fix μ
    assume μ: arr μ
    let ?src = Maps.MkIde (src μ)
    have src: Maps.ide ?src
      using μ by simp
    interpret src: identity-in-bicategory-of-spans V H a i src trg ‹src μ›
      using μ by unfold-locales auto
    interpret src: arrow-of-tabulations-in-maps V H a i src trg
                 ‹src μ› src.tab ‹tab₀ (src μ)› ‹tab₁ (src μ)›
                 ‹src μ› src.tab ‹tab₀ (src μ)› ‹tab₁ (src μ)›
                 ‹src μ›
      using src.is-arrow-of-tabulations-in-maps by simp
    interpret src: span-in-category Maps.comp ‹⦇Leg0 = ?src, Leg1 = ?src⦈›
      using src by (unfold-locales, simp)

    let ?tab₀ = Maps.MkArr (src (tab₀ (src μ))) (src μ) ⟦tab₀ (src μ)⟧
    have tab₀-src: «tab₀ (src μ) : src (tab₀ (src μ)) → src μ» ∧
                   is-left-adjoint (tab₀ (src μ)) ∧ ⟦tab₀ (src μ)⟧ = ⟦tab₀ (src μ)⟧
      using μ by simp
    have tab₀: Maps.arr ?tab₀
      using μ Maps.arr-MkArr tab₀-src by blast
    let ?tab₁ = Maps.MkArr (src (tab₀ (src μ))) (src μ) ⟦tab₁ (src μ)⟧
    have tab₁-src: «tab₁ (src μ) : src (tab₀ (src μ)) → src μ» ∧
                   is-left-adjoint (tab₁ (src μ)) ∧ ⟦tab₁ (src μ)⟧ = ⟦tab₁ (src μ)⟧
      using μ by simp
    have tab₁: Maps.arr ?tab₁
      using μ Maps.arr-MkArr tab₁-src by blast
    interpret tab: span-in-category Maps.comp ‹⦇Leg0 = ?tab₀, Leg1 = ?tab₁⦈›
      using tab₀ tab₁ Maps.dom-char Maps.cod-char by (unfold-locales, simp)

    have src μ ⋆ tab₀ (src μ) ≅ tab₀ (src μ)
      using μ iso-lunit isomorphic-def
      by (metis lunit-in-hom(2) src.ide-u src.u-simps(3) src-src)
    hence src μ ⋆ tab₀ (src μ) ≅ tab₁ (src μ)
      using μ src.obj-has-symmetric-tab isomorphic-transitive by blast

    have ⟦tab₀ (src μ)⟧ ∈ Maps.Hom (src (tab₀ (src μ))) (src μ)
```

```
    using μ tab₀-src by blast
  have ⟦src μ⟧ ∈ Maps.Hom (src μ) (src μ)
  proof −
    have «src μ : src μ → src μ» ∧ is-left-adjoint (src μ) ∧ ⟦src μ⟧ = ⟦src μ⟧
      using μ obj-is-self-adjoint by simp
    thus ?thesis by auto
  qed

  interpret SPN-src: arrow-of-spans Maps.comp ‹SPN (src μ)›
    using μ SPN.preserves-reflects-arr Span.arr-char by blast
  have SPN-src: SPN (src μ) =
              (|Chn = Maps.MkArr (src (tab₀ (src μ))) (src (tab₀ (src μ))) ⟦spn (src μ)⟧,
               Dom = (|Leg0 = ?tab₀, Leg1 = ?tab₁|),
               Cod = (|Leg0 = ?tab₀, Leg1 = ?tab₁|)|)
    unfolding SPN-def using μ by simp

  interpret src-SPN: arrow-of-spans Maps.comp ‹Span.src (SPN μ)›
    using μ SPN.preserves-reflects-arr Span.arr-char by blast
  have src-SPN: Span.src (SPN μ) =
              (|Chn = ?src,
               Dom = (|Leg0 = ?src, Leg1 = ?src|),
               Cod = (|Leg0 = ?src, Leg1 = ?src|)|)
  proof −
    let ?tab₀-dom = Maps.MkArr (src (tab₀ (dom μ))) (src μ) ⟦tab₀ (dom μ)⟧
    have Maps.arr ?tab₀-dom
    proof −
      have «tab₀ (dom μ) : src (tab₀ (dom μ)) → src μ» ∧
            is-left-adjoint (tab₀ (dom μ)) ∧ ⟦tab₀ (dom μ)⟧ = ⟦tab₀ (dom μ)⟧
        using μ by simp
      thus ?thesis
        using μ Maps.arr-MkArr by blast
    qed
    thus ?thesis
      using μ Maps.cod-char Span.src-def by simp
  qed
```

The idea of the proof is that $[\![tab_0\ (src\ \mu)]\!]$ is invertible in $Maps(B)$ and determines an invertible arrow of spans from $SPN\ (src\ \mu)$ to $Span.src\ (SPN\ \mu)$.

```
  let ?φ = (|Chn = ?tab₀, Dom = Dom (SPN (src μ)), Cod = Cod (Span.src (SPN μ))|)
  interpret φ: arrow-of-spans Maps.comp ?φ
    apply (unfold-locales, simp-all)
  proof −
    show Maps.in-hom ?tab₀ SPN-src.dom.apex src-SPN.cod.apex
      using μ tab₀ Maps.dom-char Maps.cod-char SPN-src src-SPN
            tab.apex-def src-SPN.cod.apex-def
      apply (intro Maps.in-homI) by simp-all
    show src-SPN.cod.leg0 ⊙ ?tab₀ = SPN-src.dom.leg0
    proof −
      have Maps.seq src-SPN.cod.leg0 ?tab₀
```

```
      using μ src-SPN tab₀ Maps.dom-char Maps.cod-char
      by (intro Maps.seqI, auto)
    moreover have Maps.Comp ⟦src μ⟧ ⟦tab₀ (src μ)⟧ = ⟦tab₀ (src μ)⟧
    proof −
      have tab₀ (src μ) ∈ Maps.Comp ⟦src μ⟧ ⟦tab₀ (src μ)⟧
        using μ is-iso-classI ide-in-iso-class [of src μ]
              ide-in-iso-class [of tab₀ (src μ)] ‹src μ ⋆ tab₀ (src μ) ≅ tab₀ (src μ)›
        by auto
      thus ?thesis
        using Maps.Comp-eq-iso-class-memb
              ‹⟦tab₀ (src μ)⟧ ∈ Maps.Hom (src (tab₀ (src μ))) (src μ)›
              ‹⟦src μ⟧ ∈ Maps.Hom (src μ) (src μ)›
        by meson
    qed
    ultimately show ?thesis
      using μ Maps.comp-char [of src-SPN.cod.leg0 ?tab₀] src-SPN by simp
  qed
  show src-SPN.cod.leg1 ⊙ ?tab₀ = SPN-src.dom.leg1
  proof −
    have Maps.seq src-SPN.cod.leg1 ?tab₀
      using μ src-SPN tab₀ Maps.dom-char Maps.cod-char
      by (intro Maps.seqI, auto)
    moreover have Maps.Comp ⟦src μ⟧ ⟦tab₀ (src μ)⟧ = ⟦tab₁ (src μ)⟧
    proof −
      have tab₁ (src μ) ∈ Maps.Comp ⟦src μ⟧ ⟦tab₀ (src μ)⟧
        using μ is-iso-classI ide-in-iso-class [of src μ]
              ide-in-iso-class [of tab₀ (src μ)]
              ‹isomorphic (src μ ⋆ tab₀ (src μ)) (tab₁ (src μ))›
        by auto
      thus ?thesis
        using Maps.Comp-eq-iso-class-memb
              ‹⟦tab₀ (src μ)⟧ ∈ Maps.Hom (src (tab₀ (src μ))) (src μ)›
              ‹⟦src μ⟧ ∈ Maps.Hom (src μ) (src μ)›
        by meson
    qed
    ultimately show ?thesis
      using μ Maps.comp-char [of src-SPN.cod.leg1 ?tab₀] src-SPN by simp
  qed
qed
have Span.in-hom ?φ (SPN (src μ)) (Span.src (SPN μ))
  using μ tab₀ spn-ide [of src μ] iso-class-eqI
        Span.arr-char Span.dom-char Span.cod-char φ.arrow-of-spans-axioms
        SPN-src src-SPN src.apex-def tab.apex-def Maps.dom-char
  apply (intro Span.in-homI) by auto

moreover have Maps.iso ?tab₀
  using μ tab₀ ide-in-iso-class src.is-map-iff-tab₀-is-equivalence obj-is-self-adjoint
        Maps.iso-char' [of ?tab₀]
  by auto
```

ultimately show *Span.isomorphic* (*SPN* (*src* μ)) (*Span.src* (*SPN* μ))
using *Span.isomorphic-def Span.iso-char* **by** *auto*
qed
show $\bigwedge\mu$. *arr* $\mu \Longrightarrow$ *Span.isomorphic* (*SPN* (*trg* μ)) (*Span.trg* (*SPN* μ))
proof −
fix μ
assume μ: *arr* μ
let *?trg* = *Maps.MkIde* (*trg* μ)
have *trg*: *Maps.ide ?trg*
using μ **by** *simp*
interpret *trg*: *identity-in-bicategory-of-spans V H* a i *src trg* ‹*trg* μ›
using μ **by** *unfold-locales auto*
interpret *trg*: *span-in-category Maps.comp* ‹⦇*Leg0* = *?trg*, *Leg1* = *?trg*⦈›
using *trg* **by** (*unfold-locales*, *simp*)

let *?tab*$_0$ = *Maps.MkArr* (*src* (*tab*$_0$ (*trg* μ))) (*trg* μ) ⟦*tab*$_0$ (*trg* μ)⟧
have *tab*$_0$*-trg*: «*tab*$_0$ (*trg* μ) : *src* (*tab*$_0$ (*trg* μ)) → *trg* μ» ∧
is-left-adjoint (*tab*$_0$ (*trg* μ)) ∧ ⟦*tab*$_0$ (*trg* μ)⟧ = ⟦*tab*$_0$ (*trg* μ)⟧
using μ **by** *simp*
have *tab*$_0$: *Maps.arr ?tab*$_0$
using μ *Maps.arr-MkArr tab*$_0$*-trg* **by** *blast*
let *?tab*$_1$ = *Maps.MkArr* (*src* (*tab*$_0$ (*trg* μ))) (*trg* μ) ⟦*tab*$_1$ (*trg* μ)⟧
have *tab*$_1$*-trg*: «*tab*$_1$ (*trg* μ) : *src* (*tab*$_0$ (*trg* μ)) → *trg* μ» ∧
is-left-adjoint (*tab*$_1$ (*trg* μ)) ∧ ⟦*tab*$_1$ (*trg* μ)⟧ = ⟦*tab*$_1$ (*trg* μ)⟧
using μ **by** *simp*
have *tab*$_1$: *Maps.arr ?tab*$_1$
using μ *Maps.arr-MkArr tab*$_1$*-trg* **by** *blast*
interpret *tab*: *span-in-category Maps.comp* ‹⦇*Leg0* = *?tab*$_0$, *Leg1* = *?tab*$_1$⦈›
using *tab*$_0$ *tab*$_1$ *Maps.dom-char Maps.cod-char* **by** (*unfold-locales*, *simp*)

have *trg* μ ⋆ *tab*$_1$ (*trg* μ) ≅ *tab*$_0$ (*trg* μ)
proof −
have «l[*tab*$_1$ (*trg* μ)] : *trg* μ ⋆ *tab*$_1$ (*trg* μ) ⇒ *tab*$_1$ (*trg* μ)»
using μ **by** *simp*
moreover have *iso* l[*tab*$_1$ (*trg* μ)]
using μ *iso-lunit* **by** *simp*
ultimately have *trg* μ ⋆ *tab*$_1$ (*trg* μ) ≅ *tab*$_1$ (*trg* μ)
using *isomorphic-def* **by** *auto*
also have *tab*$_1$ (*trg* μ) ≅ *tab*$_0$ (*trg* μ)
using μ *trg.obj-has-symmetric-tab isomorphic-symmetric* **by** *auto*
finally show *?thesis* **by** *blast*
qed
hence *trg* μ ⋆ *tab*$_1$ (*trg* μ) ≅ *tab*$_1$ (*trg* μ)
using μ *trg.obj-has-symmetric-tab isomorphic-transitive* **by** *blast*

have ⟦*tab*$_1$ (*trg* μ)⟧ ∈ *Maps.Hom* (*src* (*tab*$_0$ (*trg* μ))) (*trg* μ)
proof −
have «*tab*$_1$ (*trg* μ) : *src* (*tab*$_0$ (*trg* μ)) → *trg* μ» ∧ *is-left-adjoint* (*tab*$_0$ (*trg* μ)) ∧
⟦*tab*$_0$ (*trg* μ)⟧ = ⟦*tab*$_0$ (*trg* μ)⟧

```
    using μ by simp
  thus ?thesis by auto
qed
have ⟦trg μ⟧ ∈ Maps.Hom (trg μ) (trg μ)
proof −
  have «trg μ : trg μ → trg μ» ∧ is-left-adjoint (trg μ) ∧ ⟦trg μ⟧ = ⟦trg μ⟧
    using μ obj-is-self-adjoint by simp
  thus ?thesis by auto
qed

interpret SPN-trg: arrow-of-spans Maps.comp ‹SPN (trg μ)›
  using μ SPN.preserves-reflects-arr Span.arr-char by blast
have SPN-trg: SPN (trg μ) =
              ⦇Chn = Maps.MkArr (src (tab₁ (trg μ))) (src (tab₁ (trg μ))) ⟦spn (trg μ)⟧,
               Dom = ⦇Leg0 = ?tab₀, Leg1 = ?tab₁⦈,
               Cod = ⦇Leg0 = ?tab₀, Leg1 = ?tab₁⦈⦈
  unfolding SPN-def using μ by simp

interpret trg-SPN: arrow-of-spans Maps.comp ‹Span.trg (SPN μ)›
  using μ SPN.preserves-reflects-arr Span.arr-char by blast
have trg-SPN: Span.trg (SPN μ) = ⦇Chn = ?trg,
                                  Dom = ⦇Leg0 = ?trg, Leg1 = ?trg⦈,
                                  Cod = ⦇Leg0 = ?trg, Leg1 = ?trg⦈⦈
proof −
  let ?tab₁-dom = Maps.MkArr (src (tab₁ (dom μ))) (trg μ) ⟦tab₁ (dom μ)⟧
  have Maps.arr ?tab₁-dom
  proof −
    have «tab₁ (dom μ) : src (tab₁ (dom μ)) → trg μ» ∧
          is-left-adjoint (tab₁ (dom μ)) ∧ ⟦tab₁ (dom μ)⟧ = ⟦tab₁ (dom μ)⟧
      using μ by simp
    thus ?thesis
      using μ Maps.arr-MkArr by blast
  qed
  thus ?thesis
    using μ Maps.cod-char Span.trg-def by simp
qed

let ?φ = ⦇Chn = ?tab₁, Dom = Dom (SPN (trg μ)), Cod = Cod (Span.trg (SPN μ))⦈
interpret φ: arrow-of-spans Maps.comp ?φ
  apply (unfold-locales, simp-all)
proof −
  show Maps.in-hom ?tab₁ SPN-trg.dom.apex trg-SPN.cod.apex
    using μ tab₀ tab₁ Maps.dom-char Maps.cod-char SPN-trg trg-SPN
          tab.apex-def trg-SPN.cod.apex-def
    apply (intro Maps.in-homI) by simp-all

  show Maps.comp trg-SPN.cod.leg0 ?tab₁ = SPN-trg.dom.leg0
  proof −
    have Maps.seq trg-SPN.cod.leg0 ?tab₁
```

using μ *trg-SPN* tab_1 *Maps.dom-char Maps.cod-char*
by (*intro Maps.seqI*, *auto*)
moreover have *Maps.Comp* $[\![trg\ \mu]\!]$ $[\![tab_1\ (trg\ \mu)]\!] = [\![tab_1\ (trg\ \mu)]\!]$
proof −
have $tab_1\ (trg\ \mu) \in$ *Maps.Comp* $[\![trg\ \mu]\!]$ $[\![tab_1\ (trg\ \mu)]\!]$
using μ *is-iso-classI ide-in-iso-class* [*of trg* μ]
ide-in-iso-class [*of* $tab_1\ (trg\ \mu)$] ‹$trg\ \mu \star tab_1\ (trg\ \mu) \cong tab_1\ (trg\ \mu)$›
by *auto*
thus *?thesis*
using *Maps.Comp-eq-iso-class-memb*
‹*iso-class* $(tab_1\ (trg\ \mu)) \in$ *Maps.Hom* $(src\ (tab_0\ (trg\ \mu)))\ (trg\ \mu)$›
‹*iso-class* $(trg\ \mu) \in$ *Maps.Hom* $(trg\ \mu)\ (trg\ \mu)$›
by *meson*
qed
moreover have $[\![tab_1\ (trg\ \mu)]\!] = [\![tab_0\ (trg\ \mu)]\!]$
using μ *iso-class-eqI trg.obj-has-symmetric-tab* **by** *auto*
ultimately show *?thesis*
using μ *Maps.comp-char* [*of trg-SPN.cod.leg0* $?tab_1$] *trg-SPN*
by *simp*
qed
show *trg-SPN.cod.leg1* $\odot$ $?tab_1$ = *SPN-trg.dom.leg1*
proof −
have *Maps.seq trg-SPN.cod.leg1* $?tab_1$
using μ *trg-SPN* tab_1 *Maps.dom-char Maps.cod-char*
by (*intro Maps.seqI*, *auto*)
moreover have *Maps.Comp* $[\![trg\ \mu]\!]$ $[\![tab_1\ (trg\ \mu)]\!] = [\![tab_1\ (trg\ \mu)]\!]$
proof −
have $tab_1\ (trg\ \mu) \in$ *Maps.Comp* $[\![trg\ \mu]\!]$ $[\![tab_1\ (trg\ \mu)]\!]$
using μ *is-iso-classI ide-in-iso-class* [*of trg* μ]
ide-in-iso-class [*of* $tab_1\ (trg\ \mu)$] ‹$trg\ \mu \star tab_1\ (trg\ \mu) \cong tab_1\ (trg\ \mu)$›
by *auto*
thus *?thesis*
using *Maps.Comp-eq-iso-class-memb*
‹$[\![tab_1\ (trg\ \mu)]\!] \in$ *Maps.Hom* $(src\ (tab_0\ (trg\ \mu)))\ (trg\ \mu)$›
‹$[\![trg\ \mu]\!] \in$ *Maps.Hom* $(trg\ \mu)\ (trg\ \mu)$›
by *meson*
qed
ultimately show *?thesis*
using μ *Maps.comp-char* [*of trg-SPN.cod.leg1* $?tab_1$] *trg-SPN* **by** *simp*
qed
qed
have φ: *Span.in-hom* $?\varphi$ (*SPN* $(trg\ \mu)$) (*Span.trg* (*SPN* μ))
using μ tab_0 *spn-ide* [*of trg* μ] *iso-class-eqI*
Span.arr-char Span.dom-char Span.cod-char φ*.arrow-of-spans-axioms*
SPN-trg trg-SPN trg.apex-def tab.apex-def Maps.dom-char
apply (*intro Span.in-homI*) **by** *auto*
have *Maps.iso* $?tab_1$
proof −
have *Maps.iso* $?tab_0$

```
        using μ tab₀ ide-in-iso-class trg.is-map-iff-tab₀-is-equivalence obj-is-self-adjoint
              Maps.iso-char' [of ?tab₀]
        by auto
      moreover have ?tab₀ = ?tab₁
      proof −
        have ⟦tab₀ (trg μ)⟧ = ⟦tab₁ (trg μ)⟧
          using μ iso-class-eqI trg.obj-has-symmetric-tab by auto
        thus ?thesis by simp
      qed
      ultimately show ?thesis by simp
    qed
    thus Span.isomorphic (SPN (trg μ)) (Span.trg (SPN μ))
      using φ Span.isomorphic-def Span.iso-char by auto
  qed
qed

lemma SPN-is-weak-arrow-of-homs:
shows weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN
  ..

end
```

Compositors

To complete the proof that *SPN* is a pseudofunctor, we need to obtain a natural isomorphism Φ, whose component at (r, s) is an isomorphism $\Phi\ (r, s)$ from the horizontal composite $SPN\ r \circ SPN\ s$ to $SPN\ (r \star s)$ in $Span(Maps(B))$, and we need to prove that the coherence conditions are satisfied.

We have shown that the tabulations of r and s compose to yield a tabulation of $r \star s$. Since tabulations of the same arrow are equivalent, this tabulation must be equivalent to the chosen tabulation of $r \star s$. We therefore obtain an equivalence map from the apex of $SPN\ r \circ SPN\ s$ to the apex of $SPN\ (r \star s)$ which commutes with the legs of these spans up to isomorphism. This equivalence map determines an invertible arrow in $Span(Maps(B))$. Moreover, by property *T2*, any two such equivalence maps are connected by a unique 2-cell, which is consequently an isomorphism. This shows that the arrow in $Span(Maps(B))$ is uniquely determined, which fact we can exploit to establish the required coherence conditions.

```
locale two-composable-identities-in-bicategory-of-spans =
  bicategory-of-spans V H a i src trg +
  r: identity-in-bicategory-of-spans V H a i src trg r +
  s: identity-in-bicategory-of-spans V H a i src trg s
for V :: 'a comp                  (infixr ‹·› 55)
and H :: 'a ⇒ 'a ⇒ 'a             (infixr ‹⋆› 53)
and a :: 'a ⇒ 'a ⇒ 'a ⇒ 'a        (‹a[-, -, -]›)
and i :: 'a ⇒ 'a                  (‹i[-]›)
and src :: 'a ⇒ 'a
and trg :: 'a ⇒ 'a
```

and r :: $'a$
and s :: $'a$ +
assumes *composable*: *src* r = *trg* s
begin

notation *isomorphic* (**infix** ‹≅› *50*)

interpretation r: *arrow-in-bicategory-of-spans* V H a i *src trg* r r r
by *unfold-locales auto*
interpretation r: *arrow-of-tabulations-in-maps* V H a i *src trg*
r $r.tab$ ‹tab_0 r› ‹tab_1 r›
r $r.tab$ ‹tab_0 r› ‹tab_1 r›
r
using *r.is-arrow-of-tabulations-in-maps* **by** *simp*
interpretation s: *arrow-in-bicategory-of-spans* V H a i *src trg* s s s
by *unfold-locales auto*
interpretation s: *arrow-of-tabulations-in-maps* V H a i *src trg*
s $s.tab$ ‹tab_0 s› ‹tab_1 s›
s $s.tab$ ‹tab_0 s› ‹tab_1 s›
s
using *s.is-arrow-of-tabulations-in-maps* **by** *simp*

sublocale *identity-in-bicategory-of-spans* V H a i *src trg* ‹$r \star s$›
apply *unfold-locales* **by** (*simp add*: *composable*)
sublocale *horizontal-composite-of-arrows-of-tabulations-in-maps* V H a i *src trg*
r $r.tab$ ‹tab_0 r› ‹tab_1 r› s $s.tab$ ‹tab_0 s› ‹tab_1 s›
r $r.tab$ ‹tab_0 r› ‹tab_1 r› s $s.tab$ ‹tab_0 s› ‹tab_1 s›
r s
using *composable* **by** *unfold-locales auto*

abbreviation p_0 **where** $p_0 \equiv \varrho\sigma.p_0$
abbreviation p_1 **where** $p_1 \equiv \varrho\sigma.p_1$

We will take as the composition isomorphism from $SPN\ r \circ SPN\ s$ to $SPN\ (r \star s)$ the arrow of tabulations, induced by the identity $r \star s$, from the composite of the chosen tabulations of r and s to the chosen tabulation of $r \star s$. As this arrow of tabulations is induced by an identity, it is an equivalence map.

interpretation *cmp*: *identity-arrow-of-tabulations-in-maps* V H a i *src trg*
‹$r \star s$› $\varrho\sigma.tab$ ‹tab_0 $s \star \varrho\sigma.p_0$› ‹$tab_1$ $r \star \varrho\sigma.p_1$›
‹$r \star s$› tab ‹tab_0 $(r \star s)$› ‹tab_1 $(r \star s)$›
‹$r \star s$›
using *composable*
by *unfold-locales auto*

lemma *cmp-interpretation*:
shows *identity-arrow-of-tabulations-in-maps* V H a i *src trg*
$(r \star s)$ $\varrho\sigma.tab$ $(tab_0\ s \star \varrho\sigma.p_0)$ $(tab_1\ r \star \varrho\sigma.p_1)$
$(r \star s)$ tab $(tab_0\ (r \star s))$ $(tab_1\ (r \star s))$
$(r \star s)$

..

definition *cmp*
where *cmp* = *cmp.chine*

lemma *cmp-props*:
shows «*cmp* : *src* $\varrho\sigma$.*tab* → *src tab*»
and «*cmp* : *cmp* ⇒ *cmp*»
and *equivalence-map cmp*
and tab_0 (*r* ⋆ *s*) ⋆ *cmp* ≅ tab_0 *s* ⋆ $\varrho\sigma.p_0$
and tab_1 (*r* ⋆ *s*) ⋆ *cmp* ≅ tab_1 *r* ⋆ $\varrho\sigma.p_1$
using *cmp-def cmp.leg0-uniquely-isomorphic*(*1*) *cmp.leg1-uniquely-isomorphic*(*1*) *isomorphic-symmetric cmp.chine-is-equivalence*
by *auto*

lemma *cmp-in-hom* [*intro*]:
shows «*cmp* : *src* $\varrho\sigma$.*tab* → *src tab*»
and «*cmp* : *cmp* ⇒ *cmp*»
using *cmp-props* **by** *auto*

lemma *cmp-simps* [*simp*]:
shows *arr cmp* **and** *ide cmp*
and *src cmp* = *src* $\varrho\sigma$.*tab* **and** *trg cmp* = *src tab*
and *dom cmp* = *cmp* **and** *cod cmp* = *cmp*
using *cmp-props equivalence-map-is-ide* **by** *auto*

Now we have to use the above properties of the underlying bicategory to exhibit the composition isomorphisms as actual arrows in *Span*(*Maps*(*B*)) and to prove the required naturality and coherence conditions.

interpretation *Maps*: *maps-category V H* a i *src trg* **..**
interpretation *Span*: *span-bicategory Maps.comp* $Maps.PRJ_0$ $Maps.PRJ_1$ **..**

no-notation *Fun.comp* (**infixl** ‹∘› *55*)
notation *Span.vcomp* (**infixr** ‹·› *55*)
notation *Span.hcomp* (**infixr** ‹∘› *53*)
notation *Maps.comp* (**infixr** ‹⊙› *55*)

interpretation *SPN*: *functor V Span.vcomp SPN*
using *SPN-is-functor* **by** *simp*
interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
using *SPN-is-weak-arrow-of-homs* **by** *simp*

interpretation *SPN-r-SPN-s*: *arrow-of-spans Maps.comp* ‹*SPN r* ∘ *SPN s*›
using *composable Span.ide-char* [*of SPN r* ∘ *SPN s*] **by** *simp*
interpretation *SPN-r-SPN-s*: *identity-arrow-of-spans Maps.comp* ‹*SPN r* ∘ *SPN s*›
using *composable Span.ide-char* [*of SPN r* ∘ *SPN s*]
by (*unfold-locales*, *simp*)
interpretation *SPN-rs*: *arrow-of-spans Maps.comp* ‹*SPN* (*r* ⋆ *s*)›
using *composable Span.arr-char r.base-simps*(*2*) *s.base-simps*(*2*) **by** *blast*

interpretation *SPN-rs*: *identity-arrow-of-spans Maps.comp* ‹*SPN* $(r \star s)$›
using *composable Span.ide-char SPN.preserves-ide r.is-ide s.is-ide*
by (*unfold-locales*, *simp*)

The following are the legs (as arrows of *Maps*) of the spans *SPN r* and *SPN s*.

definition R_0 **where** $R_0 = [\![[\![tab_0\ r]\!]]\!]$
definition R_1 **where** $R_1 = [\![[\![tab_1\ r]\!]]\!]$
definition S_0 **where** $S_0 = [\![[\![tab_0\ s]\!]]\!]$
definition S_1 **where** $S_1 = [\![[\![tab_1\ s]\!]]\!]$

lemma *span-legs-eq*:
shows *Leg0* (*Dom* (*SPN r*)) $= R_0$ **and** *Leg1* (*Dom* (*SPN r*)) $= R_1$
and *Leg0* (*Dom* (*SPN s*)) $= S_0$ **and** *Leg1* (*Dom* (*SPN s*)) $= S_1$
using *SPN-def* R_0*-def* R_1*-def* S_0*-def* S_1*-def composable* **by** *auto*

lemma R_0*-in-hom* [*intro*]:
shows *Maps.in-hom* R_0 (*Maps.MkIde* (*src* $r.s_0$)) (*Maps.MkIde* (*src r*))
by (*simp add*: *Maps.MkArr-in-hom′* R_0*-def*)

lemma R_1*-in-hom* [*intro*]:
shows *Maps.in-hom* R_1 (*Maps.MkIde* (*src* $r.s_0$)) (*Maps.MkIde* (*trg r*))
by (*simp add*: *Maps.MkArr-in-hom′* R_1*-def*)

lemma S_0*-in-hom* [*intro*]:
shows *Maps.in-hom* S_0 (*Maps.MkIde* (*src* $s.s_0$)) (*Maps.MkIde* (*src s*))
by (*simp add*: *Maps.MkArr-in-hom′* S_0*-def*)

lemma S_1*-in-hom* [*intro*]:
shows *Maps.in-hom* S_1 (*Maps.MkIde* (*src* $s.s_0$)) (*Maps.MkIde* (*trg s*))
by (*simp add*: *Maps.MkArr-in-hom′* S_1*-def*)

lemma *RS-simps* [*simp*]:
shows *Maps.arr* R_0 **and** *Maps.dom* R_0 = *Maps.MkIde* (*src* $r.s_0$)
and *Maps.cod* R_0 = *Maps.MkIde* (*src r*)
and *Maps.Dom* R_0 = *src* $r.s_0$ **and** *Maps.Cod* R_0 = *src r*
and *Maps.arr* R_1 **and** *Maps.dom* R_1 = *Maps.MkIde* (*src* $r.s_0$)
and *Maps.cod* R_1 = *Maps.MkIde* (*trg r*)
and *Maps.Dom* R_1 = *src* $r.s_0$ **and** *Maps.Cod* R_1 = *trg r*
and *Maps.arr* S_0 **and** *Maps.dom* S_0 = *Maps.MkIde* (*src* $s.s_0$)
and *Maps.cod* S_0 = *Maps.MkIde* (*src s*)
and *Maps.Dom* S_0 = *src* $s.s_0$ **and** *Maps.Cod* S_0 = *src s*
and *Maps.arr* S_1 **and** *Maps.dom* S_1 = *Maps.MkIde* (*src* $s.s_0$)
and *Maps.cod* S_1 = *Maps.MkIde* (*trg s*)
and *Maps.Dom* S_1 = *src* $s.s_0$ **and** *Maps.Cod* S_1 = *trg s*
using R_0*-in-hom* R_1*-in-hom* S_0*-in-hom* S_1*-in-hom composable*
by (*auto simp add*: R_0*-def* R_1*-def* S_0*-def* S_1*-def*)

The apex of the composite span *SPN r* ∘ *SPN s* (defined in terms of pullback) coincides with the apex of the composite tabulation $\varrho\sigma$ (defined using the chosen tabulation of $(tab_0\ r)^* \star tab_1\ s)$). We need this to be true in order to define the compositor of a

pseudofunctor from the underlying bicategory B to $Span(Maps(B))$. It is only true if we have carefully chosen pullbacks in $Maps(B)$ in order to ensure the relationship with the chosen tabulations.

```
lemma SPN-r-SPN-s-apex-eq:
shows SPN-r-SPN-s.apex = Maps.MkIde (src ϱσ.tab)
proof −
  have obj (Maps.Cod SPN-r-SPN-s.leg0)
    using Maps.arr-char [of SPN-r-SPN-s.leg0] by simp
  moreover have obj (Maps.Dom SPN-r-SPN-s.leg0)
    using Maps.arr-char [of SPN-r-SPN-s.leg0] by simp
  moreover have SPN-r-SPN-s.leg0 ≠ Maps.Null
    using Maps.arr-char [of SPN-r-SPN-s.leg0] by simp
  moreover have Maps.Dom SPN-r-SPN-s.leg0 = src ϱσ.tab
  proof −
    interpret REP-S1: map-in-bicategory V H a i src trg ‹Maps.REP S1›
      using Maps.REP-in-Map Maps.arr-char Maps.in-HomD S1-def
      apply unfold-locales
      by (meson Maps.REP-in-hhom(2) S1-in-hom)
    interpret REP-R0: map-in-bicategory V H a i src trg ‹Maps.REP R0›
      using Maps.REP-in-Map Maps.arr-char Maps.in-HomD R0-def
      apply unfold-locales
      by (meson Maps.REP-in-hhom(2) R0-in-hom)
    have Maps.Dom SPN-r-SPN-s.leg0 = Maps.Dom (S0 ⊙ Maps.PRJ0 R0 S1)
      using composable Span.hcomp-def S0-def R0-def S1-def by simp
    also have ... = Maps.Dom ⟦⟦tab0 ((Maps.REP R0)* ⋆ Maps.REP S1)⟧⟧
    proof −
      have is-left-adjoint (tab0 ((Maps.REP R0)* ⋆ Maps.REP S1))
      proof −
        have ide ((Maps.REP R0)* ⋆ Maps.REP S1)
        proof −
          have src (Maps.REP R0)* = trg (Maps.REP S1)
            using REP-R0.is-map REP-S1.is-map left-adjoint-is-ide R0-def S1-def
            by (metis (no-types, lifting) Maps.REP-CLS REP-R0.antipar(2)
                isomorphic-implies-hpar(4) composable r.leg0-simps(3)
                r.satisfies-T0 s.leg1-is-map s.leg1-simps(3) s.leg1-simps(4))
          thus ?thesis by simp
        qed
        thus ?thesis by simp
      qed
      moreover have Maps.Dom (S0 ⊙ ⟦⟦tab0 ((Maps.REP R0)* ⋆ Maps.REP S1)⟧⟧) =
                    src (tab0 ((Maps.REP R0)* ⋆ Maps.REP S1))
      proof −
        have Maps.arr (⟦⟦prj0 (Maps.REP S1) (Maps.REP R0)⟧⟧)
         using Maps.CLS-in-hom Maps.prj0-simps(1) Maps.PRJ0-def composable by fastforce
        moreover have Maps.Dom S0 = Maps.Cod ⟦⟦prj0 (Maps.REP S1) (Maps.REP R0)⟧⟧
        proof −
          have Maps.Cod ⟦⟦prj0 (Maps.REP S1) (Maps.REP R0)⟧⟧ =
                trg (prj0 (Maps.REP S1) (Maps.REP R0))
            by simp
```

```
      also have ... = src (Maps.REP S1)
      proof −
        have ide (Maps.REP S1)
          by simp
        moreover have is-left-adjoint (Maps.REP R0)
          by auto
        moreover have trg (Maps.REP S1) = trg (Maps.REP R0)
          by (simp add: composable)
        ultimately show ?thesis
          using S1-def Maps.REP-CLS r.leg0-is-map s.leg1-is-map by simp
      qed
      also have ... = src (tab0 s)
        using tab0-in-hom(1) by simp
      also have ... = Maps.Dom S0
        using S0-def by simp
      finally show ?thesis by simp
    qed
    ultimately have
      Maps.Dom (S0 ⊙ ⟦⟦tab0 ((Maps.REP R0)* ⋆ Maps.REP S1)⟧⟧) =
       Maps.Dom ⟦⟦tab0 ((Maps.REP R0)* ⋆ Maps.REP S1)⟧⟧
      using Maps.CLS-in-hom by simp
    thus ?thesis by simp
  qed
  ultimately show ?thesis
    using Maps.PRJ0-def composable Maps.Dom.simps(1) RS-simps(1) RS-simps(16)
          RS-simps(18) RS-simps(3)
    by presburger
qed
also have ... = src (tab0 ((Maps.REP R0)* ⋆ Maps.REP S1))
  by simp
finally have
  Maps.Dom SPN-r-SPN-s.leg0 = src (tab0 ((Maps.REP R0)* ⋆ Maps.REP S1))
  by simp
also have ... = src (tab0 (r.s0* ⋆ s.s1))
proof −
  interpret r0's1: identity-in-bicategory-of-spans V H a i src trg ‹r.s0* ⋆ s.s1›
    using composable by (unfold-locales, simp)
  have (Maps.REP R0)* ⋆ Maps.REP S1 ≅ r.s0* ⋆ s.s1
  proof −
    have (Maps.REP R0)* ≅ r.s0*
    proof −
      have 1: adjoint-pair (Maps.REP R0) (Maps.REP R0)*
        using REP-R0.is-map left-adjoint-extends-to-adjoint-pair by blast
      moreover have adjoint-pair r.s0 (Maps.REP R0)*
      proof −
        have Maps.REP R0 ≅ r.s0
          unfolding R0-def
          using Maps.REP-CLS r.leg0-is-map composable by force
        thus ?thesis
```

```
            using 1 adjoint-pair-preserved-by-iso isomorphic-def
                REP-R0.triangle-in-hom(4) REP-R0.triangle-right'
            by auto
          qed
          ultimately show ?thesis
            using r.leg0-is-map left-adjoint-determines-right-up-to-iso
                left-adjoint-extends-to-adjoint-pair
            by auto
        qed
        moreover have Maps.REP S1 ≅ s.s1
          unfolding S1-def
          using Maps.REP-CLS s.leg1-is-map composable by force
        moreover have ∃ a. a ≅ (tab0 r)* ⋆ tab1 s ∧ (Maps.REP R0)* ⋆ Maps.REP S1 ≅ a
          using calculation composable isomorphic-implies-hpar(3)
              hcomp-ide-isomorphic hcomp-isomorphic-ide [of (Maps.REP R0)* r.s0* s.s1]
          by auto
        ultimately show ?thesis
          using isomorphic-transitive by blast
      qed
      thus ?thesis
        using r0's1.isomorphic-implies-same-tab isomorphic-symmetric by metis
    qed
    also have ... = src ϱσ.tab
      using VV.ide-charSbC VV.arr-charSbC composable Span.hcomp-def ϱσ.tab-simps(2)
by auto
    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    using composable Maps.arr-char Maps.dom-char SPN-r-SPN-s.dom.apex-def
    apply auto
    by (metis (no-types, lifting) Maps.not-arr-null SPN-r-SPN-s.chine-eq-apex
        SPN-r-SPN-s.chine-simps(1))
qed
```

We will be taking the arrow *CLS cmp* of *Maps* as the composition isomorphism from *SPN r* ∘ *SPN s* to *SPN* (*r* ⋆ *s*). The following result shows that it has the right domain and codomain for that purpose.

```
lemma iso-class-cmp-in-hom:
shows Maps.in-hom (Maps.MkArr (src ϱσ.tab) (src tab) ⟦cmp⟧)
              SPN-r-SPN-s.apex SPN-rs.apex
and Maps.in-hom ⟦⟦cmp⟧⟧ SPN-r-SPN-s.apex SPN-rs.apex
proof −
  show Maps.in-hom (Maps.MkArr (src ϱσ.tab) (src tab) ⟦cmp⟧)
              SPN-r-SPN-s.apex SPN-rs.apex
  proof −
    have obj (src ϱσ.tab)
      using obj-src ϱσ.tab-in-hom by blast
    moreover have obj (src tab)
      using obj-src by simp
```

moreover have $[\![cmp]\!] \in$ *Maps.Hom* (*src* $\varrho\sigma$.*tab*) (*src tab*)
by (*metis* (*mono-tags*, *lifting*) *cmp.is-map cmp-def cmp-props*(*1*) *mem-Collect-eq*)
moreover have *SPN-r-SPN-s.apex* = *Maps.MkIde* (*src* $\varrho\sigma$.*tab*)
using *SPN-r-SPN-s-apex-eq* **by** *simp*
moreover have *SPN-rs.apex* = *Maps.MkIde* (*src tab*)
using *SPN-def composable SPN-rs.cod.apex-def Maps.arr-char Maps.dom-char SPN-rs.dom.leg-simps*(*1*)
by *fastforce*
ultimately show *?thesis*
using *Maps.MkArr-in-hom* **by** *simp*
qed
thus *Maps.in-hom* $[\![[\![cmp]\!]]\!]$ *SPN-r-SPN-s.apex SPN-rs.apex* **by** *simp*
qed

interpretation $r_0{}'s_1$: *two-composable-identities-in-bicategory-of-spans* *V H* a i *src trg* ‹(*Maps.REP* R_0)*› ‹*Maps.REP* S_1›
proof
show *ide* (*Maps.REP* S_1)
using *Maps.REP-in-Map Maps.arr-char left-adjoint-is-ide*
by (*meson Maps.REP-in-hhom*(*2*) S_1*-in-hom*)
show *ide* (*Maps.REP* R_0)*
using *Maps.REP-in-Map Maps.arr-char left-adjoint-is-ide Maps.REP-in-hhom*(*2*) R_0*-in-hom* **by** *auto*
show *src* (*Maps.REP* R_0)* = *trg* (*Maps.REP* S_1)
using *Maps.REP-in-hhom*(*2*) R_0*-in-hom composable* **by** *auto*
qed

interpretation $R_0{}'S_1$: *identity-in-bicategory-of-spans V H* a i *src trg* ‹(tab_0 *r*)* $\star$ tab_1 *s*›
by (*unfold-locales*, *simp add*: *composable*)

lemma *prj-tab-agreement*:
shows (tab_0 *r*)* $\star$ tab_1 *s* $\cong$ (*Maps.REP* R_0)* $\star$ *Maps.REP* S_1
and $\varrho\sigma.p_0 \cong prj_0$ (*Maps.REP* S_1) (*Maps.REP* R_0)
and $\varrho\sigma.p_1 \cong prj_1$ (*Maps.REP* S_1) (*Maps.REP* R_0)
proof −
show *1*: (tab_0 *r*)* $\star$ tab_1 *s* $\cong$ (*Maps.REP* R_0)* $\star$ *Maps.REP* S_1
proof −
have (tab_0 *r*)* $\cong$ (*Maps.REP* R_0)*
using *Maps.REP-CLS isomorphic-symmetric* R_0*-def composable r.satisfies-T0 isomorphic-to-left-adjoint-implies-isomorphic-right-adjoint*
by *fastforce*
moreover have tab_1 *s* $\cong$ *Maps.REP* S_1
by (*metis Maps.REP-CLS isomorphic-symmetric* S_1*-def s.leg1-is-map s.leg1-simps*(*3*−*4*))
moreover have *src* (*Maps.REP* R_0)* = *trg* (tab_1 *s*)
using *composable r.T0.antipar right-adjoint-simps*(*2*) **by** *fastforce*
ultimately show *?thesis*
using *hcomp-isomorphic-ide* [*of* (tab_0 *r*)* (*Maps.REP* R_0)* tab_1 *s*] *hcomp-ide-isomorphic isomorphic-transitive composable*
by *auto*

```
    qed
    show ϱσ.p₀ ≅ tab₀ ((Maps.REP R₀)* ⋆ Maps.REP S₁)
      using 1 R₀'S₁.isomorphic-implies-same-tab isomorphic-reflexive by auto
    show ϱσ.p₁ ≅ tab₁ ((Maps.REP R₀)* ⋆ Maps.REP S₁)
      using 1 R₀'S₁.isomorphic-implies-same-tab isomorphic-reflexive by auto
  qed

  lemma chine-hcomp-SPN-SPN:
  shows Span.chine-hcomp (SPN r) (SPN s) = Maps.MkIde (src ϱσ.p₀)
  proof −
    have Span.chine-hcomp (SPN r) (SPN s) =
          Maps.MkIde (src (tab₀ ((Maps.REP R₀)* ⋆ Maps.REP S₁)))
      using Span.chine-hcomp-ide-ide [of SPN r SPN s] composable
            Maps.pbdom-def Maps.PRJ₀-def Maps.CLS-in-hom Maps.dom-char R₀-def S₁-def
      apply simp
      using Maps.prj0-simps(1) RS-simps(1) RS-simps(16) RS-simps(18) RS-simps(3)
      by presburger
    also have ... = Maps.MkIde (src ϱσ.p₀)
      using prj-tab-agreement isomorphic-implies-hpar(3) by force
    finally show ?thesis by simp
  qed

end
```

The development above focused on two specific composable 1-cells in bicategory B. Here we reformulate those results as statements about the entire bicategory.

```
context bicategory-of-spans
begin

  interpretation Maps: maps-category V H a i src trg ..
  interpretation Span: span-bicategory Maps.comp Maps.PRJ₀ Maps.PRJ₁ ..

  no-notation Fun.comp (infixl ‹∘› 55)
  notation Span.vcomp (infixr ‹•› 55)
  notation Span.hcomp (infixr ‹∘› 53)
  notation Maps.comp (infixr ‹⊙› 55)
  notation isomorphic (infix ‹≅› 50)

  interpretation SPN: functor V Span.vcomp SPN
    using SPN-is-functor by simp
  interpretation SPN: weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN
    using SPN-is-weak-arrow-of-homs by simp

  interpretation HoSPN-SPN: composite-functor VV.comp Span.VV.comp Span.vcomp
                            SPN.FF ‹λμν. fst μν ∘ snd μν›
    ..
  interpretation SPNoH: composite-functor VV.comp V
                        Span.vcomp ‹λμν. fst μν ⋆ snd μν› SPN
    ..
```

Given arbitrary composable 1-cells r and s, obtain an arrow of spans in *Maps* having the isomorphism class of *rs.cmp* as its chine.

definition *CMP*
where *CMP r s* ≡
  ⦇*Chn* = ⟦⟦*two-composable-identities-in-bicategory-of-spans.cmp V H* a i *src trg r s*⟧⟧,
   Dom = *Dom* (*SPN r* ∘ *SPN s*), *Cod* = *Dom* (*SPN* (*r* ⋆ *s*))⦈

lemma *compositor-in-hom* [*intro*]:
assumes *ide r* **and** *ide s* **and** *src r* = *trg s*
shows *Span.in-hhom* (*CMP r s*) (*SPN.map*$_0$ (*src s*)) (*SPN.map*$_0$ (*trg r*))
and *Span.in-hom* (*CMP r s*) (*HoSPN-SPN.map* (*r*, *s*)) (*SPNoH.map* (*r*, *s*))
proof −
  have *rs*: *VV.ide* (*r*, *s*)
    using *assms VV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *simp*
  interpret *rs*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg r s*
    using *rs VV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ **by** *unfold-locales auto*
  interpret *cmp*: *arrow-of-tabulations-in-maps V H* a i *src trg*
      ‹*r* ⋆ *s*› *rs.ϱσ.tab* ‹*tab*$_0$ *s* ⋆ *rs.ϱσ.p*$_0$› ‹*tab*$_1$ *r* ⋆ *rs.ϱσ.p*$_1$›
      ‹*r* ⋆ *s*› *rs.tab* ‹*tab*$_0$ (*r* ⋆ *s*)› ‹*tab*$_1$ (*r* ⋆ *s*)›
      ‹*r* ⋆ *s*›
    by *unfold-locales auto*
  have *rs.cmp* = *cmp.chine*
    using *rs.cmp-def* **by** *simp*

  interpret *SPN-r-SPN-s*: *arrow-of-spans Maps.comp* ‹*SPN r* ∘ *SPN s*›
    using *rs.composable Span.ide-char* [*of SPN r* ∘ *SPN s*] **by** *simp*
  interpret *SPN-r-SPN-s*: *identity-arrow-of-spans Maps.comp* ‹*SPN r* ∘ *SPN s*›
    using *rs.composable Span.ide-char* [*of SPN r* ∘ *SPN s*]
    by (*unfold-locales*, *simp*)
  interpret *SPN-rs*: *arrow-of-spans Maps.comp* ‹*SPN* (*r* ⋆ *s*)›
    using *Span.arr-char rs.is-ide SPN.preserves-arr* **by** *blast*
  interpret *SPN-rs*: *identity-arrow-of-spans Maps.comp* ‹*SPN* (*r* ⋆ *s*)›
    using *Span.ide-char rs.is-ide SPN.preserves-ide*
    by (*unfold-locales*, *simp*)

  interpret *Dom*: *span-in-category Maps.comp* ‹*Dom* (*CMP r s*)›
    by (*unfold-locales*, *simp add*: *CMP-def*)
  interpret *Cod*: *span-in-category Maps.comp* ‹*Cod* (*CMP r s*)›
  proof −

    have «*tab*$_0$ (*r* ⋆ *s*) : *src* (*tab*$_0$ (*r* ⋆ *s*)) → *src s*» ∧ *is-left-adjoint* (*tab*$_0$ (*r* ⋆ *s*)) ∧
        ⟦*tab*$_0$ (*r* ⋆ *s*)⟧ = ⟦*tab*$_0$ (*r* ⋆ *s*)⟧
      by *simp*
    hence ∃*f*. «*f* : *src* (*tab*$_0$ (*r* ⋆ *s*)) → *src s*» ∧ *is-left-adjoint f* ∧ ⟦*tab*$_0$ (*r* ⋆ *s*)⟧ = ⟦*f*⟧
      by *blast*
    moreover have ∃*f*. «*f* : *src* (*tab*$_0$ (*r* ⋆ *s*)) → *trg r*» ∧ *is-left-adjoint f* ∧
                ⟦*tab*$_1$ (*r* ⋆ *s*)⟧ = ⟦*f*⟧
      by (*metis rs.base-simps*(*2*) *rs.leg1-in-hom*(*1*) *rs.leg1-is-map trg-hcomp*)
    ultimately show *span-in-category Maps.comp* (*Cod* (*CMP r s*))

```
    using assms Maps.arr-char Maps.dom-char CMP-def
    by unfold-locales auto
qed

interpret r0's1: two-composable-identities-in-bicategory-of-spans
                V H a i src trg ‹(Maps.REP rs.R0)*› ‹Maps.REP rs.S1›
proof
  show ide (Maps.REP rs.S1)
    using Maps.REP-in-Map Maps.arr-char left-adjoint-is-ide
    by (meson Maps.REP-in-hhom(2) rs.S1-in-hom)
  show ide (Maps.REP rs.R0)*
    using Maps.REP-in-Map Maps.arr-char left-adjoint-is-ide
          Maps.REP-in-hhom(2) rs.R0-in-hom by auto
  show src (Maps.REP rs.R0)* = trg (Maps.REP rs.S1)
    using Maps.REP-in-hhom(2) rs.R0-in-hom rs.composable by auto
qed

interpret R0'S1: identity-in-bicategory-of-spans V H a i src trg ‹(tab0 r)* ⋆ tab1 s›
  by (unfold-locales, simp add: rs.composable)
```

Here we obtain explicit formulas for the legs and apex of *SPN-r-SPN-s* and *SPN-rs*.

```
have SPN-r-SPN-s-leg0-eq:
      SPN-r-SPN-s.leg0 = Maps.comp rs.S0 (Maps.PRJ0 rs.R0 rs.S1)
  using rs.composable Span.hcomp-def rs.S0-def rs.R0-def rs.S1-def by simp
have SPN-r-SPN-s-leg1-eq:
      SPN-r-SPN-s.leg1 = Maps.comp rs.R1 (Maps.PRJ1 rs.R0 rs.S1)
  using rs.composable Span.hcomp-def rs.R1-def rs.R0-def rs.S1-def by simp
have SPN-r-SPN-s.apex = Maps.MkIde (src rs.ϱσ.tab)
  using rs.SPN-r-SPN-s-apex-eq by auto

have SPN-rs-leg0-eq: SPN-rs.leg0 = ⟦⟦tab0 (r ⋆ s)⟧⟧
  unfolding SPN-def using rs by simp
have SPN-rs-leg1-eq: SPN-rs.leg1 = ⟦⟦tab1 (r ⋆ s)⟧⟧
  unfolding SPN-def using rs by simp
have SPN-rs.apex = Maps.MkIde (src (tab-of-ide (r ⋆ s)))
  using SPN-rs.dom.apex-def Maps.dom-char SPN-rs-leg0-eq SPN-rs.dom.leg-simps(1)
  by simp
```

The composition isomorphism *CMP r s* is an arrow of spans in *Maps*(*B*).

```
interpret arrow-of-spans Maps.comp ‹CMP r s›
proof
  show 1: Maps.in-hom (Chn (CMP r s)) Dom.apex Cod.apex
    using rs.iso-class-cmp-in-hom rs.composable CMP-def by simp
  show Cod.leg0 ⊙ Chn (CMP r s) = Dom.leg0
  proof (intro Maps.arr-eqI)
    show 2: Maps.seq Cod.leg0 (Chn (CMP r s))
      using 1 Maps.dom-char Maps.cod-char by blast
    show 3: Maps.arr Dom.leg0 by simp
    show Maps.Dom (Cod.leg0 ⊙ Chn (CMP r s)) = Maps.Dom Dom.leg0
```

```
    using 1 2 Maps.dom-char Maps.cod-char Maps.comp-char
          Dom.leg-in-hom Maps.in-hom-char Maps.seq-char
    by auto
  show Maps.Cod (Cod.leg0 ⊙ Chn (CMP r s)) = Maps.Cod Dom.leg0
    using 2 3 Maps.comp-char [of Cod.leg0 Chn (CMP r s)]
          Dom.leg-simps Dom.apex-def Maps.dom-char SPN-r-SPN-s-leg0-eq
          Maps.comp-char [of rs.S0 Maps.PRJ0 rs.R0 rs.S1] CMP-def
    by simp
  show Maps.Map (Cod.leg0 ⊙ Chn (CMP r s)) = Maps.Map Dom.leg0
  proof −
    have Maps.Map (Cod.leg0 ⊙ Chn (CMP r s)) = Maps.Comp ⟦tab0 (r ⋆ s)⟧ ⟦rs.cmp⟧
      using 1 2 Maps.dom-char Maps.cod-char
            Maps.comp-char [of Cod.leg0 Chn (CMP r s)] CMP-def
      by simp
    also have ... = Maps.Comp ⟦tab0 s⟧ ⟦rs.ϱσ.p0⟧
    proof −
      have Maps.Comp ⟦tab0 (r ⋆ s)⟧ ⟦rs.cmp⟧ = ⟦tab0 (r ⋆ s) ⋆ rs.cmp⟧
        using Maps.Comp-eq-iso-class-memb Maps.CLS-hcomp cmp.is-map rs.cmp-def
        by auto
      also have ... = Maps.Comp ⟦tab0 s⟧ ⟦rs.ϱσ.p0⟧
      using Maps.Comp-eq-iso-class-memb Maps.CLS-hcomp iso-class-eqI rs.cmp-props(4)
        by auto
      finally show ?thesis by simp
    qed
    also have ... = Maps.Map Dom.leg0
    proof −
      have Maps.seq rs.S0 (Maps.PRJ0 rs.R0 rs.S1)
        by (intro Maps.seqI, simp-all add: rs.composable)
      moreover have ⟦prj0 (Maps.REP rs.S1) (Maps.REP rs.R0)⟧ = ⟦rs.ϱσ.p0⟧
        using rs.prj-tab-agreement iso-class-eqI by auto
      moreover have Maps.Dom (Maps.PRJ0 rs.R0 rs.S1) = src rs.ϱσ.p0
        using rs.prj-tab-agreement Maps.PRJ0-def rs.composable
              isomorphic-implies-hpar(3)
        by auto
      ultimately show ?thesis
        using SPN-r-SPN-s-leg0-eq Maps.comp-char [of rs.S0 Maps.PRJ0 rs.R0 rs.S1]
              rs.S0-def Maps.PRJ0-def rs.composable CMP-def
        by simp
    qed
    finally show ?thesis by simp
  qed
qed
show Cod.leg1 ⊙ Chn (CMP r s) = Dom.leg1
proof (intro Maps.arr-eqI)
  show 2: Maps.seq Cod.leg1 (Chn (CMP r s))
    using 1 Maps.dom-char Maps.cod-char by blast
  show 3: Maps.arr Dom.leg1 by simp
  show Maps.Dom (Cod.leg1 ⊙ Chn (CMP r s)) = Maps.Dom Dom.leg1
    using 1 2 Maps.dom-char Maps.cod-char Maps.comp-char
```

Dom.leg-in-hom Maps.in-hom-char Maps.seq-char
by *auto*
show *Maps.Cod* (*Cod.leg1* $\odot$ *Chn* (*CMP r s*)) = *Maps.Cod Dom.leg1*
using *2 3 Maps.comp-char* [*of Cod.leg1 Chn* (*CMP r s*)]
Dom.apex-def Maps.dom-char SPN-r-SPN-s-leg1-eq
Maps.comp-char [*of* $rs.R_1$ $Maps.PRJ_1$ $rs.R_0$ $rs.S_1$] *CMP-def*
by *simp*
show *Maps.Map* (*Cod.leg1* $\odot$ *Chn* (*CMP r s*)) = *Maps.Map Dom.leg1*
proof −
have *Maps.Map* (*Cod.leg1* $\odot$ *Chn* (*CMP r s*)) = *Maps.Comp* $[\![tab_1\ (r \star s)]\!]$ $[\![rs.cmp]\!]$
using *1 2 Maps.dom-char Maps.cod-char*
Maps.comp-char [*of Cod.leg1 Chn* (*CMP r s*)] *CMP-def*
by *simp*
also have ... = *Maps.Comp* $[\![tab_1\ r]\!]$ $[\![rs.\varrho\sigma.p_1]\!]$
proof −
have *Maps.Comp* $[\![tab_1\ (r \star s)]\!]$ $[\![rs.cmp]\!]$ = $[\![tab_1\ (r \star s) \star rs.cmp]\!]$
using *Maps.Comp-eq-iso-class-memb Maps.CLS-hcomp cmp.is-map rs.cmp-def*
by *auto*
also have ... = *Maps.Comp* $[\![tab_1\ r]\!]$ $[\![rs.\varrho\sigma.p_1]\!]$
using *Maps.Comp-eq-iso-class-memb*
Maps.CLS-hcomp [*of* tab_1 *r* $rs.\varrho\sigma.p_1$] *iso-class-eqI rs.cmp-props*(*5*)
by *auto*
finally show *?thesis* **by** *simp*
qed
also have ... = *Maps.Map Dom.leg1*
proof −
have *Maps.seq* $rs.R_1$ ($Maps.PRJ_1$ $rs.R_0$ $rs.S_1$)
by (*intro Maps.seqI*, *simp-all add*: *rs.composable*)
moreover have $[\![prj_1\ (Maps.REP\ rs.S_1)\ (Maps.REP\ rs.R_0)]\!]$ = $[\![rs.\varrho\sigma.p_1]\!]$
using *rs.prj-tab-agreement iso-class-eqI* **by** *auto*
moreover have *Maps.Dom* ($Maps.PRJ_1$ $rs.R_0$ $rs.S_1$) = *src* $rs.\varrho\sigma.p_1$
using *rs.prj-tab-agreement* $Maps.PRJ_1$*-def rs.composable*
isomorphic-implies-hpar(*3*)
by *auto*
ultimately show *?thesis*
using *SPN-r-SPN-s-leg1-eq Maps.comp-char* [*of* $rs.R_1$ $Maps.PRJ_1$ $rs.R_0$ $rs.S_1$]
$rs.R_1$*-def* $Maps.PRJ_1$*-def rs.composable CMP-def*
by *simp*
qed
finally show *?thesis* **by** *simp*

qed
qed
qed
show *Span.in-hom* (*CMP r s*) (*HoSPN-SPN.map* (*r*, *s*)) (*SPNoH.map* (*r*, *s*))
using *Span.arr-char arrow-of-spans-axioms Span.dom-char Span.cod-char*
CMP-def SPN.FF-def $VV.arr\text{-}char_{SbC}$ *rs.composable*
by *auto*
thus *Span.in-hhom* (*CMP r s*) ($SPN.map_0$ (*src s*)) ($SPN.map_0$ (*trg r*))

using *assms* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ $VV.in\text{-}hom\text{-}char_{SbC}$ *SPN.FF-def*
apply (*intro Span.in-hhomI*)
apply *auto*
using *Span.src-dom* [*of CMP r s*] *Span.trg-dom* [*of CMP r s*]
apply (*elim Span.in-homE*)
apply *auto*
using *Span.src-dom* [*of CMP r s*] *Span.trg-dom* [*of CMP r s*]
apply (*elim Span.in-homE*)
by *auto*
qed

lemma *compositor-simps* [*simp*]:
assumes *ide r* **and** *ide s* **and** *src r* = *trg s*
shows *Span.arr* (*CMP r s*)
and *Span.src* (*CMP r s*) = $SPN.map_0$ (*src s*) **and** *Span.trg* (*CMP r s*) = $SPN.map_0$ (*trg r*)
and *Span.dom* (*CMP r s*) = *HoSPN-SPN.map* (*r*, *s*)
and *Span.cod* (*CMP r s*) = *SPNoH.map* (*r*, *s*)
using *assms compositor-in-hom* [*of r s*] **by** *auto*

lemma *compositor-is-iso*:
assumes *ide r* **and** *ide s* **and** *src r* = *trg s*
shows *Span.iso* (*CMP r s*)
proof −
interpret *rs*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg r s*
using *assms* **by** *unfold-locales auto*
have *Span.arr* (*CMP r s*)
using *assms compositor-in-hom* **by** *blast*
moreover have *Maps.iso* ⟦⟦*rs.cmp*⟧⟧
using *assms Maps.iso-char′*
by (*metis* (*mono-tags*, *lifting*) *Maps.CLS-in-hom Maps.Map.simps*(*1*) *Maps.in-homE*
equivalence-is-left-adjoint ide-in-iso-class rs.cmp-props(*3*) *rs.cmp-simps*(*2*))
ultimately show *?thesis*
unfolding *CMP-def*
using *assms Span.iso-char* **by** *simp*
qed

interpretation Ξ: *transformation-by-components VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map ‹λ*rs*. *CMP* (*fst rs*) (*snd rs*)›
proof
fix *rs*
assume *rs*: *VV.ide rs*
let *?r* = *fst rs*
let *?s* = *snd rs*
show *Span.in-hom* (*CMP ?r ?s*) (*HoSPN-SPN.map rs*) (*SPNoH.map rs*)
using *rs compositor-in-hom* [*of ?r ?s*] $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *simp*
next
fix $\mu\nu$
assume $\mu\nu$: *VV.arr* $\mu\nu$
let $?\mu$ = *fst* $\mu\nu$

```
let ?ν = snd μν
show CMP (fst (VV.cod μν)) (snd (VV.cod μν)) · HoSPN-SPN.map μν =
    SPNoH.map μν · CMP (fst (VV.dom μν)) (snd (VV.dom μν))
proof −
  let ?LHS = CMP (fst (VV.cod μν)) (snd (VV.cod μν)) · HoSPN-SPN.map μν
  let ?RHS = SPNoH.map μν · CMP (fst (VV.dom μν)) (snd (VV.dom μν))
  have LHS:
    Span.in-hom ?LHS (HoSPN-SPN.map (VV.dom μν)) (SPNoH.map (VV.cod μν))
  proof
    show Span.in-hom (HoSPN-SPN.map μν) (HoSPN-SPN.map (VV.dom μν))
                     (HoSPN-SPN.map (VV.cod μν))
      using μν by blast
    show Span.in-hom (CMP (fst (VV.cod μν)) (snd (VV.cod μν)))
                     (HoSPN-SPN.map (VV.cod μν)) (SPNoH.map (VV.cod μν))
      using μν VV.cod-simp by (auto simp add: VV.arr-char_SbC)
  qed
  have RHS:
    Span.in-hom ?RHS (HoSPN-SPN.map (VV.dom μν)) (SPNoH.map (VV.cod μν))
    using μν VV.dom-simp VV.cod-simp by (auto simp add: VV.arr-char_SbC)
  show ?LHS = ?RHS
  proof (intro Span.arr-eqI)
    show Span.par ?LHS ?RHS
      apply (intro conjI)
      using LHS RHS apply auto[2]
    proof −
      show Span.dom ?LHS = Span.dom ?RHS
      proof −
        have Span.dom ?LHS = HoSPN-SPN.map (VV.dom μν)
          using LHS by auto
        also have ... = Span.dom ?RHS
          using RHS by auto
        finally show ?thesis by simp
      qed
      show Span.cod ?LHS = Span.cod ?RHS
      proof −
        have Span.cod ?LHS = SPNoH.map (VV.cod μν)
          using LHS by auto
        also have ... = Span.cod ?RHS
          using RHS by auto
        finally show ?thesis by simp
      qed
    qed
    show Chn ?LHS = Chn ?RHS
    proof −
      interpret dom-μ-ν: two-composable-identities-in-bicategory-of-spans V H a i src trg
                    ‹dom ?μ› ‹dom ?ν›
        using μν VV.ide-char_SbC VV.arr-char_SbC by unfold-locales auto
      interpret cod-μ-ν: two-composable-identities-in-bicategory-of-spans V H a i src trg
                    ‹cod ?μ› ‹cod ?ν›
```

using $\mu\nu$ $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ **by** *unfold-locales auto*
interpret $\mu\text{-}\nu$: *horizontal-composite-of-arrows-of-tabulations-in-maps*
V H a i *src trg*
‹*dom* $?\mu$› ‹*tab-of-ide* (*dom* $?\mu$)› ‹tab_0 (*dom* $?\mu$)› ‹tab_1 (*dom* $?\mu$)›
‹*dom* $?\nu$› ‹*tab-of-ide* (*dom* $?\nu$)› ‹tab_0 (*dom* $?\nu$)› ‹tab_1 (*dom* $?\nu$)›
‹*cod* $?\mu$› ‹*tab-of-ide* (*cod* $?\mu$)› ‹tab_0 (*cod* $?\mu$)› ‹tab_1 (*cod* $?\mu$)›
‹*cod* $?\nu$› ‹*tab-of-ide* (*cod* $?\nu$)› ‹tab_0 (*cod* $?\nu$)› ‹tab_1 (*cod* $?\nu$)›
$?\mu$ $?\nu$
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ **by** *unfold-locales auto*

let $?\mu\nu = ?\mu \star ?\nu$
interpret *dom*-$\mu\nu$: *identity-in-bicategory-of-spans* V H a i *src trg* ‹*dom* $?\mu\nu$›
using $\mu\nu$ **by** (*unfold-locales*, *simp*)
interpret *cod*-$\mu\nu$: *identity-in-bicategory-of-spans* V H a i *src trg* ‹*cod* $?\mu\nu$›
using $\mu\nu$ **by** (*unfold-locales*, *simp*)
interpret $\mu\nu$: *arrow-of-tabulations-in-maps* V H a i *src trg*
‹*dom* $?\mu\nu$› ‹*tab-of-ide* (*dom* $?\mu\nu$)› ‹tab_0 (*dom* $?\mu\nu$)› ‹tab_1 (*dom* $?\mu\nu$)›
‹*cod* $?\mu\nu$› ‹*tab-of-ide* (*cod* $?\mu\nu$)› ‹tab_0 (*cod* $?\mu\nu$)› ‹tab_1 (*cod* $?\mu\nu$)›
$?\mu\nu$
using $\mu\nu$ **by** *unfold-locales auto*

have *Chn-LHS-eq*:
Chn $?LHS$ = $[\![$*cod*-μ-ν.*cmp*$]\!]$ $\odot$ *Span.chine-hcomp* (*SPN* (*fst* $\mu\nu$)) (*SPN* (*snd* $\mu\nu$))
proof −
have *Chn* $?LHS$ = *Chn* (*CMP* (*fst* (*VV.cod* $\mu\nu$)) (*snd* (*VV.cod* $\mu\nu$))) $\odot$
Chn (*HoSPN-SPN.map* $\mu\nu$)
using $\mu\nu$ *LHS Span.Chn-vcomp* **by** *blast*
also have ... = $[\![$*cod*-μ-ν.*cmp*$]\!]$ $\odot$ *Chn* (*HoSPN-SPN.map* $\mu\nu$)
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ $VV.cod\text{-}char_{SbC}$ *CMP-def* **by** *simp*
also have ... = $[\![$*cod*-μ-ν.*cmp*$]\!]$ $\odot$
Span.chine-hcomp (*SPN* (*fst* $\mu\nu$)) (*SPN* (*snd* $\mu\nu$))
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ *SPN.FF-def Span.hcomp-def* **by** *simp*
finally show *?thesis* **by** *blast*
qed
have *Chn-RHS-eq*:
Chn $?RHS$ = *Maps.MkArr* (*src* (tab_0 (*dom* $?\mu \star$ *dom* $?\nu$)))
(*src* (tab_0 (*cod* $?\mu \star$ *cod* $?\nu$)))
$[\![\mu\nu.chine]\!]$ $\odot$
Maps.MkArr (*src* *dom*-μ-ν.$\varrho\sigma.p_0$) (*src* (*tab-of-ide* (*dom* $?\mu \star$ *dom* $?\nu$)))
$[\![$*dom*-μ-ν.*cmp*$]\!]$
proof −
have *Chn* $?RHS$ = *Chn* (*SPN* ($?\mu \star ?\nu$)) $\odot$
Maps.MkArr (*src* *dom*-μ-ν.$\varrho\sigma.p_0$) (*src* (*tab-of-ide* (*dom* $?\mu \star$ *dom* $?\nu$)))
$[\![$*dom*-μ-ν.*cmp*$]\!]$
using $\mu\nu$ *RHS Span.vcomp-def* $VV.arr\text{-}char_{SbC}$ *CMP-def Span.arr-char Span.not-arr-Null*
VV.dom-simp
by *auto*
moreover have *Chn* (*SPN* ($?\mu \star ?\nu$)) =
Maps.MkArr (*src* (tab_0 (*dom* $?\mu \star$ *dom* $?\nu$)))

```
                        (src (tab₀ (cod ?μ ⋆ cod ?ν)))
                        ⟦μν.chine⟧
  proof −
    have Chn (SPN (?μ ⋆ ?ν)) =
          Maps.MkArr (src (tab₀ (dom ?μ ⋆ dom ?ν)))
                     (src (tab₀ (cod ?μ ⋆ cod ?ν)))
                     ⟦spn ?μν⟧
      using μν SPN-def by simp
    also have ... = Maps.MkArr (src (tab₀ (dom ?μ ⋆ dom ?ν)))
                               (src (tab₀ (cod ?μ ⋆ cod ?ν)))
                               ⟦μν.chine⟧
      using spn-def by simp
    finally show ?thesis by simp
  qed
  ultimately show ?thesis by simp
qed

let ?Chn-LHS =
    Maps.MkArr (src cod-μ-ν.ϱσ.p₀) (src (tab-of-ide (cod ?μ ⋆ cod ?ν)))
      ⟦cod-μ-ν.cmp⟧ ⊙
     Span.chine-hcomp (SPN ?μ) (SPN ?ν)
let ?Chn-RHS =
    Maps.MkArr (src (tab₀ (dom ?μ ⋆ dom ?ν))) (src (tab₀ (cod ?μ ⋆ cod ?ν)))
      ⟦μν.chine⟧ ⊙
     Maps.MkArr (src dom-μ-ν.ϱσ.p₀) (src (tab-of-ide (dom ?μ ⋆ dom ?ν)))
      ⟦dom-μ-ν.cmp⟧

have ?Chn-LHS = ?Chn-RHS
proof (intro Maps.arr-eqI)
  interpret LHS: arrow-of-spans Maps.comp ?LHS
    using LHS Span.arr-char by auto
  interpret RHS: arrow-of-spans Maps.comp ?RHS
    using RHS Span.arr-char by auto
  show 1: Maps.arr ?Chn-LHS
    using LHS.chine-in-hom Chn-LHS-eq by auto
  show 2: Maps.arr ?Chn-RHS
    using RHS.chine-in-hom Chn-RHS-eq by auto
```

Here is where we use *dom-μ-ν.chine-hcomp-SPN-SPN*, which depends on our having chosen the "right" pullbacks for *Maps*(*B*). The map *Chn-LHS* has as its domain the apex of the horizontal composite of the components of *VV.dom μν*, whereas *Chn-RHS* has as its domain the apex of the chosen tabulation of $r_0{}^* \star s_1$. We need these to be equal in order for *Chn-LHS* and *Chn-RHS* to be equal.

```
  show Maps.Dom ?Chn-LHS = Maps.Dom ?Chn-RHS
  proof −
    have Maps.Dom ?Chn-LHS = Maps.Dom (Maps.dom ?Chn-LHS)
      using μν 1 Maps.Dom-dom by presburger
    also have
      ... = Maps.Dom (Span.chine-hcomp (SPN (dom ?μ)) (SPN (dom ?ν)))
```

```
  proof −
    have ... = Maps.Dom (Maps.dom (Span.chine-hcomp (SPN ?μ) (SPN ?ν)))
      using 1 Maps.seq-char Maps.Dom-comp by auto
    also have ... = Maps.Dom (Maps.pbdom (Leg0 (Dom (SPN ?μ)))
                                         (Leg1 (Dom (SPN ?ν))))
      using μν VV.arr-char_SbC Span.chine-hcomp-in-hom [of SPN ?ν SPN ?μ]
      by auto
    also have ... = Maps.Dom (Maps.dom (Maps.pbdom (Leg0 (Dom (SPN ?μ)))
                                           (Leg1 (Dom (SPN ?ν)))))
    proof −
     have Maps.cospan (Leg0 (Dom (SPN (fst μν)))) (Leg1 (Dom (SPN (snd μν))))
       using μν VV.arr-char_SbC SPN-in-hom Span.arr-char Span.dom-char SPN-def
             Maps.CLS-in-hom Maps.arr-char Maps.cod-char dom-μ-ν.composable
             dom-μ-ν.RS-simps(16) dom-μ-ν.S_1-def dom-μ-ν.RS-simps(1)
             dom-μ-ν.R_0-def Maps.pbdom-in-hom
       by simp
      thus ?thesis
        using μν VV.arr-char_SbC Maps.pbdom-in-hom by simp
    qed
    also have ... = Maps.Dom
                    (Maps.dom (Maps.pbdom (Leg0 (Dom (SPN (dom ?μ))))
                                          (Leg1 (Dom (SPN (dom ?ν))))))
      using μν SPN-def VV.arr-char_SbC by simp
    also have ... = Maps.Dom
                    (Maps.dom (Span.chine-hcomp (SPN (dom ?μ)) (SPN (dom ?ν))))
      using μν VV.arr-char_SbC ide-dom
      by (simp add: Span.chine-hcomp-ide-ide)
   also have ... = Maps.Dom (Span.chine-hcomp (SPN (dom ?μ)) (SPN (dom ?ν)))
  using Maps.Dom-dom Maps.in-homE SPN.preserves-reflects-arr SPN.preserves-src
        SPN.preserves-trg Span.chine-hcomp-in-hom dom-μ-ν.composable
        dom-μ-ν.r.base-simps(2) dom-μ-ν.s.base-simps(2)
      by (metis (no-types, lifting))
    finally show ?thesis by simp
  qed
  also have ... = src dom-μ-ν.ϱσ.p_0
    using dom-μ-ν.chine-hcomp-SPN-SPN by simp
  also have ... = Maps.Dom ?Chn-RHS
    using 2 Maps.seq-char Maps.Dom-comp by auto
  finally show ?thesis by simp
qed
show Maps.Cod ?Chn-LHS = Maps.Cod ?Chn-RHS
proof −
  have Maps.Cod ?Chn-LHS = src (tab-of-ide (cod ?μ ⋆ cod ?ν))
    using μν 1 VV.arr-char_SbC Maps.seq-char by auto
  also have ... = src (tab_0 (cod ?μ ⋆ cod ?ν))
    using μν VV.arr-char_SbC cod-μν.tab-simps(2) by auto
  also have ... = Maps.Cod ?Chn-RHS
  by (metis (no-types, lifting) 2 Maps.Cod.simps(1) Maps.Cod-comp Maps.seq-char)
  finally show ?thesis by simp
```

```
qed
show Maps.Map ?Chn-LHS = Maps.Map ?Chn-RHS
proof −
  have RHS: Maps.Map ?Chn-RHS = iso-class (μν.chine ⋆ dom-μ-ν.cmp)
  proof −
    have Maps.Map ?Chn-RHS = Maps.Comp ⟦μν.chine⟧ ⟦dom-μ-ν.cmp⟧
      using μν 2 VV.arr-char_SbC Maps.Map-comp
            Maps.comp-char
              [of Maps.MkArr (src (tab_0 (dom ?μ ⋆ dom ?ν)))
                             (src (tab_0 (cod ?μ ⋆ cod ?ν)))
                             ⟦μν.chine⟧
                  Maps.MkArr (src dom-μ-ν.ϱσ.p_0)
                             (src (tab-of-ide (dom ?μ ⋆ dom ?ν)))
                             ⟦dom-μ-ν.cmp⟧]
      by simp
    also have ... = ⟦μν.chine ⋆ dom-μ-ν.cmp⟧
    proof −
      have ⟦dom-μ-ν.cmp⟧ ∈
             Maps.Hom (src dom-μ-ν.ϱσ.p_0) (src (tab_0 (dom ?μ ⋆ dom ?ν)))
      proof −
        have ⟦dom-μ-ν.cmp⟧ ∈
               Maps.Hom (src dom-μ-ν.ϱσ.tab) (src (tab-of-ide (dom ?μ ⋆ dom ?ν)))
          using μν VV.arr-char_SbC dom-μ-ν.cmp-props(1−3)
          by (metis (mono-tags, lifting) equivalence-is-left-adjoint mem-Collect-eq)
        thus ?thesis
          using μν VV.arr-char_SbC dom-μν.tab-simps(2) by simp
      qed
      moreover have ⟦μν.chine⟧ ∈
                      Maps.Hom (src (tab_0 (dom ?μ ⋆ dom ?ν)))
                               (src (tab_0 (cod ?μ ⋆ cod ?ν)))
        using μν VV.arr-char_SbC μν.chine-in-hom μν.is-map by auto
      moreover have
        μν.chine ⋆ dom-μ-ν.cmp ∈ Maps.Comp ⟦μν.chine⟧ ⟦dom-μ-ν.cmp⟧
      proof
        show is-iso-class ⟦dom-μ-ν.cmp⟧
          using is-iso-classI by simp
        show is-iso-class ⟦μν.chine⟧
          using is-iso-classI by simp
        show dom-μ-ν.cmp ∈ ⟦dom-μ-ν.cmp⟧
          using ide-in-iso-class [of dom-μ-ν.cmp] by simp
        show μν.chine ∈ ⟦μν.chine⟧
          using ide-in-iso-class by simp
        show μν.chine ⋆ dom-μ-ν.cmp ≅ μν.chine ⋆ dom-μ-ν.cmp
    using μν VV.arr-char_SbC μν.chine-simps dom-μ-ν.cmp-simps dom-μν.tab-simps(2)
              isomorphic-reflexive
          by auto
      qed
      ultimately show ?thesis
        using μν dom-μ-ν.cmp-props μν.chine-in-hom μν.chine-is-induced-map
```

```
        Maps.Comp-eq-iso-class-memb
    by blast
  qed
  finally show ?thesis by simp
qed

have LHS: Maps.Map ?Chn-LHS = ⟦cod-μ-ν.cmp ⋆ μ-ν.chine⟧
proof −
  have Maps.Map ?Chn-LHS =
         Maps.Comp ⟦cod-μ-ν.cmp⟧
                 (Maps.Map
                   (Maps.tuple (Maps.CLS (spn ?μ ⋆ dom-μ-ν.ϱσ.p1))
                               (Maps.CLS (tab0 (cod ?μ)))
                               (Maps.CLS (tab1 (cod ?ν)))
                               (Maps.CLS (spn ?ν ⋆ dom-μ-ν.ϱσ.p0))))
  proof −
    have Maps.Map ?Chn-LHS =
         Maps.Comp ⟦cod-μ-ν.cmp⟧
                 (Maps.Map (Span.chine-hcomp (SPN ?μ) (SPN ?ν)))
      using μν 1 VV.arr-charSbC Maps.Map-comp cod-μν.tab-simps(2)
            Maps.comp-char
              [of Maps.MkArr (src cod-μ-ν.ϱσ.p0)
                             (src (tab-of-ide (cod ?μ ⋆ cod ?ν)))
                             ⟦cod-μ-ν.cmp⟧
                  Span.chine-hcomp (SPN ?μ) (SPN ?ν)]
      by simp
    moreover have Span.chine-hcomp (SPN ?μ) (SPN ?ν) =
                  Maps.tuple
                    (Maps.CLS (spn ?μ ⋆ dom-μ-ν.ϱσ.p1))
                     (Maps.CLS (tab0 (cod ?μ)))
                     (Maps.CLS (tab1 (cod ?ν)))
                    (Maps.CLS (spn ?ν ⋆ dom-μ-ν.ϱσ.p0))
    proof −
      have Maps.PRJ0
             (Maps.MkArr (src (tab0 (dom ?μ))) (trg ?ν) ⟦tab0 (dom ?μ)⟧)
             (Maps.MkArr (src (tab0 (dom ?ν))) (trg ?ν) ⟦tab1 (dom ?ν)⟧) =
           ⟦⟦dom-μ-ν.ϱσ.p0⟧⟧ ∧
           Maps.PRJ1
             (Maps.MkArr (src (tab0 (dom ?μ))) (trg ?ν) ⟦tab0 (dom ?μ)⟧)
             (Maps.MkArr (src (tab0 (dom ?ν))) (trg ?ν) ⟦tab1 (dom ?ν)⟧) =
           ⟦⟦dom-μ-ν.ϱσ.p1⟧⟧
      proof −
        interpret X: identity-in-bicategory-of-spans V H a i src trg
                     ‹(tab0 (dom ?μ))* ⋆ tab1 (dom ?ν)›
          using μν VV.arr-charSbC
          by (unfold-locales, simp)
        have Maps.PRJ0
               (Maps.MkArr (src (tab0 (dom ?μ))) (trg ?ν) ⟦tab0 (dom ?μ)⟧)
               (Maps.MkArr (src (tab0 (dom ?ν))) (trg ?ν) ⟦tab1 (dom ?ν)⟧) =
```

$[\![[\![tab_0\ ((Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\mu)))\ (trg\ (snd\ \mu\nu))$
$[\![tab_0\ (dom\ ?\mu)]\!]))^* \star$
$Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\nu)))\ (trg\ ?\nu)$
$[\![tab_1\ (dom\ ?\nu)]\!]))]\!]$
unfolding $Maps.PRJ_0$-*def*
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ *dom-μ-ν.RS-simps*(*1*) *dom-μ-ν.RS-simps*(*16*) *dom-μ-ν.RS-simps*(*18*) *dom-μ-ν.RS-simps*(*3*) *dom-μ-ν.*R_0-*def* *dom-μ-ν.*S_1-*def*
by *auto*
moreover
have $Maps.PRJ_1$
$(Maps.MkArr\ (src\ (tab_0\ (dom\ ?\mu)))\ (trg\ ?\nu)\ [\![tab_0\ (dom\ ?\mu)]\!])$
$(Maps.MkArr\ (src\ (tab_0\ (dom\ ?\nu)))\ (trg\ ?\nu)\ [\![tab_1\ (dom\ ?\nu)]\!]) =$
$[\![[\![tab_1\ ((Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\mu)))\ (trg\ (snd\ \mu\nu))$
$[\![tab_0\ (dom\ ?\mu)]\!]))^* \star$
$Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\nu)))\ (trg\ ?\nu)$
$[\![tab_1\ (dom\ ?\nu)]\!]))]\!]$
unfolding $Maps.PRJ_1$-*def*
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ *dom-μ-ν.RS-simps*(*1*) *dom-μ-ν.RS-simps*(*16*) *dom-μ-ν.RS-simps*(*18*) *dom-μ-ν.RS-simps*(*3*) *dom-μ-ν.*R_0-*def* *dom-μ-ν.*S_1-*def*
by *auto*
moreover
have $(Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\mu)))\ (trg\ (snd\ \mu\nu))$
$[\![tab_0\ (dom\ ?\mu)]\!]))^* \star$
$Maps.REP\ (Maps.MkArr\ (src\ (tab_0\ (dom\ ?\nu)))\ (trg\ ?\nu)$
$[\![tab_1\ (dom\ ?\nu)]\!]) \cong$
$(tab_0\ (dom\ ?\mu))^* \star tab_1\ (dom\ ?\nu)$
using $VV.arr\text{-}char_{SbC}$ $\mu\nu$ *dom-μ-ν.*S_1-*def* *dom-μ-ν.s.leg1-simps*(*3*) *dom-μ-ν.s.leg1-simps*(*4*) *trg-dom* *dom-μ-ν.*R_0-*def* *dom-μ-ν.prj-tab-agreement*(*1*) *isomorphic-symmetric*
by *simp*
ultimately show *?thesis*
using *X.isomorphic-implies-same-tab isomorphic-symmetric* **by** *metis*
qed
thus *?thesis*
unfolding *Span.chine-hcomp-def*
using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ *SPN-def isomorphic-reflexive*
$Maps.comp\text{-}CLS$ [*of spn* $?\mu$ *dom-μ-ν.*$\varrho\sigma.p_1$ *spn* $?\mu \star$ *dom-μ-ν.*$\varrho\sigma.p_1$]
$Maps.comp\text{-}CLS$ [*of spn* $?\nu$ *dom-μ-ν.*$\varrho\sigma.p_0$ *spn* $?\nu \star$ *dom-μ-ν.*$\varrho\sigma.p_0$]
by *simp*
qed
moreover have $Maps.Dom\ (Span.chine\text{-}hcomp\ (SPN\ ?\mu)\ (SPN\ ?\nu)) =$
src dom-μ-ν.$\varrho\sigma.p_0$
by (*metis* (*no-types*, *lifting*) *1 2 Maps.Dom.simps*(*1*) *Maps.comp-char* ‹*Maps.Dom ?Chn-LHS = Maps.Dom ?Chn-RHS*›)
ultimately show *?thesis* **by** *simp*
qed
also have ... = $Maps.Comp$ [\![*cod-μ-ν.cmp*]\!] [\![*μ-ν.chine*]\!]

proof −
  let *?tuple* = *Maps.tuple* ⟦⟦*spn* (*fst* $\mu\nu$) ⋆ *dom-*μ*-*ν*.*$\varrho\sigma$*.*p_1⟧⟧
                ⟦⟦tab_0 (*cod ?*μ)⟧⟧ ⟦⟦tab_1 (*cod ?*ν)⟧⟧
              ⟦⟦*spn* (*snd* $\mu\nu$) ⋆ *dom-*μ*-*ν*.*$\varrho\sigma$*.*p_0⟧⟧
  have *iso-class* μ*-*ν*.chine* = *Maps.Map ?tuple*
    using μ*-*ν*.CLS-chine spn-def Maps.Map.simps*(*1*)
    by (*metis* (*no-types*, *lifting*))
  thus *?thesis* **by** *simp*
qed
also have ... = ⟦*cod-*μ*-*ν*.cmp* ⋆ μ*-*ν*.chine*⟧
proof −
  have ⟦μ*-*ν*.chine*⟧ ∈ *Maps.Hom* (*src dom-*μ*-*ν*.*$\varrho\sigma$*.*p_0) (*src cod-*μ*-*ν*.*$\varrho\sigma$*.*p_0)
  proof −
    have «μ*-*ν*.chine* : *src dom-*μ*-*ν*.*$\varrho\sigma$*.*p_0 → *src cod-*μ*-*ν*.*$\varrho\sigma$*.*p_0»
      using $\mu\nu$ *VV.arr-char*$_{SbC}$ **by** *simp*
    thus *?thesis*
      using μ*-*ν*.is-map ide-in-iso-class left-adjoint-is-ide* **by** *blast*
  qed
  moreover have ⟦*cod-*μ*-*ν*.cmp*⟧ ∈
              Maps.Hom (*src cod-*μ*-*ν*.*$\varrho\sigma$*.*p_0) (*src* (tab_0 (*cod ?*μ ⋆ *cod ?*ν)))
  proof −
    have «*cod-*μ*-*ν*.cmp* : *src cod-*μ*-*ν*.*$\varrho\sigma$*.*p_0 → *src* (tab_0 (*cod ?*μ ⋆ *cod ?*ν))»
      using $\mu\nu$ *VV.arr-char*$_{SbC}$ *cod-*μ*-*ν*.cmp-in-hom cod-*$\mu\nu$*.tab-simps*(*2*)
      by *simp*
    thus *?thesis*
      using *cod-*μ*-*ν*.cmp-props equivalence-is-left-adjoint left-adjoint-is-ide*
          ide-in-iso-class [*of cod-*μ*-*ν*.cmp*]
      by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
  qed
  moreover have
    cod-μ*-*ν*.cmp* ⋆ μ*-*ν*.chine* ∈ *Maps.Comp* ⟦*cod-*μ*-*ν*.cmp*⟧ ⟦μ*-*ν*.chine*⟧
  proof
    show *is-iso-class* ⟦μ*-*ν*.chine*⟧
      using μ*-*ν*.w-simps*(*1*) *is-iso-classI* **by** *blast*
    show *is-iso-class* ⟦*cod-*μ*-*ν*.cmp*⟧
      using *cod-*μ*-*ν*.cmp-simps*(*2*) *is-iso-classI* **by** *blast*
    show μ*-*ν*.chine* ∈ ⟦μ*-*ν*.chine*⟧
      using *ide-in-iso-class* **by** *simp*
    show *cod-*μ*-*ν*.cmp* ∈ ⟦*cod-*μ*-*ν*.cmp*⟧
      using *ide-in-iso-class* **by** *simp*
    show *cod-*μ*-*ν*.cmp* ⋆ μ*-*ν*.chine* ≅ *cod-*μ*-*ν*.cmp* ⋆ μ*-*ν*.chine*
      by (*simp add*: *isomorphic-reflexive*)
  qed
  ultimately show *?thesis*
    using $\mu\nu$ *cod-*μ*-*ν*.cmp-props* μ*-*ν*.chine-in-hom* μ*-*ν*.chine-is-induced-map*
        Maps.Comp-eq-iso-class-memb
    by *simp*
qed
finally show *?thesis* **by** *simp*

qed

have *EQ*: $[\![\mu\nu.chine \star dom\text{-}\mu\text{-}\nu.cmp]\!] = [\![cod\text{-}\mu\text{-}\nu.cmp \star \mu\text{-}\nu.chine]\!]$
proof (*intro iso-class-eqI*)
  show $\mu\nu.chine \star dom\text{-}\mu\text{-}\nu.cmp \cong cod\text{-}\mu\text{-}\nu.cmp \star \mu\text{-}\nu.chine$
  proof −
    interpret *dom-cmp*: *identity-arrow-of-tabulations-in-maps* V H a i *src trg*
      ‹*dom* $?\mu\nu$›
      $dom\text{-}\mu\text{-}\nu.\varrho\sigma.tab$
      ‹tab_0 (*dom* $?\nu$) $\star$ $dom\text{-}\mu\text{-}\nu.\varrho\sigma.p_0$›
      ‹tab_1 (*dom* $?\mu$) $\star$ $dom\text{-}\mu\text{-}\nu.\varrho\sigma.p_1$›
      ‹*dom* $?\mu\nu$›
      ‹*tab-of-ide* (*dom* $?\mu$ $\star$ *dom* $?\nu$)›
      ‹tab_0 (*dom* $?\mu$ $\star$ *dom* $?\nu$)›
      ‹tab_1 (*dom* $?\mu$ $\star$ *dom* $?\nu$)›
      ‹*dom* $?\mu\nu$›
    using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ $dom\text{-}\mu\text{-}\nu.cmp\text{-}interpretation$ **by** *simp*
    interpret *cod-cmp*: *identity-arrow-of-tabulations-in-maps* V H a i *src trg*
      ‹*cod* $?\mu\nu$›
      $cod\text{-}\mu\text{-}\nu.\varrho\sigma.tab$
      ‹tab_0 (*cod* $?\nu$) $\star$ $cod\text{-}\mu\text{-}\nu.\varrho\sigma.p_0$›
      ‹tab_1 (*cod* $?\mu$) $\star$ $cod\text{-}\mu\text{-}\nu.\varrho\sigma.p_1$›
      ‹*cod* $?\mu\nu$›
      ‹*tab-of-ide* (*cod* $?\mu$ $\star$ *cod* $?\nu$)›
      ‹tab_0 (*cod* $?\mu$ $\star$ *cod* $?\nu$)›
      ‹tab_1 (*cod* $?\mu$ $\star$ *cod* $?\nu$)›
      ‹*cod* $?\mu\nu$›
    using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ $cod\text{-}\mu\text{-}\nu.cmp\text{-}interpretation$ **by** *simp*
    interpret *L*: *vertical-composite-of-arrows-of-tabulations-in-maps*
      V H a i *src trg*
      ‹*dom* $?\mu\nu$›
      ‹$dom\text{-}\mu\text{-}\nu.\varrho\sigma.tab$›
      ‹tab_0 (*dom* $?\nu$) $\star$ $dom\text{-}\mu\text{-}\nu.\varrho\sigma.p_0$›
      ‹tab_1 (*dom* $?\mu$) $\star$ $dom\text{-}\mu\text{-}\nu.\varrho\sigma.p_1$›
      ‹*dom* $?\mu\nu$›
      ‹*tab-of-ide* (*dom* $?\mu\nu$)›
      ‹tab_0 (*dom* $?\mu\nu$)›
      ‹tab_1 (*dom* $?\mu\nu$)›
      ‹*cod* $?\mu\nu$›
      $cod\text{-}\mu\nu.tab$
      ‹tab_0 (*cod* $?\mu\nu$)›
      ‹tab_1 (*cod* $?\mu\nu$)›
      ‹*dom* $?\mu\nu$›
      ‹$?\mu \star ?\nu$›
    using $\mu\nu$ $VV.arr\text{-}char_{SbC}$ $dom\text{-}\mu\text{-}\nu.cmp\text{-}in\text{-}hom$
    by *unfold-locales auto*
    interpret *R*: *vertical-composite-of-arrows-of-tabulations-in-maps*
      V H a i *src trg*
      ‹*dom* $?\mu\nu$›

```
                    ‹dom-μ-ν.ϱσ.tab›
                    ‹tab₀ (dom ?ν) ⋆ dom-μ-ν.ϱσ.p₀›
                    ‹tab₁ (dom ?μ) ⋆ dom-μ-ν.ϱσ.p₁›
                    ‹cod ?μν›
                    ‹cod-μ-ν.ϱσ.tab›
                    ‹tab₀ (cod ?ν) ⋆ cod-μ-ν.ϱσ.p₀›
                    ‹tab₁ (cod ?μ) ⋆ cod-μ-ν.ϱσ.p₁›
                    ‹cod ?μν›
                    cod-μν.tab
                    ‹tab₀ (cod ?μν)›
                    ‹tab₁ (cod ?μν)›
                    ‹?μ ⋆ ?ν›
                    ‹cod ?μν›
              using μν VV.arr-char_SbC cod-μ-ν.cmp-in-hom
              by unfold-locales auto
            have μν.chine ⋆ dom-μ-ν.cmp ≅ L.chine
            using μν VV.arr-char_SbC L.chine-char dom-μ-ν.cmp-def isomorphic-symmetric
              by simp
            also have ... = R.chine
              using L.is-ide μν comp-arr-dom comp-cod-arr isomorphic-reflexive by force
            also have ... ≅ cod-μ-ν.cmp ⋆ μ-ν.chine
              using μν VV.arr-char_SbC R.chine-char cod-μ-ν.cmp-def by simp
            finally show ?thesis by simp
          qed
        qed
        show ?thesis
          using LHS RHS EQ by simp
      qed
    qed
    thus ?thesis
      using Chn-LHS-eq Chn-RHS-eq by simp
  qed
 qed
qed
qed

interpretation Ξ: natural-isomorphism VV.comp Span.vcomp
                  HoSPN-SPN.map SPNoH.map Ξ.map
  using VV.ide-char_SbC VV.arr-char_SbC Ξ.map-simp-ide compositor-is-iso
  by (unfold-locales, simp)

lemma compositor-naturalitytransformation:
shows transformation-by-components VV.comp Span.vcomp HoSPN-SPN.map SPNoH.map
        (λrs. CMP (fst rs) (snd rs))
  ..

lemma compositor-naturalityisomorphism:
shows natural-isomorphism VV.comp Span.vcomp HoSPN-SPN.map SPNoH.map Ξ.map
  ..
```

end

Associativity Coherence

locale *three-composable-identities-in-bicategory-of-spans* =
  bicategory-of-spans V H a i *src trg* +
  f: *identity-in-bicategory-of-spans V H* a i *src trg f* +
  g: *identity-in-bicategory-of-spans V H* a i *src trg g* +
  h: *identity-in-bicategory-of-spans V H* a i *src trg h*
for *V* :: *'a comp* (**infixr** ‹·› *55*)
and *H* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹⋆› *53*)
and a :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›)
and i :: *'a* ⇒ *'a* (‹i[-]›)
and *src* :: *'a* ⇒ *'a*
and *trg* :: *'a* ⇒ *'a*
and *f* :: *'a*
and *g* :: *'a*
and *h* :: *'a* +
assumes *fg*: *src f* = *trg g*
and *gh*: *src g* = *trg h*
begin

  interpretation *f*: *arrow-of-tabulations-in-maps V H* a i *src trg*
    f f.tab ‹tab_0 *f*› ‹tab_1 *f*› *f f.tab* ‹tab_0 *f*› ‹tab_1 *f*› *f*
    using *f.is-arrow-of-tabulations-in-maps* **by** *simp*
  interpretation *h*: *arrow-of-tabulations-in-maps V H* a i *src trg*
    h h.tab ‹tab_0 *h*› ‹tab_1 *h*› *h h.tab* ‹tab_0 *h*› ‹tab_1 *h*› *h*
    using *h.is-arrow-of-tabulations-in-maps* **by** *simp*

  interpretation *E*: *self-evaluation-map V H* a i *src trg* **..**
  notation *E.eval* (‹⦃-⦄›)

  interpretation *Maps*: *maps-category V H* a i *src trg* **..**
  interpretation *Span*: *span-bicategory Maps.comp* $Maps.PRJ_0$ $Maps.PRJ_1$ **..**

  no-notation *Fun.comp* (**infixl** ‹∘› *55*)
  notation *Span.vcomp* (**infixr** ‹•› *55*)
  notation *Span.hcomp* (**infixr** ‹∘› *53*)
  notation *Maps.comp* (**infixr** ‹⊙› *55*)
  notation *isomorphic* (**infix** ‹≅› *50*)

  interpretation *SPN*: *functor V Span.vcomp SPN*
    using *SPN-is-functor* **by** *simp*
  interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
    using *SPN-is-weak-arrow-of-homs* **by** *simp*
  interpretation *SPN-SPN*: *functor VV.comp Span.VV.comp SPN.FF*
    using *SPN.functor-FF* **by** *auto*
  interpretation *HoSPN-SPN*: *composite-functor VV.comp Span.VV.comp Span.vcomp*

SPN.FF ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\circ$ *snd* $\mu\nu$›

..

interpretation *SPNoH*: *composite-functor VV.comp V Span.vcomp* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$› *SPN*

..

Here come a lot of interpretations for "composite things". We need these in order to have relatively short, systematic names for entities that will appear in the lemmas to follow. The names of the interpretations use a prefix notation, where H refers to horizontal composition of 1-cells and T refers to composite of tabulations. So, for example, *THfgh* refers to the composite of the tabulation associated with the horizontal composition $f \star g$ with the tabulation associated with h.

interpretation *HHfgh*: *identity-in-bicategory-of-spans V H* a i *src trg* ‹$(f \star g) \star h$›
  using *fg gh* **by** *unfold-locales auto*
interpretation *HfHgh*: *identity-in-bicategory-of-spans V H* a i *src trg* ‹$f \star g \star h$›
  using *fg gh* **by** *unfold-locales auto*
interpretation *Tfg*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg f g*
  using *fg gh* **by** *unfold-locales auto*
interpretation *Tgh*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg g h*
  using *fg gh* **by** *unfold-locales auto*
interpretation *THfgh*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg*
    ‹$f \star g$› *h*
  using *fg gh* **by** *unfold-locales auto*
interpretation *THfgh*: *tabulation V H* a i *src trg* ‹$(f \star g) \star h$› *THfgh.$\varrho\sigma$.tab*
    ‹tab_0 $h \star$ *THfgh.$\varrho\sigma$.p_0*› ‹tab_1 $(f \star g) \star$ *THfgh.$\varrho\sigma$.p_1*›
  using *THfgh.$\varrho\sigma$.composite-is-tabulation* **by** *simp*
interpretation *TfHgh*: *two-composable-identities-in-bicategory-of-spans V H* a i *src trg*
    f ‹$g \star h$›
  using *fg gh* **by** *unfold-locales auto*
interpretation *TfHgh*: *tabulation V H* a i *src trg* ‹$f \star g \star h$› *TfHgh.$\varrho\sigma$.tab*
    ‹tab_0 $(g \star h) \star$ *TfHgh.$\varrho\sigma$.p_0*› ‹tab_1 $f \star$ *TfHgh.$\varrho\sigma$.p_1*›
  using *TfHgh.$\varrho\sigma$.composite-is-tabulation* **by** *simp*

interpretation *Tfg-Hfg*: *arrow-of-tabulations-in-maps V H* a i *src trg*
    ‹$f \star g$› *Tfg.$\varrho\sigma$.tab* ‹tab_0 $g \star$ *Tfg.$\varrho\sigma$.p_0*› ‹tab_1 $f \star$ *Tfg.$\varrho\sigma$.p_1*›
    ‹$f \star g$› ‹*tab-of-ide* $(f \star g)$› ‹tab_0 $(f \star g)$› ‹tab_1 $(f \star g)$›
    ‹$f \star g$›
  by *unfold-locales auto*
interpretation *Tgh-Hgh*: *arrow-of-tabulations-in-maps V H* a i *src trg*
    ‹$g \star h$› *Tgh.$\varrho\sigma$.tab* ‹tab_0 $h \star$ *Tgh.$\varrho\sigma$.p_0*› ‹tab_1 $g \star$ *Tgh.$\varrho\sigma$.p_1*›
    ‹$g \star h$› ‹*tab-of-ide* $(g \star h)$› ‹tab_0 $(g \star h)$› ‹tab_1 $(g \star h)$›
    ‹$g \star h$›
  by *unfold-locales auto*
interpretation *THfgh-HHfgh*:
  arrow-of-tabulations-in-maps V H a i *src trg*
    ‹$(f \star g) \star h$› *THfgh.$\varrho\sigma$.tab* ‹tab_0 $h \star$ *THfgh.$\varrho\sigma$.p_0*› ‹tab_1 $(f \star g) \star$ *THfgh.$\varrho\sigma$.p_1*›
    ‹$(f \star g) \star h$› ‹*tab-of-ide* $((f \star g) \star h)$› ‹tab_0 $((f \star g) \star h)$› ‹tab_1 $((f \star g) \star h)$›
    ‹$(f \star g) \star h$›
  using *fg gh* **by** *unfold-locales auto*

interpretation *TfHgh-HfHgh*:
arrow-of-tabulations-in-maps V H a i *src trg*
‹$f \star g \star h$› *TfHgh.ϱσ.tab* ‹$tab_0\ (g \star h) \star$ *TfHgh.ϱσ.*p_0› ‹$tab_1\ f \star$ *TfHgh.ϱσ.*p_1›
‹$f \star g \star h$› ‹*tab-of-ide* $(f \star g \star h)$› ‹$tab_0\ (f \star g \star h)$› ‹$tab_1\ (f \star g \star h)$›
‹$f \star g \star h$›
using *fg gh* **by** *unfold-locales auto*
interpretation *TTfgh*: *composite-tabulation-in-maps V H* a i *src trg*
‹$f \star g$› *Tfg.ϱσ.tab* ‹$tab_0\ g \star$ *Tfg.ϱσ.*p_0› ‹$tab_1\ f \star$ *Tfg.ϱσ.*p_1›
h ‹*tab-of-ide* h› ‹$tab_0\ h$› ‹$tab_1\ h$›
using *fg gh* **by** *unfold-locales auto*
interpretation *TTfgh-THfgh*:
horizontal-composite-of-arrows-of-tabulations-in-maps V H a i *src trg*
‹$f \star g$› *Tfg.ϱσ.tab* ‹$tab_0\ g \star$ *Tfg.ϱσ.*p_0› ‹$tab_1\ f \star$ *Tfg.ϱσ.*p_1›
h ‹*tab-of-ide* h› ‹$tab_0\ h$› ‹$tab_1\ h$›
‹$f \star g$› ‹*tab-of-ide* $(f \star g)$› ‹$tab_0\ (f \star g)$› ‹$tab_1\ (f \star g)$›
h ‹*tab-of-ide* h› ‹$tab_0\ h$› ‹$tab_1\ h$›
‹$f \star g$› h
..
interpretation *TfTgh*: *composite-tabulation-in-maps V H* a i *src trg*
f ‹*tab-of-ide* f› ‹$tab_0\ f$› ‹$tab_1\ f$›
‹$g \star h$› *Tgh.ϱσ.tab* ‹$tab_0\ h \star$ *Tgh.ϱσ.*p_0› ‹$tab_1\ g \star$ *Tgh.ϱσ.*p_1›
using *fg gh* **by** *unfold-locales auto*
interpretation *TfTgh-TfHgh*:
horizontal-composite-of-arrows-of-tabulations-in-maps V H a i *src trg*
f ‹*tab-of-ide* f› ‹$tab_0\ f$› ‹$tab_1\ f$›
‹$g \star h$› *Tgh.ϱσ.tab* ‹$tab_0\ h \star$ *Tgh.ϱσ.*p_0› ‹$tab_1\ g \star$ *Tgh.ϱσ.*p_1›
f ‹*tab-of-ide* f› ‹$tab_0\ f$› ‹$tab_1\ f$›
‹$g \star h$› ‹*tab-of-ide* $(g \star h)$› ‹$tab_0\ (g \star h)$› ‹$tab_1\ (g \star h)$›
f ‹$g \star h$›
..
interpretation *TfTgh-TfTgh*:
horizontal-composite-of-arrows-of-tabulations-in-maps V H a i *src trg*
f ‹*tab-of-ide* f› ‹$tab_0\ f$› ‹$tab_1\ f$›
‹$g \star h$› *Tgh.ϱσ.tab* ‹$tab_0\ h \star$ *Tgh.ϱσ.*p_0› ‹$tab_1\ g \star$ *Tgh.ϱσ.*p_1›
f ‹*tab-of-ide* f› ‹$tab_0\ f$› ‹$tab_1\ f$›
‹$g \star h$› *Tgh.ϱσ.tab* ‹$tab_0\ h \star$ *Tgh.ϱσ.*p_0› ‹$tab_1\ g \star$ *Tgh.ϱσ.*p_1›
f ‹$g \star h$›
..

The following interpretation defines the associativity between the peaks of the two composite tabulations *TTfgh* (associated to the left) and *TfTgh* (associated to the right).

interpretation *TTfgh-TfTgh*:
arrow-of-tabulations-in-maps V H a i *src trg*
‹$(f \star g) \star h$› *TTfgh.tab* ‹$tab_0\ h \star$ *TTfgh.*p_0› ‹$(tab_1\ f \star$ *Tfg.ϱσ.*$p_1) \star$ *TTfgh.*p_1›
‹$f \star g \star h$› *TfTgh.tab* ‹$(tab_0\ h \star$ *Tgh.ϱσ.*$p_0) \star$ *TfTgh.*p_0› ‹$tab_1\ f \star$ *TfTgh.*p_1›
‹a$[f, g, h]$›
using *fg gh* **by** *unfold-locales auto*

This interpretation defines the map, from the apex of the tabulation associated with the horizontal composite $(f \star g) \star h$ to the apex of the tabulation associated with the

horizontal composite $f \star g \star h$, induced by the associativity isomorphism $\mathrm{a}[f, g, h]$ from $(f \star g) \star h$ to $f \star g \star h$.

interpretation *HHfgh-HfHgh*: *arrow-of-tabulations-in-maps V H* a i *src trg*
‹*dom* (α (f, g, h))› ‹*tab-of-ide* (*dom* (α (f, g, h)))›
‹tab_0 (*dom* (α (f, g, h)))› ‹tab_1 (*dom* (α (f, g, h)))›
‹*cod* (α (f, g, h))› ‹*tab-of-ide* (*cod* (α (f, g, h)))›
‹tab_0 (*cod* (α (f, g, h)))› ‹tab_1 (*cod* (α (f, g, h)))›
‹α (f, g, h)›
proof –
have *arrow-of-tabulations-in-maps V H* a i *src trg*
$((f \star g) \star h)$ (*tab-of-ide* $((f \star g) \star h)$) (tab_0 $((f \star g) \star h)$) (tab_1 $((f \star g) \star h)$)
$(f \star g \star h)$ (*tab-of-ide* $(f \star g \star h)$) (tab_0 $(f \star g \star h)$) (tab_1 $(f \star g \star h)$)
$\mathrm{a}[f, g, h]$
using *fg gh* **by** *unfold-locales auto*
thus *arrow-of-tabulations-in-maps V H* a i *src trg*
(*dom* (α (f, g, h))) (*tab-of-ide* (*dom* (α (f, g, h))))
(tab_0 (*dom* (α (f, g, h)))) (tab_1 (*dom* (α (f, g, h))))
(*cod* (α (f, g, h))) (*tab-of-ide* (*cod* (α (f, g, h))))
(tab_0 (*cod* (α (f, g, h)))) (tab_1 (*cod* (α (f, g, h))))
(α (f, g, h))
using *fg gh* α*-def* **by** *auto*
qed

interpretation *SPN-f*: *arrow-of-spans Maps.comp* ‹*SPN f*›
using *SPN-in-hom Span.arr-char* [*of SPN f*] **by** *simp*
interpretation *SPN-g*: *arrow-of-spans Maps.comp* ‹*SPN g*›
using *SPN-in-hom Span.arr-char* [*of SPN g*] **by** *simp*
interpretation *SPN-h*: *arrow-of-spans Maps.comp* ‹*SPN h*›
using *SPN-in-hom Span.arr-char* [*of SPN h*] **by** *simp*
interpretation *SPN-fgh*: *three-composable-identity-arrows-of-spans Maps.comp*
Maps.PRJ$_0$ *Maps.PRJ*$_1$ ‹*SPN f*› ‹*SPN g*› ‹*SPN h*›
using *fg gh Span.arr-char SPN-in-hom SPN.preserves-ide Span.ide-char*
apply *unfold-locales* **by** *auto*

The following relates the projections associated with the composite span *SPN-fgh* with tabulations in the underlying bicategory.

lemma *prj-char*:
shows *SPN-fgh.Prj*$_{11}$ = $[\![\![$*Tfg.*$\varrho\sigma$*.*p_1 $\star$ *TTfgh.*$p_1]\!]\!]$
and *SPN-fgh.Prj*$_{01}$ = $[\![\![$*Tfg.*$\varrho\sigma$*.*p_0 $\star$ *TTfgh.*$p_1]\!]\!]$
and *SPN-fgh.Prj*$_0$ = $[\![\![$*TTfgh.*$p_0]\!]\!]$
and *SPN-fgh.Prj*$_1$ = $[\![\![$*TfTgh.*$p_1]\!]\!]$
and *SPN-fgh.Prj*$_{10}$ = $[\![\![$*Tgh.*$\varrho\sigma$*.*p_1 $\star$ *TfTgh.*$p_0]\!]\!]$
and *SPN-fgh.Prj*$_{00}$ = $[\![\![$*Tgh.*$\varrho\sigma$*.*p_0 $\star$ *TfTgh.*$p_0]\!]\!]$
proof –
show *SPN-fgh.Prj*$_{11}$ = $[\![\![$*Tfg.*$\varrho\sigma$*.*p_1 $\star$ *TTfgh.*$p_1]\!]\!]$
proof –
have *ide* (*Tfg.*$\varrho\sigma$*.*p_1 $\star$ *TTfgh.*p_1)
by (*metis TTfgh.composable TTfgh.leg1-simps*(*2*) *Tfg.*$\varrho\sigma$*.T0.antipar*(*2*)
Tfg.$\varrho\sigma$*.T0.ide-right Tfg-Hfg.u-simps*(*3*) *f.T0.antipar*(*2*) *f.T0.ide-right*

f.u-simps(*3*) *fg g.ide-leg1 g.leg1-simps*(*4*) *h.ide-leg1 h.leg1-simps*(*4*)
ide-hcomp hseqE hcomp-simps(*1*) tab_1*-simps*(*1*))
thus *?thesis*
using *fg gh Tfg.ϱσ.prj-char TTfgh.prj-char isomorphic-reflexive*
Maps.comp-CLS [*of* tab_0 *g Tfg.ϱσ.*p_0 tab_0 *g* ⋆ *Tfg.ϱσ.*p_0]
Maps.comp-CLS [*of Tfg.ϱσ.*p_1 *TTfgh.*p_1 *Tfg.ϱσ.*p_1 ⋆ *TTfgh.*p_1]
by (*simp add*: *TTfgh.composable Tfg.ϱσ.T0.antipar*(*2*))
qed
show *SPN-fgh.Prj*$_{01}$ = ⟦⟦*Tfg.ϱσ.*p_0 ⋆ *TTfgh.*p_1⟧⟧
proof −
have *ide* (*Tfg.ϱσ.*p_0 ⋆ *TTfgh.*p_1)
by (*metis TTfgh.leg1-simps*(*2*) *bicategory-of-spans.tab*$_0$*-simps*(*1*)
bicategory-of-spans.tab$_1$*-simps*(*1*) *bicategory-of-spans-axioms*
Tfg.ϱσ.T0.antipar(*2*) *Tfg.ϱσ.T0.ide-right Tfg.composable f.T0.antipar*(*2*)
f.T0.ide-right f.u-simps(*3*) *g.ide-leg1 g.leg1-simps*(*4*)
Tfg.u-simps(*3*) *THfgh.composable h.ide-leg1 h.leg1-simps*(*4*)
ide-hcomp hseqE hcomp-simps(*1*) tab_1*-simps*(*3*))
thus *?thesis*
using *fg gh Tfg.ϱσ.prj-char TTfgh.prj-char isomorphic-reflexive*
Maps.comp-CLS [*of* tab_0 *g Tfg.ϱσ.*p_0 tab_0 *g* ⋆ *Tfg.ϱσ.*p_0]
Maps.comp-CLS [*of Tfg.ϱσ.*p_0 *TTfgh.*p_1 *Tfg.ϱσ.*p_0 ⋆ *TTfgh.*p_1]
by (*simp add*: *Tfg.ϱσ.T0.antipar*(*2*) *THfgh.composable*)
qed
show *SPN-fgh.Prj*$_0$ = ⟦⟦*TTfgh.*p_0⟧⟧
using *isomorphic-reflexive TTfgh.prj-char Tfg.ϱσ.prj-char*
Maps.comp-CLS [*of* tab_0 *g Tfg.ϱσ.*p_0 tab_0 *g* ⋆ *Tfg.ϱσ.*p_0]
by (*simp add*: *Tfg.composable*)
show *SPN-fgh.Prj*$_1$ = ⟦⟦*TfTgh.*p_1⟧⟧
using *Tgh.ϱσ.prj-char isomorphic-reflexive Tgh.composable*
Maps.comp-CLS [*of* tab_1 *g Tgh.ϱσ.*p_1 tab_1 *g* ⋆ *Tgh.ϱσ.*p_1]
TfTgh.prj-char
by *simp*
show *SPN-fgh.Prj*$_{10}$ = ⟦⟦*Tgh.ϱσ.*p_1 ⋆ *TfTgh.*p_0⟧⟧
using *fg gh Tgh.ϱσ.prj-char TfTgh.prj-char isomorphic-reflexive*
Maps.comp-CLS [*of* tab_1 *g* prj_1 (tab_1 *h*) (tab_0 *g*) tab_1 *g* ⋆ *Tgh.ϱσ.*p_1]
Maps.comp-CLS [*of Tgh.ϱσ.*p_1 *TfTgh.*p_0 *Tgh.ϱσ.*p_1 ⋆ *TfTgh.*p_0]
by *simp*
show *SPN-fgh.Prj*$_{00}$ = ⟦⟦*Tgh.ϱσ.*p_0 ⋆ *TfTgh.*p_0⟧⟧
using *fg gh Tgh.ϱσ.prj-char TfTgh.prj-char isomorphic-reflexive*
Maps.comp-CLS [*of* tab_1 *g Tgh.ϱσ.*p_1 tab_1 *g* ⋆ *Tgh.ϱσ.*p_1]
Maps.comp-CLS [*of Tgh.ϱσ.*p_0 *TfTgh.*p_0 *Tgh.ϱσ.*p_0 ⋆ *TfTgh.*p_0]
by *simp*
qed

interpretation Φ: *transformation-by-components VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map ‹λ*rs. CMP* (*fst rs*) (*snd rs*)›
using *compositor-naturalitytransformation* **by** *simp*
interpretation Φ: *natural-isomorphism VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map Φ*.map*

using *compositor-naturalityisomorphism* **by** *simp*

interpretation *VVV'*: *subcategory VxVxV.comp*
‹$\lambda\tau\mu\nu$. *arr* (*fst* $\tau\mu\nu$) $\wedge$ *arr* (*fst* (*snd* $\tau\mu\nu$)) $\wedge$ *arr* (*snd* (*snd* $\tau\mu\nu$)) $\wedge$
src (*fst* (*snd* $\tau\mu\nu$)) = *trg* (*snd* (*snd* $\tau\mu\nu$)) $\wedge$
src (*fst* $\tau\mu\nu$) = *trg* (*fst* (*snd* $\tau\mu\nu$))›
using *fg gh VVV.arr-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VVV.subcategory-axioms* **by** *simp*

We define abbreviations for the left and right-hand sides of the equation for associativity coherence.

abbreviation *LHS*
where *LHS* $\equiv$ *SPN* a[*f*, *g*, *h*] $\cdot$ Φ.*map* (*f* $\star$ *g*, *h*) $\cdot$ (Φ.*map* (*f*, *g*) $\circ$ *SPN h*)

abbreviation *RHS*
where *RHS* $\equiv$ Φ.*map* (*f*, *g* $\star$ *h*) $\cdot$ (*SPN f* $\circ$ Φ.*map* (*g*, *h*)) $\cdot$
Span.assoc (*SPN f*) (*SPN g*) (*SPN h*)

lemma *arr-LHS*:
shows *Span.arr LHS*
using *fg gh VVV.arr-char*$_{SbC}$ *VVV.ide-char*$_{SbC}$ *VV.arr-char*$_{SbC}$ *VV.ide-char*$_{SbC}$ *Span.hseqI'*
HoHV-def compositor-in-hom α*-def*
apply (*intro Span.seqI*)
apply *simp-all*
using *SPN.FF-def*
apply *simp*
proof −
have *SPN* ((*f* $\star$ *g*) $\star$ *h*) = *Span.cod* (*CMP* (*f* $\star$ *g*) *h*)
using *fg gh compositor-in-hom* **by** *simp*
also have ... = *Span.cod* (*CMP* (*f* $\star$ *g*) *h* $\cdot$ (*CMP f g* $\circ$ *SPN h*))
proof −
have *Span.seq* (*CMP* (*f* $\star$ *g*) *h*) (*CMP f g* $\circ$ *SPN h*)
proof (*intro Span.seqI Span.hseqI*)
show *1*: *Span.in-hhom* (*SPN h*) (*SPN.map*$_0$ (*src h*)) (*SPN.map*$_0$ (*trg h*))
using *SPN.preserves-src SPN.preserves-trg* **by** *simp*
show *2*: *Span.in-hhom* (*CMP f g*) (*SPN.map*$_0$ (*trg h*)) (*SPN.map*$_0$ (*trg f*))
using *compositor-in-hom SPN-fgh.*$\nu\pi$*.composable fg* **by** *auto*
show *3*: *Span.arr* (*CMP* (*f* $\star$ *g*) *h*)
using *TTfgh.composable Tfg.*$\varrho\sigma$*.ide-base compositor-simps*(*1*) *h.is-ide* **by** *auto*
show *Span.dom* (*CMP* (*f* $\star$ *g*) *h*) = *Span.cod* (*CMP f g* $\circ$ *SPN h*)
using *1 2 3 fg gh compositor-in-hom SPN-fgh.*$\nu\pi$*.composable SPN-in-hom SPN.FF-def*
by *auto*
qed
thus *?thesis* **by** *simp*
qed
finally show *SPN* ((*f* $\star$ *g*) $\star$ *h*) = *Span.cod* (*CMP* (*f* $\star$ *g*) *h* $\cdot$ (*CMP f g* $\circ$ *SPN h*))
by *blast*
qed

lemma *arr-RHS*:
shows *Span.arr RHS*
  using *fg gh VV.ide-char$_{SbC}$ VV.arr-char$_{SbC}$ Φ.map-simp-ide SPN.FF-def Span.hseqI'*
  by (*intro Span.seqI*, *simp-all*)

lemma *par-LHS-RHS*:
shows *Span.par LHS RHS*
proof (*intro conjI*)
  show *Span.arr LHS*
    using *arr-LHS* **by** *simp*
  show *Span.arr RHS*
    using *arr-RHS* **by** *simp*
  show *Span.dom LHS* = *Span.dom RHS*
  proof −
    have *Span.dom LHS* = *Span.dom* (Φ.*map* (*f*, *g*) ∘ *SPN h*)
      using *arr-LHS* **by** *auto*
    also have ... = *Span.dom* (Φ.*map* (*f*, *g*)) ∘ *Span.dom* (*SPN h*)
      using *arr-LHS Span.dom-hcomp* [*of SPN h* Φ.*map* (*f*, *g*)] **by** *blast*
    also have ... = (*SPN f* ∘ *SPN g*) ∘ *SPN h*
      using *fg* Φ.*map-simp-ide VV.ide-char$_{SbC}$ VV.arr-char$_{SbC}$ SPN.FF-def* **by** *simp*
    also have ... = *Span.dom* (*Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*))
      using *fg gh VVV.arr-char$_{SbC}$ VVV.ide-char$_{SbC}$ VV.arr-char$_{SbC}$ VV.ide-char$_{SbC}$* **by** *simp*
    also have ... = *Span.dom RHS*
      using ‹*Span.arr RHS*› **by** *auto*
    finally show *?thesis* **by** *blast*
  qed
  show *Span.cod LHS* = *Span.cod RHS*
  proof −
    have *Span.cod LHS* = *Span.cod* (*SPN* a[*f*, *g*, *h*])
      using *arr-LHS* **by** *simp*
    also have ... = *SPN* (*f* ⋆ *g* ⋆ *h*)
      unfolding *α-def*
        using *fg gh VVV.ide-char$_{SbC}$ VVV.arr-char$_{SbC}$ VV.ide-char$_{SbC}$ VV.arr-char$_{SbC}$ HoVH-def*
      by *simp*
    also have ... = *Span.cod RHS*
      using *arr-RHS fg gh* Φ.*map-simp-ide VV.ide-char$_{SbC}$ VV.arr-char$_{SbC}$ SPN.FF-def*
          compositor-in-hom
      by *simp*
    finally show *?thesis* **by** *blast*
  qed
qed

lemma *Chn-LHS-eq*:
shows *Chn LHS* =
    ⟦⟦*HHfgh-HfHgh.chine*⟧⟧ ⊙ ⟦⟦*THfgh-HHfgh.chine*⟧⟧ ⊙ ⟦⟦*TTfgh-THfgh.chine*⟧⟧
proof −
  have *Chn LHS* = ⟦⟦*HHfgh-HfHgh.chine*⟧⟧ ⊙ ⟦⟦*THfgh-HHfgh.chine*⟧⟧ ⊙

Span.chine-hcomp (*CMP f g*) (*SPN h*)
proof −
have *Chn LHS* = *Chn* (*SPN* a[*f*, *g*, *h*]) ⊙ *Chn* (*CMP* (*f* ⋆ *g*) *h*) ⊙
Chn (*CMP f g* ∘ *SPN h*)
using *fg gh arr-LHS* Φ*.map-simp-ide* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *Span.Chn-vcomp*
by *auto*
moreover have *Chn* (*SPN* a[*f*, *g*, *h*]) = *Maps.CLS HHfgh-HfHgh.chine*
using *fg gh SPN-def* $VVV.arr\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *spn-def* α*-def* **by** *simp*
moreover have *Chn* (*CMP* (*f* ⋆ *g*) *h*) = *Maps.CLS THfgh-HHfgh.chine*
using *fg gh CMP-def THfgh.cmp-def* **by** *simp*
moreover have *Chn* (*CMP f g* ∘ *SPN h*) = *Span.chine-hcomp* (*CMP f g*) (*SPN h*)
using *fg gh Span.hcomp-def* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have *Span.chine-hcomp* (*CMP f g*) (*SPN h*) = ⟦⟦*TTfgh-THfgh.chine*⟧⟧
proof −
have *Span.chine-hcomp* (*CMP f g*) (*SPN h*) =
Maps.tuple
(⟦⟦*Tfg.cmp*⟧⟧ ⊙ $Maps.PRJ_1$ ⟦⟦tab_0 *g* ⋆ *Tfg.ϱσ.*p_0⟧⟧ ⟦⟦tab_1 *h*⟧⟧)
⟦⟦tab_0 (*f* ⋆ *g*)⟧⟧ ⟦⟦tab_1 *h*⟧⟧
(⟦⟦*spn h*⟧⟧ ⊙ $Maps.PRJ_0$ ⟦⟦tab_0 *g* ⋆ *Tfg.ϱσ.*p_0⟧⟧ ⟦⟦tab_1 *h*⟧⟧)
proof −
have ⟦⟦tab_0 *g*⟧⟧ ⊙ ⟦⟦*Tfg.ϱσ.*p_0⟧⟧ = ⟦⟦tab_0 *g* ⋆ *Tfg.ϱσ.*p_0⟧⟧
using *fg isomorphic-reflexive*
Maps.comp-CLS [*of* tab_0 *g Tfg.ϱσ.*p_0 tab_0 *g* ⋆ *Tfg.ϱσ.*p_0]
by *simp*
moreover have *span-in-category.apex Maps.comp*
(|*Leg0* = ⟦⟦tab_0 *h*⟧⟧, *Leg1* = ⟦⟦tab_1 *h*⟧⟧|) =
⟦⟦*spn h*⟧⟧
proof −
interpret *h*: *span-in-category Maps.comp* ‹(|*Leg0* = ⟦⟦tab_0 *h*⟧⟧, *Leg1* = ⟦⟦tab_1 *h*⟧⟧|)›
using *h.determines-span* **by** *simp*
interpret *dom-h*: *identity-arrow-of-tabulations-in-maps V H* a i *src trg*
‹*dom h*› ‹*tab-of-ide* (*dom h*)› ‹tab_0 (*dom h*)› ‹tab_1 (*dom h*)›
‹*cod h*› ‹*tab-of-ide* (*cod h*)› ‹tab_0 (*cod h*)› ‹tab_1 (*cod h*)›
h
by (*simp add*: *h.is-arrow-of-tabulations-in-maps*
identity-arrow-of-tabulations-in-maps.intro
identity-arrow-of-tabulations-in-maps-axioms.intro)
have *Maps.arr h.leg0*
using *h.leg-simps*(*1*) **by** *simp*
hence *Maps.dom h.leg0* = ⟦⟦*dom-h.chine*⟧⟧
using *Maps.dom-char Maps.CLS-in-hom*
apply *simp*
proof −
have *h.is-induced-map* (*src* (tab_0 *h*))
using *h.is-induced-map-iff dom-h.*Δ*-eq-*ϱ *h.apex-is-induced-by-cell* **by** *force*
hence *src* (tab_0 *h*) ≅ *h.chine*
using *h.chine-is-induced-map h.induced-map-unique* **by** *simp*

```
      thus ⟦src (tab₀ h)⟧ = ⟦h.chine⟧
        using iso-class-eqI by simp
    qed
    thus ?thesis
      using h.apex-def spn-def by simp
  qed
  ultimately show ?thesis
    unfolding Span.chine-hcomp-def
    using fg gh CMP-def Tfg.ϱσ.prj-char Span.hcomp-def by simp
qed
also have ... = ⟦⟦TTfgh-THfgh.chine⟧⟧
proof −
  have ⟦⟦TTfgh-THfgh.chine⟧⟧ =
        Maps.tuple ⟦⟦Tfg-Hfg.chine ⋆ TTfgh.p₁⟧⟧
                   ⟦⟦tab₀ (f ⋆ g)⟧⟧ ⟦⟦tab₁ h⟧⟧
                 ⟦⟦h.chine ⋆ TTfgh.p₀⟧⟧
    using TTfgh-THfgh.CLS-chine by simp
  also have ... =
            Maps.tuple (⟦⟦Tfg-Hfg.chine⟧⟧ ⊙ ⟦⟦TTfgh.p₁⟧⟧)
                       ⟦⟦tab₀ (f ⋆ g)⟧⟧ ⟦⟦tab₁ h⟧⟧
                     (⟦⟦h.chine⟧⟧ ⊙ ⟦⟦TTfgh.p₀⟧⟧)
  proof −
    have ⟦⟦Tfg-Hfg.chine ⋆ TTfgh.p₁⟧⟧ = ⟦⟦Tfg-Hfg.chine⟧⟧ ⊙ ⟦⟦TTfgh.p₁⟧⟧
    proof −
      have is-left-adjoint TTfgh.p₁
        using Tfg.ϱσ.T0.antipar(2) THfgh.composable by simp
      moreover have Tfg-Hfg.chine ⋆ TTfgh.p₁ ≅ Tfg-Hfg.chine ⋆ TTfgh.p₁
        using TTfgh-THfgh.prj-chine(2) isomorphic-reflexive isomorphic-implies-hpar(2)
        by blast
      ultimately show ?thesis
        using Tfg-Hfg.is-map
              Maps.comp-CLS [of Tfg-Hfg.chine TTfgh.p₁ Tfg-Hfg.chine ⋆ TTfgh.p₁]
        by simp
    qed
    moreover have ⟦⟦h.chine ⋆ TTfgh.p₀⟧⟧ = ⟦⟦h.chine⟧⟧ ⊙ ⟦⟦TTfgh.p₀⟧⟧
    proof −
      have is-left-adjoint TTfgh.p₀
        by (simp add: Tfg.ϱσ.T0.antipar(2) THfgh.composable)
      moreover have h.chine ⋆ TTfgh.p₀ ≅ h.chine ⋆ TTfgh.p₀
        using TTfgh-THfgh.prj-chine(1) isomorphic-reflexive isomorphic-implies-hpar(2)
        by blast
      ultimately show ?thesis
        using h.is-map Maps.comp-CLS [of h.chine TTfgh.p₀ h.chine ⋆ TTfgh.p₀]
        by simp
    qed
    ultimately show ?thesis by argo
  qed
  also have ... =
            Maps.tuple (⟦⟦Tfg.cmp⟧⟧ ⊙ Maps.PRJ₁ ⟦⟦tab₀ g ⋆ Tfg.ϱσ.p₀⟧⟧ ⟦⟦tab₁ h⟧⟧)
```

$[\![[\![tab_0\ (f \star g)]\!]\ [\![tab_1\ h]\!]]\!]$
$([\![[\![spn\ h]\!]] \odot Maps.PRJ_0\ [\![[\![tab_0\ g \star Tfg.\varrho\sigma.p_0]\!]\ [\![tab_1\ h]\!]]\!])$
using *Tfg.cmp-def spn-def TTfgh.prj-char* **by** *simp*
finally show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
qed

abbreviation *tuple-BC*
where *tuple-BC* $\equiv$ *Maps.tuple SPN-fgh.Prj*$_{01}$ *SPN-fgh.ν.leg0 SPN-fgh.π.leg1 SPN-fgh.Prj*$_0$

abbreviation *tuple-ABC*
where *tuple-ABC* $\equiv$ *Maps.tuple SPN-fgh.Prj*$_{11}$
SPN-fgh.μ.leg0
(*SPN-fgh.ν.leg1* $\odot$ *SPN-fgh.νπ.prj*$_1$)
tuple-BC

abbreviation *tuple-BC′*
where *tuple-BC′* $\equiv$ *Maps.tuple* $[\![[\![Tfg.\varrho\sigma.p_0 \star TTfgh.p_1]\!]\ [\![tab_0\ g]\!]\ [\![tab_1\ h]\!]\ [\![TTfgh.p_0]\!]]\!]$

abbreviation *tuple-ABC′*
where *tuple-ABC′* $\equiv$ *Maps.tuple* $[\![[\![Tfg.\varrho\sigma.p_1 \star TTfgh.p_1]\!]$
$[\![tab_0\ f]\!]\ [\![tab_1\ g \star Tgh.\varrho\sigma.p_1]\!]]\!]$
tuple-BC′

lemma *csq*:
shows *Maps.commutative-square SPN-fgh.ν.leg0 SPN-fgh.π.leg1*
SPN-fgh.Prj$_{01}$ *SPN-fgh.Prj*$_0$
and *Maps.commutative-square SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* $\odot$ *SPN-fgh.νπ.prj*$_1$)
SPN-fgh.Prj$_{11}$ *tuple-BC*
proof −
show *1*: *Maps.commutative-square SPN-fgh.ν.leg0 SPN-fgh.π.leg1*
SPN-fgh.Prj$_{01}$ *SPN-fgh.Prj*$_0$
proof
show *Maps.cospan SPN-fgh.ν.leg0 SPN-fgh.π.leg1*
using *SPN-fgh.νπ.legs-form-cospan*(*1*) **by** *simp*
show *Maps.span SPN-fgh.Prj*$_{01}$ *SPN-fgh.Prj*$_0$
using *SPN-fgh.prj-simps*(*2*−*3*,*5*−*6*) **by** *presburger*
show *Maps.dom SPN-fgh.ν.leg0* = *Maps.cod SPN-fgh.Prj*$_{01}$
using *SPN-fgh.prj-simps*(*8*) *SPN-g.dom.is-span SPN-g.dom.leg-simps*(*2*)
by *auto*
show *SPN-fgh.ν.leg0* $\odot$ *SPN-fgh.Prj*$_{01}$ = *SPN-fgh.π.leg1* $\odot$ *SPN-fgh.Prj*$_0$
by (*metis* (*no-types*, *lifting*) *Maps.cod-comp Maps.comp-assoc*
Maps.pullback-commutes′ SPN-fgh.μν.dom.leg-simps(*1*)
SPN-fgh.μν.leg0-composite SPN-fgh.cospan-νπ)
qed
show *Maps.commutative-square*

SPN-fgh.μ.leg0 (*Maps.comp SPN-fgh.ν.leg1 SPN-fgh.νπ.prj*$_1$)
SPN-fgh.Prj$_{11}$ *tuple-BC*
proof
show *Maps.cospan SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
using *fg gh SPN-fgh.prj-simps*(*10*) **by** *blast*
show *Maps.span SPN-fgh.Prj*$_{11}$ *tuple-BC*
using *fg gh 1 Maps.tuple-simps*(*1*) *Maps.tuple-simps*(*2*) *SPN-fgh.prj-simps*(*1*)
SPN-fgh.prj-simps(*4*) *SPN-fgh.prj-simps*(*5*)
by *presburger*
show *Maps.dom SPN-fgh.μ.leg0* = *Maps.cod SPN-fgh.Prj*$_{11}$
using *fg gh SPN-f.dom.leg-simps*(*2*) *SPN-fgh.prj-simps*(*7*) **by** *auto*
show *SPN-fgh.μ.leg0* ⊙ *SPN-fgh.Prj*$_{11}$ = (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$) ⊙ *tuple-BC*
using *1 fg gh Maps.comp-assoc Maps.prj-tuple*
by (*metis* (*no-types*, *lifting*) *Maps.pullback-commutes′ SPN-fgh.cospan-μν*)
qed
qed

lemma *tuple-ABC-eq-ABC′*:
shows *tuple-BC* = *tuple-BC′*
and *tuple-ABC* = *tuple-ABC′*
proof −
have *SPN-fgh.Prj*$_{11}$ = ⟦⟦*Tfg.ϱσ.p*$_1$ ⋆ *TTfgh.p*$_1$⟧⟧
using *prj-char* **by** *simp*
moreover have *SPN-fgh.μ.leg0* = ⟦⟦*tab*$_0$ *f*⟧⟧
by *simp*
moreover have *SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$ = ⟦⟦*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$⟧⟧
using *Tgh.ϱσ.prj-char isomorphic-reflexive Tgh.composable*
Maps.comp-CLS [*of tab*$_1$ *g Tgh.ϱσ.p*$_1$ *tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$]
by (*simp add*: *g.T0.antipar*(*2*))
moreover show *tuple-BC* = *tuple-BC′*
using *prj-char Tfg.ϱσ.prj-char* **by** *simp*
ultimately show *tuple-ABC* = *tuple-ABC′*
by *argo*
qed

lemma *tuple-BC-in-hom*:
shows *Maps.in-hom tuple-BC* (*Maps.MkIde* (*src TTfgh.p*$_0$)) (*Maps.MkIde* (*src Tgh.ϱσ.p*$_0$))
proof
show *1*: *Maps.arr tuple-BC*
using *csq*(*1*) **by** *simp*
have *2*: *Maps.commutative-square*
⟦⟦*tab*$_0$ *g*⟧⟧ ⟦⟦*tab*$_1$ *h*⟧⟧ ⟦⟦*Tfg.ϱσ.p*$_0$ ⋆ *TTfgh.p*$_1$⟧⟧ ⟦⟦*TTfgh.p*$_0$⟧⟧
by (*metis Tfg.S*$_0$*-def Tfg.span-legs-eq*(*3*) *Tgh.S*$_1$*-def Tgh.span-legs-eq*(*4*) *csq*(*1*)
prj-char(*2*) *prj-char*(*3*))
show *Maps.dom tuple-BC* = *Maps.MkIde* (*src TTfgh.p*$_0$)
proof −
have *Maps.dom tuple-BC′* = *Maps.dom* ⟦⟦*Tfg.ϱσ.p*$_0$ ⋆ *TTfgh.p*$_1$⟧⟧
using *2 Maps.tuple-simps* **by** *simp*
also have ... = *Chn* (*Span.hcomp* (*Span.hcomp* (*SPN f*) (*SPN g*)) (*SPN h*))

```
      using Maps.dom-char
      by (metis SPN-fgh.prj-simps(5) prj-char(2))
    also have ... = Maps.MkIde (src TTfgh.p0)
      using 1 fg gh Maps.dom-char csq(1) prj-char(3) tuple-ABC-eq-ABC'(1)
            Maps.Dom.simps(1) Maps.tuple-simps(2) SPN-fgh.prj-simps(3,5−6)
      by presburger
    finally have Maps.dom tuple-BC' = Maps.MkIde (src TTfgh.p0)
      by blast
    thus ?thesis
      using tuple-ABC-eq-ABC' by simp
  qed
  show Maps.cod tuple-BC = Maps.MkIde (src Tgh.ϱσ.p0)
  proof −
    have Maps.cod tuple-BC' = Maps.pbdom ⟦⟦tab0 g⟧⟧ ⟦⟦tab1 h⟧⟧
      using 1 2 fg gh Maps.tuple-in-hom by blast
    also have ... = Maps.MkIde (src Tgh.ϱσ.p0)
      using 1 2 fg gh Maps.pbdom-def
      by (metis (no-types, lifting) SPN.preserves-ide SPN-fgh.νπ.are-identities(2)
          SPN-fgh.νπ.composable Span.chine-hcomp-ide-ide Tfg.S0-def Tfg.span-legs-eq(3)
          Tgh.S1-def Tgh.chine-hcomp-SPN-SPN Tgh.span-legs-eq(4) g.is-ide)
    finally show ?thesis
      using tuple-ABC-eq-ABC' by simp
  qed
qed

lemma tuple-ABC-in-hom:
shows Maps.in-hom tuple-ABC (Maps.MkIde (src TTfgh.p0)) (Maps.MkIde (src TfTgh.p0))
proof
  show 1: Maps.arr tuple-ABC
    using SPN-fgh.chine-assoc-def SPN-fgh.chine-assoc-in-hom by auto
  show Maps.dom tuple-ABC = Maps.MkIde (src TTfgh.p0)
  proof −
    have Maps.dom tuple-ABC = Maps.dom SPN-fgh.chine-assoc
      by (simp add: SPN-fgh.chine-assoc-def)
    also have ... = Chn ((SPN f ∘ SPN g) ∘ SPN h)
      using SPN-fgh.chine-assoc-in-hom by blast
    also have ... = Maps.MkIde (src TTfgh.p0)
      by (metis (lifting) Maps.Dom.simps(1) Maps.dom-char SPN-fgh.prj-simps(3)
          SPN-fgh.prj-simps(6) prj-char(3))
    finally show ?thesis by blast
  qed
  show Maps.cod tuple-ABC = Maps.MkIde (src TfTgh.p0)
  proof −
    have Maps.cod tuple-ABC = Maps.cod SPN-fgh.chine-assoc
      by (simp add: SPN-fgh.chine-assoc-def)
    also have 1: ... = Chn (SPN f ∘ SPN g ∘ SPN h)
      using SPN-fgh.chine-assoc-in-hom by blast
    also have ... = Maps.MkIde (src TfTgh.p0)
      by (metis (lifting) Maps.Dom.simps(1) Maps.cod-char Maps.seq-char
```

SPN-fgh.prj-chine-assoc(*1*) *SPN-fgh.prj-simps*(*1*) *TfTgh.leg1-in-hom*(*1*)
TfTgh-TfTgh.u-in-hom 1 in-hhomE prj-char(*4*) *src-hcomp*)
finally show *?thesis* **by** *argo*
qed
qed

lemma *Chn-RHS-eq*:
shows *Chn RHS* = ⟦⟦*TfHgh-HfHgh.chine*⟧⟧ ⊙ ⟦⟦*TfTgh-TfHgh.chine*⟧⟧ ⊙ *tuple-ABC′*
proof −
have *Chn RHS* =
Chn (Φ.*map* (*f*, *g* ⋆ *h*)) ⊙ *Chn* (*SPN f* ∘ Φ.*map* (*g*, *h*)) ⊙
Chn (*Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*))
proof −
have *Chn RHS* = *Chn* (Φ.*map* (*f*, *g* ⋆ *h*)) ⊙
Chn ((*SPN f* ∘ Φ.*map* (*g*, *h*)) · *Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*))
using *arr-RHS Span.vcomp-eq Span.Chn-vcomp* **by** *blast*
also have ... = *Chn* (Φ.*map* (*f*, *g* ⋆ *h*)) ⊙ *Chn* (*SPN f* ∘ Φ.*map* (*g*, *h*)) ⊙
Chn (*Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*))
proof −
have *Span.seq* (*SPN f* ∘ Φ.*map* (*g*, *h*)) (*Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*))
using *arr-RHS* **by** *auto*
thus *?thesis*
using *fg gh Span.vcomp-eq* [*of SPN f* ∘ Φ.*map* (*g*, *h*)
Span.assoc (*SPN f*) (*SPN g*) (*SPN h*)]
by *simp*
qed
finally show *?thesis* **by** *blast*
qed
moreover have *Chn* (Φ.*map* (*f*, *g* ⋆ *h*)) = ⟦⟦*TfHgh-HfHgh.chine*⟧⟧
using *arr-RHS fg gh* Φ.*map-simp-ide* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *CMP-def TfHgh.cmp-def*
by *simp*
moreover have *Chn* (*SPN f* ∘ Φ.*map* (*g*, *h*)) = *Span.chine-hcomp* (*SPN f*) (*CMP g h*)
using *fg gh Span.hcomp-def* Φ.*map-simp-ide* $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$ *SPN.FF-def*
by *simp*
moreover have *Chn* (*Span.assoc* (*SPN f*) (*SPN g*) (*SPN h*)) = *tuple-ABC*
using *fg gh Span.*α*-ide* $VVV.ide\text{-}char_{SbC}$ $VVV.arr\text{-}char_{SbC}$ $VV.ide\text{-}char_{SbC}$ $VV.arr\text{-}char_{SbC}$
*SPN-fgh.chine-assoc-def Span.*α*-def*
by *simp*
moreover have *Span.chine-hcomp* (*SPN f*) (*CMP g h*) = ⟦⟦*TfTgh-TfHgh.chine*⟧⟧
proof −
have *Span.chine-hcomp* (*SPN f*) (*CMP g h*) =
Maps.tuple
(⟦⟦*f.chine*⟧⟧ ⊙ ⟦⟦$TfTgh.p_1$⟧⟧)
⟦⟦tab_0 *f*⟧⟧ ⟦⟦tab_1 (*g* ⋆ *h*)⟧⟧
(⟦⟦*Tgh-Hgh.chine*⟧⟧ ⊙ ⟦⟦$TfTgh.p_0$⟧⟧)
proof −
interpret *f*: *span-in-category Maps.comp*
‹⦇*Leg0* = *Maps.MkArr* (*src* (tab_0 *f*)) (*trg g*) ⟦tab_0 *f*⟧,
Leg1 = *Maps.MkArr* (*src* (tab_0 *f*)) (*trg f*) ⟦tab_1 *f*⟧⦈›

```
      using f.determines-span
      by (simp add: Tfg.composable)
    interpret f: arrow-of-tabulations-in-maps V H a i src trg
                f f.tab ‹tab₀ f› ‹tab₁ f› f f.tab ‹tab₀ f› ‹tab₁ f› f
      using f.is-arrow-of-tabulations-in-maps by simp
    have f.apex = Maps.CLS f.chine
    proof (intro Maps.arr-eqI)
      show Maps.arr f.apex by simp
      show Maps.arr ⟦⟦f.chine⟧⟧
        using Maps.CLS-in-hom f.is-map by blast
      show Maps.Dom f.apex = Maps.Dom ⟦⟦f.chine⟧⟧
        using f.apex-def Tfg.RS-simps(2) Tfg.R₀-def Tfg.composable by auto
      show Maps.Cod f.apex = Maps.Cod ⟦⟦f.chine⟧⟧
        using f.apex-def Tfg.RS-simps(2) Tfg.R₀-def Tfg.composable by auto
      show Maps.Map f.apex = Maps.Map ⟦⟦f.chine⟧⟧
      proof −
        have Maps.Map f.apex = ⟦src (tab₀ f)⟧
          using f.apex-def Maps.dom-char Tfg.RS-simps(2) Tfg.R₀-def Tfg.composable
          by auto
        also have ... = ⟦f.chine⟧
        proof (intro iso-class-eqI)
          have f.is-induced-map (src (tab₀ f))
            using f.apex-is-induced-by-cell comp-cod-arr by auto
          thus src (tab₀ f) ≅ f.chine
            using f.induced-map-unique f.chine-is-induced-map by simp
        qed
        also have ... = Maps.Map ⟦⟦f.chine⟧⟧
          by simp
        finally show ?thesis by simp
      qed
    qed
    thus ?thesis
      unfolding Span.chine-hcomp-def
      using fg gh CMP-def Tgh.ϱσ.prj-char Span.hcomp-def isomorphic-reflexive
            Maps.comp-CLS [of tab₁ g Tgh.ϱσ.p₁ tab₁ g ⋆ Tgh.ϱσ.p₁]
            Tgh.cmp-def TfTgh.prj-char
      by simp
  qed
  also have ... = Maps.tuple ⟦⟦f.chine ⋆ TfTgh.p₁⟧⟧
                              ⟦⟦tab₀ f⟧⟧ ⟦⟦tab₁ (g ⋆ h)⟧⟧
                              ⟦⟦Tgh-Hgh.chine ⋆ TfTgh.p₀⟧⟧
  using isomorphic-reflexive TfHgh.composable f.is-map TfHgh.composable Tgh-Hgh.is-map
        Maps.comp-CLS [of f.chine TfTgh.p₁ f.chine ⋆ TfTgh.p₁]
        Maps.comp-CLS [of Tgh-Hgh.chine TfTgh.p₀ Tgh-Hgh.chine ⋆ TfTgh.p₀]
    by auto
  also have ... = ⟦⟦TfTgh-TfHgh.chine⟧⟧
    using TfTgh-TfHgh.CLS-chine by simp
  finally show ?thesis by blast
qed
```

ultimately have *Chn RHS* $= [\![[\![\mathit{TfHgh\text{-}HfHgh.chine}]\!] \odot [\![\mathit{TfTgh\text{-}TfHgh.chine}]\!] \odot \mathit{tuple\text{-}ABC}$
  by *simp*
 also have ... $= [\![[\![\mathit{TfHgh\text{-}HfHgh.chine}]\!] \odot [\![\mathit{TfTgh\text{-}TfHgh.chine}]\!] \odot \mathit{tuple\text{-}ABC'}$
  using *tuple-ABC-eq-ABC'* **by** *simp*
 finally show *?thesis* **by** *simp*
qed

interpretation g_0h_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* ‹tab_1 *h*› ‹tab_0 *g*›
 using *gh* **by** *unfold-locales auto*
interpretation f_0g_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* ‹tab_1 *g*› ‹tab_0 *f*›
 using *fg* **by** *unfold-locales auto*
interpretation f_0gh_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg*
  ‹tab_1 $g \star \mathit{Tgh.}\varrho\sigma.p_1$› ‹$\mathit{tab}_0$ *f*›
 using *fg gh Tgh.*$\varrho\sigma$*.leg1-is-map*
 by *unfold-locales auto*
interpretation fg_0h_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg*
  ‹tab_1 *h*› ‹tab_0 $g \star \mathit{Tfg.}p_0$›
 using *TTfgh.*r_0s_1*-is-cospan* **by** *simp*

lemma *src-tab-eq*:
shows $(\mathrm{a}^{-1}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0) \cdot$
  TfTgh.composite-cell TTfgh-TfTgh.chine TTfgh-TfTgh.the-ϑ $\cdot$ *TTfgh-TfTgh.the-*ν $=$
  TTfgh.tab
proof −
 have *TfTgh.composite-cell TTfgh-TfTgh.chine TTfgh-TfTgh.the-*ϑ $\cdot$ *TTfgh-TfTgh.the-*ν $=$
  $(\mathrm{a}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0) \cdot$ *TTfgh.tab*
  unfolding *TTfgh.tab-def*
  using *TTfgh-TfTgh.chine-is-induced-map TTfgh.tab-def TTfgh-TfTgh.*Δ*-simps*(*4*)
  by *auto*
 moreover have *iso* $(\mathrm{a}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0)$
  by (*simp add*: *fg gh*)
 moreover have *inv* $(\mathrm{a}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0) = \mathrm{a}^{-1}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0$
  using *fg gh* **by** *simp*
 ultimately show *?thesis*
  using *TTfgh-TfTgh.*Δ*-simps*(*1*)
   invert-side-of-triangle(*1*)
   [*of TfTgh.composite-cell TTfgh-TfTgh.chine TTfgh-TfTgh.the-*ϑ $\cdot$ *TTfgh-TfTgh.the-*ν
    $\mathrm{a}[f,\ g,\ h] \star \mathit{tab}_0\ h \star \mathit{TTfgh.}p_0$ *TTfgh.tab*]
  by *argo*
qed

We need to show that the associativity isomorphism (defined in terms of tupling) coincides with *TTfgh-TfTgh.chine* (defined in terms of tabulations). In order to do this, we need to know how the latter commutes with projections. That is the purpose of the following lemma. Unfortunately, it requires some lengthy calculations, which I haven't seen any way to avoid.

lemma *prj-chine*:
shows $[\![[\![\mathit{TfTgh.}p_1 \star \mathit{TTfgh\text{-}TfTgh.chine}]\!]]\!] = [\![[\![\mathit{Tfg.}p_1 \star \mathit{TTfgh.}p_1]\!]]\!]$
and $[\![[\![\mathit{Tgh.}p_1 \star \mathit{TfTgh.}p_0 \star \mathit{TTfgh\text{-}TfTgh.chine}]\!]]\!] = [\![[\![\mathit{Tfg.}p_0 \star \mathit{TTfgh.}p_1]\!]]\!]$

and $[\![\![Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]\!] = [\![\![TTfgh.p_0]\!]\!]$
proof −
  have *1*: *ide* $TfTgh.p_1$
    by (*simp add*: *TfTgh.composable*)
  have *2*: *ide* *TTfgh-TfTgh.chine*
    by *simp*
  have *3*: *src* $TfTgh.p_1$ = *trg* *TTfgh-TfTgh.chine*
    using *TTfgh-TfTgh.chine-in-hom*(*1*) **by** *simp*
  have *4*: *src* (tab_1 f) = *trg* $TfTgh.p_1$
    using *TfTgh.leg1-simps*(*2*) **by** *blast*

The required isomorphisms will each be established via *T2*, using the equation *src-tab-eq* (associativities omitted from diagram):

=

There is a sequential dependence between the proofs, such as we have already seen for *horizontal-composite-of-arrows-of-tabulations-in-maps.prj-chine*.

define u_f **where** $u_f = g \star h \star tab_0\ h \star TTfgh.p_0$
define w_f **where** $w_f = Tfg.p_1 \star TTfgh.p_1$

define $w_f{}'$ **where** $w_f{}' = \mathit{TfTgh.p_1} \star \mathit{TTfgh\text{-}TfTgh.chine}$
define ϑ_f
where $\vartheta_f = (g \star \mathrm{a}[h,\ \mathit{tab}_0\ h,\ \mathit{TTfgh.p_0}]) \cdot (g \star \mathit{h.tab} \star \mathit{TTfgh.p_0}) \cdot (g \star \mathit{fg_0h_1.\varphi}) \cdot$
$\mathrm{a}[g,\ \mathit{tab}_0\ g \star \mathit{Tfg.p_0},\ \mathit{TTfgh.p_1}] \cdot (\mathrm{a}[g,\ \mathit{tab}_0\ g,\ \mathit{Tfg.p_0}] \star \mathit{TTfgh.p_1}) \cdot$
$((\mathit{g.tab} \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1}) \cdot (\mathit{f_0g_1.\varphi} \star \mathit{TTfgh.p_1}) \cdot$
$\mathrm{a}^{-1}[\mathit{tab}_0\ f,\ \mathit{Tfg.p_1},\ \mathit{TTfgh.p_1}]$
define $\vartheta_f{}'$
where $\vartheta_f{}' = (g \star h \star \mathit{TTfgh\text{-}TfTgh.the\text{-}\vartheta}) \cdot$
$\mathit{can}\ (\langle g\rangle \star \langle h\rangle \star ((\langle \mathit{tab}_0\ h\rangle \star \langle \mathit{Tgh.p_0}\rangle) \star \langle \mathit{TfTgh.p_0}\rangle) \star \langle \mathit{TTfgh\text{-}TfTgh.chine}\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle \mathit{tab}_0\ h\rangle) \star \langle \mathit{Tgh.p_0}\rangle) \star \langle \mathit{TfTgh.p_0}\rangle) \star \langle \mathit{TTfgh\text{-}TfTgh.chine}\rangle) \cdot$
$(((g \star \mathit{h.tab} \star \mathit{Tgh.p_0}) \star \mathit{TfTgh.p_0}) \star \mathit{TTfgh\text{-}TfTgh.chine}) \cdot$
$(((g \star \mathit{g_0h_1.\varphi}) \star \mathit{TfTgh.p_0}) \star \mathit{TTfgh\text{-}TfTgh.chine}) \cdot$
$((\mathrm{a}[g,\ \mathit{tab}_0\ g,\ \mathit{Tgh.p_1}] \star \mathit{TfTgh.p_0}) \star \mathit{TTfgh\text{-}TfTgh.chine}) \cdot$
$(((\mathit{g.tab} \star \mathit{Tgh.p_1}) \star \mathit{TfTgh.p_0}) \star \mathit{TTfgh\text{-}TfTgh.chine}) \cdot$
$(\mathit{f_0gh_1.\varphi} \star \mathit{TTfgh\text{-}TfTgh.chine}) \cdot$
$\mathrm{a}^{-1}[\mathit{tab}_0\ f,\ \mathit{TfTgh.p_1},\ \mathit{TTfgh\text{-}TfTgh.chine}]$
define β_f
where $\beta_f = \mathrm{a}[\mathit{tab}_1\ f,\ \mathit{TfTgh.p_1},\ \mathit{TTfgh\text{-}TfTgh.chine}] \cdot$
$\mathit{TTfgh\text{-}TfTgh.the\text{-}\nu} \cdot$
$\mathrm{a}^{-1}[\mathit{tab}_1\ f,\ \mathit{Tfg.p_1},\ \mathit{TTfgh.p_1}]$
have w_f: *ide* w_f
using w_f*-def* $\mathit{fg_0h_1.p_1\text{-}simps}$ **by** *auto*
have w_f*-is-map*: *is-left-adjoint* w_f
using w_f*-def* $\mathit{fg_0h_1.p_1\text{-}simps}$
by (*simp add*: *left-adjoints-compose*)
have $w_f{}'$: *ide* $w_f{}'$
unfolding $w_f{}'$*-def* **by** *simp*
have $w_f{}'$*-is-map*: *is-left-adjoint* $w_f{}'$
unfolding $w_f{}'$*-def*
using *3* *TTfgh-TfTgh.is-map* $\mathit{f_0gh_1.leg1\text{-}is\text{-}map}$
by (*simp add*: *left-adjoints-compose*)
have ϑ_f: «$\vartheta_f : \mathit{tab}_0\ f \star w_f \Rightarrow u_f$»
proof (*unfold* ϑ_f*-def* w_f*-def* u_f*-def*, *intro comp-in-homI*)
show «$\mathrm{a}^{-1}[\mathit{tab}_0\ f,\ \mathit{Tfg.p_1},\ \mathit{TTfgh.p_1}]$:
$\mathit{tab}_0\ f \star \mathit{Tfg.p_1} \star \mathit{TTfgh.p_1} \Rightarrow (\mathit{tab}_0\ f \star \mathit{Tfg.p_1}) \star \mathit{TTfgh.p_1}$»
using $\mathit{f_0g_1.leg1\text{-}simps}$ $\mathit{fg_0h_1.p_1\text{-}simps}$ $\mathit{f_0g_1.cospan}$ $\mathit{g_0h_1.cospan}$ **by** *auto*
show «$\mathit{f_0g_1.\varphi} \star \mathit{TTfgh.p_1}$:
$(\mathit{tab}_0\ f \star \mathit{Tfg.p_1}) \star \mathit{TTfgh.p_1} \Rightarrow (\mathit{tab}_1\ g \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1}$»
using $\mathit{f_0g_1.\varphi\text{-}in\text{-}hom}(2)$ $\mathit{Tfg.\varrho\sigma.T0.antipar}(1)$
by (*intro hcomp-in-vhom*, *auto*)
show «$(\mathit{g.tab} \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1}$:
$(\mathit{tab}_1\ g \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1} \Rightarrow ((g \star \mathit{tab}_0\ g) \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1}$»
using $\mathit{Tfg.\varrho\sigma.T0.antipar}(1)$
by (*intro hcomp-in-vhom*, *auto*)
show «$\mathrm{a}[g,\ \mathit{tab}_0\ g,\ \mathit{Tfg.p_0}] \star \mathit{TTfgh.p_1}$:
$((g \star \mathit{tab}_0\ g) \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1} \Rightarrow (g \star \mathit{tab}_0\ g \star \mathit{Tfg.p_0}) \star \mathit{TTfgh.p_1}$»
using $\mathit{fg_0h_1.p_1\text{-}simps}$
by (*intro hcomp-in-vhom*, *auto*)
show «$\mathrm{a}[g,\ \mathit{tab}_0\ g \star \mathit{Tfg.p_0},\ \mathit{TTfgh.p_1}]$:

$(g \star tab_0\ g \star Tfg.p_0) \star TTfgh.p_1 \Rightarrow g \star (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1$»
using $fg_0h_1.p_1$-*simps* **by** *auto*
show «$g \star fg_0h_1.\varphi : g \star (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1 \Rightarrow g \star tab_1\ h \star TTfgh.p_0$»
using $fg_0h_1.\varphi$-*in-hom* $fg_0h_1.p_1$-*simps*
by (*intro hcomp-in-vhom*, *auto*)
show «$g \star h.tab \star TTfgh.p_0 : g \star tab_1\ h \star TTfgh.p_0 \Rightarrow g \star (h \star tab_0\ h) \star TTfgh.p_0$»
using *gh* $fg_0h_1.\varphi$-*in-hom* $fg_0h_1.p_1$-*simps*
by (*intro hcomp-in-vhom*, *auto*)
show «$g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]$:
$g \star (h \star tab_0\ h) \star TTfgh.p_0 \Rightarrow g \star h \star tab_0\ h \star TTfgh.p_0$»
using *gh* $fg_0h_1.\varphi$-*in-hom* $fg_0h_1.p_1$-*simps*
by (*intro hcomp-in-vhom*, *auto*)
qed
have $\vartheta_f{}'$: «$\vartheta_f{}' : tab_0\ f \star w_f{}' \Rightarrow u_f$»
proof (*unfold* $\vartheta_f{}'$-*def* $w_f{}'$-*def* u_f-*def*, *intro comp-in-homI*)
show «$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]$:
$tab_0\ f \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine \Rightarrow (tab_0\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine$»
using *1 2 3 4 assoc'-in-hom*(*2*) *f.ide-u f.leg1-simps*(*3*) **by** *auto*
show «$f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine$:
$(tab_0\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine \Rightarrow$
$((tab_1\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.\varphi$-*in-hom*(*2*)
by (*intro hcomp-in-vhom*, *auto*)
show «$((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$:
$((tab_1\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow (((g \star tab_0\ g) \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.cospan$ $g_0h_1.cospan$
by (*intro hcomp-in-vhom*, *auto*)
show «$(\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$:
$(((g \star tab_0\ g) \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow ((g \star tab_0\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.cospan$ $g_0h_1.cospan$
by (*intro hcomp-in-vhom*, *auto*)
show «$((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$:
$((g \star tab_0\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow ((g \star tab_1\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.cospan$ $g_0h_1.cospan$ $g_0h_1.\varphi$-*in-hom*(*2*)
by (*intro hcomp-in-vhom*, *auto*)
show «$((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$:
$((g \star tab_1\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow ((g \star (h \star tab_0\ h) \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.cospan$ $g_0h_1.cospan$
by (*intro hcomp-in-vhom*, *auto*)
show «$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$:
$((g \star (h \star tab_0\ h) \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow g \star h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.cospan$ $g_0h_1.cospan$ **by** *auto*
show «$g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta$:

$g \star h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow g \star h \star tab_0\ h \star TTfgh.p_0$»
using $f_0gh_1.cospan\ g_0h_1.cospan\ TTfgh\text{-}TfTgh.the\text{-}\vartheta\text{-}in\text{-}hom$
by (*intro hcomp-in-vhom*, *auto*)
qed
have β_f: «$\beta_f : tab_1\ f \star w_f \Rightarrow tab_1\ f \star w_f{}'$»
proof (*unfold* β_f*-def* w_f*-def* $w_f{}'$*-def*, *intro comp-in-homI*)
show «$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$:
$tab_1\ f \star Tfg.p_1 \star TTfgh.p_1 \Rightarrow (tab_1\ f \star Tfg.p_1) \star TTfgh.p_1$»
using *TTfgh.leg1-in-hom*(*2*) *assoc′-in-hom* **by** *auto*
show «$TTfgh\text{-}TfTgh.the\text{-}\nu$:
$(tab_1\ f \star Tfg.p_1) \star TTfgh.p_1 \Rightarrow (tab_1\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine$»
using *TTfgh-TfTgh.the-ν-in-hom TTfgh-TfTgh.the-ν-props* **by** *simp*
show «$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]$:
$(tab_1\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine \Rightarrow tab_1\ f \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine$»
using *1 2 3 4* **by** *auto*
qed
have *iso-*β_f: *iso* β_f
unfolding β_f*-def*
using *1 2 3 4* β_f β_f*-def isos-compose*
apply (*intro isos-compose*)
apply (*metis TTfgh.composable TTfgh.leg1-in-hom*(*2*) *Tfg.ϱσ.T0.antipar*(*2*)
Tfg.ϱσ.T0.ide-right Tfg.ϱσ.leg1-in-hom(*2*) *Tfg-Hfg.u-simps*(*3*)
f.T0.antipar(*2*) *f.T0.ide-right f.ide-leg1* f_0g_1*.cospan g.ide-leg1*
h.ide-leg1 h.leg1-simps(*4*) *hcomp-in-vhomE ide-hcomp*
iso-assoc′ tab_1*-simps*(*1*))
using *TTfgh-TfTgh.the-ν-props*(*2*) *f.ide-leg1 iso-assoc* **by** *blast+*
have u_f: *ide* u_f
using ϑ_f *ide-cod* **by** *blast*
have w_f*-in-hhom*: *in-hhom* w_f (*src* u_f) (*src* (tab_0 *f*))
using u_f w_f u_f*-def* w_f*-def* **by** *simp*
have $w_f{}'$*-in-hhom*: *in-hhom* $w_f{}'$ (*src* u_f) (*src* (tab_0 *f*))
using u_f $w_f{}'$ $w_f{}'$*-def* u_f*-def* **by** *simp*
have *5*: $\exists!\gamma.$ «$\gamma : w_f \Rightarrow w_f{}'$» $\wedge\ \beta_f = tab_1\ f \star \gamma \wedge \vartheta_f = \vartheta_f{}' \cdot (tab_0\ f \star \gamma)$
proof −
have eq_f: $f.composite\text{-}cell\ w_f\ \vartheta_f = f.composite\text{-}cell\ w_f{}'\ \vartheta_f{}' \cdot \beta_f$
proof −

I don't see any alternative here to just grinding out the calculation. The idea is to bring $f.composite\text{-}cell\ w_f\ \vartheta_f$ into a form in which *src-tab-eq* can be applied to eliminate ϑ_f in favor of $\vartheta_f{}'$.

have $f.composite\text{-}cell\ w_f\ \vartheta_f =$
$((f \star g \star \mathrm{a}[h,\ tab_0\ h,\ fg_0h_1.p_0]) \cdot$
$(f \star g \star h.tab \star fg_0h_1.p_0) \cdot$
$(f \star g \star fg_0h_1.\varphi) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star f_0g_1.p_0,\ fg_0h_1.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ f_0g_1.p_0] \star fg_0h_1.p_1) \cdot$
$(f \star (g.tab \star f_0g_1.p_0) \star fg_0h_1.p_1) \cdot$
$(f \star f_0g_1.\varphi \star fg_0h_1.p_1) \cdot$

$(f \star \mathrm{a}^{-1}[tab_0\ f,\ f_0 g_1.p_1,\ fg_0 h_1.p_1])) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ f_0 g_1.p_1 \star fg_0 h_1.p_1] \cdot$
$(f.tab \star f_0 g_1.p_1 \star fg_0 h_1.p_1)$
unfolding *w_f-def ϑ_f-def*
using *$fg_0 h_1.p_1$-simps Tgh.composable whisker-left* **by** *simp*
also have ... =
$(f \star g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(f \star g \star h.tab \star TTfgh.p_0) \cdot$
$(f \star g \star fg_0 h_1.\varphi) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
using *comp-assoc* **by** *simp*
also have ... =
$(\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}^{-1}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$(f \star g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0])) \cdot$
$(f \star g \star h.tab \star TTfgh.p_0) \cdot$
$(f \star g \star fg_0 h_1.\varphi) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
proof −
have $(\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}^{-1}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0]) \cdot$
$(f \star g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) =$
$f \star g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]$
using *fg gh $fg_0 h_1.p_1$-simps comp-cod-arr comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$(f \star g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0])) \cdot$
$(f \star g \star h.tab \star TTfgh.p_0) \cdot$
$(f \star g \star fg_0 h_1.\varphi) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0 g_1.\varphi \star TTfgh.p_1) \cdot$

$(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ (h \star tab_0\ h) \star TTfgh.p_0] \cdot$
$(f \star g \star h.tab \star TTfgh.p_0)) \cdot$
$(f \star g \star fg_0h_1.\varphi) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps comp-assoc*
assoc′-naturality [*of f g* $\mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]$]
by *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_1\ h \star TTfgh.p_0] \cdot$
$(f \star g \star fg_0h_1.\varphi)) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps comp-assoc*
assoc′-naturality [*of f g* $h.tab \star TTfgh.p_0$]
by *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f \star f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$((f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1]) \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$

using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps comp-assoc*
assoc′-naturality [*of f g* $fg_0h_1.\varphi$]
by *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$((f \star f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ tab_0\ f \star Tfg.p_1,\ TTfgh.p_1]) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$
proof −
have $(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot \mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] =$
$\mathrm{a}[f,\ tab_0\ f \star Tfg.p_1,\ TTfgh.p_1] \cdot (\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $(f \star \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot \mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1 \star TTfgh.p_1] =$
$\{\!|(\langle f\rangle \star \mathbf{a}^{-1}[\langle tab_0\ f\rangle,\ \langle Tfg.p_1\rangle,\ \langle TTfgh.p_1\rangle]) \cdot$
$\mathbf{a}[\langle f\rangle,\ \langle tab_0\ f\rangle,\ \langle Tfg.p_1\rangle \star \langle TTfgh.p_1\rangle]|\!\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
a'*-def* α*-def*
by *simp*
also have ... =
$\{\!|\mathbf{a}[\langle f\rangle,\ \langle tab_0\ f\rangle \star \langle Tfg.p_1\rangle,\ \langle TTfgh.p_1\rangle] \cdot$
$(\mathbf{a}[\langle f\rangle,\ \langle tab_0\ f\rangle,\ \langle Tfg.p_1\rangle] \star \langle TTfgh.p_1\rangle) \cdot$
$\mathbf{a}^{-1}[\langle f\rangle \star \langle tab_0\ f\rangle,\ \langle Tfg.p_1\rangle,\ \langle TTfgh.p_1\rangle]|\!\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
by (*intro E.eval-eqI*, *simp-all*)
also have ... = $\mathrm{a}[f,\ tab_0\ f \star Tfg.p_1,\ TTfgh.p_1] \cdot (\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
a'*-def* α*-def*
by *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$

$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$((f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ tab_1\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$((f \star f_0 g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$

proof −
have $(f \star f_0 g_1.\varphi \star TTfgh.p_1) \cdot \mathrm{a}[f,\ tab_0\ f \star Tfg.p_1,\ TTfgh.p_1] =$
$\mathrm{a}[f,\ tab_1\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot ((f \star f_0 g_1.\varphi) \star TTfgh.p_1)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.\varphi$*-in-hom*
assoc-naturality [*of f* $f_0g_1.\varphi$ $TTfgh.p_1$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) \cdot$
$((f \star f_0 g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(f.tab \star Tfg.p_1 \star TTfgh.p_1)$

proof −
have $(f \star (g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot \mathrm{a}[f,\ tab_1\ g \star Tfg.p_0,\ TTfgh.p_1] =$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot ((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.\varphi$*-in-hom*
assoc-naturality [*of f* $g.tab \star Tfg.p_0$ $TTfgh.p_1$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$

$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) \cdot$
$((f \star f_0 g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$((f.tab \star Tfg.p_1) \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *$fg_0h_1.p_1$-simps assoc′-naturality* [*of f.tab Tfg.p_1 TTfgh.p_1*] *comp-assoc*
by *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1)) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1)) \cdot$
$((f \star f_0 g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$((f.tab \star Tfg.p_1) \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $(((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1)) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) =$
$(f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1$
using *fg gh $fg_0h_1.p_1$-simps comp-cod-arr whisker-right comp-assoc-assoc′ whisker-left* [*of f* $\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]$ $\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) \cdot$
$((f \star f_0 g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$

$((f.tab \star Tfg.p_1) \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$(((\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1)) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1)) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) \cdot$
$((f \star f_0g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$
$((f.tab \star Tfg.p_1) \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $((\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1)) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) =$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1$
using *fg* $fg_0h_1.p_1$*-simps comp-cod-arr comp-assoc-assoc′*
whisker-right
[*of* $TTfgh.p_1$ $\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0]$ $\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0]$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1) \cdot$
$((\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$((f \star (g.tab \star Tfg.p_0)) \star TTfgh.p_1) \cdot$
$((f \star f_0g_1.\varphi) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \star TTfgh.p_1) \cdot$

$((f.tab \star Tfg.p_1) \star TTfgh.p_1)) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1)) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \cdot$
$(f \star (g.tab \star Tfg.p_0)) \cdot$
$(f \star f_0g_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \cdot$
$(f.tab \star Tfg.p_1)$
$\star\ TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
whisker-right comp-assoc
by *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[f \star g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \cdot$
$(f \star (g.tab \star Tfg.p_0)) \cdot$
$(f \star f_0g_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \cdot$
$(f.tab \star Tfg.p_1)$
$\star\ TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $\mathrm{a}^{-1}[f,\ g,\ (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$\mathrm{a}[f,\ (g \star tab_0\ g) \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$((f \star \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0]) \star TTfgh.p_1) \cdot$
$(\mathrm{a}[f,\ g,\ tab_0\ g \star Tfg.p_0] \star TTfgh.p_1) =$
$\{\mathbf{a}^{-1}[\langle f\rangle,\ \langle g\rangle,\ (\langle tab_0\ g\rangle \star \langle Tfg.p_0\rangle) \star \langle TTfgh.p_1\rangle] \cdot$
$(\langle f\rangle \star \mathbf{a}[\langle g\rangle,\ \langle tab_0\ g\rangle \star \langle Tfg.p_0\rangle,\ \langle TTfgh.p_1\rangle]) \cdot$
$(\langle f\rangle \star \mathbf{a}[\langle g\rangle,\ \langle tab_0\ g\rangle,\ \langle Tfg.p_0\rangle] \star \langle TTfgh.p_1\rangle) \cdot$

$\mathbf{a}[\langle f\rangle, (\langle g\rangle \star \langle tab_0\ g\rangle) \star \langle Tfg.p_0\rangle, \langle TTfgh.p_1\rangle] \cdot$
$((\langle f\rangle \star \mathbf{a}^{-1}[\langle g\rangle, \langle tab_0\ g\rangle, \langle Tfg.p_0\rangle]) \star \langle TTfgh.p_1\rangle) \cdot$
$(\mathbf{a}[\langle f\rangle, \langle g\rangle, \langle tab_0\ g\rangle \star \langle Tfg.p_0\rangle] \star \langle TTfgh.p_1\rangle)\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps* a'*-def* α*-def*
by *simp*
also have ... $= \{\!|\mathbf{a}[\langle f\rangle \star \langle g\rangle, \langle tab_0\ g\rangle \star \langle Tfg.p_0\rangle, \langle TTfgh.p_1\rangle]|\!\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
by (*intro E.eval-eqI*, *auto*)
also have ... $= \mathrm{a}[f \star g, tab_0\ g \star Tfg.p_0, TTfgh.p_1]$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps* a'*-def* α*-def*
by *simp*
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $=$
$\mathrm{a}[f, g, h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((\mathrm{a}[f \star g, h, tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}^{-1}[f \star g, h, tab_0\ h \star TTfgh.p_0]) \cdot$
$((f \star g) \star \mathrm{a}[h, tab_0\ h, TTfgh.p_0])) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[f \star g, tab_0\ g \star Tfg.p_0, TTfgh.p_1] \cdot$
$(\mathrm{a}^{-1}[f, g, tab_0\ g \star Tfg.p_0] \cdot$
$(f \star \mathrm{a}[g, tab_0\ g, Tfg.p_0]) \cdot$
$(f \star (g.tab \star Tfg.p_0)) \cdot$
$(f \star f_0g_1.\varphi) \cdot$
$\mathrm{a}[f, tab_0\ f, Tfg.p_1] \cdot$
$(f.tab \star Tfg.p_1)$
$\star\ TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ f, Tfg.p_1, TTfgh.p_1]$
proof −
have $(\mathrm{a}[f \star g, h, tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}^{-1}[f \star g, h, tab_0\ h \star TTfgh.p_0]) \cdot$
$((f \star g) \star \mathrm{a}[h, tab_0\ h, TTfgh.p_0]) =$
$((f \star g) \star \mathrm{a}[h, tab_0\ h, TTfgh.p_0])$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps* *comp-cod-arr comp-assoc-assoc′*
by *simp*
thus *?thesis* **by** *simp*
qed
also have ... $=$
$\mathrm{a}[f, g, h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g, h, tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f \star g, h, tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star \mathrm{a}[h, tab_0\ h, TTfgh.p_0]) \cdot$
$((f \star g) \star h.tab \star TTfgh.p_0) \cdot$
$((f \star g) \star fg_0h_1.\varphi) \cdot$

$\mathrm{a}[f \star g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ tab_0\ g \star Tfg.p_0] \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0]) \cdot$
$(f \star (g.tab \star Tfg.p_0)) \cdot$
$(f \star f_0 g_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ Tfg.p_1] \cdot$
$(f.tab \star Tfg.p_1)$
$\star\ TTfgh.p_1)) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$TTfgh.tab \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *TTfgh.tab-def Tfg.ϱσ.tab-def* **by** *simp*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$((\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \cdot$
$(f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \cdot$
$(g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \cdot$
$(g \star h.tab \star Tgh.p_0) \cdot$
$(g \star g_0 h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \cdot$
$(g.tab \star Tgh.p_1)$
$\star\ TfTgh.p_0) \cdot$
$(f \star f_0 gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1)$
$\star\ TTfgh\text{-}TfTgh.chine) \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *src-tab-eq TfTgh.tab-def Tgh.ϱσ.tab-def comp-assoc* **by** *simp*

Now we have to make this look like $f.composite\text{-}cell\ w_f{}'\ \vartheta_f{}' \cdot \beta_f$.

also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$((\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$

$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi)) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1)$
$\star\ TTfgh\text{-}TfTgh.chine) \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu) \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$

proof −
have $f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \cdot$
$(g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \cdot$
$(g \star h.tab \star Tgh.p_0) \cdot$
$(g \star g_0h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \cdot$
$(g.tab \star Tgh.p_1)$
$\star\ TfTgh.p_0 =$
$(f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$
$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0)$
using *fg gh whisker-right whisker-left* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f.tab \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine) \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$

proof −

have $\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \cdot$
$(f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$
$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1)$
$\star\ TTfgh\text{-}TfTgh.chine =$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f.tab \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine)$

proof −
have $arr\ (\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \cdot$
$(f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$
$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *fg gh*
by (*intro seqI′ comp-in-homI*) *auto*
moreover
have $arr\ ((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \cdot$
$(f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$
$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$

using *calculation* **by** *blast*
moreover
have $arr\ ((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \cdot$
$(f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *calculation* **by** *blast*
moreover
have $arr\ ((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \cdot$
$(f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *calculation* **by** *blast*
moreover
have $arr\ ((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \cdot$
$(f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *calculation* **by** *blast*
moreover
have $arr\ ((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \cdot$
$(f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *calculation* **by** *blast*
moreover
have $arr\ ((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \cdot$
$(f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \cdot$
$(f.tab \star TfTgh.p_1))$
using *calculation* **by** *blast*
moreover
have $arr\ ((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \cdot$
$(f \star f_0gh_1.\varphi) \cdot$

```
          a[f, tab₀ f, TfTgh.p₁] ·
          (f.tab ⋆ TfTgh.p₁))
      using calculation by blast
    moreover
    have arr ((f ⋆ f₀gh₁.φ) ·
          a[f, tab₀ f, TfTgh.p₁] ·
          (f.tab ⋆ TfTgh.p₁))
      using calculation by blast
    moreover
    have arr (a[f, tab₀ f, TfTgh.p₁] ·
          (f.tab ⋆ TfTgh.p₁))
      using calculation by blast
    ultimately show ?thesis
      using whisker-right [of TTfgh-TfTgh.chine] TTfgh-TfTgh.is-ide by presburger
  qed
  thus ?thesis
    using comp-assoc by presburger
qed
also have ... =
          a[f, g, h ⋆ tab₀ h ⋆ TTfgh.p₀] ·
          a[f ⋆ g, h, tab₀ h ⋆ TTfgh.p₀] ·
          (a⁻¹[f, g, h] ⋆ tab₀ h ⋆ TTfgh.p₀) ·
          ((f ⋆ g ⋆ h) ⋆ TTfgh-TfTgh.the-ϑ) ·
          a[f ⋆ g ⋆ h, (tab₀ h ⋆ Tgh.p₀) ⋆ TfTgh.p₀, TTfgh-TfTgh.chine] ·
          (a⁻¹[f, g ⋆ h, (tab₀ h ⋆ Tgh.p₀) ⋆ TfTgh.p₀] ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ a[g ⋆ h, tab₀ h ⋆ Tgh.p₀, TfTgh.p₀]) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ a⁻¹[g, h, tab₀ h ⋆ Tgh.p₀] ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ (g ⋆ a[h, tab₀ h, Tgh.p₀]) ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ (g ⋆ h.tab ⋆ Tgh.p₀) ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ (g ⋆ g₀h₁.φ) ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ a[g, tab₀ g, Tgh.p₁] ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ (g.tab ⋆ Tgh.p₁) ⋆ TfTgh.p₀) ⋆ TTfgh-TfTgh.chine) ·
          ((f ⋆ f₀gh₁.φ) ⋆ TTfgh-TfTgh.chine) ·
          (a[f, tab₀ f, TfTgh.p₁] ⋆ TTfgh-TfTgh.chine) ·
          (((f.tab ⋆ TfTgh.p₁) ⋆ TTfgh-TfTgh.chine) ·
          a⁻¹[tab₁ f, TfTgh.p₁, TTfgh-TfTgh.chine] ·
          a[tab₁ f, TfTgh.p₁, TTfgh-TfTgh.chine]) ·
          TTfgh-TfTgh.the-ν ·
          a⁻¹[tab₁ f, Tfg.p₁, TTfgh.p₁]
proof −
  have ((f.tab ⋆ TfTgh.p₁) ⋆ TTfgh-TfTgh.chine) ·
          a⁻¹[tab₁ f, TfTgh.p₁, TTfgh-TfTgh.chine] ·
          a[tab₁ f, TfTgh.p₁, TTfgh-TfTgh.chine] =
        (f.tab ⋆ TfTgh.p₁) ⋆ TTfgh-TfTgh.chine
    using fg gh fg₀h₁.p₀-simps fg₀h₁.p₁-simps f₀g₁.p₀-simps f₀g₁.p₁-simps
          comp-arr-dom comp-assoc-assoc'
    by simp
  thus ?thesis by simp
qed
```

also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0]\ \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0]\ \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0)\ \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta)\ \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$(((f.tab \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])\ \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu\ \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0]\ \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0]\ \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0)\ \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta)\ \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu\ \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $((f.tab \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*

assoc′-naturality [*of f.tab TfTgh.p*$_1$ *TTfgh-TfTgh.chine*]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
a[*f*, *g*, *h* ⋆ *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$] ·
a[*f* ⋆ *g*, *h*, *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$] ·
(a^{-1}[*f*, *g*, *h*] ⋆ *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$) ·
((*f* ⋆ *g* ⋆ *h*) ⋆ *TTfgh-TfTgh.the-ϑ*) ·
a[*f* ⋆ *g* ⋆ *h*, (*tab*$_0$ *h* ⋆ *Tgh.p*$_0$) ⋆ *TfTgh.p*$_0$, *TTfgh-TfTgh.chine*] ·
(a^{-1}[*f*, *g* ⋆ *h*, (*tab*$_0$ *h* ⋆ *Tgh.p*$_0$) ⋆ *TfTgh.p*$_0$] ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ a[*g* ⋆ *h*, *tab*$_0$ *h* ⋆ *Tgh.p*$_0$, *TfTgh.p*$_0$]) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ a^{-1}[*g*, *h*, *tab*$_0$ *h* ⋆ *Tgh.p*$_0$] ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ (*g* ⋆ a[*h*, *tab*$_0$ *h*, *Tgh.p*$_0$]) ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ (*g* ⋆ *h.tab* ⋆ *Tgh.p*$_0$) ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ (*g* ⋆ *g*$_0$*h*$_1$.*φ*) ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ a[*g*, *tab*$_0$ *g*, *Tgh.p*$_1$] ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ (*g.tab* ⋆ *Tgh.p*$_1$) ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ *f*$_0$*gh*$_1$.*φ*) ⋆ *TTfgh-TfTgh.chine*) ·
(a[*f*, *tab*$_0$ *f*, *TfTgh.p*$_1$] ⋆ *TTfgh-TfTgh.chine*) ·
a^{-1}[*f* ⋆ *tab*$_0$ *f*, *TfTgh.p*$_1$, *TTfgh-TfTgh.chine*] ·
((a^{-1}[*f*, *tab*$_0$ *f*, *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*] ·
a[*f*, *tab*$_0$ *f*, *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*]) ·
(*f.tab* ⋆ *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*)) ·
a[*tab*$_1$ *f*, *TfTgh.p*$_1$, *TTfgh-TfTgh.chine*] ·
TTfgh-TfTgh.the-ν ·
a^{-1}[*tab*$_1$ *f*, *Tfg.p*$_1$, *TTfgh.p*$_1$]
proof −
have (a^{-1}[*f*, *tab*$_0$ *f*, *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*] ·
a[*f*, *tab*$_0$ *f*, *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*]) ·
(*f.tab* ⋆ *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*) =
f.tab ⋆ *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*
using *fg gh fg*$_0$*h*$_1$*.p*$_0$*-simps fg*$_0$*h*$_1$*.p*$_1$*-simps f*$_0$*g*$_1$*.p*$_0$*-simps f*$_0$*g*$_1$*.p*$_1$*-simps*
comp-cod-arr comp-assoc-assoc′
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... =
a[*f*, *g*, *h* ⋆ *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$] ·
a[*f* ⋆ *g*, *h*, *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$] ·
(a^{-1}[*f*, *g*, *h*] ⋆ *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$) ·
((*f* ⋆ *g* ⋆ *h*) ⋆ *TTfgh-TfTgh.the-ϑ*) ·
a[*f* ⋆ *g* ⋆ *h*, (*tab*$_0$ *h* ⋆ *Tgh.p*$_0$) ⋆ *TfTgh.p*$_0$, *TTfgh-TfTgh.chine*] ·
(a^{-1}[*f*, *g* ⋆ *h*, (*tab*$_0$ *h* ⋆ *Tgh.p*$_0$) ⋆ *TfTgh.p*$_0$] ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ a[*g* ⋆ *h*, *tab*$_0$ *h* ⋆ *Tgh.p*$_0$, *TfTgh.p*$_0$]) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ a^{-1}[*g*, *h*, *tab*$_0$ *h* ⋆ *Tgh.p*$_0$] ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·
((*f* ⋆ (*g* ⋆ a[*h*, *tab*$_0$ *h*, *Tgh.p*$_0$]) ⋆ *TfTgh.p*$_0$) ⋆ *TTfgh-TfTgh.chine*) ·

$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... =
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f \star TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ tab_0\ f \star TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$
proof −
have $(\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f \star tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] =$
$\{\!|(\mathbf{a}[\langle f\rangle,\ \langle tab_0\ f\rangle,\ \langle TfTgh.p_1\rangle] \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathbf{a}^{-1}[\langle f\rangle \star \langle tab_0\ f\rangle,\ \langle TfTgh.p_1\rangle,\ \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$\mathbf{a}^{-1}[\langle f\rangle,\ \langle tab_0\ f\rangle,\ \langle TfTgh.p_1\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$

using a'*-def* α*-def* **by** *simp*
also have ... $= \{\!|\mathbf{a}^{-1}[\langle f\rangle, \langle tab_0\ f\rangle \star \langle TfTgh.p_1\rangle, \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$(\langle f\rangle \star \mathbf{a}^{-1}[\langle tab_0\ f\rangle, \langle TfTgh.p_1\rangle, \langle TTfgh\text{-}TfTgh.chine\rangle])|\!\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
by (*intro E.eval-eqI*, *auto*)
also have ... $= \mathrm{a}^{-1}[f,\ tab_0\ f \star TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$
using a'*-def* α*-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $=$
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof $-$
have $((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f \star TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$

$(f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$

proof −

have $((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star f_0gh_1.\varphi) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ tab_0\ f \star TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] =$
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (tab_1\ g \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
assoc′-naturality [*of f* $f_0gh_1.\varphi$ *TTfgh-TfTgh.chine*] *comp-assoc*
by *simp*
also have ... =
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ ((g \star tab_0\ g) \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$
proof −
have $((f \star (g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (tab_1\ g \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ ((g \star tab_0\ g) \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
assoc′-naturality [*of f* $(g.tab \star Tgh.p_1) \star TfTgh.p_0$ *TTfgh-TfTgh.chine*]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed

also have ... =
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star tab_0\ g \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$
proof −
have $((f \star \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ ((g \star tab_0\ g) \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ (g \star tab_0\ g \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
assoc′-naturality
$[of\ f\ \mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0\ \ TTfgh\text{-}TfTgh.chine]$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star tab_1\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$
proof −
have $((f \star (g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star tab_0\ g \star Tgh.p_1) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ (g \star tab_1\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
assoc′-naturality $[of\ f\ (g \star g_0h_1.\varphi) \star TfTgh.p_0\ \ TTfgh\text{-}TfTgh.chine]$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$(((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star (h \star tab_0\ h) \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$

$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$

proof −
have $((f \star (g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star tab_1\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[f,\ (g \star (h \star tab_0\ h) \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
assoc′-naturality
[*of f* $(g \star h.tab \star Tgh.p_0) \star TfTgh.p_0$ *TTfgh-TfTgh.chine*]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$

proof −

have $((f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0]) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star \mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((f \star (g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star (h \star tab_0\ h) \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\{\!|((\langle f\rangle \star \mathbf{a}[\langle g\rangle \star \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle])$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((\langle f\rangle \star \mathbf{a}^{-1}[\langle g\rangle,\ \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle] \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((\langle f\rangle \star (\langle g\rangle \star \mathbf{a}[\langle h\rangle,\ \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle]) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathbf{a}^{-1}[\langle f\rangle,\ (\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle,$
$\langle TTfgh\text{-}TfTgh.chine\rangle]\}\!|$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
a'*-def* α*-def*
by *simp*
also have ... =
$\{\!|\mathbf{a}^{-1}[\langle f\rangle,\ (\langle g\rangle \star \langle h\rangle) \star (\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle,$
$\langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$(\langle f\rangle \star (\mathbf{a}[\langle g\rangle \star \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle])$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\langle f\rangle \star (\mathbf{a}^{-1}[\langle g\rangle,\ \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle] \star \langle TfTgh.p_0\rangle)$

$\star$ $\langle$ *TTfgh-TfTgh.chine*$\rangle$) $\cdot$
($\langle f\rangle$ $\star$ (($\langle g\rangle$ $\star$ **a**[$\langle h\rangle$, $\langle tab_0\ h\rangle$, $\langle Tgh.p_0\rangle$]) $\star$ $\langle$ *TfTgh.*$p_0\rangle$)
$\star$ $\langle$ *TTfgh-TfTgh.chine*$\rangle$)$\}$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
apply (*intro E.eval-eqI*) **by** *simp-all*
also have ... =
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]$ $\cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh* $fg_0h_1.p_0$*-simps* $fg_0h_1.p_1$*-simps* $f_0g_1.p_0$*-simps* $f_0g_1.p_1$*-simps*
a'*-def* α*-def*
by *simp*
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$(\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0]$ $\cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0]$ $\cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0)$ $\cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta))$ $\cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]$ $\cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]$ $\cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star (((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$ $\cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine]$ $\cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]$ $\cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu$ $\cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof $-$
have $(f \star ((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star ((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$
$(f \star ((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$ $\cdot$

$(f \star f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]) =$
$(f \star (((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$
using *fg gh whisker-left* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$(f \star g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(\mathrm{a}[f,\ g,\ h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[f \star g,\ h,\ ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)) \cdot$
$(f \star (((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot$
$((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) =$
$\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$(\mathrm{a}[f \star g,\ h,\ tab_0\ h \star TTfgh.p_0] \cdot$
$(((f \star g) \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta)) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
proof −
have $(\mathrm{a}^{-1}[f,\ g,\ h] \star tab_0\ h \star TTfgh.p_0) \cdot ((f \star g \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) =$
$\mathrm{a}^{-1}[f,\ g,\ h] \star TTfgh\text{-}TfTgh.the\text{-}\vartheta$
using *fg gh comp-arr-dom comp-cod-arr*
interchange [*of* $\mathrm{a}^{-1}[f,\ g,\ h]\ f \star g \star h$
$tab_0\ h \star TTfgh.p_0\ TTfgh\text{-}TfTgh.the\text{-}\vartheta$]

by *simp*
also have ... = $(((f \star g) \star h) \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh comp-arr-dom comp-cod-arr*
interchange [*of* $(f \star g) \star h\ \mathrm{a}^{-1}[f,\ g,\ h]\ TTfgh\text{-}TfTgh.the\text{-}\vartheta$
$((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$]
by *simp*
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$(\mathrm{a}[f,\ g,\ h \star tab_0\ h \star TTfgh.p_0] \cdot$
$((f \star g) \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta)) \cdot$
$\mathrm{a}[f \star g,\ h,\ ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh assoc-naturality* [*of* $f \star g\ h\ TTfgh\text{-}TfTgh.the\text{-}\vartheta$] *comp-assoc*
by *simp*
also have ... =
$(f \star g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$\mathrm{a}[f,\ g,\ h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[f \star g,\ h,\ ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using *fg gh assoc-naturality* [*of* $f\ g\ h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta$] *comp-assoc*
by *simp*
finally show *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
$((f \star g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(f \star can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)))$
.

$(f \star (((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])) \cdot$
$\mathrm{a}[f,\ tab_0\ f,\ TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(f.tab \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$TTfgh\text{-}TfTgh.the\text{-}\nu \cdot$
$\mathrm{a}^{-1}[tab_1\ f,\ Tfg.p_1,\ TTfgh.p_1]$
proof −
have $\mathrm{a}[f,\ g,\ h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[f \star g,\ h,\ ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$

$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) =$
$f \star can$
$(\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$

proof −

have $\mathrm{a}[f,\ g,\ h \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[f \star g,\ h,\ ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g,\ h] \star ((tab_0\ h \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[f \star g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[f,\ g \star h,\ (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[f,\ (g \star h) \star (tab_0\ h \star Tgh.p_0) \star TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f \star \mathrm{a}[g \star h,\ tab_0\ h \star Tgh.p_0,\ TfTgh.p_0] \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star (\mathrm{a}^{-1}[g,\ h,\ tab_0\ h \star Tgh.p_0] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f \star ((g \star \mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0]) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) =$
$\{\!|\mathbf{a}[\langle f\rangle,\ \langle g\rangle,$
$\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$\mathbf{a}[\langle f\rangle \star \langle g\rangle,\ \langle h\rangle,$
$((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle f\rangle,\ \langle g\rangle,\ \langle h\rangle]$
$\star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathbf{a}[\langle f\rangle \star \langle g\rangle \star \langle h\rangle,\ (\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle,$
$\langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$(\mathbf{a}^{-1}[\langle f\rangle,\ \langle g\rangle \star \langle h\rangle,$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle] \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathbf{a}^{-1}[\langle f\rangle,\ (\langle g\rangle \star \langle h\rangle) \star (\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle,$
$\langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$(\langle f\rangle \star \mathbf{a}[\langle g\rangle \star \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle] \star$
$\langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\langle f\rangle \star (\mathbf{a}^{-1}[\langle g\rangle,\ \langle h\rangle,\ \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle] \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\langle f\rangle \star ((\langle g\rangle \star \mathbf{a}[\langle h\rangle,\ \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle]) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)\}\!|\}$

using a'*-def* α*-def* **by** *simp*

also have ... =
$can\ (\langle f\rangle \star (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle))$
$(\langle f\rangle \star ((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle))$

using *fg gh*
apply (*unfold can-def*)
apply (*intro E.eval-eqI*)
by *simp-all*

also have ... =
$f \star can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$

$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$)
using *fg gh whisker-can-left-0* **by** *simp*
finally show *?thesis* **by** *blast*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... =
(f $\star$ (g $\star$ h $\star$ *TTfgh-TfTgh.the-*ϑ) $\cdot$
can ($\langle g\rangle$ $\star$ $\langle h\rangle$ $\star$ (($\langle tab_0\ h\rangle$ $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$)
((($\langle g\rangle$ $\star$ ($\langle h\rangle$ $\star$ $\langle tab_0\ h\rangle$) $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$) $\cdot$
(((g $\star$ h.tab $\star$ Tgh.p_0) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g $\star$ $g_0h_1.\varphi$) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
((a[g, tab_0 g, Tgh.p_1] $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g.tab $\star$ Tgh.p_1) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
($f_0gh_1.\varphi$ $\star$ TTfgh-TfTgh.chine) $\cdot$
a^{-1}[tab_0 f, TfTgh.p_1, TTfgh-TfTgh.chine]) $\cdot$
a[f, tab_0 f, TfTgh.p_1 $\star$ TTfgh-TfTgh.chine] $\cdot$
(f.tab $\star$ TfTgh.p_1 $\star$ TTfgh-TfTgh.chine) $\cdot$
a[tab_1 f, TfTgh.p_1, TTfgh-TfTgh.chine] $\cdot$
TTfgh-TfTgh.the-ν $\cdot$
a^{-1}[tab_1 f, Tfg.p_1, TTfgh.p_1]
proof −
have ((f $\star$ g $\star$ h $\star$ TTfgh-TfTgh.the-ϑ) $\cdot$
(f $\star$ *can* ($\langle g\rangle$ $\star$ $\langle h\rangle$ $\star$ (($\langle tab_0\ h\rangle$ $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$)
((($\langle g\rangle$ $\star$ ($\langle h\rangle$ $\star$ $\langle tab_0\ h\rangle$) $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$)) $\cdot$
(f $\star$ (((g $\star$ h.tab $\star$ Tgh.p_0) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g $\star$ $g_0h_1.\varphi$) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
((a[g, tab_0 g, Tgh.p_1] $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g.tab $\star$ Tgh.p_1) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
($f_0gh_1.\varphi$ $\star$ TTfgh-TfTgh.chine) $\cdot$
a^{-1}[tab_0 f, TfTgh.p_1, TTfgh-TfTgh.chine])) =
f $\star$ (g $\star$ h $\star$ TTfgh-TfTgh.the-ϑ) $\cdot$
can ($\langle g\rangle$ $\star$ $\langle h\rangle$ $\star$ (($\langle tab_0\ h\rangle$ $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$)
((($\langle g\rangle$ $\star$ ($\langle h\rangle$ $\star$ $\langle tab_0\ h\rangle$) $\star$ $\langle Tgh.p_0\rangle$) $\star$ $\langle TfTgh.p_0\rangle$)
$\star$ $\langle$*TTfgh-TfTgh.chine*$\rangle$) $\cdot$
(((g $\star$ h.tab $\star$ Tgh.p_0) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g $\star$ $g_0h_1.\varphi$) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
((a[g, tab_0 g, Tgh.p_1] $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
(((g.tab $\star$ Tgh.p_1) $\star$ TfTgh.p_0) $\star$ TTfgh-TfTgh.chine) $\cdot$
($f_0gh_1.\varphi$ $\star$ TTfgh-TfTgh.chine) $\cdot$
a^{-1}[tab_0 f, TfTgh.p_1, TTfgh-TfTgh.chine]
proof −
have *1*: *arr* ((g $\star$ h $\star$ TTfgh-TfTgh.the-ϑ) $\cdot$

$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$
using *fg gh*
apply (*intro seqI′ comp-in-homI*) **by** *auto*
moreover
have *2*: $arr\ (can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine])$
using *calculation* **by** *blast*
ultimately show *?thesis*
using *whisker-left f.ide-base* **by** *presburger*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= f.composite\text{-}cell\ w_f{}'\ \vartheta_f{}' \cdot \beta_f$
unfolding $w_f{}'$*-def* $\vartheta_f{}'$*-def* β_f*-def*
using *comp-assoc* **by** *presburger*
finally show *?thesis* **by** *blast*
qed
show *?thesis*
using $w_f\ w_f{}'\ \vartheta_f\ \vartheta_f{}'\ \beta_f\ f.T2\ [of\ w_f\ w_f{}'\ \vartheta_f\ u_f\ \vartheta_f{}'\ \beta_f]\ eq_f$ **by** *fast*
qed
obtain γ_f **where** γ_f: «$\gamma_f : w_f \Rightarrow w_f{}'$» $\wedge\ \beta_f = tab_1\ f \star \gamma_f \wedge \vartheta_f = \vartheta_f{}' \cdot (tab_0\ f \star \gamma_f)$
using *5* **by** *auto*
show $[\![\![TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine]\!]\!] = [\![\![Tfg.p_1 \star TTfgh.p_1]\!]\!]$
proof −
have *iso* γ_f
using γ_f *BS3* w_f*-is-map* $w_f{}'$*-is-map* **by** *blast*
hence *isomorphic* $w_f\ w_f{}'$
using γ_f *isomorphic-def isomorphic-symmetric* **by** *auto*
thus *?thesis*
using w_f*-def* $w_f{}'$*-def Maps.CLS-eqI isomorphic-symmetric* **by** *auto*
qed

On to the next equation:

$$[\![\![Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]\!] = [\![\![Tfg.p_0 \star TTfgh.p_1]\!]\!].$$

We have to make use of the equation $\vartheta_f = \vartheta_f{}' \cdot (tab_0\ f \star \gamma_f)$ in this part, similarly to how the equation *src-tab-eq* was used to replace *TTfgh.tab* in the first part.

define u_g **where** $u_g = h \star tab_0\ h \star TTfgh.p_0$
define w_g **where** $w_g = Tfg.p_0 \star TTfgh.p_1$
define $w_g{}'$ **where** $w_g{}' = Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
define ϑ_g
where $\vartheta_g = \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot (h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]$
define $\vartheta_g{}'$
where $\vartheta_g{}' = (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
define β_g
where $\beta_g = \mathrm{a}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot \mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ f \star \gamma_f) \cdot \mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot (inv\ f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
have u_g: *ide* u_g
unfolding u_g*-def* **by** *simp*
have w_g: *ide* w_g
unfolding w_g*-def* **using** $fg_0h_1.p_1$*-simps* **by** *auto*
have w_g*-is-map*: *is-left-adjoint* w_g
unfolding w_g*-def*
using $fg_0h_1.p_1$*-simps left-adjoints-compose* **by** *simp*
have $w_g{}'$: *ide* $w_g{}'$
unfolding $w_g{}'$*-def* **by** *simp*
have $w_g{}'$*-is-map*: *is-left-adjoint* $w_g{}'$
unfolding $w_g{}'$*-def*
using *TTfgh-TfTgh.is-map left-adjoints-compose* **by** *simp*
have ϑ_g: «$\vartheta_g : tab_0\ g \star w_g \Rightarrow u_g$»
using w_g*-def* u_g*-def* ϑ_g*-def* $fg_0h_1.p_1$*-simps* $fg_0h_1.\varphi$*-in-hom* **by** *auto*
have $\vartheta_g{}'$: «$\vartheta_g{}' : tab_0\ g \star w_g{}' \Rightarrow u_g$»
unfolding $w_g{}'$*-def* u_g*-def* $\vartheta_g{}'$*-def*
by (*intro comp-in-homI*) *auto*
have w_g*-in-hhom*: *in-hhom* w_g (*src* u_g) (*src* ($tab_0\ g$))
unfolding w_g*-def* u_g*-def* **by** *auto*
have $w_g{}'$*-in-hhom*: *in-hhom* $w_g{}'$ (*src* u_g) (*src* ($tab_0\ g$))
unfolding $w_g{}'$*-def* u_g*-def* **by** *auto*
have β_g: «$\beta_g : tab_1\ g \star w_g \Rightarrow tab_1\ g \star w_g{}'$»
proof (*unfold* β_g*-def* w_g*-def*, *intro comp-in-homI*)

show «$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$:
$tab_1\ g \star Tfg.p_0 \star TTfgh.p_1 \Rightarrow (tab_1\ g \star Tfg.p_0) \star TTfgh.p_1$»
using $fg_0h_1.p_1$-*simps* **by** *auto*
show «*inv* $f_0g_1.\varphi \star TTfgh.p_1$:
$(tab_1\ g \star Tfg.p_0) \star TTfgh.p_1 \Rightarrow (tab_0\ f \star Tfg.p_1) \star TTfgh.p_1$»
using $fg_0h_1.p_1$-*simps* $f_0g_1.\varphi$-*in-hom* $f_0g_1.\varphi$-*uniqueness*(*2*)
by (*intro hcomp-in-vhom*) *auto*
show «$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]$:
$(tab_0\ f \star Tfg.p_1) \star TTfgh.p_1 \Rightarrow tab_0\ f \star Tfg.p_1 \star TTfgh.p_1$»
using $fg_0h_1.p_1$-*simps* γ_f w_f-*def* $w_f{}'$-*def* **by** *auto*
show «$tab_0\ f \star \gamma_f : tab_0\ f \star Tfg.p_1 \star TTfgh.p_1 \Rightarrow tab_0\ f \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine$»
using $fg_0h_1.p_1$-*simps* γ_f w_f-*def* $w_f{}'$-*def* **by** *auto*
show «$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine]$:
$tab_0\ f \star TfTgh.p_1 \star TTfgh\text{-}TfTgh.chine \Rightarrow (tab_0\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine$»
by *auto*
show «$f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine$:
$(tab_0\ f \star TfTgh.p_1) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow ((tab_1\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$»
using $f_0gh_1.\varphi$-*in-hom*
by (*intro hcomp-in-vhom*) *auto*
show «$\mathrm{a}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]$:
$((tab_1\ g \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow (tab_1\ g \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$»
by *auto*
show «$\mathrm{a}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$:
$(tab_1\ g \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine \Rightarrow tab_1\ g \star w_g{}'$»
using $w_g{}'$-*def* **by** *auto*
qed
have eq_g: *g.composite-cell* $w_g\ \vartheta_g$ = *g.composite-cell* $w_g{}'\ \vartheta_g{}' \cdot \beta_g$
proof −
have *g.composite-cell* $w_g\ \vartheta_g$ =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot$
$(h.tab \star TTfgh.p_0) \cdot$
$fg_0h_1.\varphi \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0 \star TTfgh.p_1] \cdot$
$(g.tab \star Tfg.p_0 \star TTfgh.p_1)$
unfolding w_g-*def* ϑ_g-*def* **by** *simp*
also have ... =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$((g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0 \star TTfgh.p_1]) \cdot$
$(g.tab \star Tfg.p_0 \star TTfgh.p_1)$
using *fg gh* $f_0g_1.p_0$-*simps* $fg_0h_1.p_0$-*simps* $fg_0h_1.p_1$-*simps whisker-left*
comp-assoc
by *simp*
also have ... =

$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$(\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}^{-1}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]))) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0 \star TTfgh.p_1] \cdot$
$(g.tab \star Tfg.p_0 \star TTfgh.p_1)$

proof −
have $(\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot$
$(g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) =$
$g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using *$fg_0h_1.p_1$-simps comp-cod-arr comp-assoc-assoc′* **by** *simp*
thus *?thesis*
by (*simp add*: *comp-assoc*)
qed
also have ... =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}^{-1}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0 \star TTfgh.p_1]) \cdot$
$(g.tab \star Tfg.p_0 \star TTfgh.p_1)$
using *comp-assoc* **by** *presburger*
also have ... =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$(\mathrm{a}^{-1}[g \star tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$(g.tab \star Tfg.p_0 \star TTfgh.p_1))$
using *fg gh $f_0g_1.p_0$-simps $fg_0h_1.p_0$-simps $fg_0h_1.p_1$-simps $fg_0h_1.p_1$-simps comp-assoc pentagon′*
invert-opposite-sides-of-square
[*of* $\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1$
$(\mathrm{a}^{-1}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1]) \cdot (g \star \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1])$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]\ \mathrm{a}^{-1}[g,\ tab_0\ g,\ Tfg.p_0 \star TTfgh.p_1]]$
by *simp*
also have ... =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$((g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$

$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using $fg_0h_1.p_1$-*simps assoc'-naturality* [*of g.tab* $Tfg.p_0$ $TTfgh.p_1$] **by** *simp*
also have ... =
$(g \star \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]) \cdot$
$(g \star h.tab \star TTfgh.p_0) \cdot$
$(g \star fg_0h_1.\varphi) \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tfg.p_0,\ TTfgh.p_1] \cdot$
$(\mathrm{a}[g,\ tab_0\ g,\ Tfg.p_0] \star TTfgh.p_1) \cdot$
$((g.tab \star Tfg.p_0) \star TTfgh.p_1) \cdot$
$(f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
proof −
have $(f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1] =$
$((f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$(\mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1]) \cdot$
$(inv\ f_0g_1.\varphi \star TTfgh.p_1)) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using *comp-assoc* **by** *presburger*
also have ... = $((f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$((tab_0\ f \star Tfg.p_1) \star TTfgh.p_1) \cdot$
$(inv\ f_0g_1.\varphi \star TTfgh.p_1)) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using $fg_0h_1.p_1$-*simps whisker-right comp-assoc-assoc'* **by** *simp*
also have ... = $((f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$(inv\ f_0g_1.\varphi \star TTfgh.p_1)) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using $fg_0h_1.p_1$-*simps* $f_0g_1.\varphi$-*uniqueness comp-cod-arr* **by** *simp*
also have ... = $((tab_1\ g \star Tfg.p_0) \star TTfgh.p_1) \cdot \mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
proof −
have $(f_0g_1.\varphi \star TTfgh.p_1) \cdot (inv\ f_0g_1.\varphi \star TTfgh.p_1) =$
$f_0g_1.\varphi \cdot inv\ f_0g_1.\varphi \star TTfgh.p_1$
using $f_0g_1.\varphi$-*uniqueness whisker-right* **by** *simp*
also have ... = $(tab_1\ g \star Tfg.p_0) \star TTfgh.p_1$
using $f_0g_1.\varphi$-*uniqueness comp-arr-inv'* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... = $\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using $fg_0h_1.p_1$-*simps comp-cod-arr* **by** *simp*
finally have $(f_0g_1.\varphi \star TTfgh.p_1) \cdot \mathrm{a}^{-1}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot (inv\ f_0g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1] = \mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$

by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \vartheta_f \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
unfolding ϑ_f*-def* **using** *comp-assoc* **by** *presburger*
also have ... $= \vartheta_f{}' \cdot (tab_0\ f \star \gamma_f) \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
using γ_f *comp-assoc* **by** *simp*
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0 h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(f_0 gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ f \star \gamma_f) \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$
unfolding $\vartheta_f{}'$*-def* **using** *comp-assoc* **by** *presburger*
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0 h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(\mathrm{a}^{-1}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(f_0 gh_1.\varphi \star TTfgh\text{-}TfTgh.chine)) \cdot$
$\mathrm{a}^{-1}[tab_0\ f,\ TfTgh.p_1,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ f \star \gamma_f) \cdot$
$\mathrm{a}[tab_0\ f,\ Tfg.p_1,\ TTfgh.p_1] \cdot$
$(inv\ f_0 g_1.\varphi \star TTfgh.p_1) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tfg.p_0,\ TTfgh.p_1]$

proof −
have $(\mathrm{a}^{-1}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$(f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine) =$
$f_0gh_1.\varphi \star TTfgh\text{-}TfTgh.chine$
using $f_0gh_1.p_0$*-simps comp-cod-arr comp-arr-dom comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *fastforce*
qed
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
β_g
unfolding β_g*-def* **using** *comp-assoc* **by** *presburger*
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(((g.tab \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
β_g
proof −
have $(((g.tab \star Tgh.p_1) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_1\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((g.tab \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* $(g.tab \star Tgh.p_1)$ $TfTgh.p_0$ $TTfgh\text{-}TfTgh.chine$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$

$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
β_g

proof −
have $((g.tab \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_1\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* $g.tab\ Tgh.p_1\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$((\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
β_g

proof −
have $(\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) =$
$g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
using *comp-cod-arr comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *comp-assoc* $g_0h_1.\varphi$*-in-hom* **by** *simp*
qed
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$

$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$((\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
β_g

proof −
have $(\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) =$
$(\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((g \star (tab_0\ g \star Tgh.p_1)) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps comp-assoc comp-assoc-assoc′* **by** *simp*
also have ... $= (\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps comp-cod-arr comp-assoc-assoc′* **by** *simp*
also have ... $= (((g \star (tab_0\ g \star Tgh.p_1)) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)$
using $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps whisker-right comp-assoc-assoc′* **by** *simp*
also have ... $= (\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine$
using $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps comp-cod-arr* **by** *simp*
finally show *?thesis* **by** *presburger*
qed
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$

$(\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
β_g
using *comp-assoc* **by** *presburger*
also have ... $= (g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(g \star (h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
β_g
proof −
have $(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g \star (h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
proof −
have $(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$((((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
using *comp-assoc* **by** *presburger*
also have ... $= ((((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_1\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]) \cdot$
$((g \star g_0h_1.\varphi) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
proof −
have $(((g \star g_0h_1.\varphi) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$

$\mathrm{a}^{-1}[g \star tab_1\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((g \star g_0h_1.\varphi) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using *gh* $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* $g \star g_0h_1.\varphi$ $TfTgh.p_0$ $TTfgh\text{-}TfTgh.chine$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$((g \star g_0h_1.\varphi) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
proof −
have $(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g \star tab_1\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using *gh* $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* $g \star h.tab \star Tgh.p_0$ $TfTgh.p_0$ $TTfgh\text{-}TfTgh.chine$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$(((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g,\ tab_1\ h \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
proof −
have $((g \star g_0h_1.\varphi) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[g,\ tab_1\ h \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using *gh* $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* g $g_0h_1.\varphi$ $TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$]
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= \mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g \star (h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
proof −
have $((g \star h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[g,\ tab_1\ h \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g \star (h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)$
using *gh* $f_0gh_1.p_0$*-simps*
assoc′-naturality [*of* g $h.tab \star Tgh.p_0$ $TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$]

by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
finally show *?thesis* **by** *simp*
qed
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= ((g \star h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(g \star can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)) \cdot$
$(g \star (h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g \star g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g \star \mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine])) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
β_g
proof −
have $can\ (\langle g\rangle \star \langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$g \star can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
proof −
have $\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$can\ ((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
proof −
have $\mathrm{a}^{-1}[g \star (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] =$
$\{\!|\mathbf{a}^{-1}[\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle,\ \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$
using *gh* $f_0gh_1.p_0$*-simps canI-associator-0* a'*-def* α*-def* **by** *simp*
also have ... $= can\ ((((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)$
unfolding *can-def*
using *gh*
apply (*intro E.eval-eqI*) **by** *simp-all*
finally show *?thesis* **by** *blast*
qed
moreover
have $\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$can\ ((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle g\rangle \star ((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
proof −

have $\mathrm{a}^{-1}[g,\ (h \star tab_0\ h) \star Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\{\!|\mathbf{a}^{-1}[\langle g\rangle,\ (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$
using *gh* $f_0gh_1.p_0$*-simps canI-associator-0* a'*-def* α*-def* **by** *simp*
also have ... $= can\ ((\langle g\rangle \star (\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle g\rangle \star ((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$
unfolding *can-def*
using *gh*
apply (*intro E.eval-eqI*) **by** *simp-all*
finally show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
using *gh whisker-can-left-0* **by** *simp*
qed
moreover have $\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$g \star \mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
proof −
have $\mathrm{a}[g,\ tab_0\ g \star Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}[g \star tab_0\ g \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$((\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1] \star TfTgh.p_0) \star TTfgh\text{-}TfTgh.chine)\ \cdot$
$\mathrm{a}^{-1}[(g \star tab_0\ g) \star Tgh.p_1,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}^{-1}[g \star tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\ \cdot$
$\mathrm{a}^{-1}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] =$
$\{\!|\mathbf{a}[\langle g\rangle,\ \langle tab_0\ g\rangle \star \langle Tgh.p_1\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]\ \cdot$
$\mathbf{a}[\langle g\rangle \star \langle tab_0\ g\rangle \star \langle Tgh.p_1\rangle,\ \langle TfTgh.p_0\rangle,\ \langle TTfgh\text{-}TfTgh.chine\rangle]\ \cdot$
$((\mathbf{a}[\langle g\rangle,\ \langle tab_0\ g\rangle,\ \langle Tgh.p_1\rangle] \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)\ \cdot$
$\mathbf{a}^{-1}[(\langle g\rangle \star \langle tab_0\ g\rangle) \star \langle Tgh.p_1\rangle,\ \langle TfTgh.p_0\rangle,\ \langle TTfgh\text{-}TfTgh.chine\rangle]\ \cdot$
$\mathbf{a}^{-1}[\langle g\rangle \star \langle tab_0\ g\rangle,\ \langle Tgh.p_1\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]\ \cdot$
$\mathbf{a}^{-1}[\langle g\rangle,\ \langle tab_0\ g\rangle,\ \langle Tgh.p_1\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$
using *gh* $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps* a'*-def* α*-def* **by** *simp*
also have ... $= \{\!|\langle g\rangle \star \mathbf{a}^{-1}[\langle tab_0\ g\rangle,\ \langle Tgh.p_1\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$
apply (*intro E.eval-eqI*) **by** *simp-all*
also have ... $= g \star \mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
using *gh* $g_0h_1.p_1$*-simps* $f_0gh_1.p_0$*-simps* a'*-def* α*-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (g \star$
$(h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta)\ \cdot$
$(can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle)$
$\star\ \langle TTfgh\text{-}TfTgh.chine\rangle)$

$(((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle$
$\star \langle TTfgh\text{-}TfTgh.chine\rangle)) \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[g,\ tab_0\ g,\ Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g.tab \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
β_g
using *gh whisker-left* **by** *auto*
also have ... $= g.composite\text{-}cell\ w_g{}'\ \vartheta_g{}' \cdot \beta_g$
unfolding $w_g{}'$*-def* $\vartheta_g{}'$*-def*
using *comp-assoc* **by** *presburger*
finally show *?thesis* **by** *blast*
qed
have *6*: $\exists!\gamma.\ «\gamma : w_g \Rightarrow w_g{}'» \wedge \beta_g = tab_1\ g \star \gamma \wedge \vartheta_g = \vartheta_g{}' \cdot (tab_0\ g \star \gamma)$
using $w_g\ w_g{}'\ \vartheta_g\ \vartheta_g{}'\ \beta_g\ eq_g$ *g.T2* [*of* $w_g\ w_g{}'\ \vartheta_g\ u_g\ \vartheta_g{}'\ \beta_g$] **by** *blast*
obtain γ_g **where** γ_g: $«\gamma_g : w_g \Rightarrow w_g{}'» \wedge \beta_g = tab_1\ g \star \gamma_g \wedge \vartheta_g = \vartheta_g{}' \cdot (tab_0\ g \star \gamma_g)$
using *6* **by** *auto*
show $[\![\![Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]\!] = [\![\![Tfg.p_0 \star TTfgh.p_1]\!]\!]$
proof −
have *iso* γ_g
using γ_g *BS3* w_g*-is-map* $w_g{}'$*-is-map* **by** *blast*
hence *isomorphic* $w_g\ w_g{}'$
using γ_g *isomorphic-def isomorphic-symmetric* **by** *auto*
thus *?thesis*
using $w_g\ w_g{}'\ w_g$*-def* $w_g{}'$*-def Maps.CLS-eqI* **by** *auto*
qed

Now the last equation: similar, but somewhat simpler.

define u_h **where** $u_h = tab_0\ h \star TTfgh.p_0$
define w_h **where** $w_h = TTfgh.p_0$
define $w_h{}'$ **where** $w_h{}' = Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
define ϑ_h
where $\vartheta_h = tab_0\ h \star TTfgh.p_0$
define $\vartheta_h{}'$
where $\vartheta_h{}' = TTfgh\text{-}TfTgh.the\text{-}\vartheta \cdot \mathrm{a}^{-1}[tab_0\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
define β_h
where $\beta_h = \mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot (tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot inv\ fg_0h_1.\varphi$
have u_h: *ide* u_h
unfolding u_h*-def* **by** *simp*
have w_h: *ide* w_h
unfolding w_h*-def* **by** *simp*
have w_h*-is-map*: *is-left-adjoint* w_h
unfolding w_h*-def* **by** *simp*
have $w_h{}'$: *ide* $w_h{}'$

unfolding w_h'*-def* **by** *simp*
have w_h'*-is-map*: *is-left-adjoint* w_h'
unfolding w_h'*-def*
using $g_0h_1.p_0$*-simps* $f_0gh_1.p_0$*-simps TTfgh-TfTgh.is-map left-adjoints-compose* **by** *simp*
have ϑ_h: «ϑ_h : $tab_0\ h \star w_h \Rightarrow u_h$»
unfolding ϑ_h*-def* w_h*-def* u_h*-def* **by** *auto*
have ϑ_h': «ϑ_h' : $tab_0\ h \star w_h' \Rightarrow u_h$»
unfolding ϑ_h'*-def* w_h'*-def* u_h*-def*
using $g_0h_1.p_0$*-simps* $f_0gh_1.p_0$*-simps*
by (*intro comp-in-homI*) *auto*
have β_h: «β_h : $tab_1\ h \star w_h \Rightarrow tab_1\ h \star w_h'$»
proof (*unfold* β_h*-def* w_h*-def* w_h'*-def*, *intro comp-in-homI*)

show «*inv* $fg_0h_1.\varphi$: $tab_1\ h \star TTfgh.p_0 \Rightarrow (tab_0\ g \star Tfg.p_0) \star TTfgh.p_1$»
using $fg_0h_1.\varphi$*-uniqueness* **by** *blast*
show «$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]$:
$(tab_0\ g \star Tfg.p_0) \star TTfgh.p_1 \Rightarrow tab_0\ g \star Tfg.p_0 \star TTfgh.p_1$»
using $fg_0h_1.p_1$*-simps* **by** *auto*
show «$tab_0\ g \star \gamma_g$:
$tab_0\ g \star Tfg.p_0 \star TTfgh.p_1 \Rightarrow tab_0\ g \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$»
using γ_g w_g*-def* w_g'*-def* $fg_0h_1.p_1$*-simps* **by** *auto*
show «$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$:
$tab_0\ g \star Tgh.p_1 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow (tab_0\ g \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$»
using $fg_0h_1.p_1$*-simps* **by** *auto*
show «$g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$:
$(tab_0\ g \star Tgh.p_1) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow (tab_1\ h \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$»
using $fg_0h_1.p_1$*-simps* **by** *force*
show «$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$:
$(tab_1\ h \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
$\Rightarrow tab_1\ h \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$»
using $fg_0h_1.p_1$*-simps* **by** *auto*
qed
have eq_h: *h.composite-cell* $w_h\ \vartheta_h$ = *h.composite-cell* $w_h'\ \vartheta_h' \cdot \beta_h$
proof −

Once again, the strategy is to form the subexpression

$$\mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot (h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]$$

which is equal to ϑ_g, so that we can make use of the equation $\vartheta_g = \vartheta_g' \cdot (tab_0\ g \star \gamma_g)$.

have *h.composite-cell* $w_h\ \vartheta_h$ =
$(h \star tab_0\ h \star TTfgh.p_0) \cdot \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot (h.tab \star TTfgh.p_0)$
unfolding w_h*-def* ϑ_h*-def* **by** *simp*
also have ... = $\mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot (h.tab \star TTfgh.p_0)$
proof −
have $(h \star tab_0\ h \star TTfgh.p_0) \cdot \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] = \mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0]$
using *comp-cod-arr* **by** *simp*
thus *?thesis*

using *comp-assoc* **by** *metis*
qed
also have ... $= (\mathrm{a}[h,\ tab_0\ h,\ TTfgh.p_0] \cdot (h.tab \star TTfgh.p_0) \cdot$
$fg_0h_1.\varphi \cdot \mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot inv\ fg_0h_1.\varphi$
proof −
have $(h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot (\mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot inv\ fg_0h_1.\varphi =$
$(h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot ((tab_0\ g \star Tfg.p_0) \star TTfgh.p_1) \cdot inv\ fg_0h_1.\varphi$
using $fg_0h_1.p_1$*-simps comp-assoc-assoc′* **by** *simp*
also have ... $= (h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot inv\ fg_0h_1.\varphi$
using $fg_0h_1.p_1$*-simps* $fg_0h_1.\varphi$*-uniqueness comp-cod-arr* **by** *simp*
also have ... $= (h.tab \star TTfgh.p_0) \cdot (tab_1\ h \star TTfgh.p_0)$
using *comp-arr-inv′* $fg_0h_1.\varphi$*-uniqueness* **by** *simp*
also have ... $= h.tab \star TTfgh.p_0$
using *comp-arr-dom* $fg_0h_1.p_0$*-simps* **by** *simp*
finally have $(h.tab \star TTfgh.p_0) \cdot fg_0h_1.\varphi \cdot (\mathrm{a}^{-1}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1]) \cdot inv\ fg_0h_1.\varphi =$
$h.tab \star TTfgh.p_0$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= \vartheta_g \cdot \mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot inv\ fg_0h_1.\varphi$
unfolding ϑ_g*-def* **by** *simp*
also have ... $= (\vartheta_g{}' \cdot (tab_0\ g \star \gamma_g)) \cdot \mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot inv\ fg_0h_1.\varphi$
using γ_g **by** *simp*
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
unfolding $\vartheta_g{}'$*-def*
using *comp-assoc* **by** *presburger*
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$((\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$

proof −
have $(\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) =$
$(h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
using *comp-cod-arr comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(\mathrm{a}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$((h.tab \star Tgh.p_0) \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
using *comp-assoc* **by** *presburger*
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
using *assoc-naturality* [*of h.tab Tgh.p*$_0$ *TfTgh.p*$_0$ $\star$ *TTfgh-TfTgh.chine*] *comp-assoc*
by *simp*
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$((\mathrm{a}^{-1}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine)) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
proof −
have $(\mathrm{a}^{-1}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$

$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) =$
$h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
using *comp-cod-arr comp-assoc-assoc′* **by** *simp*
thus *?thesis*
using *comp-assoc* **by** *simp*
qed
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine])) \cdot$
$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
using *comp-assoc* **by** *presburger*
also have ... $= ((h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta) \cdot$
$(h \star can\ ((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)))) \cdot$
$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
proof −
have $can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) =$
$can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$\{\!|\mathbf{a}^{-1}[\langle h\rangle \star \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$\mathbf{a}^{-1}[\langle h\rangle,\ \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\}$
using a'*-def* α*-def* **by** *simp*
also have ... $= can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$can\ ((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle h\rangle \star \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
proof −
have $\{\!|\mathbf{a}^{-1}[\langle h\rangle \star \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$
$\mathbf{a}^{-1}[\langle h\rangle,\ \langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]|\!\} =$

$can\ ((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle h\rangle \star \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
unfolding *can-def*
apply (*intro E.eval-eqI*) **by** *simp-all*
thus *?thesis* **by** *simp*
qed
also have ... $= can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle h\rangle \star \langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
by *simp*
also have ... $= h \star can\ ((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
using *whisker-can-left-0* **by** *simp*
finally have $can\ (\langle h\rangle \star ((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$((((\langle h\rangle \star \langle tab_0\ h\rangle) \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) \cdot$
$(\mathrm{a}^{-1}[h \star tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) =$
$h \star can\ ((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
by *simp*
thus *?thesis*
using *comp-assoc* **by** *presburger*
qed
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta \cdot$
$can\ ((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle)) \cdot$
$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
using *whisker-left* [*of h*] *comp-assoc* **by** *simp*
also have ... $= (h \star TTfgh\text{-}TfTgh.the\text{-}\vartheta \cdot$
$\mathrm{a}^{-1}[tab_0\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]) \cdot$
$\mathrm{a}[h,\ tab_0\ h,\ Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(h.tab \star Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}[tab_1\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(g_0h_1.\varphi \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine) \cdot$
$\mathrm{a}^{-1}[tab_0\ g,\ Tgh.p_1,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine] \cdot$
$(tab_0\ g \star \gamma_g) \cdot$
$\mathrm{a}[tab_0\ g,\ Tfg.p_0,\ TTfgh.p_1] \cdot$
$inv\ fg_0h_1.\varphi$
proof −
have $can\ ((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) =$
$\{\!|\mathbf{a}^{-1}[\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle,\ \langle TTfgh\text{-}TfTgh.chine\rangle] \cdot$

$\mathbf{a}^{-1}[\langle tab_0\ h\rangle,\ \langle Tgh.p_0\rangle,\ \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle]\}$
unfolding *can-def*
apply (*intro E.eval-eqI*) **by** *auto*
also have ... $= \mathrm{a}^{-1}[tab_0\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
using a'*-def* α*-def* **by** *simp*
finally have *can* $((((\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle) \star \langle TfTgh.p_0\rangle) \star \langle TTfgh\text{-}TfTgh.chine\rangle)$
$(\langle tab_0\ h\rangle \star \langle Tgh.p_0\rangle \star \langle TfTgh.p_0\rangle \star \langle TTfgh\text{-}TfTgh.chine\rangle) =$
$\mathrm{a}^{-1}[tab_0\ h \star Tgh.p_0,\ TfTgh.p_0,\ TTfgh\text{-}TfTgh.chine] \cdot$
$\mathrm{a}^{-1}[tab_0\ h,\ Tgh.p_0,\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
by *simp*
thus *?thesis* **by** *simp*
qed
also have ... $= h.composite\text{-}cell\ w_h{}'\ \vartheta_h{}' \cdot \beta_h$
unfolding $w_h{}'$*-def* $\vartheta_h{}'$*-def* β_h*-def*
using *comp-assoc* **by** *presburger*
finally show *?thesis* **by** *simp*
qed
have $\mathcal{7}$: $\exists!\gamma.\ «\gamma : w_h \Rightarrow w_h{}'» \wedge \beta_h = tab_1\ h \star \gamma \wedge \vartheta_h = \vartheta_h{}' \cdot (tab_0\ h \star \gamma)$
using $w_h\ w_h{}'\ \vartheta_h\ \vartheta_h{}'\ \beta_h\ eq_h\ h.T2$ [*of* $w_h\ w_h{}'\ \vartheta_h\ u_h\ \vartheta_h{}'\ \beta_h$] **by** *blast*
obtain γ_h **where** γ_h: $«\gamma_h : w_h \Rightarrow w_h{}'» \wedge \beta_h = tab_1\ h \star \gamma_h \wedge \vartheta_h = \vartheta_h{}' \cdot (tab_0\ h \star \gamma_h)$
using $\mathcal{7}$ **by** *auto*
show $[\![\![Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]\!] = [\![\![TTfgh.p_0]\!]\!]$
proof −
have *iso* γ_h
using γ_h *BS3* w_h*-is-map* $w_h{}'$*-is-map* **by** *blast*
hence *isomorphic* $w_h\ w_h{}'$
using γ_h *isomorphic-def isomorphic-symmetric* **by** *auto*
thus *?thesis*
using $w_h\ w_h{}'$ w_h*-def* $w_h{}'$*-def Maps.CLS-eqI* [*of* $w_h\ w_h{}'$] **by** *simp*
qed
qed

Finally, we can show that *TTfgh-TfTgh.chine* is given by tupling.

lemma *CLS-chine*:
shows $[\![\![TTfgh\text{-}TfTgh.chine]\!]\!] = tuple\text{-}ABC$
proof −
have *tuple-ABC* = *SPN-fgh.chine-assoc*
using *SPN-fgh.chine-assoc-def* **by** *simp*
also have ... $= [\![\![TTfgh\text{-}TfTgh.chine]\!]\!]$
proof (*intro Maps.arr-eqI*)
show *Maps.arr SPN-fgh.chine-assoc*
using *SPN-fgh.chine-assoc-in-hom* **by** *auto*
show *Maps.arr* $[\![\![TTfgh\text{-}TfTgh.chine]\!]\!]$
using *Maps.CLS-in-hom TTfgh-TfTgh.is-map* **by** *blast*
show *Maps.Dom SPN-fgh.chine-assoc* = *Maps.Dom* $[\![\![TTfgh\text{-}TfTgh.chine]\!]\!]$
using *SPN-fgh.chine-assoc-def Maps.dom-char tuple-ABC-in-hom TTfgh-TfTgh.chine-in-hom*
by *fastforce*
show *Maps.Cod SPN-fgh.chine-assoc* = *Maps.Cod* $[\![\![TTfgh\text{-}TfTgh.chine]\!]\!]$

```
proof −
  have Maps.Cod SPN-fgh.chine-assoc = Maps.Cod tuple-ABC
    using SPN-fgh.chine-assoc-def by simp
  also have ... = src (prj0 (tab1 g ⋆ prj1 (tab1 h) (tab0 g)) (tab0 f))
    by (metis (lifting) Maps.Dom.simps(1) Maps.seq-char SPN-fgh.prj-chine-assoc(1)
        SPN-fgh.prj-simps(1) calculation f0gh1.leg1-simps(3) prj-char(4))
  also have ... = Maps.Cod ⟦⟦TTfgh-TfTgh.chine⟧⟧
    using Maps.cod-char TTfgh-TfTgh.chine-in-hom by simp
  finally show ?thesis by blast
qed
show Maps.Map SPN-fgh.chine-assoc = Maps.Map ⟦⟦TTfgh-TfTgh.chine⟧⟧
proof −
  have 0: Chn (Span.hcomp (SPN f) (Span.hcomp (SPN g) (SPN h))) =
          Maps.MkIde (src TfTgh.p0)
    using fg gh
    by (metis (mono-tags, lifting) Maps.in-homE Maps.seqE SPN-fgh.prj-chine-assoc(1)
        SPN-fgh.prj-simps(1) SPN-fgh.prj-simps(13) calculation tuple-ABC-in-hom)
  have tuple-ABC = ⟦⟦TTfgh-TfTgh.chine⟧⟧
  proof (intro Maps.prj-joint-monic
                [of SPN-fgh.μ.leg0 SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1
                    tuple-ABC ⟦⟦TTfgh-TfTgh.chine⟧⟧])
    show Maps.cospan SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1)
      using SPN-fgh.νπ.dom.is-span SPN-fgh.νπ.leg1-composite SPN-fgh.cospan-μν
      by auto
    show Maps.seq SPN-fgh.Prj1 tuple-ABC
      using 0 tuple-ABC-in-hom SPN-fgh.prj-in-hom(4) by auto
    show Maps.seq SPN-fgh.Prj1 ⟦⟦TTfgh-TfTgh.chine⟧⟧
    proof
      show Maps.in-hom ⟦⟦TTfgh-TfTgh.chine⟧⟧ ⟦⟦src TTfgh-TfTgh.chine⟧⟧
                       ⟦⟦trg TTfgh-TfTgh.chine⟧⟧
        using fg gh TTfgh-TfTgh.chine-in-hom Maps.CLS-in-hom TTfgh-TfTgh.is-map
        by blast
      show Maps.in-hom SPN-fgh.Prj1 ⟦⟦trg TTfgh-TfTgh.chine⟧⟧ SPN-fgh.μ.apex
      proof
        show Maps.cospan SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1)
          using SPN-fgh.prj-in-hom(4) by blast
        show ⟦⟦trg TTfgh-TfTgh.chine⟧⟧ =
             Maps.pbdom SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1)
        proof −
          have ⟦⟦trg TTfgh-TfTgh.chine⟧⟧ = Maps.MkIde (src TfTgh.p0)
            by simp
          also have ... = Maps.pbdom SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1)
            using 0 Maps.pbdom-def SPN-fgh.chine-composite(2) by presburger
          finally show ?thesis by blast
        qed
        show SPN-fgh.μ.apex = Maps.dom SPN-fgh.μ.leg0
          using SPN-f.dom.apex-def by blast
      qed
    qed
```

have *2*: *Maps.commutative-square SPN-fgh.ν.leg0 SPN-fgh.π.leg1*
SPN-fgh.Prj$_{01}$ *SPN-fgh.Prj*$_0$
proof
show *Maps.cospan SPN-fgh.ν.leg0 SPN-fgh.π.leg1*
using *SPN-fgh.νπ.legs-form-cospan*(*1*) **by** *simp*
show *Maps.span SPN-fgh.Prj*$_{01}$ *SPN-fgh.Prj*$_0$
using *SPN-fgh.prj-simps*(*2*−*3*,*5*−*6*) **by** *presburger*
show *Maps.dom SPN-fgh.ν.leg0* = *Maps.cod SPN-fgh.Prj*$_{01}$
using *SPN-fgh.prj-simps*(*8*) *SPN-g.dom.is-span SPN-g.dom.leg-simps*(*2*)
by *auto*
show *SPN-fgh.ν.leg0* ⊙ *SPN-fgh.Prj*$_{01}$ = *SPN-fgh.π.leg1* ⊙ *SPN-fgh.Prj*$_0$
by (*metis* (*no-types*, *lifting*) *Maps.cod-comp Maps.comp-assoc*
Maps.pullback-commutes′ SPN-fgh.μν.dom.leg-simps(*1*)
SPN-fgh.μν.leg0-composite SPN-fgh.cospan-νπ)
qed
have *1*: *Maps.commutative-square*
SPN-fgh.μ.leg0 (*Maps.comp SPN-fgh.ν.leg1 SPN-fgh.νπ.prj*$_1$)
SPN-fgh.Prj$_{11}$ *tuple-BC*
proof
show *Maps.cospan SPN-fgh.μ.leg0* (*Maps.comp SPN-fgh.ν.leg1 SPN-fgh.νπ.prj*$_1$)
using *fg gh SPN-fgh.prj-simps*(*10*) **by** *blast*
show *Maps.span SPN-fgh.Prj*$_{11}$ *tuple-BC*
using *fg gh csq*(*2*) **by** *blast*
show *Maps.dom SPN-fgh.μ.leg0* = *Maps.cod SPN-fgh.Prj*$_{11}$
using *fg gh SPN-f.dom.leg-simps*(*2*) *SPN-fgh.prj-simps*(*7*) **by** *auto*
show *SPN-fgh.μ.leg0* ⊙ *SPN-fgh.Prj*$_{11}$ =
(*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$) ⊙ *tuple-BC*
using *2 fg gh Maps.comp-assoc csq*(*2*)
Maps.prj-tuple [*of SPN-fgh.ν.leg0 SPN-fgh.π.leg1 SPN-fgh.Prj*$_{01}$ *SPN-fgh.Prj*$_0$]
by *blast*
qed
show *SPN-fgh.Prj*$_1$ ⊙ *tuple-ABC* = *SPN-fgh.Prj*$_1$ ⊙ *Maps.CLS TTfgh-TfTgh.chine*
proof −
have *SPN-fgh.Prj*$_1$ ⊙ *tuple-ABC* = *SPN-fgh.Prj*$_{11}$
using *csq*(*2*) **by** *simp*
also have ... = ⟦⟦*Tfg.p*$_1$ ⋆ *TTfgh.p*$_1$⟧⟧
using *prj-char* **by** *simp*
also have ... = ⟦⟦*TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*⟧⟧
using *prj-chine*(*1*) **by** *simp*
also have ... = ⟦⟦*TfTgh.p*$_1$⟧⟧ ⊙ ⟦⟦*TTfgh-TfTgh.chine*⟧⟧
proof −
have *is-left-adjoint TfTgh.p*$_1$
by (*simp add*: *fg*)
moreover have *is-left-adjoint TTfgh-TfTgh.chine*
using *TTfgh-TfTgh.is-map* **by** *simp*
moreover have *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine* ≅ *TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*
using *fg gh isomorphic-reflexive* **by** *simp*
ultimately show *?thesis*
using *Maps.comp-CLS*

```
            [of TfTgh.p1 TTfgh-TfTgh.chine TfTgh.p1 ⋆ TTfgh-TfTgh.chine]
      by simp
  qed
  also have ... = SPN-fgh.Prj1 ⊙ Maps.CLS TTfgh-TfTgh.chine
    using prj-char by simp
  finally show ?thesis by blast
qed
show
  Maps.PRJ0 SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1) ⊙ tuple-ABC =
   Maps.PRJ0 SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1) ⊙
    ⟦⟦TTfgh-TfTgh.chine⟧⟧
proof −
  have
    Maps.PRJ0 SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1) ⊙ tuple-ABC =
     tuple-BC
    using csq(2)
          Maps.prj-tuple [of SPN-fgh.μ.leg0 SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1
                          SPN-fgh.Prj11 tuple-BC]
    by simp
  also have ... =
            Maps.comp
            (Maps.PRJ0 SPN-fgh.μ.leg0 (Maps.comp SPN-fgh.ν.leg1 SPN-fgh.νπ.prj1))
              ⟦⟦TTfgh-TfTgh.chine⟧⟧
  proof (intro Maps.prj-joint-monic
                [of SPN-fgh.ν.leg0 SPN-fgh.π.leg1 tuple-BC
                    Maps.PRJ0 SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1) ⊙
                      ⟦⟦TTfgh-TfTgh.chine⟧⟧])
    show Maps.cospan SPN-fgh.ν.leg0 SPN-fgh.π.leg1
      using SPN-fgh.νπ.legs-form-cospan(1) by simp
    show Maps.seq SPN-fgh.νπ.prj1 tuple-BC
    proof
      show Maps.in-hom tuple-BC
                    (Maps.MkIde (src TTfgh.p0)) (Maps.MkIde (src Tgh.p0))
        using tuple-BC-in-hom by simp
      show Maps.in-hom SPN-fgh.νπ.prj1 (Maps.MkIde (src Tgh.p0)) SPN-fgh.ν.apex
      proof −
        have Maps.pbdom SPN-fgh.ν.leg0 SPN-fgh.π.leg1 = Maps.MkIde (src Tgh.p0)
          using fg gh Maps.pbdom-def
          by (metis (no-types, lifting) SPN.preserves-ide SPN-fgh.νπ.are-identities(2)
           SPN-fgh.νπ.composable Span.chine-hcomp-ide-ide Tgh.chine-hcomp-SPN-SPN
              g.is-ide)
        thus ?thesis
          using SPN-fgh.νπ.prj-in-hom(1) by simp
      qed
    qed
    show Maps.seq SPN-fgh.νπ.prj1
                (Maps.PRJ0 SPN-fgh.μ.leg0 (SPN-fgh.ν.leg1 ⊙ SPN-fgh.νπ.prj1) ⊙
                  ⟦⟦TTfgh-TfTgh.chine⟧⟧)
    proof
```

show *Maps.in-hom SPN-fgh.νπ.prj*$_1$
(*Maps.pbdom SPN-fgh.ν.leg0 SPN-fgh.π.leg1*) *SPN-fgh.ν.apex*
using *SPN-fgh.νπ.prj-in-hom*(*1*) **by** *simp*
show *Maps.in-hom*
(*Maps.PRJ*$_0$ *SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$) ⊙
⟦⟦*TTfgh-TfTgh.chine*⟧⟧)
⟦⟦*src TTfgh-TfTgh.chine*⟧⟧
(*Maps.pbdom SPN-fgh.ν.leg0 SPN-fgh.π.leg1*)
proof
show *Maps.in-hom* ⟦⟦*TTfgh-TfTgh.chine*⟧⟧ ⟦⟦*src TTfgh-TfTgh.chine*⟧⟧
⟦⟦*trg TTfgh-TfTgh.chine*⟧⟧
using *fg gh TTfgh-TfTgh.chine-in-hom Maps.CLS-in-hom TTfgh-TfTgh.is-map*
by *blast*
show *Maps.in-hom*
(*Maps.PRJ*$_0$ *SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$))
⟦⟦*trg TTfgh-TfTgh.chine*⟧⟧
(*Maps.pbdom SPN-fgh.ν.leg0 SPN-fgh.π.leg1*)
proof
show *Maps.cospan SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
using *SPN-fgh.prj-in-hom*(*4*) **by** *blast*
show ⟦⟦*trg TTfgh-TfTgh.chine*⟧⟧ =
Maps.pbdom SPN-fgh.μ.leg0 (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
proof −
have ⟦⟦*trg TTfgh-TfTgh.chine*⟧⟧ = *Maps.MkIde* (*src TfTgh.p*$_0$)
by *simp*
also have ... = *Maps.pbdom SPN-fgh.μ.leg0*
(*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
using *0 Maps.pbdom-def SPN-fgh.chine-composite*(*2*) **by** *presburger*
finally show *?thesis* **by** *blast*
qed
show *Maps.pbdom SPN-fgh.ν.leg0 SPN-fgh.π.leg1* =
Maps.dom (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
using *fg gh Maps.pbdom-def SPN-fgh.νπ.apex-composite*
SPN-fgh.νπ.dom.apex-def SPN-fgh.νπ.dom.is-span
SPN-fgh.νπ.leg1-composite
by *presburger*
qed
qed
qed
show *SPN-fgh.νπ.prj*$_0$ ⊙ *tuple-BC* =
SPN-fgh.νπ.prj$_0$ ⊙
Maps.PRJ$_0$ *SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$) ⊙
⟦⟦*TTfgh-TfTgh.chine*⟧⟧
proof −
have *SPN-fgh.νπ.prj*$_0$ ⊙ *tuple-BC* = *SPN-fgh.Prj*$_0$
using *csq*(*1*) **by** *simp*
also have ... = *SPN-fgh.νπ.prj*$_0$ ⊙
Maps.PRJ$_0$ *SPN-fgh.μ.leg0* (*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$) ⊙
⟦⟦*TTfgh-TfTgh.chine*⟧⟧

proof −
have $SPN\text{-}fgh.\nu\pi.prj_0 \odot$
$Maps.PRJ_0\ SPN\text{-}fgh.\mu.leg0\ (SPN\text{-}fgh.\nu.leg1 \odot SPN\text{-}fgh.\nu\pi.prj_1) \odot$
$[\![TTfgh\text{-}TfTgh.chine]\!] =$
$[\![Tgh.p_0]\!] \odot [\![TfTgh.p_0]\!] \odot [\![TTfgh\text{-}TfTgh.chine]\!]$
using *fg gh Tgh.ϱσ.prj-char TfTgh.prj-char(1) isomorphic-reflexive*
$Maps.comp\text{-}CLS\ [of\ tab_1\ g\ prj_1\ (tab_1\ h)\ (tab_0\ g)\ tab_1\ g \star Tgh.p_1]$
by *simp*
also have ... $= [\![Tgh.p_0]\!] \odot [\![TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]$
using *fg gh TTfgh-TfTgh.is-map isomorphic-reflexive*
Maps.comp-CLS
$[of\ TfTgh.p_0\ TTfgh\text{-}TfTgh.chine\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
by *simp*
also have ... $= [\![Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]$
using *fg gh TTfgh-TfTgh.is-map left-adjoints-compose isomorphic-reflexive*
$Maps.comp\text{-}CLS\ [of\ Tgh.p_0\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine$
$Tgh.p_0 \star TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
by *simp*
also have ... $= [\![TTfgh.p_0]\!]$
using *prj-chine(3)* **by** *simp*
also have ... $= SPN\text{-}fgh.Prj_0$
using *prj-char* **by** *simp*
finally show *?thesis* **by** *argo*
qed
finally show *?thesis* **by** *blast*
qed
show $SPN\text{-}fgh.\nu\pi.prj_1 \odot tuple\text{-}BC =$
$SPN\text{-}fgh.\nu\pi.prj_1 \odot$
$Maps.PRJ_0\ SPN\text{-}fgh.\mu.leg0\ (SPN\text{-}fgh.\nu.leg1 \odot SPN\text{-}fgh.\nu\pi.prj_1) \odot$
$[\![TTfgh\text{-}TfTgh.chine]\!]$
proof −
have $SPN\text{-}fgh.\nu\pi.prj_1 \odot tuple\text{-}BC = SPN\text{-}fgh.Prj_{01}$
using *csq(1)* **by** *simp*
also have ... $= SPN\text{-}fgh.\nu\pi.prj_1 \odot$
$Maps.PRJ_0\ SPN\text{-}fgh.\mu.leg0\ (SPN\text{-}fgh.\nu.leg1 \odot SPN\text{-}fgh.\nu\pi.prj_1) \odot$
$[\![TTfgh\text{-}TfTgh.chine]\!]$
proof −
have $SPN\text{-}fgh.\nu\pi.prj_1 \odot$
$Maps.PRJ_0\ SPN\text{-}fgh.\mu.leg0\ (SPN\text{-}fgh.\nu.leg1 \odot SPN\text{-}fgh.\nu\pi.prj_1) \odot$
$[\![TTfgh\text{-}TfTgh.chine]\!] =$
$[\![Tgh.p_1]\!] \odot [\![TfTgh.p_0]\!] \odot [\![TTfgh\text{-}TfTgh.chine]\!]$
using *fg gh Tgh.ϱσ.prj-char TfTgh.prj-char(1) isomorphic-reflexive*
$Maps.comp\text{-}CLS\ [of\ tab_1\ g\ prj_1\ (tab_1\ h)\ (tab_0\ g)\ tab_1\ g \star Tgh.p_1]$
by *simp*
also have ... $= [\![Tgh.p_1]\!] \odot [\![TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]\!]$
using *fg gh TTfgh-TfTgh.is-map isomorphic-reflexive*
Maps.comp-CLS
$[of\ TfTgh.p_0\ TTfgh\text{-}TfTgh.chine\ TfTgh.p_0 \star TTfgh\text{-}TfTgh.chine]$
by *simp*

```
            also have ... = ⟦⟦Tgh.p₁ ⋆ TfTgh.p₀ ⋆ TTfgh-TfTgh.chine⟧⟧
              using fg gh TTfgh-TfTgh.is-map left-adjoints-compose isomorphic-reflexive
                    Maps.comp-CLS [of Tgh.p₁ TfTgh.p₀ ⋆ TTfgh-TfTgh.chine
                                      Tgh.p₁ ⋆ TfTgh.p₀ ⋆ TTfgh-TfTgh.chine]
              by simp
            also have ... = ⟦⟦Tfg.p₀ ⋆ TTfgh.p₁⟧⟧
              using prj-chine(2) by simp
            also have ... = SPN-fgh.Prj₀₁
              using prj-char by simp
            finally show ?thesis by argo
          qed
          finally show ?thesis by blast
        qed
      qed
      finally show ?thesis by simp
    qed
  qed
  thus ?thesis
    using SPN-fgh.chine-assoc-def by simp
 qed
qed
finally show ?thesis by simp
qed
```

At long last, we can show associativity coherence for *SPN*.

```
lemma assoc-coherence:
shows LHS = RHS
proof (intro Span.arr-eqI)
  show Span.par LHS RHS
    using par-LHS-RHS by blast
  show Chn LHS = Chn RHS
  proof −
  have Chn LHS = ⟦⟦HHfgh-HfHgh.chine⟧⟧ ⊙ ⟦⟦THfgh-HHfgh.chine⟧⟧ ⊙ ⟦⟦TTfgh-THfgh.chine⟧⟧
      using Chn-LHS-eq by simp
    also have ... = ⟦⟦HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine⟧⟧
    proof −
      have ⟦⟦THfgh-HHfgh.chine⟧⟧ ⊙ ⟦⟦TTfgh-THfgh.chine⟧⟧ =
           ⟦⟦THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine⟧⟧
        using fg gh isomorphic-reflexive HHfgh-HfHgh.is-map THfgh-HHfgh.is-map
              TTfgh-THfgh.is-map left-adjoints-compose
              Maps.comp-CLS
          [of THfgh-HHfgh.chine TTfgh-THfgh.chine THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine]
        by simp
      moreover
      have ⟦⟦HHfgh-HfHgh.chine⟧⟧ ⊙ ⟦⟦THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine⟧⟧ =
           ⟦⟦HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine⟧⟧
      proof −
        have ide (HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine)
        proof −
```

have *ide* (*THfgh-HHfgh.chine* ⋆ *TTfgh-THfgh.chine*)
using *fg gh HHfgh-HfHgh.is-map THfgh-HHfgh.is-map TTfgh-THfgh.is-map*
left-adjoint-is-ide left-adjoints-compose
by *auto*
moreover have *src HHfgh-HfHgh.chine* = *trg* (*THfgh-HHfgh.chine* ⋆ *TTfgh-THfgh.chine*)
using *fg gh HHfgh-HfHgh.chine-in-hom* α*-def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
thus *?thesis*
using *fg gh isomorphic-reflexive HHfgh-HfHgh.is-map THfgh-HHfgh.is-map*
TTfgh-THfgh.is-map left-adjoints-compose
Maps.comp-CLS
[*of HHfgh-HfHgh.chine THfgh-HHfgh.chine* ⋆ *TTfgh-THfgh.chine*
HHfgh-HfHgh.chine ⋆ *THfgh-HHfgh.chine* ⋆ *TTfgh-THfgh.chine*]
by *auto*
qed
ultimately show *?thesis* **by** *argo*
qed
also have ... = ⟦⟦*TfHgh-HfHgh.chine* ⋆ *TfTgh-TfHgh.chine* ⋆ *TTfgh-TfTgh.chine*⟧⟧
proof −
interpret *A*: *vertical-composite-of-arrows-of-tabulations-in-maps V H* a i *src trg*
‹(*f* ⋆ *g*) ⋆ *h*› *TTfgh.tab* ‹tab_0 *h* ⋆ *TTfgh.*p_0› ‹(tab_1 *f* ⋆ *Tfg.*p_1) ⋆ *TTfgh.*p_1›
‹*f* ⋆ *g* ⋆ *h*› *TfTgh.tab* ‹(tab_0 *h* ⋆ *Tgh.*p_0) ⋆ *TfTgh.*p_0› ‹tab_1 *f* ⋆ *TfTgh.*p_1›
‹*f* ⋆ *g* ⋆ *h*› *TfHgh.*$\varrho\sigma$*.tab* ‹tab_0 (*g* ⋆ *h*) ⋆ *TfHgh.*$\varrho\sigma$*.*p_0› ‹tab_1 *f* ⋆ *TfHgh.*$\varrho\sigma$*.*p_1›
‹a[*f*, *g*, *h*]› ‹*f* ⋆ *g* ⋆ *h*›
..
interpret *B*: *vertical-composite-of-arrows-of-tabulations-in-maps V H* a i *src trg*
‹(*f* ⋆ *g*) ⋆ *h*› *TTfgh.tab* ‹tab_0 *h* ⋆ *TTfgh.*p_0› ‹(tab_1 *f* ⋆ *Tfg.*p_1) ⋆ *TTfgh.*p_1›
‹*f* ⋆ *g* ⋆ *h*› *TfHgh.*$\varrho\sigma$*.tab* ‹tab_0 (*g* ⋆ *h*) ⋆ *TfHgh.*$\varrho\sigma$*.*p_0› ‹tab_1 *f* ⋆ *TfHgh.*$\varrho\sigma$*.*p_1›
‹*f* ⋆ *g* ⋆ *h*› *HfHgh.tab* ‹tab_0 (*f* ⋆ *g* ⋆ *h*)› ‹tab_1 (*f* ⋆ *g* ⋆ *h*)›
‹a[*f*, *g*, *h*]› ‹*f* ⋆ *g* ⋆ *h*›
using *fg gh* **by** *unfold-locales auto*
interpret *C*: *vertical-composite-of-arrows-of-tabulations-in-maps V H* a i *src trg*
‹(*f* ⋆ *g*) ⋆ *h*› *THfgh.*$\varrho\sigma$*.tab* ‹tab_0 *h* ⋆ *THfgh.*$\varrho\sigma$*.*p_0› ‹tab_1 (*f* ⋆ *g*) ⋆ *THfgh.*$\varrho\sigma$*.*p_1›
‹(*f* ⋆ *g*) ⋆ *h*› *HHfgh.tab* ‹tab_0 ((*f* ⋆ *g*) ⋆ *h*)› ‹tab_1 ((*f* ⋆ *g*) ⋆ *h*)›
‹*f* ⋆ *g* ⋆ *h*› *HfHgh.tab* ‹tab_0 (*f* ⋆ *g* ⋆ *h*)› ‹tab_1 (*f* ⋆ *g* ⋆ *h*)›
‹(*f* ⋆ *g*) ⋆ *h*› ‹a[*f*, *g*, *h*]›
using *fg gh* **by** *unfold-locales auto*
interpret *D*: *vertical-composite-of-arrows-of-tabulations-in-maps V H* a i *src trg*
‹(*f* ⋆ *g*) ⋆ *h*› *TTfgh.tab* ‹tab_0 *h* ⋆ *TTfgh.*p_0› ‹(tab_1 *f* ⋆ *Tfg.*p_1) ⋆ *TTfgh.*p_1›
‹(*f* ⋆ *g*) ⋆ *h*› *THfgh.*$\varrho\sigma$*.tab* ‹tab_0 *h* ⋆ *THfgh.*$\varrho\sigma$*.*p_0› ‹tab_1 (*f* ⋆ *g*) ⋆ *THfgh.*$\varrho\sigma$*.*p_1›
‹*f* ⋆ *g* ⋆ *h*› *HfHgh.tab* ‹tab_0 (*f* ⋆ *g* ⋆ *h*)› ‹tab_1 (*f* ⋆ *g* ⋆ *h*)›
‹(*f* ⋆ *g*) ⋆ *h*› ‹a[*f*, *g*, *h*]›
using *fg gh* **by** *unfold-locales auto*
have *HHfgh-HfHgh.chine* ⋆ *THfgh-HHfgh.chine* ⋆ *TTfgh-THfgh.chine* ≅ *D.chine*
proof −
have *D.chine* ≅ *D.*π*.chine* ⋆ *TTfgh-THfgh.chine*
using *D.chine-char* **by** *simp*
also have ... ≅ *C.chine* ⋆ *TTfgh-THfgh.chine*

```
    using fg gh comp-arr-dom isomorphic-reflexive by simp
  also have ... ≅ (C.π.chine ⋆ THfgh-HHfgh.chine) ⋆ TTfgh-THfgh.chine
    using C.chine-char hcomp-isomorphic-ide by simp
  also have ... ≅ (HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine) ⋆ TTfgh-THfgh.chine
  proof −
    have C.π.chine = HHfgh-HfHgh.chine
      using fg gh comp-arr-dom comp-cod-arr α-def by simp
    hence isomorphic C.π.chine HHfgh-HfHgh.chine
      using isomorphic-reflexive by simp
    thus ?thesis
      using hcomp-isomorphic-ide by simp
  qed
  also have ... ≅ HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine
  proof −
    have ide HHfgh-HfHgh.chine ∧ ide THfgh-HHfgh.chine ∧ ide TTfgh-THfgh.chine
      by simp
    moreover have src HHfgh-HfHgh.chine = trg THfgh-HHfgh.chine ∧
                  src THfgh-HHfgh.chine = trg TTfgh-THfgh.chine
      using fg gh HHfgh-HfHgh.chine-in-hom THfgh-HHfgh.chine-in-hom
            TTfgh-THfgh.chine-in-hom α-def
       by auto
    ultimately show ?thesis
      using fg gh iso-assoc isomorphic-def
            assoc-in-hom [of HHfgh-HfHgh.chine THfgh-HHfgh.chine TTfgh-THfgh.chine]
      by auto
  qed
  finally show ?thesis
    using isomorphic-symmetric by blast
qed
also have ... ≅ B.chine
proof −
  have D.chine = B.chine
    using fg gh comp-arr-dom comp-cod-arr by simp
  thus ?thesis
    using isomorphic-reflexive by simp
qed
also have ... ≅ TfHgh-HfHgh.chine ⋆ TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine
proof −
  have B.chine ≅ TfHgh-HfHgh.chine ⋆ B.μ.chine
    using B.chine-char by simp
  also have ... ≅ TfHgh-HfHgh.chine ⋆ A.chine
      using fg gh comp-cod-arr isomorphic-reflexive by simp
  also have ... ≅ TfHgh-HfHgh.chine ⋆ TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine
    using A.chine-char hcomp-ide-isomorphic by simp
  finally show ?thesis by blast
qed
finally have HHfgh-HfHgh.chine ⋆ THfgh-HHfgh.chine ⋆ TTfgh-THfgh.chine ≅
             TfHgh-HfHgh.chine ⋆ TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine
  by blast
```

```
      thus ?thesis
        using fg gh Maps.CLS-eqI isomorphic-implies-hpar(1) by blast
    qed
  also have ... = ⟦⟦TfHgh-HfHgh.chine⟧⟧ ⊙ ⟦⟦TfTgh-TfHgh.chine⟧⟧ ⊙ ⟦⟦TTfgh-TfTgh.chine⟧⟧
   using fg gh isomorphic-reflexive TfTgh-TfHgh.is-map TfHgh-HfHgh.is-map TTfgh-TfTgh.is-map
          left-adjoints-compose
          Maps.comp-CLS
         [of TfTgh-TfHgh.chine TTfgh-TfTgh.chine TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine]
          Maps.comp-CLS
            [of TfHgh-HfHgh.chine TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine
                TfHgh-HfHgh.chine ⋆ TfTgh-TfHgh.chine ⋆ TTfgh-TfTgh.chine]
      by simp
    also have ... = Chn RHS
      using Chn-RHS-eq CLS-chine tuple-ABC-eq-ABC'(2) by simp
    finally show ?thesis
      by blast
  qed
qed

end
```

SPN is an Equivalence Pseudofunctor

```
context bicategory-of-spans
begin

  interpretation Maps: maps-category V H a i src trg ..
  interpretation Span: span-bicategory Maps.comp Maps.PRJ0 Maps.PRJ1 ..

  no-notation Fun.comp (infixl ‹∘› 55)
  notation Span.vcomp (infixr ‹•› 55)
  notation Span.hcomp (infixr ‹∘› 53)
  notation Maps.comp (infixr ‹⊙› 55)
  notation isomorphic (infix ‹≅› 50)

  interpretation SPN: functor V Span.vcomp SPN
    using SPN-is-functor by simp
  interpretation SPN: weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN
    using SPN-is-weak-arrow-of-homs by simp
  interpretation HoSPN-SPN: composite-functor VV.comp Span.VV.comp Span.vcomp
                            SPN.FF ‹λμν. Span.hcomp (fst μν) (snd μν)›
    ..
  interpretation SPNoH: composite-functor VV.comp V Span.vcomp
                        ‹λμν. fst μν ⋆ snd μν› SPN
    ..

  interpretation Φ: transformation-by-components VV.comp Span.vcomp
                    HoSPN-SPN.map SPNoH.map ‹λrs. CMP (fst rs) (snd rs)›
    using compositor-naturalitytransformation by simp
```

interpretation Φ: *natural-isomorphism VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map Φ*.map*
using *compositor-naturalityisomorphism* **by** *simp*

abbreviation Φ
where $\Phi \equiv \Phi.map$

interpretation *SPN*: *pseudofunctor V H* a i *src trg*
Span.vcomp Span.hcomp Span.assoc Span.unit Span.src Span.trg SPN Φ
proof
show $\bigwedge f\ g\ h.\ [\![\ ide\ f;\ ide\ g;\ ide\ h;\ src\ f = trg\ g;\ src\ g = trg\ h\ ]\!] \Longrightarrow$
$SPN\ \mathrm{a}[f,\ g,\ h] \cdot \Phi\ (f \star g,\ h) \cdot (\Phi\ (f,\ g) \circ SPN\ h) =$
$\Phi\ (f,\ g \star h) \cdot (SPN\ f \circ \Phi\ (g,\ h)) \cdot Span.assoc\ (SPN\ f)\ (SPN\ g)\ (SPN\ h)$
proof −
fix *f g h*
assume *f*: *ide f* **and** *g*: *ide g* **and** *h*: *ide h*
assume *fg*: *src f* = *trg g* **and** *gh*: *src g* = *trg h*
interpret *fgh*: *three-composable-identities-in-bicategory-of-spans V H* a i *src trg f g h*
using *f g h fg gh*
by (*unfold-locales*, *simp*)
show *fgh.LHS* = *fgh.RHS*
using *fgh.assoc-coherence* **by** *simp*
qed
qed

lemma *SPN-is-pseudofunctor*:
shows *pseudofunctor V H* a i *src trg*
Span.vcomp Span.hcomp Span.assoc Span.unit Span.src Span.trg SPN Φ
..

interpretation *SPN*: *equivalence-pseudofunctor V H* a i *src trg*
Span.vcomp Span.hcomp Span.assoc Span.unit Span.src Span.trg SPN Φ
proof
show $\bigwedge \mu\ \mu'.\ [\![par\ \mu\ \mu';\ SPN\ \mu = SPN\ \mu']\!] \Longrightarrow \mu = \mu'$
proof −
fix $\mu\ \mu'$
assume *par*: *par* $\mu\ \mu'$
assume *eq*: $SPN\ \mu = SPN\ \mu'$
interpret *dom-*μ: *identity-in-bicategory-of-spans V H* a i *src trg* ‹*dom* μ›
using *par* **apply** *unfold-locales* **by** *auto*
interpret *cod-*μ: *identity-in-bicategory-of-spans V H* a i *src trg* ‹*cod* μ›
using *par* **apply** *unfold-locales* **by** *auto*
interpret μ: *arrow-of-tabulations-in-maps V H* a i *src trg*
‹*dom* μ› ‹*tab-of-ide* (*dom* μ)› ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
‹*cod* μ› ‹*tab-of-ide* (*cod* μ)› ‹tab_0 (*cod* μ)› ‹tab_1 (*cod* μ)›
μ
using *par* **apply** *unfold-locales* **by** *auto*
interpret μ: *arrow-in-bicategory-of-spans V H* a i *src trg* ‹*dom* μ› ‹*cod* μ› μ
using *par* **apply** *unfold-locales* **by** *auto*

interpret μ': *arrow-of-tabulations-in-maps V H* a i *src trg*
‹*dom* μ› ‹*tab-of-ide* (*dom* μ)› ‹tab_0 (*dom* μ)› ‹tab_1 (*dom* μ)›
‹*cod* μ› ‹*tab-of-ide* (*cod* μ)› ‹tab_0 (*cod* μ)› ‹tab_1 (*cod* μ)›
μ'
using *par* **apply** *unfold-locales* **by** *auto*
interpret μ': *arrow-in-bicategory-of-spans V H* a i *src trg* ‹*dom* μ› ‹*cod* μ› μ'
using *par* **apply** *unfold-locales* **by** *auto*
have μ.*chine* $\cong$ μ'.*chine*
using *par eq SPN-def spn-def Maps.CLS-eqI* μ.*is-ide* **by** *auto*
hence $\mu.\Delta = \mu'.\Delta$
using $\mu.\Delta$-*naturality* $\mu'.\Delta$-*naturality*
by (*metis* $\mu.\Delta$-*simps*(*4*) $\mu'.\Delta$-*simps*(*4*) μ.*chine-is-induced-map* μ'.*chine-is-induced-map* μ.*induced-map-preserved-by-iso*)
thus $\mu = \mu'$
using *par* $\mu.\mu$-*in-terms-of*-Δ $\mu'.\mu$-*in-terms-of*-Δ **by** *metis*
qed
show $\bigwedge a'$. *Span.obj* $a' \Longrightarrow \exists\, a$. *obj* $a \wedge$ *Span.equivalent-objects* (*SPN.*map_0 a) a'
proof −
fix a'
assume a': *Span.obj* a'
let *?a* = *Maps.Dom* (*Chn* a')
have a: *obj ?a*
using a' *Span.obj-char Span.ide-char Maps.ide-char*$_{CC}$ **by** *blast*
moreover have *Span.equivalent-objects* (*SPN.*map_0 *?a*) a'
proof −
have *SPN.*map_0 *?a* = a'
proof (*intro Span.arr-eqI*)
have *Chn* (*SPN.*map_0 *?a*) = *Chn* (*Span.src* (*SPN ?a*))
using a a' **by** *auto*
also have ... = *Maps.MkIde* (*Maps.Dom* (*Chn* a'))
proof −
have *Maps.arr* ⟦⟦tab_0 (*dom* (*Maps.Dom* (*Chn* a')))⟧⟧
proof −
have *is-left-adjoint* (tab_0 (*dom* (*Maps.Dom* (*Chn* a'))))
using a **by** *auto*
thus *?thesis*
using *Maps.CLS-in-hom* **by** *auto*
qed
moreover have *arr* (*Maps.Dom* (*Chn* a'))
using a **by** *auto*
moreover have *Span.arr* (*SPN* (*Maps.Dom* (*Chn* a')))
using a a' *SPN-in-hom* **by** *auto*
ultimately show *?thesis*
using a a' *SPN-def Span.src-def Maps.cod-char obj-simps*(*2*) **by** *simp*
qed
also have ... = *Chn* a'
using a' *Maps.MkIde-Dom Span.obj-char Span.ide-char* **by** *simp*
finally show *Chn* (*SPN.*map_0 *?a*) = *Chn* a' **by** *simp*
show *Span.par* (*SPN.*map_0 (*Maps.Dom* (*Chn* a'))) a'

```
        using a a′ Span.obj-char
        apply (intro conjI)
        using SPN.map₀-simps(1) Span.obj-def
          apply blast
         apply simp
         apply (metis (no-types, lifting) SPN.map₀-def SPN.preserves-arr Span.obj-src
            ‹Chn (SPN.map₀ (Maps.Dom (Chn a′))) = Chn a′› obj-def)
        by (metis (no-types, lifting) SPN.map₀-def SPN.preserves-arr Span.obj-src
            ‹Chn (SPN.map₀ (Maps.Dom (Chn a′))) = Chn a′› obj-def)
    qed
    thus ?thesis
      using Span.equivalent-objects-reflexive
      by (simp add: a′)
  qed
  ultimately show ∃ a. obj a ∧ Span.equivalent-objects (SPN.map₀ a) a′
    by auto
qed
show ⋀a b g. ⟦obj a; obj b; Span.in-hhom g (SPN.map₀ a) (SPN.map₀ b); Span.ide g⟧
              ⟹ ∃ f. «f : a → b» ∧ ide f ∧ Span.isomorphic (SPN f) g
proof −
  fix a b g
  assume a: obj a and b: obj b
  and g-in-hhom: Span.in-hhom g (SPN.map₀ a) (SPN.map₀ b)
  and ide-g: Span.ide g
  have arr-a: arr a
    using a by auto
  have arr-b: arr b
   using b by auto
  show ∃ f. «f : a → b» ∧ ide f ∧ Span.isomorphic (SPN f) g
  proof −
    interpret g: arrow-of-spans Maps.comp g
      using ide-g Span.ide-char by blast
    interpret g: identity-arrow-of-spans Maps.comp g
      using ide-g Span.ide-char
      by unfold-locales auto
    interpret REP-leg0: map-in-bicategory V H a i src trg ‹Maps.REP g.leg0›
      using Maps.REP-in-Map [of g.leg0]
      by unfold-locales auto
    have 0: «Maps.REP g.leg0 : src (Maps.REP g.apex) → Maps.Cod g.leg0»
      using g.dom.leg-in-hom Maps.REP-in-hhom
      by (metis (no-types, lifting) Maps.Dom-cod Maps.REP-simps(2) Maps.arr-cod
          g.dom.leg-simps(1))
    have 1: «Maps.REP g.leg1 : src (Maps.REP g.apex) → Maps.Cod g.leg1»
      using g.dom.leg-in-hom Maps.REP-in-hhom
      by (metis (no-types, lifting) Maps.Dom-cod Maps.REP-simps(2) Maps.arr-cod
          g.dom.leg-simps(3))
    let ?f = Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*
    have f-in-hhom: «?f : a → b»
    proof
```

```
  show «Maps.REP g.leg1 : src (Maps.REP g.apex) → b»
  proof −
    have «Maps.REP g.leg1 : src (Maps.REP g.apex) → Maps.Cod g.leg1»
      using 1 by simp
    moreover have Maps.Cod g.leg1 = b
    proof −
      have src (Maps.REP g.dtrg) = src (Maps.REP (Leg0 (Dom (SPN.map0 b))))
        using g-in-hhom Span.trg-def [of g] by auto
      also have ... = src (Maps.REP (Maps.cod ⟦⟦tab0 b⟧⟧))
        using b arr-b SPN.map0-def Span.src-def SPN-in-hom by auto
      also have ... = src (Maps.REP ⟦⟦trg (tab0 b)⟧⟧)
        using b Maps.CLS-in-hom [of tab0 b] by force
      also have ... = src (Maps.REP ⟦⟦b⟧⟧)
        using b by fastforce
      also have ... = b
        using b obj-simps by auto
      finally show ?thesis by simp
    qed
    ultimately show ?thesis by argo
  qed
  show «(Maps.REP g.leg0)* : a → src (Maps.REP g.apex)»
  proof −
    have «Maps.REP g.leg0 : src (Maps.REP g.apex) → a»
    proof −
      have src (Maps.REP g.dsrc) = src (Maps.REP (Leg0 (Dom (SPN.map0 a))))
        using g-in-hhom Span.src-def [of g] by auto
      also have ... = src (Maps.REP (Maps.cod ⟦⟦tab0 a⟧⟧))
        using a arr-a SPN.map0-def Span.src-def SPN-in-hom by auto
      also have ... = src (Maps.REP ⟦⟦trg (tab0 a)⟧⟧)
        using a Maps.CLS-in-hom [of tab0 a] by force
      also have ... = src (Maps.REP ⟦⟦a⟧⟧)
        using a by fastforce
      also have ... = a
        using a obj-simps by auto
      finally show ?thesis by fast
    qed
    thus ?thesis
      using REP-leg0.antipar REP-leg0.ide-right
      apply (intro in-hhomI) by auto
  qed
qed
moreover have ide-f: ide ?f
  using REP-leg0.antipar f-in-hhom by fastforce
moreover have Span.isomorphic (SPN ?f) g
proof −
  have SPN-f-eq: SPN ?f = ⦇Chn = ⟦⟦spn ?f⟧⟧,
                          Dom = ⦇Leg0 = ⟦⟦tab0 ?f⟧⟧, Leg1 = ⟦⟦tab1 ?f⟧⟧⦈,
                          Cod = ⦇Leg0 = ⟦⟦tab0 ?f⟧⟧, Leg1 = ⟦⟦tab1 ?f⟧⟧⦈⦈
    using calculation(1) SPN-def [of ?f] REP-leg0.antipar by auto
```

We need an invertible arrow of spans from *SPN f* to *g*. There exists a tabulation (*REP g.leg0*, ϱ, *REP g.leg1*) of *f*. There is also a tabulation (tab_0 *f*, ϱ', tab_1 *f*) *of f*. As these are tabulations of the same arrow, they are equivalent. This yields an equivalence map which is an arrow of spans from (tab_0 *f*, tab_1 *f*) to (*REP g.leg0*, ϱ, *REP g.leg1*). Its isomorphism class is an invertible arrow of spans in maps from (*CLS* (tab_0 *f*), *CLS* (tab_1 *f*)) to (*g.leg0*, *g.leg1*).

interpret *f*: *identity-in-bicategory-of-spans V H* a i *src trg ?f*
  using *ide-f* **apply** *unfold-locales* **by** *auto*
interpret *f*: *arrow-of-tabulations-in-maps V H* a i *src trg*
    ?f f.tab ‹tab_0 *?f*› ‹tab_1 *?f*› *?f f.tab* ‹tab_0 *?f*› ‹tab_1 *?f*› *?f*
  using *f.is-arrow-of-tabulations-in-maps* **by** *simp*
interpret *g*: *span-of-maps V H* a i *src trg* ‹*Maps.REP g.leg0*› ‹*Maps.REP g.leg1*›
  using *Span.arr-char*
  by (*unfold-locales*, *blast*+)

have *2*: *src* (*Maps.REP g.leg0*) = *src* (*Maps.REP g.leg1*)
  using *0 1* **by** *fastforce*
hence $\exists\,\varrho$. *tabulation* $(\cdot)$ $(\star)$ a i *src trg ?f* ϱ (*Maps.REP g.leg0*) (*Maps.REP g.leg1*)
  using *BS2'* [*of Maps.REP g.leg0 Maps.REP g.leg1 ?f*] *isomorphic-reflexive*
    Span.arr-char
  by *auto*
hence *tabulation V H* a i *src trg ?f*
    (*REP-leg0.trnr*$_\eta$ (*Maps.REP g.leg1*) *?f*) (*Maps.REP g.leg0*) (*Maps.REP g.leg1*)
  using *2 REP-leg0.canonical-tabulation* [*of Maps.REP g.leg1*] **by** *auto*
hence *3*: $\exists\, w\ w'\ \varphi\ \psi\ \vartheta\ \nu\ \vartheta'\ \nu'$.
    equivalence-in-bicategory $(\cdot)$ $(\star)$ a i *src trg* w' w ψ φ $\wedge$
    «w : *src* (tab_0 *?f*) $\rightarrow$ *src* (*Maps.REP g.leg0*)» $\wedge$
    «w' : *src* (*Maps.REP g.leg0*) $\rightarrow$ *src* (tab_0 *?f*)» $\wedge$
    «ϑ : *Maps.REP g.leg0* $\star$ w $\Rightarrow$ tab_0 *?f*» $\wedge$
    «ν : tab_1 *?f* $\Rightarrow$ *Maps.REP g.leg1* $\star$ w» $\wedge$ *iso* ν $\wedge$
    f.tab = (*?f* $\star$ ϑ) $\cdot$ a[*?f*, *Maps.REP g.leg0*, w] $\cdot$
      (*REP-leg0.trnr*$_\eta$ (*Maps.REP g.leg1*) *?f* $\star$ w) $\cdot$ ν $\wedge$
    «ϑ' : tab_0 *?f* $\star$ w' $\Rightarrow$ *Maps.REP g.leg0*» $\wedge$
    «ν' : *Maps.REP g.leg1* $\Rightarrow$ tab_1 *?f* $\star$ w'» $\wedge$ *iso* ν' $\wedge$
    REP-leg0.trnr$_\eta$ (*Maps.REP g.leg1*) *?f* =
    (*?f* $\star$ ϑ') $\cdot$ a[*?f*, tab_0 *?f*, w'] $\cdot$ (*f.tab* $\star$ w') $\cdot$ ν'
      using *f.apex-unique-up-to-equivalence*
        [*of REP-leg0.trnr*$_\eta$ (*Maps.REP g.leg1*) *?f*
          Maps.REP g.leg0 Maps.REP g.leg1]
  by *simp*
obtain w w' φ ψ ϑ ν ϑ' ν'
  where *4*: *equivalence-in-bicategory* $(\cdot)$ $(\star)$ a i *src trg* w' w ψ φ $\wedge$
    «w : *src* (tab_0 *?f*) $\rightarrow$ *src* (*Maps.REP g.leg0*)» $\wedge$
    «w' : *src* (*Maps.REP g.leg0*) $\rightarrow$ *src* (tab_0 *?f*)» $\wedge$
    «ϑ : *Maps.REP g.leg0* $\star$ w $\Rightarrow$ tab_0 *?f*» $\wedge$
    «ν : tab_1 *?f* $\Rightarrow$ *Maps.REP g.leg1* $\star$ w» $\wedge$ *iso* ν $\wedge$
    f.tab = (*?f* $\star$ ϑ) $\cdot$ a[*?f*, *Maps.REP g.leg0*, w] $\cdot$
      (*REP-leg0.trnr*$_\eta$ (*Maps.REP g.leg1*) *?f* $\star$ w) $\cdot$ ν $\wedge$

«ϑ' : tab_0 ?f ⋆ w' ⇒ Maps.REP g.leg0» ∧
«ν' : Maps.REP g.leg1 ⇒ tab_1 ?f ⋆ w'» ∧ iso ν' ∧
REP-leg0.trnr$_\eta$ (Maps.REP g.leg1) ?f =
(?f ⋆ ϑ') · a[?f, tab_0 ?f, w'] · (f.tab ⋆ w') · ν'
using *3* **by** *meson*
hence *w$\vartheta\nu$*: *equivalence-map w* ∧ «w : src (tab_0 ?f) → src (Maps.REP g.leg0)» ∧
«ϑ : Maps.REP g.leg0 ⋆ w ⇒ tab_0 ?f» ∧
«ν : tab_1 ?f ⇒ Maps.REP g.leg1 ⋆ w» ∧ iso ν
using *equivalence-map-def quasi-inverses-def quasi-inverses-symmetric*
by *meson*
let ?W = ⦇Chn = ⟦⟦w⟧⟧, Dom = Dom (SPN ?f), Cod = Dom g⦈
have *W-in-hom*: *Span.in-hom* ?W (SPN ?f) g
proof
have *arrow-of-spans Maps.comp* ?W
proof
interpret *Dom-W*: *span-in-category Maps.comp* ‹Dom ?W›
proof (*unfold-locales*, *intro conjI*)
show Maps.arr (Leg0 (Dom ?W))
apply (*intro* Maps.arrI_{CC})
apply *auto*
by (*metis f.base-simps*(*2*) *f.satisfies-T0 f.u-in-hom src-hcomp*)
show Maps.arr (Leg1 (Dom ?W))
using *1*
apply (*intro* Maps.arrI_{CC})
apply *auto*
proof −
let ?f = tab_1 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)
assume *1*: «Maps.REP g.leg1 : Maps.Dom g.apex → Maps.Cod g.leg1»
have «?f : src (tab_0 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*))
→ Maps.Cod g.leg1» ∧
is-left-adjoint ?f ∧ ⟦tab_1 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧ = ⟦?f⟧
using *1* **by** *simp*
thus ∃f. «f : src (tab_0 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*))
→ Maps.Cod g.leg1» ∧
is-left-adjoint f ∧
⟦tab_1 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧ = ⟦f⟧
by *blast*
qed
show Maps.dom (Leg0 (Dom ?W)) = Maps.dom (Leg1 (Dom ?W))
proof −
have Maps.dom (Leg0 (Dom ?W)) =
Maps.MkIde (src (tab_0 (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)))
using *Maps.dom-char*
apply *simp*
by (*metis* (*no-types*, *lifting*) *Maps.CLS-in-hom Maps.in-homE f.base-simps*(*2*)
f.satisfies-T0 f.u-simps(*3*) *hcomp-simps*(*1*))
also have ... = Maps.dom (Leg1 (Dom ?W))
using *Maps.dom-char Maps.CLS-in-hom f.leg1-is-map f-in-hhom*
apply *simp*

```
    by (metis (no-types, lifting) Maps.in-homE Maps.REP-simps(3) f.base-simps(2)
          f.leg1-is-map f.leg1-simps(3) f.leg1-simps(4) g.dom.leg-simps(3)
          trg-hcomp)
    finally show ?thesis by blast
  qed
qed
show Maps.span (Leg0 (Dom ?W)) (Leg1 (Dom ?W))
  using Dom-W.span-in-category-axioms Dom-W.is-span by blast
interpret Cod-W: span-in-category Maps.comp ‹Cod ?W›
  apply unfold-locales by fastforce
show Maps.span (Leg0 (Cod ?W)) (Leg1 (Cod ?W))
  by fastforce
show Maps.in-hom (Chn ?W) Dom-W.apex Cod-W.apex
proof
  show 1: Maps.arr (Chn ?W)
    using wϑν Maps.CLS-in-hom [of w] equivalence-is-adjoint by auto
  show Maps.dom (Chn ?W) = Dom-W.apex
  proof −
    have Maps.dom (Chn ?W) = Maps.MkIde (src w)
      using 1 wϑν Maps.dom-char by simp
    also have ... = Dom-W.apex
    proof −
      have src w = src (tab₀ ?f)
        using wϑν by blast
      thus ?thesis
        using Dom-W.apex-def Maps.arr-char Maps.dom-char
        apply simp
        by (metis (no-types, lifting) f.base-simps(2) f.satisfies-T0
            f.u-in-hom hcomp-simps(1))
    qed
    finally show ?thesis by fastforce
  qed
  show Maps.cod (Chn ?W) = Cod-W.apex
  proof −
    have Maps.cod (Chn ?W) = Maps.MkIde (trg w)
      using 1 wϑν Maps.cod-char by simp
    also have ... = Cod-W.apex
    proof −
      have trg w = src (Maps.REP g.leg0)
        using wϑν by blast
      thus ?thesis
        using Cod-W.apex-def Maps.arr-char Maps.cod-char
        apply simp
        using g.dom.apex-def Maps.dom-char Maps.REP-simps(2) g.dom.is-span
        by presburger
    qed
    finally show ?thesis by fastforce
  qed
qed
```

```
show Cod-W.leg0 ⊙ Chn ?W = Dom-W.leg0
proof −
  have Cod-W.leg0 ⊙ Chn ?W = g.leg0 ⊙ ⟦⟦w⟧⟧
    by simp
  also have ... = ⟦⟦Maps.REP g.leg0⟧⟧ ⊙ ⟦⟦w⟧⟧
    using g.dom.leg-simps(1) Maps.CLS-REP [of g.leg0]
    by simp
  also have ... = ⟦⟦Maps.REP g.leg0 ⋆ w⟧⟧
  proof −
    have is-left-adjoint (Maps.REP g.leg0)
      by fast
    moreover have is-left-adjoint w
      using wϑν equivalence-is-adjoint by simp
    moreover have Maps.REP g.leg0 ⋆ w ≅ Maps.REP g.leg0 ⋆ w
      using wϑν isomorphic-reflexive Maps.REP-in-hhom
      by (metis (no-types, lifting) REP-leg0.ide-left adjoint-pair-antipar(1)
          calculation(2) ide-hcomp in-hhomE)
    ultimately show ?thesis
      using wϑν Maps.comp-CLS isomorphic-reflexive equivalence-is-adjoint
      by blast
  qed
  also have ... = ⟦⟦tab₀ ?f⟧⟧
  proof −
    have iso ϑ
    proof −
      have is-left-adjoint (Maps.REP g.leg0 ⋆ w)
        using wϑν equivalence-is-adjoint Maps.REP-in-hhom
        by (simp add: g.leg0-is-map in-hhom-def left-adjoints-compose)
      moreover have is-left-adjoint (tab₀ ?f)
        by simp
      ultimately show ?thesis
        using wϑν BS3 by blast
    qed
    thus ?thesis
      using wϑν Maps.CLS-eqI equivalence-is-adjoint
      by (meson isomorphic-def isomorphic-implies-hpar(1))
  qed
  finally show ?thesis by fastforce
qed
show Cod-W.leg1 ⊙ Chn ?W = Dom-W.leg1
proof −
  have Cod-W.leg1 ⊙ Chn ?W = g.leg1 ⊙ ⟦⟦w⟧⟧
    by simp
  also have ... = ⟦⟦Maps.REP g.leg1⟧⟧ ⊙ ⟦⟦w⟧⟧
    using g.dom.leg-simps(3) Maps.CLS-REP by presburger
  also have ... = ⟦⟦Maps.REP g.leg1 ⋆ w⟧⟧
  proof −
    have is-left-adjoint (Maps.REP g.leg1)
      by fast
```

```
      moreover have is-left-adjoint w
        using wϑν equivalence-is-adjoint by simp
      moreover have Maps.REP g.leg1 ⋆ w ≅ Maps.REP g.leg1 ⋆ w
        using wϑν isomorphic-reflexive Maps.REP-in-hhom
        by (metis (no-types, lifting) 2 calculation(2) g.dom.is-span
            hcomp-ide-isomorphic Maps.ide-REP in-hhomE
            right-adjoint-determines-left-up-to-iso)
      ultimately show ?thesis
        using wϑν Maps.comp-CLS isomorphic-reflexive equivalence-is-adjoint
        by blast
    qed
    also have ... = ⟦⟦tab₁ ?f⟧⟧
    proof −
      have ide (Maps.REP g.leg1 ⋆ w)
        using 2 wϑν equivalence-map-is-ide by auto
      moreover have Maps.REP g.leg1 ⋆ w ≅
                    tab₁ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)
        using wϑν equivalence-is-adjoint f.leg1-is-map
              right-adjoint-determines-left-up-to-iso adjoint-pair-preserved-by-iso
        by (meson adjoint-pair-antipar(2) ide-in-hom(2) ide-is-iso)
      ultimately show ?thesis
        using Maps.CLS-eqI by blast
    qed
    finally show ?thesis by fastforce
  qed
qed
thus W: Span.arr ?W
  using Span.arr-char by blast
interpret Dom-W:
  span-in-category Maps.comp
    ‹⦇Leg0 = Maps.MkArr (src (tab₀ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)))
                        (src (Maps.REP g.leg0)*)
                        (iso-class (tab₀ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*))),
      Leg1 = Maps.MkArr (src (tab₀ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)))
                        (Maps.Cod g.leg1)
                        (iso-class (tab₁ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)))⦈›
  using W Span.arr-char
  by (simp add: arrow-of-spans-def)
interpret Cod-W: span-in-category Maps.comp ‹Cod ?W›
  using W Span.arr-char
  by (simp add: arrow-of-spans-def)
show Span.dom ?W = SPN ?f
proof −
  have Span.dom ?W =
        ⦇Chn = Dom-W.apex,
         Dom = ⦇Leg0 = ⟦⟦tab₀ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧⟧,
               Leg1 = ⟦⟦tab₁ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧⟧⦈,
         Cod = ⦇Leg0 = ⟦⟦tab₀ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧⟧,
               Leg1 = ⟦⟦tab₁ (Maps.REP g.leg1 ⋆ (Maps.REP g.leg0)*)⟧⟧⦈⦈
```

```
          using 0 W Span.dom-char by simp
        also have ... = SPN ?f
          using SPN-def Dom-W.apex-def Maps.dom-char Dom-W.is-span iso-class-eqI
                spn-ide
          apply simp
          using ide-f by blast
        finally show ?thesis by blast
      qed
      show Span.cod ?W = g
        using 0 W Span.cod-char Cod-W.apex-def by simp
    qed
    moreover have Span.iso ?W
    proof −
      have Maps.iso ⟦⟦w⟧⟧
      proof −
        have Maps.arr ⟦⟦w⟧⟧ ∧ w ∈ Maps.Map ⟦⟦w⟧⟧ ∧ equivalence-map w
        proof (intro conjI)
          show 1: Maps.arr ⟦⟦w⟧⟧
            using wϑν Maps.CLS-in-hom equivalence-is-adjoint by blast
          show equivalence-map w
            using wϑν by blast
          show w ∈ Maps.Map ⟦⟦w⟧⟧
            using 1 wϑν equivalence-is-adjoint Maps.arr-char
            by (simp add: equivalence-map-is-ide ide-in-iso-class)
        qed
        thus ?thesis
          using Maps.iso-char' by blast
      qed
      thus ?thesis
        using wϑν W-in-hom Span.iso-char by auto
    qed
    ultimately show ?thesis
      using Span.isomorphic-def by blast
  qed
  ultimately show ?thesis by blast
 qed
qed
show ⋀r s τ. ⟦ide r; ide s; src r = src s; trg r = trg s; Span.in-hom τ (SPN r) (SPN s)⟧
              ⟹ ∃μ. «μ : r ⇒ s» ∧ SPN μ = τ
proof −
 fix r s τ
 assume r: ide r and s: ide s
 assume src-eq: src r = src s and trg-eq: trg r = trg s
 assume τ: Span.in-hom τ (SPN r) (SPN s)
 interpret τ: arrow-of-spans Maps.comp τ
   using τ Span.arr-char by auto
 interpret r: identity-in-bicategory-of-spans V H a i src trg r
   using r by unfold-locales auto
 interpret s: identity-in-bicategory-of-spans V H a i src trg s
```

```
  using s by unfold-locales auto
interpret s: arrow-of-tabulations-in-maps V H a i src trg
              s s.tab ‹tab₀ s› ‹tab₁ s› s s.tab ‹tab₀ s› ‹tab₁ s› s
  using s.is-arrow-of-tabulations-in-maps by simp
have τ-dom-leg0-eq: τ.dom.leg0 = ⟦⟦tab₀ r⟧⟧
  using τ Span.dom-char SPN-def [of r] by auto
have τ-dom-leg1-eq: τ.dom.leg1 = ⟦⟦tab₁ r⟧⟧
  using τ Span.dom-char SPN-def [of r] by auto
have τ-cod-leg0-eq: τ.cod.leg0 = ⟦⟦tab₀ s⟧⟧
  using τ Span.cod-char SPN-def [of s] by auto
have τ-cod-leg1-eq: τ.cod.leg1 = ⟦⟦tab₁ s⟧⟧
  using τ Span.cod-char SPN-def [of s] by auto

have 1: tab₀ s ⋆ Maps.REP τ.chine ≅ tab₀ r
proof −
  have tab₀ s ⋆ Maps.REP τ.chine ≅ Maps.REP τ.cod.leg0 ⋆ Maps.REP τ.chine
  proof −
    have Maps.REP τ.cod.leg0 ≅ tab₀ s
      using τ-cod-leg0-eq Maps.CLS-REP Maps.CLS-eqI Maps.REP-CLS s.satisfies-T0
      by presburger
    moreover have src (tab₀ s) = trg (Maps.REP τ.chine)
      using Maps.seq-char [of ⟦⟦tab₀ s⟧⟧ τ.chine] τ.cod.leg-simps(1)
            τ.leg0-commutes τ-cod-leg0-eq
      by auto
    ultimately show ?thesis
      using hcomp-isomorphic-ide [of tab₀ s Maps.REP τ.cod.leg0 Maps.REP τ.chine]
            isomorphic-symmetric
      by fastforce
  qed
  also have ... ≅ Maps.REP τ.dom.leg0
  proof −
   have ⟦⟦Maps.REP τ.cod.leg0⟧⟧ ⊙ ⟦⟦Maps.REP τ.chine⟧⟧ = ⟦⟦Maps.REP τ.dom.leg0⟧⟧
      using τ.leg0-commutes Maps.CLS-REP τ.chine-simps(1)
            τ.cod.leg-simps(1) τ.dom.leg-simps(1)
      by presburger
    hence ⟦⟦Maps.REP τ.cod.leg0 ⋆ Maps.REP τ.chine⟧⟧ = ⟦⟦Maps.REP τ.dom.leg0⟧⟧
      using Maps.comp-CLS [of Maps.REP τ.cod.leg0 Maps.REP τ.chine
                              Maps.REP τ.cod.leg0 ⋆ Maps.REP τ.chine]
            isomorphic-reflexive
  by (metis (no-types, lifting) Maps.seq-char Maps.REP-in-hhom(2) Maps.REP-simps(2−3)
        τ.chine-in-hom τ.cod.leg-in-hom(1) τ.dom.leg-simps(1) τ.leg0-commutes
        ide-hcomp Maps.ide-REP)
    thus ?thesis
      using Maps.CLS-eqI Maps.seq-char Maps.ide-REP
      by (meson calculation isomorphic-implies-ide(2))
  qed
  also have ... ≅ tab₀ r
    using τ-dom-leg0-eq Maps.CLS-REP Maps.CLS-eqI Maps.REP-CLS r.satisfies-T0
    by presburger
```

```
  finally show ?thesis by blast
qed
obtain ϑ where ϑ: «ϑ : tab₀ s ⋆ Maps.REP τ.chine ⇒ tab₀ r» ∧ iso ϑ
  using 1 by blast
have 2: tab₁ s ⋆ Maps.REP τ.chine ≅ tab₁ r
proof −
  have tab₁ s ⋆ Maps.REP τ.chine ≅ Maps.REP τ.cod.leg1 ⋆ Maps.REP τ.chine
  proof −
    have Maps.REP τ.cod.leg1 ≅ tab₁ s
      using τ-cod-leg1-eq Maps.CLS-REP Maps.CLS-eqI Maps.REP-CLS s.leg1-is-map
      by presburger
    moreover have src (Maps.REP τ.cod.leg1) = trg (Maps.REP τ.chine)
      using Maps.seq-char by auto
    ultimately show ?thesis
      using hcomp-isomorphic-ide [of Maps.REP τ.cod.leg1 tab₁ s Maps.REP τ.chine]
            isomorphic-symmetric
      by fastforce
  qed
  also have ... ≅ Maps.REP τ.dom.leg1
  proof −
   have ⟦⟦Maps.REP τ.cod.leg1⟧⟧ ⊙ ⟦⟦Maps.REP τ.chine⟧⟧ = ⟦⟦Maps.REP τ.dom.leg1⟧⟧
      using τ.leg1-commutes Maps.CLS-REP τ.chine-simps(1)
            τ.cod.leg-simps(3) τ.dom.leg-simps(3)
      by presburger
    hence ⟦⟦Maps.REP τ.cod.leg1 ⋆ Maps.REP τ.chine⟧⟧ = ⟦⟦Maps.REP τ.dom.leg1⟧⟧
      using Maps.comp-CLS [of Maps.REP τ.cod.leg1 Maps.REP τ.chine
                              Maps.REP τ.cod.leg1 ⋆ Maps.REP τ.chine]
            isomorphic-reflexive
      by (metis (no-types, lifting) Maps.seq-char Maps.REP-in-hhom(2)
          Maps.REP-simps(2) Maps.REP-simps(3) τ.chine-in-hom τ.cod.leg-in-hom(2)
          τ.dom.leg-simps(3) τ.leg1-commutes ide-hcomp Maps.ide-REP)
    thus ?thesis
      using Maps.CLS-eqI Maps.seq-char Maps.ide-REP
      by (meson calculation isomorphic-implies-ide(2))
  qed
  also have ... ≅ tab₁ r
    using τ-dom-leg1-eq Maps.CLS-REP Maps.CLS-eqI Maps.REP-CLS r.leg1-is-map
    by presburger
  finally show ?thesis by blast
qed
obtain ν where ν: «ν : tab₁ r ⇒ tab₁ s ⋆ Maps.REP τ.chine» ∧ iso ν
  using 2 isomorphic-symmetric by blast

define Δ
  where Δ ≡ (s ⋆ ϑ) · a[s, tab₀ s, Maps.REP τ.chine] · (s.tab ⋆ Maps.REP τ.chine) · ν
have Δ: «Δ : tab₁ r ⇒ s ⋆ tab₀ r»
proof (unfold Δ-def, intro comp-in-homI)
  show «ν : tab₁ r ⇒ tab₁ s ⋆ Maps.REP τ.chine»
    using ν by simp
```

show *3*: «*s.tab* ⋆ *Maps.REP* τ*.chine* :
tab_1 *s* ⋆ *Maps.REP* τ*.chine* ⇒ (*s* ⋆ tab_0 *s*) ⋆ *Maps.REP* τ*.chine*»
apply (*intro hcomp-in-vhom*)
apply *auto*
using *1* **by** *fastforce*
show «a[*s*, tab_0 *s*, *Maps.REP* τ*.chine*] :
(*s* ⋆ tab_0 *s*) ⋆ *Maps.REP* τ*.chine* ⇒ *s* ⋆ tab_0 *s* ⋆ *Maps.REP* τ*.chine*»
using *s assoc-in-hom* [*of s* tab_0 *s Maps.REP* τ*.chine*]
by (*metis* (*no-types*, *lifting*) *Maps.ide-REP 3* τ*.chine-simps*(*1*) *hcomp-in-vhomE*
ideD(*2*) *ideD*(*3*) *s.ide-u s.tab-simps*(*2*) *s.u-simps*(*3*))
show «*s* ⋆ ϑ : *s* ⋆ tab_0 *s* ⋆ *Maps.REP* τ*.chine* ⇒ *s* ⋆ tab_0 *r*»
using *1 s* ϑ *isomorphic-implies-hpar*(*4*) *src-eq* **by** *auto*
qed
define μ **where** $\mu \equiv r.T0.trnr_\varepsilon\ s\ \Delta \cdot inv\ (r.T0.trnr_\varepsilon\ r\ r.tab)$
have μ: «$\mu : r \Rightarrow s$»
proof (*unfold* μ*-def*, *intro comp-in-homI*)
show «$inv\ (r.T0.trnr_\varepsilon\ r\ r.tab) : r \Rightarrow tab_1\ r \star (tab_0\ r)^*$»
using *r.yields-isomorphic-representation* **by** *fastforce*
show «$r.T0.trnr_\varepsilon\ s\ \Delta : tab_1\ r \star (tab_0\ r)^* \Rightarrow s$»
using *s* Δ *src-eq r.T0.adjoint-transpose-right*(*2*) [*of s* tab_1 *r*] **by** *auto*
qed
interpret μ: *arrow-in-bicategory-of-spans V H* a i *src trg r s* μ
using μ **by** *unfold-locales auto*
interpret μ: *arrow-of-tabulations-in-maps V H* a i *src trg*
r r.tab ‹tab_0 *r*› ‹tab_1 *r*› *s s.tab* ‹tab_0 *s*› ‹tab_1 *s*› μ
using μ*.is-arrow-of-tabulations-in-maps* **by** *simp*
have Δ*-eq*: $\Delta = \mu.\Delta$
proof −
have $r.T0.trnr_\varepsilon\ s\ \Delta \cdot inv\ (r.T0.trnr_\varepsilon\ r\ r.tab) =$
$r.T0.trnr_\varepsilon\ s\ \mu.\Delta \cdot inv\ (r.T0.trnr_\varepsilon\ r\ r.tab)$
using μ μ*-def* $\mu.\mu$*-in-terms-of-*Δ **by** *auto*
hence $r.T0.trnr_\varepsilon\ s\ \Delta = r.T0.trnr_\varepsilon\ s\ \mu.\Delta$
using *r s* Δ *r.T0.adjoint-transpose-right*(*2*) *r.yields-isomorphic-representation*
iso-inv-iso iso-is-retraction retraction-is-epi epi-cancel
by (*metis* μ*.in-hom* μ*-def arrI*)
thus *?thesis*
using Δ $\mu.\Delta$*-in-hom*(*2*) *src-eq r.T0.adjoint-transpose-right*(*6*)
bij-betw-imp-inj-on
[*of* $r.T0.trnr_\varepsilon$ *s hom* (tab_1 *r*) (*s* ⋆ tab_0 *r*) *hom* (tab_1 *r* ⋆ (tab_0 *r*)*) *s*]
inj-on-def [*of* $r.T0.trnr_\varepsilon$ *s hom* (tab_1 *r*) (*s* ⋆ tab_0 *r*)]
by *simp*
qed
have μ*.is-induced-map* (*Maps.REP* τ*.chine*)
using ϑ ν Δ*-eq* Δ*-def* μ*.is-induced-map-iff* τ*.chine-simps*(*1*) *Maps.ide-REP* **by** *blast*
hence *3*: *Maps.REP* τ*.chine* ≅ μ*.chine*
using μ*.chine-is-induced-map* μ*.induced-map-unique* **by** *simp*
have *SPN* $\mu = \tau$
proof (*intro Span.arr-eqI*)
show *Span.par* (*SPN* μ) τ

```
        using μ τ SPN-in-hom
        by (metis (no-types, lifting) SPN.preserves-cod SPN.preserves-dom Span.in-homE
            in-homE)
      show Chn (SPN μ) = τ.chine
      proof −
        have Chn (SPN μ) = ⟦⟦spn μ⟧⟧
          using μ SPN-def spn-def by auto
        also have ... = ⟦⟦μ.chine⟧⟧
          using μ spn-def by fastforce
        also have ... = ⟦⟦Maps.REP τ.chine⟧⟧
        using 3 isomorphic-symmetric Maps.CLS-eqI iso-class-eqI isomorphic-implies-hpar(3)
              isomorphic-implies-hpar(4)
          by auto
        also have ... = τ.chine
          using Maps.CLS-REP τ.chine-simps(1) by blast
        finally show ?thesis by blast
      qed
    qed
    thus ∃μ. «μ : r ⇒ s» ∧ SPN μ = τ
      using μ by auto
  qed
qed

theorem SPN-is-equivalence-pseudofunctor:
shows equivalence-pseudofunctor V H a i src trg
        Span.vcomp Span.hcomp Span.assoc Span.unit Span.src Span.trg SPN Φ
  ..
```

We have completed the proof of the second half of the main result (CKS Theorem 4): B is biequivalent (via SPN) to $Span(Maps(B))$.

```
corollary
shows equivalent-bicategories V H a i src trg
        Span.vcomp Span.hcomp Span.assoc Span.unit Span.src Span.trg
  using SPN-is-equivalence-pseudofunctor equivalent-bicategories-def by blast

end

end
```

Bibliography

[1] J. Armstrong. The "strictification" theorem, 2007. https://unapologetic.wordpress.com/2007/07/01/the-strictification-theorem/, [Online; accessed 4-November-2019].

[2] J. Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, volume 47 of *Lecture Notes in Mathematics*, pages 1–77. Springer-Verlag, 1967.

[3] A. Carboni. Bicategories of partial maps. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, XXVIII(2):111–126, 1987.

[4] A. Carboni, S. Kasangian, and R. Street. Bicategories of spans and relations. *Journal of Pure and Applied Algebra*, 33:259–267, 1984.

[5] A. Carboni, G. M. Kelly, R. F. C. Walters, and R. J. Wood. Cartesian bicategories II. *Theory and Applications of Categories*, 19(6):93–124, 2008.

[6] A. Carboni and R. F. C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49:11–32, 1987.

[7] N. Johnson and D. Yau. 2-dimensional categories, 2020. https://arxiv.org/pdf/2002.06055.pdf.

[8] T. Leinster. Basic bicategories, 1998. https://arxiv.org/pdf/math/9810017.pdf.

[9] nLab (various contributors). Adjoint equivalence, 2010. https://ncatlab.org/nlab/show/adjoint+equivalence, [Online; accessed 4-November-2019].

[10] nLab (various contributors). Pseudonatural transformation, 2019. https://ncatlab.org/nlab/show/pseudonatural+transformation, [Online; accessed 11-October-2020].

[11] nLab (various contributors). Lax natural transformation, 2020. https://ncatlab.org/nlab/show/lax+natural+transformation, [Online; accessed 11-October-2020].

[12] M. Shulman. zigzag identity 1 implies 2, 2010. https://ncatlab.org/nlab/show/zigzag%20identity%201%20implies%202%20-%20SVG, [Online; accessed 4-November-2019].

[13] E. W. Stark. Category theory with adjunctions and limits. *Archive of Formal Proofs*, June 2016. http://isa-afp.org/entries/Category3.shtml, Formal proof development.

[14] E. W. Stark. Monoidal categories. *Archive of Formal Proofs*, May 2017. http://isa-afp.org/entries/MonoidalCategory.shtml, Formal proof development.

[15] R. Street. Fibrations in bicategories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, XXI(2):111–159, 1980.

[16] R. H. Street. Fibrations and Yoneda's lemma in a 2-category. In *Lecture Notes in Mathematics 420*, pages 104–133. Springer-Verlag, 1974.