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 ([12] [13]) 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

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 [14] 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.* [15] [14]), 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 [12] and monoidal categories [13]; (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 [12]. 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 [13]. 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 [12], 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 [12], 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 [12] 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* ≡ {*f'*. *f* ≅ *f'*}

definition *is-iso-class*
where *is-iso-class F* ≡ ∃*f*. *f* ∈ *F* ∧ *F* = *iso-class f*

definition *iso-class-rep*
where *iso-class-rep F* ≡ *SOME f*. *f* ∈ *F*

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

lemma *inv-in-homI* [*intro*]:
assumes *iso f* **and** «*f* : *a* → *b*»
shows «*inv f* : *b* → *a*»
⟨*proof*⟩

lemma *iso-class-is-nonempty*:
assumes *is-iso-class F*
shows *F* ≠ {}
⟨*proof*⟩

lemma *iso-class-memb-is-ide*:
assumes *is-iso-class F* **and** *f* ∈ *F*

shows *ide f*
⟨*proof*⟩

lemma *ide-in-iso-class*:
assumes *ide f*
shows $f \in [\![f]\!]$
⟨*proof*⟩

lemma *rep-in-iso-class*:
assumes *is-iso-class F*
shows *iso-class-rep F* $\in$ *F*
⟨*proof*⟩

lemma *is-iso-classI*:
assumes *ide f*
shows *is-iso-class* $[\![f]\!]$
⟨*proof*⟩

lemma *rep-iso-class*:
assumes *ide f*
shows *iso-class-rep* $[\![f]\!] \cong f$
⟨*proof*⟩

lemma *iso-class-elems-isomorphic*:
assumes *is-iso-class F* **and** $f \in F$ **and** $f' \in F$
shows $f \cong f'$
⟨*proof*⟩

lemma *iso-class-eqI* [*intro*]:
assumes $f \cong g$
shows $[\![f]\!] = [\![g]\!]$
⟨*proof*⟩

lemma *iso-class-eq*:
assumes *is-iso-class F* **and** *is-iso-class G* **and** $F \cap G \neq \{\}$
shows $F = G$
⟨*proof*⟩

lemma *iso-class-rep* [*simp*]:
assumes *is-iso-class F*
shows $[\![$*iso-class-rep F*$]\!] = F$
⟨*proof*⟩

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 [12] 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 «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" «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 «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 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** ‹$\star$› *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*⟩

interpretation *H*: *partial-magma H*
⟨*proof*⟩
interpretation *H*: *partial-composition H*
⟨*proof*⟩

lemma *is-partial-composition*:
shows *partial-composition H*
⟨*proof*⟩

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 $H.null = null$
⟨*proof*⟩

lemma *composable-implies-arr*:
assumes $\nu \star \mu \neq null$
shows *arr* μ **and** *arr* ν
⟨*proof*⟩

lemma *hcomp-null* [*simp*]:
shows $null \star \mu = null$ **and** $\mu \star null = null$
⟨*proof*⟩

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** $cod\ (\nu \star \mu) = cod\ \nu \star cod\ \mu$
⟨*proof*⟩

lemma $ide\text{-}hcomp_{WC}$:
assumes *ide f* **and** *ide g* **and** $g \star f \neq null$
shows *ide* $(g \star f)$
⟨*proof*⟩

lemma $hcomp\text{-}in\text{-}hom_{WC}$ [*intro*]:
assumes $\nu \star \mu \neq null$
shows $«\nu \star \mu : dom\ \nu \star dom\ \mu \Rightarrow cod\ \nu \star cod\ \mu»$
⟨*proof*⟩

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*⟩

lemma *isomorphic-implies-equicomposable*:
assumes $f \cong g$
shows $\tau \star f \neq null \longleftrightarrow \tau \star g \neq null$
and $f \star \sigma \neq null \longleftrightarrow g \star \sigma \neq null$
⟨*proof*⟩

lemma *interchange*:
assumes *seq* $\nu\ \mu$ **and** *seq* $\tau\ \sigma$
shows $(\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$
⟨*proof*⟩

lemma *paste-1*:
shows $\nu \star \mu = (cod\ \nu \star \mu) \cdot (\nu \star dom\ \mu)$
⟨*proof*⟩

lemma *paste-2*:
shows $\nu \star \mu = (\nu \star cod\ \mu) \cdot (dom\ \nu \star \mu)$
⟨*proof*⟩

lemma *whisker-left*:
assumes *seq* $\nu\ \mu$ **and** *ide* f
shows $f \star (\nu \cdot \mu) = (f \star \nu) \cdot (f \star \mu)$
⟨*proof*⟩

lemma *whisker-right*:
assumes *seq* $\nu\ \mu$ **and** *ide* f
shows $(\nu \cdot \mu) \star f = (\nu \star f) \cdot (\mu \star f)$
⟨*proof*⟩

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 $right\ \sigma\ \tau \longleftrightarrow left\ \tau\ \sigma$
⟨*proof*⟩

lemma *left-respects-isomorphic*:
assumes $f \cong g$
shows *left* f = *left* g
⟨*proof*⟩

lemma *right-respects-isomorphic*:
assumes $f \cong g$
shows *right* f = *right* g
⟨*proof*⟩

lemma *left-iff-left-inv*:
assumes *iso* μ
shows *left* τ $\mu \longleftrightarrow$ *left* τ (*inv* μ)
⟨*proof*⟩

lemma *right-iff-right-inv*:
assumes *iso* μ
shows *right* σ $\mu \longleftrightarrow$ *right* σ (*inv* μ)
⟨*proof*⟩

lemma *left-hom-is-subcategory*:
assumes *arr* μ
shows *subcategory* V (*left* μ)
⟨*proof*⟩

lemma *right-hom-is-subcategory*:
assumes *arr* μ
shows *subcategory* V (*right* μ)
⟨*proof*⟩

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*⟩

lemma *endofunctor-*H_R:
assumes *ide f* **and** $f \star f \neq null$
shows *endofunctor* (*Right f*) (H_R *f*)
⟨*proof*⟩

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** ‹·$_S$› *55*)
notation *S.in-hom* (‹«- : - ⇒$_S$ -»›)

lemma *right-hcomp-closed*:
assumes «μ : *x* ⇒$_S$ *y*» **and** «ν : *c* ⇒ *d*» **and** $\mu \star \nu \neq null$
shows «μ ⋆ ν : *x* ⋆ *c* ⇒$_S$ *y* ⋆ *d*»
⟨*proof*⟩

lemma *interchange*:
assumes *S.seq* ν μ **and** *S.seq* τ σ **and** $\mu \star \sigma \neq null$
shows (ν ·$_S$ μ) ⋆ (τ ·$_S$ σ) = (ν ⋆ τ) ·$_S$ (μ ⋆ σ)
⟨*proof*⟩

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

lemma *iso-char*:
assumes *S.arr* φ
shows *S.iso* φ ⟷ *iso* φ
⟨*proof*⟩

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 $\omega :: 'a +$
assumes *arr-ω*: $arr\ \omega$
begin

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

lemma *left-hcomp-closed*:
assumes «$\mu : x \Rightarrow_S y$» **and** «$\nu : c \Rightarrow d$» **and** $\nu \star \mu \neq null$
shows «$\nu \star \mu : c \star x \Rightarrow_S d \star y$»
⟨*proof*⟩

lemma *interchange*:
assumes *S.seq* $\nu\ \mu$ **and** *S.seq* $\tau\ \sigma$ **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*⟩

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

lemma *iso-char*:
assumes *S.arr* φ
shows *S.iso* $\varphi \longleftrightarrow$ *iso* φ
⟨*proof*⟩

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-unit a* ≡ *a* ⋆ *a* ≅ *a* ∧
fully-faithful-functor (*Left a*) (*Left a*) (*H*$_L$ *a*) ∧
fully-faithful-functor (*Right a*) (*Right a*) (*H*$_R$ *a*)

lemma *weak-unit-self-composable*:
assumes *weak-unit a*
shows *ide a* **and** *ide* (*a* ⋆ *a*) **and** *a* ⋆ *a* ≠ *null*
⟨*proof*⟩

lemma *weak-unit-self-right*:
assumes *weak-unit a*
shows *right a a*
⟨*proof*⟩

lemma *weak-unit-self-left*:
assumes *weak-unit a*
shows *left a a*
⟨*proof*⟩

lemma *weak-unit-in-vhom*:
assumes *weak-unit a*
shows «*a* : *a* ⇒ *a*»
⟨*proof*⟩

If *a* is a weak unit, then there exists a "unit isomorphism" «*ι* : *a* ⋆ *a* ⇒ *a*». It need not be unique, but we may choose one arbitrarily.

definition *some-unit*
where *some-unit a* ≡ *SOME ι*. *iso ι* ∧ «*ι* : *a* ⋆ *a* ⇒ *a*»

lemma *iso-some-unit*:
assumes *weak-unit a*
shows *iso* (*some-unit a*)
and «*some-unit a* : *a* ⋆ *a* ⇒ *a*»
⟨*proof*⟩

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 μ* ≡ {*a*. *weak-unit a* ∧ *μ* ⋆ *a* ≠ *null*}

lemma *sourcesI* [*intro*]:
assumes *weak-unit a* **and** *μ* ⋆ *a* ≠ *null*
shows *a* ∈ *sources μ*
⟨*proof*⟩

lemma *sourcesD* [*dest*]:
assumes *a* ∈ *sources μ*
shows *ide a* **and** *weak-unit a* **and** *μ* ⋆ *a* ≠ *null*

⟨*proof*⟩

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

lemma *targetsI* [*intro*]:
assumes *weak-unit* b **and** $b \star \mu \neq \textit{null}$
shows $b \in \textit{targets}\ \mu$
⟨*proof*⟩

lemma *targetsD* [*dest*]:
assumes $b \in \textit{targets}\ \mu$
shows *ide* b **and** *weak-unit* b **and** $b \star \mu \neq \textit{null}$
⟨*proof*⟩

lemma *sources-dom* [*simp*]:
assumes *arr* μ
shows *sources* (*dom* μ) = *sources* μ
⟨*proof*⟩

lemma *sources-cod* [*simp*]:
assumes *arr* μ
shows *sources* (*cod* μ) = *sources* μ
⟨*proof*⟩

lemma *targets-dom* [*simp*]:
assumes *arr* μ
shows *targets* (*dom* μ) = *targets* μ
⟨*proof*⟩

lemma *targets-cod* [*simp*]:
assumes *arr* μ
shows *targets* (*cod* μ) = *targets* μ
⟨*proof*⟩

lemma *weak-unit-iff-self-source*:
shows *weak-unit* $a \longleftrightarrow a \in \textit{sources}\ a$
⟨*proof*⟩

lemma *weak-unit-iff-self-target*:
shows *weak-unit* $b \longleftrightarrow b \in \textit{targets}\ b$
⟨*proof*⟩

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

lemma *sources-hcomp*:
assumes $\nu \star \mu \neq \textit{null}$
shows *sources* ($\nu \star \mu$) = *sources* μ

⟨*proof*⟩

lemma *targets-hcomp*:
assumes $\nu \star \mu \neq null$
shows *targets* $(\nu \star \mu) =$ *targets* ν
⟨*proof*⟩

lemma H_R*-preserved-along-iso*:
assumes *weak-unit* a **and** $a \cong a'$
shows *endofunctor* (*Right* a) (H_R a')
⟨*proof*⟩

lemma H_L*-preserved-along-iso*:
assumes *weak-unit* a **and** $a \cong a'$
shows *endofunctor* (*Left* a) (H_L a')
⟨*proof*⟩

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

lemma *sources-determine-composability*:
assumes $a \in sources\ \tau$
shows $\tau \star \mu \neq null \longleftrightarrow a \star \mu \neq null$
⟨*proof*⟩

lemma *targets-determine-composability*:
assumes $b \in targets\ \mu$
shows $\tau \star \mu \neq null \longleftrightarrow \tau \star b \neq null$
⟨*proof*⟩

lemma *composable-if-connected*:
assumes *sources* $\nu \cap$ *targets* $\mu \neq \{\}$
shows $\nu \star \mu \neq null$
⟨*proof*⟩

lemma *connected-if-composable*:
assumes $\nu \star \mu \neq null$
shows *sources* $\nu =$ *targets* μ
⟨*proof*⟩

lemma *iso-hcomp*$_{RWC}$:
assumes *iso* μ **and** *iso* ν **and** *sources* $\nu \cap$ *targets* $\mu \neq \{\}$
shows *iso* $(\nu \star \mu)$
and *inverse-arrows* $(\nu \star \mu)$ $(inv\ \nu \star inv\ \mu)$
⟨*proof*⟩

lemma *inv-hcomp*$_{RWC}$:
assumes *iso* μ **and** *iso* ν **and** *sources* $\nu \cap$ *targets* $\mu \neq \{\}$
shows *inv* $(\nu \star \mu) = inv\ \nu \star inv\ \mu$
⟨*proof*⟩

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*$_{AWC}$:
$[\![\ \tau \star \mu \neq null;\ \mu \star \nu \neq null\ ]\!] \Longrightarrow$
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]
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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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* ⇒ *'a* ⇒ *'a* ⇒ *'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*
⟨*proof*⟩
interpretation R: *fully-faithful-functor S.comp S.comp R*
⟨*proof*⟩

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

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

lemma *iso-unit*:
shows *S.iso* ι **and** «ι : $a \star a \Rightarrow_S a$»
⟨*proof*⟩

lemma *characteristic-iso*:
assumes *S.ide f*
shows «$\mathrm{a}[f, a, a] : (f \star a) \star a \Rightarrow_S f \star a \star a$»
and «$f \star \iota : f \star a \star a \Rightarrow_S f \star a$»
and «$(f \star \iota) \cdot_S \mathrm{a}[f, a, a] : R\ (R\ f) \Rightarrow_S R\ f$»
and *S.iso* $((f \star \iota) \cdot_S \mathrm{a}[f, a, a])$
⟨*proof*⟩

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 $\exists!\mu$. «$\mu : R\ f \Rightarrow_S f$» ∧ $R\ \mu = (f \star \iota) \cdot_S \mathrm{a}[f, a, a]$
⟨*proof*⟩

lemma *iso-runit*:
assumes *S.ide f*
shows *S.iso* $\mathrm{r}[f]$
⟨*proof*⟩

lemma *runit-eqI*:
assumes «$f : a \Rightarrow_S b$» **and** «$\mu : R\ f \Rightarrow_S f$»
and $R\ \mu = ((f \star \iota) \cdot_S \mathrm{a}[f, a, a])$
shows $\mu = \mathrm{r}[f]$
⟨*proof*⟩

lemma *runit-naturality*:
assumes *S.arr* μ
shows $\mathrm{r}[S.cod\ \mu] \cdot_S R\ \mu = \mu \cdot_S \mathrm{r}[S.dom\ \mu]$
⟨*proof*⟩

abbreviation $\mathfrak{r}$
where $\mathfrak{r}\ \mu \equiv$ *if S.arr* μ *then* $\mu \cdot_S \mathrm{r}[S.dom\ \mu]$ *else null*

interpretation $\mathfrak{r}$: *natural-transformation S.comp S.comp R S.map* $\mathfrak{r}$
⟨*proof*⟩

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

interpretation $\mathfrak{r}$: *natural-isomorphism S.comp S.comp R S.map* $\mathfrak{r}$
⟨*proof*⟩

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

interpretation *R*: *equivalence-functor S.comp S.comp R*
⟨*proof*⟩

lemma *equivalence-functor-R*:
shows *equivalence-functor S.comp S.comp R*
⟨*proof*⟩

lemma *runit-commutes-with-R*:
assumes *S.ide f*
shows r[*R f*] = *R* r[*f*]
⟨*proof*⟩

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* ≡ H_L *b*

interpretation *L*: *endofunctor S.comp L*
⟨*proof*⟩
interpretation *L*: *fully-faithful-functor S.comp S.comp L*
⟨*proof*⟩

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

definition *lunit* (‹l[-]›)
where *lunit f* ≡ *THE* μ. «μ : *L f* ⇒$_S$ *f*» ∧ *L* μ = (ι ⋆ *f*) ·$_S$ (*inv* a[*b*, *b*, *f*])

lemma *iso-unit*:
shows *S.iso* ι **and** «ι : *b* ⋆ *b* ⇒$_S$ *b*»
⟨*proof*⟩

lemma *characteristic-iso*:
assumes *S.ide f*
shows «*inv* a[*b*, *b*, *f*] : *b* ⋆ *b* ⋆ *f* ⇒$_S$ (*b* ⋆ *b*) ⋆ *f*»
and «ι ⋆ *f* : (*b* ⋆ *b*) ⋆ *f* ⇒$_S$ *b* ⋆ *f*»
and «(ι ⋆ *f*) ·$_S$ *inv* a[*b*, *b*, *f*] : *L* (*L f*) ⇒$_S$ *L f*»
and *S.iso* ((ι ⋆ *f*) ·$_S$ *inv* a[*b*, *b*, *f*])
⟨*proof*⟩

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*⟩

lemma *iso-lunit*:
assumes *S.ide f*
shows *S.iso* l[*f*]
⟨*proof*⟩

lemma *lunit-eqI*:
assumes «*f* : *a* ⇒$_S$ *b*» **and** «μ : *L f* ⇒$_S$ *f*»
and *L* μ = ((ι ⋆ *f*) ·$_S$ *inv* a[*b*, *b*, *f*])
shows μ = l[*f*]
⟨*proof*⟩

lemma *lunit-naturality*:
assumes *S.arr* μ
shows l[*S.cod* μ] ·$_S$ *L* μ = μ ·$_S$ l[*S.dom* μ]
⟨*proof*⟩

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

interpretation ƚ: *natural-transformation S.comp S.comp L S.map* ƚ
⟨*proof*⟩

lemma *natural-transformation*-ƚ:

shows *natural-transformation S.comp S.comp L S.map* ι ⟨*proof*⟩

interpretation ι: *natural-isomorphism S.comp S.comp L S.map* ι
⟨*proof*⟩

lemma *natural-isomorphism-*ι:
shows *natural-isomorphism S.comp S.comp L S.map* ι ⟨*proof*⟩

interpretation *L*: *equivalence-functor S.comp S.comp L*
⟨*proof*⟩

lemma *equivalence-functor-L*:
shows *equivalence-functor S.comp S.comp L*
⟨*proof*⟩

lemma *lunit-commutes-with-L*:
assumes *S.ide f*
shows l[*L f*] = *L* l[*f*]
⟨*proof*⟩

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* μ ⟹ *sources* μ ≠ {}
and *arr-has-target*: *arr* μ ⟹ *targets* μ ≠ {}
begin

lemma *arr-iff-has-src*:
shows *arr* μ ⟷ *sources* μ ≠ {}
⟨*proof*⟩

lemma *arr-iff-has-trg*:
shows *arr* μ ⟷ *targets* μ ≠ {}
⟨*proof*⟩

end

The horizontal composition of a prebicategory is regular.

sublocale *prebicategory* ⊆ *regular-weak-composition V H*
⟨*proof*⟩

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 $a \in$ *sources* μ **and** $a' \in$ *sources* μ
shows $a \cong a'$
⟨*proof*⟩

lemma *targets-are-isomorphic*:
assumes $b \in$ *targets* μ **and** $b' \in$ *targets* μ
shows $b \cong b'$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *isomorphism-respects-weak-units*:
assumes *weak-unit a* **and** $a \cong a'$
shows *weak-unit* a'
⟨*proof*⟩

lemma *sources-iso-closed*:
assumes $a \in$ *sources* μ **and** $a \cong a'$
shows $a' \in$ *sources* μ
⟨*proof*⟩

lemma *targets-iso-closed*:
assumes $a \in$ *targets* μ **and** $a \cong a'$
shows $a' \in$ *targets* μ
⟨*proof*⟩

lemma *sources-eqI*:
assumes *sources* $\mu \cap$ *sources* $\nu \neq \{\}$
shows *sources* $\mu =$ *sources* ν
⟨*proof*⟩

lemma *targets-eqI*:
assumes *targets* $\mu \cap$ *targets* $\nu \neq \{\}$
shows *targets* $\mu =$ *targets* ν
⟨*proof*⟩

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

lemma *sources-char*:
assumes *weak-unit a*
shows *sources* $a = \{x.\ x \cong a\}$
⟨*proof*⟩

lemma *targets-char*:
assumes *weak-unit a*
shows *targets a* = {*x*. *x* ≅ *a*}
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *objI-src*:
assumes *arr a* **and** *src a* = *a*
shows *obj a*
⟨*proof*⟩

lemma *objI-trg*:
assumes *arr a* **and** *trg a* = *a*
shows *obj a*

⟨*proof*⟩

lemma *objE* [*elim*]:
assumes *obj a* **and** ⟦ *ide a*; *src a* = *a*; *trg a* = *a* ⟧ ⟹ *T*
shows *T*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *obj-src* [*intro*, *simp*]:
assumes *arr* μ
shows *obj* (*src* μ)
⟨*proof*⟩

lemma *obj-trg* [*intro*, *simp*]:
assumes *arr* μ
shows *obj* (*trg* μ)
⟨*proof*⟩

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 $hseq_{HH}$ ≡ λμ 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*»
⟨*proof*⟩

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

lemma *ide-in-hom* [*intro*]:
assumes *ide f*
shows «*f* : *src f* → *trg f*» **and** «*f* : *f* ⇒ *f*»
⟨*proof*⟩

lemma *src-dom* [*simp*]:
shows *src* (*dom* μ) = *src* μ
⟨*proof*⟩

lemma *src-cod* [*simp*]:
shows *src* (*cod* μ) = *src* μ
⟨*proof*⟩

lemma *trg-dom* [*simp*]:
shows *trg* (*dom* μ) = *trg* μ
⟨*proof*⟩

lemma *trg-cod* [*simp*]:
shows *trg* (*cod* μ) = *trg* μ
⟨*proof*⟩

lemma *dom-src* [*simp*]:
shows *dom* (*src* μ) = *src* μ
⟨*proof*⟩

lemma *cod-src* [*simp*]:
shows *cod* (*src* μ) = *src* μ
⟨*proof*⟩

lemma *dom-trg* [*simp*]:
shows *dom* (*trg* μ) = *trg* μ
⟨*proof*⟩

lemma *cod-trg* [*simp*]:
shows *cod* (*trg* μ) = *trg* μ
⟨*proof*⟩

lemma *vcomp-in-hhom* [*intro*, *simp*]:
assumes *seq* ν μ **and** *src* ν = a **and** *trg* ν = b
shows «$\nu \cdot \mu : a \to b$»
⟨*proof*⟩

lemma *src-vcomp* [*simp*]:
assumes *seq* ν μ
shows *src* ($\nu \cdot \mu$) = *src* ν
⟨*proof*⟩

lemma *trg-vcomp* [*simp*]:
assumes *seq* ν μ
shows *trg* ($\nu \cdot \mu$) = *trg* ν
⟨*proof*⟩

lemma *vseq-implies-hpar*:

assumes *seq* ν μ
shows *src* ν = *src* μ **and** *trg* ν = *trg* μ
⟨*proof*⟩

lemma *vconn-implies-hpar*:
assumes «μ : f ⇒ g»
shows *src* μ = *src* f **and** *trg* μ = *trg* f **and** *src* g = *src* f **and** *trg* g = *trg* f
⟨*proof*⟩

lemma *src-inv* [*simp*]:
assumes *iso* μ
shows *src* (*inv* μ) = *src* μ
⟨*proof*⟩

lemma *trg-inv* [*simp*]:
assumes *iso* μ
shows *trg* (*inv* μ) = *trg* μ
⟨*proof*⟩

lemma *inv-in-hhom* [*intro*, *simp*]:
assumes *iso* μ **and** *src* μ = a **and** *trg* μ = b
shows «*inv* μ : a → b»
⟨*proof*⟩

lemma *hhom-is-subcategory*:
shows *subcategory* V (λμ. «μ : a → b»)
⟨*proof*⟩

lemma *isomorphic-objects-are-equal*:
assumes *obj* a **and** *obj* b **and** $a \cong b$
shows a = b
⟨*proof*⟩

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 ⟨*proof*⟩
sublocale *VV*: *subcategory* *VxV.comp* ‹λ$\mu\nu$. *hseq*$_{H\,H}$ (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
⟨*proof*⟩

lemma *subcategory-VV*:
shows *subcategory* *VxV.comp* (λ$\mu\nu$. *hseq*$_{H\,H}$ (*fst* $\mu\nu$) (*snd* $\mu\nu$))
⟨*proof*⟩

sublocale *VxVxV*: *product-category* V *VxV.comp* ⟨*proof*⟩
sublocale *VVV*: *subcategory* *VxVxV.comp*
‹λ$\tau\mu\nu$. *arr* (*fst* $\tau\mu\nu$) ∧ *VV.arr* (*snd* $\tau\mu\nu$) ∧
src (*fst* $\tau\mu\nu$) = *trg* (*fst* (*snd* $\tau\mu\nu$))›
⟨*proof*⟩

lemma *subcategory-VVV*:
shows *subcategory VxVxV.comp*
(λτμν. *arr* (*fst* τμν) ∧ *VV.arr* (*snd* τμν) ∧
src (*fst* τμν) = *trg* (*fst* (*snd* τμν)))
⟨*proof*⟩

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* μ ⟹ *src* μ ∈ *sources* μ
and *trg-in-targets*: *arr* μ ⟹ *trg* μ ∈ *targets* μ
and *src-hcomp'*: ν ⋆ μ ≠ *null* ⟹ *src* (ν ⋆ μ) = *src* μ
and *trg-hcomp'*: ν ⋆ μ ≠ *null* ⟹ *trg* (ν ⋆ μ) = *trg* ν
and *seq-if-composable*: ν ⋆ μ ≠ *null* ⟹ *src* ν = *trg* μ

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

lemma *composable-char*$_{PBH}$:
shows ν ⋆ μ ≠ *null* ⟷ *arr* μ ∧ *arr* ν ∧ *src* ν = *trg* μ
⟨*proof*⟩

lemma *hcomp-in-hom*$_{PBH}$:
assumes «μ : *a* →$_{WC}$ *b*» **and** «ν : *b* →$_{WC}$ *c*»
shows «ν ⋆ μ : *a* →$_{WC}$ *c*»
and «ν ⋆ μ : *dom* ν ⋆ *dom* μ ⇒ *cod* ν ⋆ *cod* μ»
⟨*proof*⟩

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*⟩

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* ≡ *SOME f′*. *f′* ∈ { *f′*. *f* ≅ *f′* }

definition *some-src*
where *some-src μ* ≡ *if arr μ then rep* (*SOME a*. *a* ∈ *sources μ*) *else null*

definition *some-trg*
where *some-trg μ* ≡ *if arr μ then rep* (*SOME b*. *b* ∈ *targets μ*) *else null*

lemma *isomorphic-ide-rep*:
assumes *ide f*
shows *f* ≅ *rep f*
⟨*proof*⟩

lemma *rep-rep*:
assumes *ide f*
shows *rep* (*rep f*) = *rep f*
⟨*proof*⟩

lemma *some-src-in-sources*:
assumes *arr μ*
shows *some-src μ* ∈ *sources μ*
⟨*proof*⟩

lemma *some-trg-in-targets*:
assumes *arr μ*
shows *some-trg μ* ∈ *targets μ*
⟨*proof*⟩

lemma *some-src-dom*:
assumes *arr μ*
shows *some-src* (*dom μ*) = *some-src μ*
⟨*proof*⟩

lemma *some-src-cod*:
assumes *arr μ*
shows *some-src* (*cod μ*) = *some-src μ*
⟨*proof*⟩

lemma *some-trg-dom*:

assumes *arr* μ
shows *some-trg* (*dom* μ) = *some-trg* μ
⟨*proof*⟩

lemma *some-trg-cod*:
assumes *arr* μ
shows *some-trg* (*cod* μ) = *some-trg* μ
⟨*proof*⟩

lemma *ide-some-src*:
assumes *arr* μ
shows *ide* (*some-src* μ)
⟨*proof*⟩

lemma *ide-some-trg*:
assumes *arr* μ
shows *ide* (*some-trg* μ)
⟨*proof*⟩

lemma *some-src-composable*:
assumes *arr* τ
shows $\tau \star \mu \neq$ *null* $\longleftrightarrow$ *some-src* $\tau \star \mu \neq$ *null*
⟨*proof*⟩

lemma *some-trg-composable*:
assumes *arr* σ
shows $\mu \star \sigma \neq$ *null* $\longleftrightarrow$ $\mu \star$ *some-trg* $\sigma \neq$ *null*
⟨*proof*⟩

lemma *sources-some-src*:
assumes *arr* μ
shows *sources* (*some-src* μ) = *sources* μ
⟨*proof*⟩

lemma *targets-some-trg*:
assumes *arr* μ
shows *targets* (*some-trg* μ) = *targets* μ
⟨*proof*⟩

lemma *src-some-src*:
assumes *arr* μ
shows *some-src* (*some-src* μ) = *some-src* μ
⟨*proof*⟩

lemma *trg-some-trg*:
assumes *arr* μ
shows *some-trg* (*some-trg* μ) = *some-trg* μ
⟨*proof*⟩

lemma *sources-char′*:
assumes *arr* μ
shows $a \in$ *sources* $\mu \longleftrightarrow$ *some-src* $\mu \cong a$
⟨*proof*⟩

lemma *targets-char′*:
assumes *arr* μ
shows $a \in$ *targets* $\mu \longleftrightarrow$ *some-trg* $\mu \cong a$
⟨*proof*⟩

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-src* $\tau =$ *some-trg* σ
⟨*proof*⟩

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-src* $a \longleftrightarrow a =$ *some-trg* a
⟨*proof*⟩

lemma *some-trg-some-src*:
assumes *arr* μ
shows *some-trg* (*some-src* μ) = *some-src* μ
⟨*proof*⟩

lemma *src-some-trg*:
assumes *arr* μ
shows *some-src* (*some-trg* μ) = *some-trg* μ
⟨*proof*⟩

lemma *some-src-eqI*:
assumes $a \in$ *sources* μ **and** *some-src* $a = a$
shows *some-src* $\mu = a$
⟨*proof*⟩

lemma *some-trg-eqI*:
assumes $b \in$ *targets* μ **and** *some-trg* $b = b$
shows *some-trg* $\mu = b$
⟨*proof*⟩

lemma *some-src-comp*:
assumes $\tau \star \sigma \neq$ *null*
shows *some-src* $(\tau \star \sigma)$ = *some-src* σ
⟨*proof*⟩

lemma *some-trg-comp*:

assumes $\tau \star \sigma \neq \mathit{null}$
shows *some-trg* $(\tau \star \sigma) = \mathit{some\text{-}trg}\ \tau$
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation T: *endofunctor V some-trg*
⟨*proof*⟩

interpretation *weak-composition-with-homs V H some-src some-trg*
⟨*proof*⟩

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

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

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* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›)
and i :: *'a* ⇒ *'a* (‹i[-]›) +
assumes *unit-in-vhom*$_{PBU}$: *weak-unit a* ⟹ «i[a] : $a \star a \Rightarrow a$»
and *iso-unit*$_{PBU}$: *weak-unit a* ⟹ *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*⟩

lemma *unit-simps* [*simp*]:
assumes *weak-unit a*
shows *arr* i[a] **and** *dom* i[a] = $a \star a$ **and** *cod* i[a] = a

⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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* ν μ ≡ *arr* (ν ⋆ μ)

lemma *hseq-char*:
shows *hseq* ν μ ⟷ *arr* μ ∧ *arr* ν ∧ *src* ν = *trg* μ
⟨*proof*⟩

lemma *hseq-char'*:
shows *hseq* ν μ $\longleftrightarrow$ $\nu \star \mu \neq$ *null*
⟨*proof*⟩

lemma *hseqI'* [*intro*, *simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *hseq* ν μ
⟨*proof*⟩

lemma *hseqI*:
assumes «μ : $a \to b$» **and** «ν : $b \to c$»
shows *hseq* ν μ
⟨*proof*⟩

lemma *hseqE* [*elim*]:
assumes *hseq* ν μ
and *arr* $\mu \Longrightarrow$ *arr* $\nu \Longrightarrow$ *src* ν = *trg* $\mu \Longrightarrow T$
shows T
⟨*proof*⟩

lemma *hcomp-simps* [*simp*]:
assumes *hseq* ν μ
shows *src* ($\nu \star \mu$) = *src* μ **and** *trg* ($\nu \star \mu$) = *trg* ν
and *dom* ($\nu \star \mu$) = *dom* $\nu \star$ *dom* μ **and** *cod* ($\nu \star \mu$) = *cod* $\nu \star$ *cod* μ
⟨*proof*⟩

lemma *ide-hcomp* [*intro*, *simp*]:
assumes *ide* ν **and** *ide* μ **and** *src* ν = *trg* μ
shows *ide* ($\nu \star \mu$)
⟨*proof*⟩

lemma *hcomp-in-hhom* [*intro*]:
assumes «μ : $a \to b$» **and** «ν : $b \to c$»
shows «$\nu \star \mu$: $a \to c$»
⟨*proof*⟩

lemma *hcomp-in-hhom'* :
assumes *arr* μ **and** *arr* ν **and** *src* μ = a **and** *trg* ν = c **and** *src* ν = *trg* μ
shows «$\nu \star \mu$: $a \to c$»
⟨*proof*⟩

lemma *hcomp-in-hhomE* [*elim*]:
assumes «$\nu \star \mu$: $a \to c$»
and ⟦ *arr* μ; *arr* ν; *src* ν = *trg* μ; *src* μ = a; *trg* ν = c ⟧ $\Longrightarrow T$
shows T
⟨*proof*⟩

lemma *hcomp-in-vhom* [*intro*]:

assumes «$\mu : f \Rightarrow g$» **and** «$\nu : h \Rightarrow k$» **and** $src\ h = trg\ f$
shows «$\nu \star \mu : h \star f \Rightarrow k \star g$»
⟨*proof*⟩

lemma *hcomp-in-vhom′* :
assumes $hseq\ \nu\ \mu$
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 «$\nu \star \mu : h \star f \Rightarrow k \star g$»
⟨*proof*⟩

lemma *hcomp-in-vhomE* [*elim*]:
assumes «$\nu \star \mu : f \Rightarrow g$»
and ⟦ $arr\ \mu$; $arr\ \nu$; $src\ \nu = trg\ \mu$; $src\ \mu = src\ f$; $src\ \mu = src\ g$;
$trg\ \nu = trg\ f$; $trg\ \nu = trg\ g$ ⟧ ⟹ T
shows T
⟨*proof*⟩

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 $\mu \star \tau \neq null$
⟨*proof*⟩

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

lemma *match-3*:
assumes $\mu \star \tau \neq null$ **and** $\nu \star \mu \neq null$
shows $(\nu \star \mu) \star \tau \neq null$
⟨*proof*⟩

lemma *match-4*:
assumes $\mu \star \tau \neq null$ **and** $\nu \star \mu \neq null$
shows $\nu \star (\mu \star \tau) \neq null$
⟨*proof*⟩

lemma *left-connected*:
assumes $seq\ \nu\ \nu'$
shows $\nu \star \mu \neq null \longleftrightarrow \nu' \star \mu \neq null$
⟨*proof*⟩

lemma *right-connected*:
assumes $seq\ \mu\ \mu'$
shows $H\ \nu\ \mu \neq null \longleftrightarrow H\ \nu\ \mu' \neq null$
⟨*proof*⟩

proposition *is-weak-composition*:
shows *weak-composition V H*
⟨*proof*⟩

interpretation *weak-composition V H*
⟨*proof*⟩

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 $(\nu \cdot \mu) \star (\tau \cdot \sigma) = (\nu \star \tau) \cdot (\mu \star \sigma)$
⟨*proof*⟩

lemma *whisker-right*:
assumes *ide f* **and** *seq* ν μ
shows $(\nu \cdot \mu) \star f = (\nu \star f) \cdot (\mu \star f)$
⟨*proof*⟩

lemma *whisker-left*:
assumes *ide f* **and** *seq* ν μ
shows $f \star (\nu \cdot \mu) = (f \star \nu) \cdot (f \star \mu)$
⟨*proof*⟩

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*⟩

lemma *iso-hcomp* [*intro*, *simp*]:
assumes *iso* μ **and** *iso* ν **and** *src* ν = *trg* μ
shows *iso* $(\nu \star \mu)$
⟨*proof*⟩

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* $(\nu \star \mu)$ $(h \star f)$ $(k \star g)$
⟨*proof*⟩

lemma *isomorphic-implies-ide*:
assumes $f \cong g$
shows *ide f* **and** *ide g*
⟨*proof*⟩

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*⟩

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*⟩

lemma *isomorphic-implies-hpar*:
assumes $f \cong f'$
shows *ide f* **and** *ide f'* **and** *src f* = *src f'* **and** *trg f* = *trg f'*
⟨*proof*⟩

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
⟨*proof*⟩

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*
⟨*proof*⟩

sublocale *HoHV*: *functor VVV.comp V HoHV*
⟨*proof*⟩

lemma *functor-HoVH*:
shows *functor VVV.comp V HoVH*
⟨*proof*⟩

sublocale *HoVH*: *functor VVV.comp V HoVH*
⟨*proof*⟩

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

abbreviation *L*
where $L \equiv \lambda\mu.$ *if arr* μ *then trg* $\mu \star \mu$ *else null*

abbreviation *R*
where $R \equiv \lambda\mu.$ *if arr* μ *then* $\mu \star$ *src* μ *else null*

sublocale *L*: *endofunctor V L*
⟨*proof*⟩

lemma *endofunctor-L*:
shows *endofunctor V L*
⟨*proof*⟩

sublocale *R*: *endofunctor V R*
⟨*proof*⟩

lemma *endofunctor-R*:
shows *endofunctor V R*
⟨*proof*⟩

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 $\alpha\ \mu\nu\tau \equiv \mathrm{a}\ (fst\ \mu\nu\tau)\ (fst\ (snd\ \mu\nu\tau))\ (snd\ (snd\ \mu\nu\tau))$

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* $\mathrm{a}[\mu, \nu, \tau]$ (*src* τ) (*trg* μ)
and «$\mathrm{a}[\mu, \nu, \tau] : (dom\ \mu \star dom\ \nu) \star dom\ \tau \Rightarrow cod\ \mu \star cod\ \nu \star cod\ \tau$»
⟨*proof*⟩

lemma *assoc-naturality1*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}[\mu, \nu, \tau] = (\mu \star \nu \star \tau) \cdot \mathrm{a}[dom\ \mu, dom\ \nu, dom\ \tau]$
⟨*proof*⟩

lemma *assoc-naturality2*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}[\mu, \nu, \tau] = \mathrm{a}[cod\ \mu, cod\ \nu, cod\ \tau] \cdot ((\mu \star \nu) \star \tau)$
⟨*proof*⟩

lemma *assoc-naturality*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}[cod\ \mu, cod\ \nu, cod\ \tau] \cdot ((\mu \star \nu) \star \tau) = (\mu \star \nu \star \tau) \cdot \mathrm{a}[dom\ \mu, dom\ \nu, dom\ \tau]$
⟨*proof*⟩

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* $\mathrm{a}[f, g, h]$ (*src h*) (*trg f*)
and «$\mathrm{a}[f, g, h] : (dom\ f \star dom\ g) \star dom\ h \Rightarrow cod\ f \star cod\ g \star cod\ h$»
⟨*proof*⟩

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* $\mathrm{a}[f, g, h]$
and *src* $\mathrm{a}[f, g, h]$ = *src h* **and** *trg* $\mathrm{a}[f, g, h]$ = *trg f*
and *dom* $\mathrm{a}[f, g, h] = (dom\ f \star dom\ g) \star dom\ h$
and *cod* $\mathrm{a}[f, g, h] = cod\ f \star cod\ g \star cod\ h$
⟨*proof*⟩

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* $\mathrm{a}[f, g, h]$
⟨*proof*⟩

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*
⟨*proof*⟩

abbreviation V
where $V \equiv V.comp$

interpretation *src*: *functor V V dom*
⟨*proof*⟩
interpretation *trg*: *functor V V cod*
⟨*proof*⟩

interpretation H: *horizontal-homs V dom cod*
⟨*proof*⟩

interpretation H: *functor H.VV.comp V* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ · *snd* $\mu\nu$›
⟨*proof*⟩

interpretation H: *horizontal-composition V C dom cod*
⟨*proof*⟩

abbreviation a
where a *f g h* ≡ *f* · *g* · *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* $\mu\nu\tau$))›
⟨*proof*⟩

interpretation *fully-faithful-functor V V H.R*
⟨*proof*⟩
interpretation *fully-faithful-functor V V H.L*
⟨*proof*⟩

abbreviation i
where i ≡ λx. x

proposition *induces-bicategory*:
shows *bicategory V C* a i *dom cod*
⟨*proof*⟩

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* $\mathcal{I}$
⟨*proof*⟩
interpretation *horizontal-homs C I.map I.map*
⟨*proof*⟩

lemma *CC-eq-VV*:
shows $CC.comp = VV.comp$
⟨*proof*⟩

lemma *CCC-eq-VVV*:
shows $CCC.comp = VVV.comp$
⟨*proof*⟩

interpretation H: *functor VV.comp C* ‹$\lambda\mu\nu$. *fst* $\mu\nu \otimes$ *snd* $\mu\nu$›
⟨*proof*⟩
interpretation H: *horizontal-composition C tensor I.map I.map*
⟨*proof*⟩

lemma *HoHV-eq-ToTC*:
shows $H.HoHV = T.ToTC$
⟨*proof*⟩

lemma *HoVH-eq-ToCT*:
shows $H.HoVH = T.ToCT$
⟨*proof*⟩

interpretation α: *natural-isomorphism VVV.comp C H.HoHV H.HoVH* α
⟨*proof*⟩

lemma R'*-eq-R*:
shows $H.R = R$
⟨*proof*⟩

lemma L'*-eq-L*:
shows $H.L = L$
⟨*proof*⟩

interpretation *R′*: *fully-faithful-functor C C H.R*
⟨*proof*⟩
interpretation *L′*: *fully-faithful-functor C C H.L*
⟨*proof*⟩

lemma *obj-char*:
shows *obj a* ⟷ *a* = ℐ
⟨*proof*⟩

proposition *induces-bicategory*:
shows *bicategory C tensor* (λμ ν τ. α (μ, ν, τ)) (λ-. ι) *I.map I.map*
⟨*proof*⟩

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*⟩

end

sublocale *prebicategory-with-homs* ⊆ *horizontal-composition V H src trg*
⟨*proof*⟩

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* →$_{WC}$ *trg f*» **and** «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
⟨*proof*⟩

lemma *runit-in-hom*:
assumes *ide f*
shows «r[*f*] : *src f* →$_{WC}$ *trg f*» **and** «r[*f*] : *f* ⋆ *src f* ⇒ *f*»
⟨*proof*⟩

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* →$_{WC}$ *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*⟩

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*⟩

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*]
⟨*proof*⟩

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*]
⟨*proof*⟩

lemma *iso-lunit*:
assumes *ide f*
shows *iso* l[*f*]
⟨*proof*⟩

lemma *iso-runit*:
assumes *ide f*
shows *iso* r[*f*]
⟨*proof*⟩

lemma *lunit-naturality*:
assumes *arr μ*
shows *μ* ⋅ l[*dom μ*] = l[*cod μ*] ⋅ (*trg μ* ⋆ *μ*)
⟨*proof*⟩

lemma *runit-naturality*:
assumes *arr μ*
shows *μ* ⋅ r[*dom μ*] = r[*cod μ*] ⋅ (*μ* ⋆ *src μ*)
⟨*proof*⟩

interpretation *L*: *endofunctor V L*
⟨*proof*⟩
interpretation 𝔩: *transformation-by-components V V L map lunit*
⟨*proof*⟩
interpretation 𝔩: *natural-isomorphism V V L map* 𝔩.*map*
⟨*proof*⟩

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

interpretation *L*: *equivalence-functor V V L*
⟨*proof*⟩

lemma *equivalence-functor-L*:
shows *equivalence-functor V V L*
⟨*proof*⟩

lemma *lunit-commutes-with-L*:
assumes *ide f*
shows l[*L f*] = *L* l[*f*]
⟨*proof*⟩

interpretation *R*: *endofunctor V R*
⟨*proof*⟩
interpretation 𝔯: *transformation-by-components V V R map runit*
⟨*proof*⟩
interpretation 𝔯: *natural-isomorphism V V R map* 𝔯.*map*
⟨*proof*⟩

lemma *natural-isomorphism-*𝔯:
shows *natural-isomorphism V V R map* 𝔯.*map*
⟨*proof*⟩

interpretation R: *equivalence-functor V V R*
⟨*proof*⟩

lemma *equivalence-functor-R*:
shows *equivalence-functor V V R*
⟨*proof*⟩

lemma *runit-commutes-with-R*:
assumes *ide f*
shows r[R f] = R r[f]
⟨*proof*⟩

definition α
where $\alpha\ \mu\ \nu\ \tau \equiv$ *if VVV.arr* (μ, ν, τ) *then*
($\mu \star \nu \star \tau$) · 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 α *f g h* = a[f, g, h]
⟨*proof*⟩

no-notation *in-hom* (‹«- : - → -»›)

lemma *natural-isomorphism-*α:
shows *natural-isomorphism VVV.comp V HoHV HoVH*
($\lambda\mu\nu\tau$. α (*fst* $\mu\nu\tau$) (*fst* (*snd* $\mu\nu\tau$)) (*snd* (*snd* $\mu\nu\tau$)))
⟨*proof*⟩

proposition *induces-bicategory*:
shows *bicategory V H* α i *src trg*
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation *prebicategory-with-units V H* a *some-unit*
⟨*proof*⟩

interpretation *prebicategory-with-homs-and-units V H* a *some-unit some-src some-trg* ⟨*proof*⟩

theorem *extends-to-bicategory*:
shows *bicategory V H α some-unit some-src some-trg*
⟨*proof*⟩

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* $\mu\nu\tau$))› ⟨*proof*⟩

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* μ ⋆ *dom* ν ⋆ *dom* τ ⇒ (*cod* μ ⋆ *cod* ν) ⋆ *cod* τ»
⟨*proof*⟩

lemma *assoc′-naturality1*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}^{-1}[\mu, \nu, \tau] = ((\mu \star \nu) \star \tau) \cdot \mathrm{a}^{-1}[dom\ \mu, dom\ \nu, dom\ \tau]$
⟨*proof*⟩

lemma *assoc′-naturality2*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}^{-1}[\mu, \nu, \tau] = \mathrm{a}^{-1}[cod\ \mu, cod\ \nu, cod\ \tau] \cdot (\mu \star \nu \star \tau)$

⟨*proof*⟩

lemma *assoc′-naturality*:
assumes *arr* μ **and** *arr* ν **and** *arr* τ **and** *src* μ = *trg* ν **and** *src* ν = *trg* τ
shows $\mathrm{a}^{-1}[cod\ \mu,\ cod\ \nu,\ cod\ \tau] \cdot (\mu \star \nu \star \tau) = ((\mu \star \nu) \star \tau) \cdot \mathrm{a}^{-1}[dom\ \mu,\ dom\ \nu,\ dom\ \tau]$
⟨*proof*⟩

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*»
⟨*proof*⟩

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*
⟨*proof*⟩

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* $\mathrm{a}[f,\ g,\ h]$
⟨*proof*⟩

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* $\mathrm{a}[f,\ g,\ h]$ $\mathrm{a}^{-1}[f,\ g,\ h]$
⟨*proof*⟩

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* $\mathrm{a}^{-1}[f,\ g,\ h]$
⟨*proof*⟩

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 $\mathrm{a}^{-1}[f,\ g,\ h] \cdot \mathrm{a}[f,\ g,\ h] = (f \star g) \star h$
⟨*proof*⟩

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*⟩

interpretation *weak-composition V H*
⟨*proof*⟩

lemma *seq-if-composable*:
assumes $\nu \star \mu \neq null$
shows $src\ \nu = trg\ \mu$
⟨*proof*⟩

lemma *obj-self-composable*:
assumes *obj a*
shows $a \star a \neq null$
and *isomorphic* $(a \star a)\ a$
⟨*proof*⟩

lemma *obj-is-weak-unit*:
assumes *obj a*
shows *weak-unit a*
⟨*proof*⟩

lemma *src-in-sources*:
assumes *arr* μ
shows $src\ \mu \in sources\ \mu$
⟨*proof*⟩

lemma *trg-in-targets*:
assumes *arr* μ
shows $trg\ \mu \in targets\ \mu$
⟨*proof*⟩

lemma *weak-unit-cancel-left*:
assumes *weak-unit a* **and** *ide f* **and** *ide g*
and $a \star f \cong a \star g$
shows $f \cong g$
⟨*proof*⟩

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*⟩

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** $a \in sources\ \mu$
shows $a \cong src\ \mu$
⟨*proof*⟩

lemma *target-iso-trg*:
assumes *arr* μ **and** $b \in$ *targets* μ
shows $b \cong$ *trg* μ
⟨*proof*⟩

lemma *is-weak-composition-with-homs*:
shows *weak-composition-with-homs V H src trg*
⟨*proof*⟩

interpretation *weak-composition-with-homs V H src trg*
⟨*proof*⟩

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* $\nu =$ *trg* μ
⟨*proof*⟩

lemma *is-associative-weak-composition*:
shows *associative-weak-composition V H* a
⟨*proof*⟩

interpretation *associative-weak-composition V H* a
⟨*proof*⟩

theorem *is-prebicategory*:
shows *prebicategory V H* a
⟨*proof*⟩

interpretation *prebicategory V H* a
⟨*proof*⟩

corollary *is-prebicategory-with-homs*:
shows *prebicategory-with-homs V H* a *src trg*
⟨*proof*⟩

interpretation *prebicategory-with-homs V H* a *src trg*
⟨*proof*⟩

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*⟩

interpretation *H*: *partial-composition H*

⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

Every identity for horizontal composition is a weak unit.

lemma *horizontal-identity-is-weak-unit*:
assumes *H.ide* μ
shows *weak-unit* μ
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation *H*: *partial-composition H*

⟨*proof*⟩

lemma *weak-unit-is-horizontal-identity*:
assumes *weak-unit a*
shows *H.ide a*
⟨*proof*⟩

lemma *sources-eq-domains*:
shows *sources* μ = *H.domains* μ
⟨*proof*⟩

lemma *targets-eq-codomains*:
shows *targets* μ = *H.codomains* μ
⟨*proof*⟩

lemma *arr-agreement*:
shows *arr* = *H.arr*
⟨*proof*⟩

interpretation *H*: *category H*
⟨*proof*⟩

proposition *is-category*:
shows *category H*
⟨*proof*⟩

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
⟨*proof*⟩

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

lemma i′-*extends*-i:
assumes *weak-unit a*
shows *iso* (i′ *a*) **and** «i′ *a* : *a* ⋆ *a* ⇒ *a*» **and** *obj a* ⟹ i′ *a* = i[*a*]
⟨*proof*⟩

proposition *extends-to-prebicategory-with-units*:
shows *prebicategory-with-units V H* a i′
⟨*proof*⟩

interpretation *PB*: *prebicategory-with-units V H* a i′
⟨*proof*⟩
interpretation *PB*: *prebicategory-with-homs V H* a *src trg*
⟨*proof*⟩
interpretation *PB*: *prebicategory-with-homs-and-units V H* a i′ *src trg* ⟨*proof*⟩

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

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*] ≡ *inv* l[*a*]

abbreviation *runit′* (‹r⁻¹[-]›)
where r⁻¹[*a*] ≡ *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*⟩

lemma *lunit-in-hom* [*intro*]:
assumes *ide f*
shows «l[*f*] : *src f* → *trg f*» **and** «l[*f*] : *trg f* ⋆ *f* ⇒ *f*»
⟨*proof*⟩

lemma *lunit-in-vhom* [*simp*]:
assumes *ide f* **and** *trg f* = *b*
shows «l[*f*] : *b* ⋆ *f* ⇒ *f*»
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *runit-in-hom* [*intro*]:
assumes *ide f*
shows «r[*f*] : *src f* → *trg f*» **and** «r[*f*] : *f* ⋆ *src f* ⇒ *f*»
⟨*proof*⟩

lemma *runit-in-vhom* [*simp*]:
assumes *ide f* **and** *src f* = *a*
shows «r[*f*] : *f* ⋆ *a* ⇒ *f*»
⟨*proof*⟩

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* ⋆ *src f* **and** *cod* r[*f*] = *f*
⟨*proof*⟩

lemma *lunit-eqI*:
assumes *ide f* **and** «*μ* : *trg f* ⋆ *f* ⇒ *f*»
and *trg f* ⋆ *μ* = (i[*trg f*] ⋆ *f*) · a^{-1}[*trg f*, *trg f*, *f*]
shows *μ* = l[*f*]
⟨*proof*⟩

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*]
⟨*proof*⟩

lemma *lunit-naturality*:
assumes *arr μ*
shows *μ* · l[*dom μ*] = l[*cod μ*] · (*trg μ* ⋆ *μ*)
⟨*proof*⟩

lemma *runit-naturality*:
assumes *arr μ*
shows *μ* · r[*dom μ*] = r[*cod μ*] · (*μ* ⋆ *src μ*)

⟨*proof*⟩

lemma *iso-lunit* [*simp*]:
assumes *ide f*
shows *iso* l[*f*]
⟨*proof*⟩

lemma *iso-runit* [*simp*]:
assumes *ide f*
shows *iso* r[*f*]
⟨*proof*⟩

lemma *iso-lunit'* [*simp*]:
assumes *ide f*
shows *iso* l^{-1}[*f*]
⟨*proof*⟩

lemma *iso-runit'* [*simp*]:
assumes *ide f*
shows *iso* r^{-1}[*f*]
⟨*proof*⟩

lemma *lunit'-in-hom* [*intro*]:
assumes *ide f*
shows «l^{-1}[*f*] : *src f* → *trg f*» **and** «l^{-1}[*f*] : *f* ⇒ *trg f* ⋆ *f*»
⟨*proof*⟩

lemma *lunit'-in-vhom* [*simp*]:
assumes *ide f* **and** *trg f* = *b*
shows «l^{-1}[*f*] : *f* ⇒ *b* ⋆ *f*»
⟨*proof*⟩

lemma *lunit'-simps* [*simp*]:
assumes *ide f*
shows *arr* l^{-1}[*f*] **and** *src* l^{-1}[*f*] = *src f* **and** *trg* l^{-1}[*f*] = *trg f*
and *dom* l^{-1}[*f*] = *f* **and** *cod* l^{-1}[*f*] = *trg f* ⋆ *f*
⟨*proof*⟩

lemma *runit'-in-hom* [*intro*]:
assumes *ide f*
shows «r^{-1}[*f*] : *src f* → *trg f*» **and** «r^{-1}[*f*] : *f* ⇒ *f* ⋆ *src f*»
⟨*proof*⟩

lemma *runit'-in-vhom* [*simp*]:
assumes *ide f* **and** *src f* = *a*
shows «r^{-1}[*f*] : *f* ⇒ *f* ⋆ *a*»
⟨*proof*⟩

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* $\mathrm{r}^{-1}[f] = f \star src\ f$
⟨*proof*⟩

interpretation *L*: *endofunctor V L* ⟨*proof*⟩
interpretation $\mathfrak{l}$: *transformation-by-components V V L map lunit*
⟨*proof*⟩
interpretation $\mathfrak{l}$: *natural-isomorphism V V L map* $\mathfrak{l}$.*map*
⟨*proof*⟩

lemma *natural-isomorphism-*$\mathfrak{l}$:
shows *natural-isomorphism V V L map* $\mathfrak{l}$.*map*
⟨*proof*⟩

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

lemma $\mathfrak{l}$-*ide-simp*:
assumes *ide f*
shows $\mathfrak{l}\ f = \mathrm{l}[f]$
⟨*proof*⟩

interpretation *L*: *equivalence-functor V V L*
⟨*proof*⟩

lemma *equivalence-functor-L*:
shows *equivalence-functor V V L*
⟨*proof*⟩

lemma *lunit-commutes-with-L*:
assumes *ide f*
shows $\mathrm{l}[L\ f] = L\ \mathrm{l}[f]$
⟨*proof*⟩

interpretation *R*: *endofunctor V R* ⟨*proof*⟩
interpretation $\mathfrak{r}$: *transformation-by-components V V R map runit*
⟨*proof*⟩
interpretation $\mathfrak{r}$: *natural-isomorphism V V R map* $\mathfrak{r}$.*map*
⟨*proof*⟩

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

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

lemma $\mathfrak{r}$-*ide-simp*:

assumes *ide f*
shows $\mathfrak{r}\ f = \mathrm{r}[f]$
⟨*proof*⟩

interpretation *R*: *equivalence-functor V V R*
⟨*proof*⟩

lemma *equivalence-functor-R*:
shows *equivalence-functor V V R*
⟨*proof*⟩

lemma *runit-commutes-with-R*:
assumes *ide f*
shows $\mathrm{r}[R\ f] = R\ \mathrm{r}[f]$
⟨*proof*⟩

lemma *lunit′-naturality*:
assumes *arr* μ
shows $(trg\ \mu \star \mu) \cdot \mathrm{l}^{-1}[dom\ \mu] = \mathrm{l}^{-1}[cod\ \mu] \cdot \mu$
⟨*proof*⟩

lemma *runit′-naturality*:
assumes *arr* μ
shows $(\mu \star src\ \mu) \cdot \mathrm{r}^{-1}[dom\ \mu] = \mathrm{r}^{-1}[cod\ \mu] \cdot \mu$
⟨*proof*⟩

lemma *isomorphic-unit-right*:
assumes *ide f*
shows $f \star src\ f \cong f$
⟨*proof*⟩

lemma *isomorphic-unit-left*:
assumes *ide f*
shows $trg\ f \star f \cong f$
⟨*proof*⟩

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* i[*a*] **and** *src* i[*a*] = *a* **and** *trg* i[*a*] = *a*
and *dom* i[*a*] = *a* ⋆ *a* **and** *cod* i[*a*] = *a*
⟨*proof*⟩

lemma *triangle*:
assumes *ide f* **and** *ide g* **and** *src g* = *trg f*
shows (*g* ⋆ l[*f*]) · a[*g*, *src g*, *f*] = r[*g*] ⋆ *f*
⟨*proof*⟩

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* ⋆ l[*g* ⋆ *h*]) · (*f* ⋆ a[*trg g*, *g*, *h*]) = *f* ⋆ l[*g*] ⋆ *h*
⟨*proof*⟩

lemma *lunit-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows l[*f* ⋆ *g*] · a[*trg f*, *f*, *g*] = l[*f*] ⋆ *g*
and a^{-1}[*trg f*, *f*, *g*] · l^{-1}[*f* ⋆ *g*] = l^{-1}[*f*] ⋆ *g*
and l[*f* ⋆ *g*] = (l[*f*] ⋆ *g*) · a^{-1}[*trg f*, *f*, *g*]
and l^{-1}[*f* ⋆ *g*] = a[*trg f*, *f*, *g*] · (l^{-1}[*f*] ⋆ *g*)
⟨*proof*⟩

lemma *runit-hcomp-gen*:
assumes *ide f* **and** *ide g* **and** *ide h*
and *src f* = *trg g* **and** *src g* = *trg h*
shows r[*f* ⋆ *g*] ⋆ *h* = ((*f* ⋆ r[*g*]) ⋆ *h*) · (a[*f*, *g*, *src g*] ⋆ *h*)
⟨*proof*⟩

lemma *runit-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows r[*f* ⋆ *g*] = (*f* ⋆ r[*g*]) · a[*f*, *g*, *src g*]
and r^{-1}[*f* ⋆ *g*] = a^{-1}[*f*, *g*, *src g*] · (*f* ⋆ r^{-1}[*g*])
and r[*f* ⋆ *g*] · a^{-1}[*f*, *g*, *src g*] = *f* ⋆ r[*g*]
and a[*f*, *g*, *src g*] · r^{-1}[*f* ⋆ *g*] = *f* ⋆ r^{-1}[*g*]
⟨*proof*⟩

lemma *unitor-coincidence*:
assumes *obj a*
shows l[*a*] = i[*a*] **and** r[*a*] = i[*a*]
⟨*proof*⟩

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*⟩

lemma *hcomp-assoc-isomorphic*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*
shows $(f \star g) \star h \cong f \star g \star h$
⟨*proof*⟩

lemma *hcomp-arr-obj*:
assumes *arr* μ **and** *obj a* **and** *src* μ = *a*
shows $\mu \star a = \mathrm{r}^{-1}[cod\ \mu] \cdot \mu \cdot \mathrm{r}[dom\ \mu]$
and $\mathrm{r}[cod\ \mu] \cdot (\mu \star a) \cdot \mathrm{r}^{-1}[dom\ \mu] = \mu$
⟨*proof*⟩

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*⟩

lemma *hcomp-reassoc*:
assumes *arr* τ **and** *arr* μ **and** *arr* ν
and *src* τ = *trg* μ **and** *src* μ = *trg* ν
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*⟩

lemma *triangle′*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows $(f \star \mathrm{l}[g]) = (\mathrm{r}[f] \star g) \cdot \mathrm{a}^{-1}[f,\ src\ f,\ g]$
⟨*proof*⟩

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*⟩

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*⟩

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* ∈ *Hom A A* for each object *A*, "horizontal composition" functors *Comp C B A* : *Hom B C* × *Hom A B* → *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* ⇒ *'o* ⇒ *'a comp*
and *Id* :: *'o* ⇒ *'a*
and *Comp* :: *'o* ⇒ *'o* ⇒ *'o* ⇒ *'a* ⇒ *'a* ⇒ *'a*
and *Unit* :: *'o* ⇒ *'a*
and *Assoc* :: *'o* ⇒ *'o* ⇒ *'o* ⇒ *'o* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a*
assumes *category-Hom*: ⟦ *A* ∈ *Obj*; *B* ∈ *Obj* ⟧ ⟹ *category* (*Hom A B*)
and *binary-functor-Comp*:
⟦ *A* ∈ *Obj*; *B* ∈ *Obj*; *C* ∈ *Obj* ⟧
⟹ *binary-functor* (*Hom B C*) (*Hom A B*) (*Hom A C*) (λ(*f*, *g*). *Comp C B A f g*)
and *ide-Id*: *A* ∈ *Obj* ⟹ *partial-composition.ide* (*Hom A A*) (*Id A*)
and *Unit-in-hom*:
A ∈ *Obj* ⟹
partial-composition.in-hom (*Hom A A*) (*Unit A*) (*Comp A A A* (*Id A*) (*Id A*)) (*Id A*)
and *iso-Unit*: *A* ∈ *Obj* ⟹ *category.iso* (*Hom A A*) (*Unit A*)
and *natural-isomorphism-Assoc*:
⟦ *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)
and left-unit-Id:
    ⋀A B. ⟦ 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))
and right-unit-Id:
    ⋀A B. ⟦ 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))
and pentagon:
    ⋀A B C D E 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)
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-composition.arr (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)

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-composition.seq (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ) (*Map* ν)
then MkCell (*Src* μ) (*Trg* μ) (*Hom* (*Src* μ) (*Trg* μ) (*Map* μ) (*Map* ν))
else Null

interpretation *partial-composition vcomp*
⟨*proof*⟩

lemma *null-char*:
shows *null* = *Null*
⟨*proof*⟩

lemma *MkCell-Map*:
assumes $\mu \neq null$
shows μ = *MkCell* (*Src* μ) (*Trg* μ) (*Map* μ)
⟨*proof*⟩

lemma *ide-char''*:
shows *ide* $\mu \longleftrightarrow$ *Cell* $\mu \wedge$ *partial-composition.ide* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
⟨*proof*⟩

lemma *MkCell-in-domains*:
assumes *Cell* μ
shows *MkCell* (*Src* μ) (*Trg* μ) (*partial-composition.dom* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ))
$\in$ *domains* μ
⟨*proof*⟩

lemma *MkCell-in-codomains*:
assumes *Cell* μ
shows *MkCell* (*Src* μ) (*Trg* μ) (*partial-composition.cod* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ))
$\in$ *codomains* μ
⟨*proof*⟩

lemma *has-domain-char*:
shows *domains* $\mu \neq \{\} \longleftrightarrow$ *Cell* μ
⟨*proof*⟩

lemma *has-codomain-char*:
shows *codomains* $\mu \neq \{\} \longleftrightarrow$ *Cell* μ
⟨*proof*⟩

lemma *arr-char*:
shows *arr* $\mu \longleftrightarrow$ *Cell* μ
⟨*proof*⟩

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

lemma *seq-char*:
shows *seq* $\mu\ \nu \longleftrightarrow$ *Cell* $\mu \wedge$ *Cell* $\nu \wedge$ *Src* μ = *Src* $\nu \wedge$ *Trg* μ = *Trg* $\nu \wedge$
partial-composition.seq (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ) (*Map* ν)
⟨*proof*⟩

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)
⟨*proof*⟩

interpretation *category vcomp*
⟨*proof*⟩

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'*
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *Src-vcomp* [*simp*]:
assumes *seq* $\mu\ \nu$
shows *Src* (*vcomp* $\mu\ \nu$) = *Src* μ
⟨*proof*⟩

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

assumes *seq* μ ν
shows *Trg* (*vcomp* μ ν) = *Trg* μ
⟨*proof*⟩

lemma *Map-vcomp* [*simp*]:
assumes *seq* μ ν
shows *Map* (*vcomp* μ ν) = *Hom* (*Src* μ) (*Trg* μ) (*Map* μ) (*Map* ν)
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *iso-char*:
shows *iso* $\mu \longleftrightarrow$ *arr* $\mu \wedge$ *category.iso* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
⟨*proof*⟩

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

definition *src*
where *src* $\mu \equiv$ *if arr* μ *then MkObj* (*Src* μ) *else null*

definition *trg*
where *trg* $\mu \equiv$ *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
⟨*proof*⟩

lemma *trg-MkCell* [*simp*]:
assumes *arr* (*MkCell* A B f)
shows *trg* (*MkCell* A B f) = *MkObj* B
⟨*proof*⟩

lemma *src-dom*:
assumes *arr* μ
shows *src* (*dom* μ) = *src* μ

⟨*proof*⟩

lemma *src-cod*:
assumes *arr* μ
shows *src* (*cod* μ) = *src* μ
⟨*proof*⟩

lemma *trg-dom*:
assumes *arr* μ
shows *trg* (*dom* μ) = *trg* μ
⟨*proof*⟩

lemma *trg-cod*:
assumes *arr* μ
shows *trg* (*cod* μ) = *trg* μ
⟨*proof*⟩

lemma *Src-src* [*simp*]:
assumes *arr* μ
shows *Src* (*src* μ) = *Src* μ
⟨*proof*⟩

lemma *Trg-src* [*simp*]:
assumes *arr* μ
shows *Trg* (*src* μ) = *Src* μ
⟨*proof*⟩

lemma *Map-src* [*simp*]:
assumes *arr* μ
shows *Map* (*src* μ) = *Id* (*Src* μ)
⟨*proof*⟩

lemma *Src-trg* [*simp*]:
assumes *arr* μ
shows *Src* (*trg* μ) = *Trg* μ
⟨*proof*⟩

lemma *Trg-trg* [*simp*]:
assumes *arr* μ
shows *Trg* (*trg* μ) = *Trg* μ
⟨*proof*⟩

lemma *Map-trg* [*simp*]:
assumes *arr* μ
shows *Map* (*trg* μ) = *Id* (*Trg* μ)
⟨*proof*⟩

lemma *Src-dom* [*simp*]:
assumes *arr* μ

shows *Src* (*dom* μ) = *Src* μ
⟨*proof*⟩

lemma *Src-cod* [*simp*]:
assumes *arr* μ
shows *Src* (*cod* μ) = *Src* μ
⟨*proof*⟩

lemma *Trg-dom* [*simp*]:
assumes *arr* μ
shows *Trg* (*dom* μ) = *Trg* μ
⟨*proof*⟩

lemma *Trg-cod* [*simp*]:
assumes *arr* μ
shows *Trg* (*cod* μ) = *Trg* μ
⟨*proof*⟩

lemma *Map-dom* [*simp*]:
assumes *arr* μ
shows *Map* (*dom* μ) = *partial-composition.dom* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
⟨*proof*⟩

lemma *Map-cod* [*simp*]:
assumes *arr* μ
shows *Map* (*cod* μ) = *partial-composition.cod* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
⟨*proof*⟩

lemma *ide-MkObj*:
assumes $A \in Obj$
shows *ide* (*MkObj* A)
⟨*proof*⟩

interpretation *src*: *functor vcomp vcomp src*
⟨*proof*⟩

interpretation *trg*: *functor vcomp vcomp trg*
⟨*proof*⟩

interpretation H: *horizontal-homs vcomp src trg*
⟨*proof*⟩

lemma *obj-MkObj*:
assumes $A \in Obj$
shows $H.obj$ (*MkObj* A)
⟨*proof*⟩

lemma *MkCell-in-hom* [*intro*]:
assumes $A \in Obj$ **and** $B \in Obj$ **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*)»
⟨*proof*⟩

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*⟩

lemma *src-hcomp*:
assumes *arr μ* **and** *arr ν* **and** *src μ* = *trg ν*
shows *src* (*hcomp μ ν*) = *src ν*
⟨*proof*⟩

lemma *trg-hcomp*:
assumes *arr μ* **and** *arr ν* **and** *src μ* = *trg ν*
shows *trg* (*hcomp μ ν*) = *trg μ*
⟨*proof*⟩

lemma *Src-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src μ* = *trg ν*
shows *Src* (*hcomp μ ν*) = *Src ν*
⟨*proof*⟩

lemma *Trg-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src μ* = *trg ν*
shows *Trg* (*hcomp μ ν*) = *Trg μ*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

interpretation *H*: *functor H.VV.comp vcomp* ‹λμν. *hcomp* (*fst* μν) (*snd* μν)›
⟨*proof*⟩

interpretation *H*: *horizontal-composition vcomp hcomp src trg*
⟨*proof*⟩

lemma *Map-obj*:
assumes *H.obj a*
shows *Map a* = *Id* (*Src a*) **and** *Map a* = *Id* (*Trg a*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

interpretation α_0: *transformation-by-components H.VVV.comp vcomp H.HoHV H.HoVH*
‹λ(*f*, *g*, *h*). *assoc f g h*›
⟨*proof*⟩

definition a (‹a[-,-,-]›)
where a *f g h* == α_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*))
⟨*proof*⟩

interpretation α: *natural-isomorphism H.VVV.comp vcomp H.HoHV H.HoVH*
‹λ*fgh*. a (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*))›
⟨*proof*⟩

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

interpretation *endofunctor vcomp H.L*
⟨*proof*⟩
interpretation *endofunctor vcomp H.R*
⟨*proof*⟩

interpretation *R*: *fully-faithful-functor vcomp vcomp H.R*
⟨*proof*⟩

interpretation *L*: *fully-faithful-functor vcomp vcomp H.L*
⟨*proof*⟩

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*)
⟨*proof*⟩

The main result: the construction produces a bicategory.

proposition *induces-bicategory*:
shows *bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

sublocale *bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

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* $\mu \in$ *Obj*
⟨*proof*⟩

lemma *Trg-in-Obj*:
assumes *arr* μ
shows *Trg* $\mu \in$ *Obj*
⟨*proof*⟩

lemma *arr-Map*:
assumes *arr* μ
shows *partial-composition.arr* (*Hom* (*Src* μ) (*Trg* μ)) (*Map* μ)
⟨*proof*⟩

lemma *obj-MkObj-Src*:
assumes *arr* μ
shows *obj* (*MkObj* (*Src* μ))
⟨*proof*⟩

lemma *obj-MkObj-Trg*:
assumes *arr* μ
shows *obj* (*MkObj* (*Trg* μ))
⟨*proof*⟩

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*)

⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *obj-char*:
shows *obj a* ⟷ *Src a* ∈ *Obj* ∧ *a* = *MkCell* (*Src a*) (*Src a*) (*Id* (*Src a*))
⟨*proof*⟩

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))
⟨*proof*⟩

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* (λ*f*. *f* ∈ *hhom a b*)) (*Hom* (*Src a*) (*Src b*))
⟨*proof*⟩

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 "bicategorical 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 $t{\downarrow}$ 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* :: $'a \Rightarrow 'a$
  and *trg* :: $'a \Rightarrow 'a$
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* =
  $Prim_0$ $'t$ (‹⟨-⟩$_0$›)
| *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* $\Rightarrow$ $'a$ *term*
where *Src* $\langle\mu\rangle_0$ = $\langle\mu\rangle_0$
  | *Src* $\langle\mu\rangle$ = $\langle$*src* $\mu\rangle_0$
  | *Src* $(t \star u)$ = *Src* u
  | *Src* $(t \cdot u)$ = *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
  | Src a⁻¹[t, u, v] = Src v

primrec Trg :: 'a term ⇒ 'a term
where Trg ⟨μ⟩₀ = ⟨μ⟩₀
  | Trg ⟨μ⟩ = ⟨trg μ⟩₀
  | Trg (t ⋆ u) = Trg t
  | Trg (t · u) = Trg t
  | Trg l[t] = Trg t
  | Trg l⁻¹[t] = Trg t
  | Trg r[t] = Trg t
  | Trg r⁻¹[t] = Trg t
  | Trg a[t, u, v] = Trg t
  | Trg a⁻¹[t, u, v] = Trg t

primrec Dom :: 'a term ⇒ 'a term
where Dom ⟨μ⟩₀ = ⟨μ⟩₀
  | Dom ⟨μ⟩ = ⟨dom μ⟩
  | Dom (t ⋆ u) = Dom t ⋆ Dom u
  | Dom (t · u) = Dom u
  | Dom l[t] = Trg t ⋆ Dom t
  | Dom l⁻¹[t] = Dom t
  | Dom r[t] = Dom t ⋆ Src t
  | Dom r⁻¹[t] = Dom t
  | Dom a[t, u, v] = (Dom t ⋆ Dom u) ⋆ Dom v
  | Dom a⁻¹[t, u, v] = Dom t ⋆ (Dom u ⋆ Dom v)

primrec Cod :: 'a term ⇒ 'a term
where Cod ⟨μ⟩₀ = ⟨μ⟩₀
  | Cod ⟨μ⟩ = ⟨cod μ⟩
  | Cod (t ⋆ u) = Cod t ⋆ Cod u
  | Cod (t · u) = Cod t
  | Cod l[t] = Cod t
  | Cod l⁻¹[t] = Trg t ⋆ Cod t
  | Cod r[t] = Cod t
  | Cod r⁻¹[t] = Cod t ⋆ Src t
  | Cod a[t, u, v] = Cod t ⋆ (Cod u ⋆ Cod v)
  | Cod a⁻¹[t, u, v] = (Cod t ⋆ Cod u) ⋆ 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 ⇒ bool
where Arr ⟨μ⟩₀ = obj μ
  | Arr ⟨μ⟩ = arr μ
  | Arr (t ⋆ u) = (Arr t ∧ Arr u ∧ Src t = Trg u)
  | Arr (t · u) = (Arr t ∧ Arr u ∧ Dom t = Cod u)
  | Arr l[t] = Arr t
  | Arr l⁻¹[t] = Arr t
```

| *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* ⇒ *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* ⇒ *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* ⇒ *'a term* ⇒ *bool*
where *HSeq t u* ≡ *Arr t* ∧ *Arr u* ∧ *Src t* = *Trg u*

abbreviation *VSeq* :: *'a term* ⇒ *'a term* ⇒ *bool*
where *VSeq t u* ≡ *Arr t* ∧ *Arr u* ∧ *Dom t* = *Cod u*

abbreviation *HPar* :: *'a term* => *'a term* ⇒ *bool*
where *HPar t u* ≡ *Arr t* ∧ *Arr u* ∧ *Src t* = *Src u* ∧ *Trg t* = *Trg u*

abbreviation *VPar* :: *'a term* => *'a term* ⇒ *bool*
where *VPar t u* ≡ *Arr t* ∧ *Arr u* ∧ *Dom t* = *Dom u* ∧ *Cod t* = *Cod u*

abbreviation *HHom* :: *'a term* ⇒ *'a term* ⇒ *'a term set*
where *HHom a b* ≡ { *t*. *Arr t* ∧ *Src t* = *a* ∧ *Trg t* = *b* }

abbreviation *VHom* :: *'a term* ⇒ *'a term* ⇒ *'a term set*
where *VHom f g* ≡ { *t*. *Arr t* ∧ *Dom t* = *f* ∧ *Cod t* = *g* }

lemma *is-Prim0-Src*:
shows *is-Prim*$_0$ (*Src t*)

⟨*proof*⟩

lemma *is-Prim0-Trg*:
shows *is-Prim*$_0$ (*Trg t*)
⟨*proof*⟩

lemma *Src-Src* [*simp*]:
shows *Arr t* ⟹ *Src* (*Src t*) = *Src t*
⟨*proof*⟩

lemma *Trg-Trg* [*simp*]:
shows *Arr t* ⟹ *Trg* (*Trg t*) = *Trg t*
⟨*proof*⟩

lemma *Src-Trg* [*simp*]:
shows *Arr t* ⟹ *Src* (*Trg t*) = *Trg t*
⟨*proof*⟩

lemma *Trg-Src* [*simp*]:
shows *Arr t* ⟹ *Trg* (*Src t*) = *Src t*
⟨*proof*⟩

lemma *Dom-Src* [*simp*]:
shows *Arr t* ⟹ *Dom* (*Src t*) = *Src t*
⟨*proof*⟩

lemma *Dom-Trg* [*simp*]:
shows *Arr t* ⟹ *Dom* (*Trg t*) = *Trg t*
⟨*proof*⟩

lemma *Cod-Src* [*simp*]:
shows *Arr t* ⟹ *Cod* (*Src t*) = *Src t*
⟨*proof*⟩

lemma *Cod-Trg* [*simp*]:
shows *Arr t* ⟹ *Cod* (*Trg t*) = *Trg t*
⟨*proof*⟩

lemma *Src-Dom-Cod*:
shows *Arr t* ⟹ *Src* (*Dom t*) = *Src t* ∧ *Src* (*Cod t*) = *Src t*
⟨*proof*⟩

lemma *Src-Dom* [*simp*]:
shows *Arr t* ⟹ *Src* (*Dom t*) = *Src t*
⟨*proof*⟩

lemma *Src-Cod* [*simp*]:
shows *Arr t* ⟹ *Src* (*Cod t*) = *Src t*
⟨*proof*⟩

lemma *Trg-Dom-Cod*:
shows $Arr\ t \Longrightarrow Trg\ (Dom\ t) = Trg\ t \wedge Trg\ (Cod\ t) = Trg\ t$
⟨*proof*⟩

lemma *Trg-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Trg\ (Dom\ t) = Trg\ t$
⟨*proof*⟩

lemma *Trg-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Trg\ (Cod\ t) = Trg\ t$
⟨*proof*⟩

lemma *VSeq-implies-HPar*:
shows $VSeq\ t\ u \Longrightarrow HPar\ t\ u$
⟨*proof*⟩

lemma *Dom-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Dom\ (Dom\ t) = Dom\ t$
⟨*proof*⟩

lemma *Cod-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Cod\ (Cod\ t) = Cod\ t$
⟨*proof*⟩

lemma *Dom-Cod* [*simp*]:
shows $Arr\ t \Longrightarrow Dom\ (Cod\ t) = Cod\ t$
⟨*proof*⟩

lemma *Cod-Dom* [*simp*]:
shows $Arr\ t \Longrightarrow Cod\ (Dom\ t) = Dom\ t$
⟨*proof*⟩

lemma *Obj-implies-Ide* :
shows $Obj\ t \Longrightarrow Ide\ t$
⟨*proof*⟩

lemma *Ide-implies-Arr* [*simp*]:
shows $Ide\ t \Longrightarrow Arr\ t$
⟨*proof*⟩

lemma *Dom-Ide*:
shows $Ide\ t \Longrightarrow Dom\ t = t$
⟨*proof*⟩

lemma *Cod-Ide*:
shows $Ide\ t \Longrightarrow Cod\ t = t$
⟨*proof*⟩

lemma *Obj-Src* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Obj* (*Src t*)
⟨*proof*⟩

lemma *Obj-Trg* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Obj* (*Trg t*)
⟨*proof*⟩

lemma *Ide-Dom* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Ide* (*Dom t*)
⟨*proof*⟩

lemma *Ide-Cod* [*simp*]:
shows *Arr t* $\Longrightarrow$ *Ide* (*Cod t*)
⟨*proof*⟩

lemma *Arr-in-Hom*:
assumes *Arr t*
shows *t* ∈ *HHom* (*Src t*) (*Trg t*) **and** *t* ∈ *VHom* (*Dom t*) (*Cod t*)
⟨*proof*⟩

lemma *Ide-in-Hom*:
assumes *Ide t*
shows *t* ∈ *HHom* (*Src t*) (*Trg t*) **and** *t* ∈ *VHom t t*
⟨*proof*⟩

lemma *Obj-in-Hom*:
assumes *Obj t*
shows *t* ∈ *HHom t t* **and** *t* ∈ *VHom t t*
⟨*proof*⟩

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* ⟨*μ*⟩$_0$ = *obj μ*
| *Can* ⟨*μ*⟩ = *ide μ*
| *Can* (*t* ⋆ *u*) = (*Can t* ∧ *Can u* ∧ *Src t* = *Trg u*)
| *Can* (*t* ⋅ *u*) = (*Can t* ∧ *Can u* ∧ *Dom t* = *Cod u*)
| *Can* **l**[*t*] = *Can t*
| *Can* **l**$^{-1}$[*t*] = *Can t*
| *Can* **r**[*t*] = *Can t*
| *Can* **r**$^{-1}$[*t*] = *Can t*
| *Can* **a**[*t*, *u*, *v*] = (*Can t* ∧ *Can u* ∧ *Can v* ∧ *Src t* = *Trg u* ∧ *Src u* = *Trg v*)
| *Can* **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* $\Longrightarrow$ *Can t*
⟨*proof*⟩

lemma *Can-implies-Arr*:
shows *Can t* $\Longrightarrow$ *Arr t*
⟨*proof*⟩

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* ⋆ *u*) = (*Inv t* ⋆ *Inv u*)
| *Inv* (*t* ⋅ *u*) = (*Inv u* ⋅ *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* $\Longrightarrow$ *Src* (*Inv t*) = *Src t*
⟨*proof*⟩

lemma *Trg-Inv* [*simp*]:
shows *Can t* $\Longrightarrow$ *Trg* (*Inv t*) = *Trg t*
⟨*proof*⟩

lemma *Dom-Inv* [*simp*]:
shows *Can t* $\Longrightarrow$ *Dom* (*Inv t*) = *Cod t*
⟨*proof*⟩

lemma *Cod-Inv* [*simp*]:
shows *Can t* $\Longrightarrow$ *Cod* (*Inv t*) = *Dom t*
⟨*proof*⟩

lemma *Inv-preserves-Ide*:
shows *Ide t* $\Longrightarrow$ *Ide* (*Inv t*)
⟨*proof*⟩

lemma *Can-Inv* [*simp*]:
shows *Can t* $\Longrightarrow$ *Can* (*Inv t*)
⟨*proof*⟩

lemma *Inv-in-Hom* [*intro*]:
assumes *Can t*
shows *Inv t* ∈ *HHom* (*Src t*) (*Trg t*) **and** *Inv t* ∈ *VHom* (*Cod t*) (*Dom t*)
⟨*proof*⟩

lemma *Inv-Ide* [*simp*]:
assumes *Ide a*
shows *Inv a* = *a*

⟨*proof*⟩

lemma *Inv-Inv* [*simp*]:
assumes *Can t*
shows *Inv* (*Inv t*) = *t*
⟨*proof*⟩

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 (·), where the empty list is represented by a formal object and ⋆ is used as the list constructor.

fun *Nml* :: *'a term* ⇒ *bool*
where *Nml* ⟨*μ*⟩$_0$ = *obj μ*
| *Nml* ⟨*μ*⟩ = *arr μ*
| *Nml* (⟨*ν*⟩ ⋆ *u*) = (*arr ν* ∧ *Nml u* ∧ ¬ *is-Prim*$_0$ *u* ∧ ⟨*src ν*⟩$_0$ = *Trg u*)
| *Nml* - = *False*

lemma *Nml-HcompD*:
assumes *Nml* (*t* ⋆ *u*)
shows ⟨*un-Prim t*⟩ = *t* **and** *arr* (*un-Prim t*) **and** *Nml t* **and** *Nml u*
and ¬ *is-Prim*$_0$ *u* **and** ⟨*src* (*un-Prim t*)⟩$_0$ = *Trg u* **and** *Src t* = *Trg u*
⟨*proof*⟩

lemma *Nml-implies-Arr*:
shows *Nml t* ⟹ *Arr t*
⟨*proof*⟩

lemma *Nml-Src* [*simp*]:
shows *Nml t* ⟹ *Nml* (*Src t*)
⟨*proof*⟩

lemma *Nml-Trg* [*simp*]:
shows *Nml t* ⟹ *Nml* (*Trg t*)
⟨*proof*⟩

lemma *Nml-Dom* [*simp*]:
shows *Nml t* ⟹ *Nml* (*Dom t*)
⟨*proof*⟩

lemma *Nml-Cod* [*simp*]:
shows *Nml t* ⟹ *Nml* (*Cod t*)
⟨*proof*⟩

lemma *Nml-Inv* [*simp*]:
assumes *Can t* **and** *Nml t*
shows *Nml* (*Inv t*)
⟨*proof*⟩

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** ‹$\lfloor\star\rfloor$› *53*)
where $\langle\nu\rangle_0\ \lfloor\star\rfloor\ u = u$
| $t\ \lfloor\star\rfloor\ \langle\mu\rangle_0 = t$
| $\langle\nu\rangle\ \lfloor\star\rfloor\ u = \langle\nu\rangle\ \star\ u$
| $(t \star u)\ \lfloor\star\rfloor\ v = t\ \lfloor\star\rfloor\ (u\ \lfloor\star\rfloor\ v)$
| $t\ \lfloor\star\rfloor\ u =$ *undefined*

lemma *HcompNml-Prim* [*simp*]:
assumes ¬ *is-Prim*$_0$ t
shows $\langle\nu\rangle\ \lfloor\star\rfloor\ t = \langle\nu\rangle\ \star\ t$
⟨*proof*⟩

lemma *HcompNml-term-Prim*$_0$ [*simp*]:
assumes *Src* t = *Trg* $\langle\mu\rangle_0$
shows $t\ \lfloor\star\rfloor\ \langle\mu\rangle_0 = t$
⟨*proof*⟩

lemma *HcompNml-Nml*:
assumes *Nml* $(t \star u)$
shows $t\ \lfloor\star\rfloor\ u = t \star u$
⟨*proof*⟩

lemma *Nml-HcompNml*:
assumes *Nml* t **and** *Nml* u **and** *Src* t = *Trg* u
shows *Nml* $(t\ \lfloor\star\rfloor\ u)$
and *Dom* $(t\ \lfloor\star\rfloor\ u)$ = *Dom* $t\ \lfloor\star\rfloor$ *Dom* u
and *Cod* $(t\ \lfloor\star\rfloor\ u)$ = *Cod* $t\ \lfloor\star\rfloor$ *Cod* u
and *Src* $(t\ \lfloor\star\rfloor\ u)$ = *Src* u
and *Trg* $(t\ \lfloor\star\rfloor\ u)$ = *Trg* t
⟨*proof*⟩

lemma *HcompNml-in-Hom* [*intro*]:
assumes *Nml* t **and** *Nml* u **and** *Src* t = *Trg* u
shows $t\ \lfloor\star\rfloor\ u \in$ *HHom* (*Src* u) (*Trg* t)
and $t\ \lfloor\star\rfloor\ u \in$ *VHom* (*Dom* $t\ \lfloor\star\rfloor$ *Dom* u) (*Cod* $t\ \lfloor\star\rfloor$ *Cod* u)
⟨*proof*⟩

lemma *Src-HcompNml*:
assumes *Nml* t **and** *Nml* u **and** *Src* t = *Trg* u
shows *Src* $(t\ \lfloor\star\rfloor\ u)$ = *Src* u
⟨*proof*⟩

lemma *Trg-HcompNml*:
assumes *Nml* t **and** *Nml* u **and** *Src* t = *Trg* u
shows *Trg* $(t\ \lfloor\star\rfloor\ u)$ = *Trg* t
⟨*proof*⟩

lemma *Dom-HcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows *Dom* (*t* ⌊⋆⌋ *u*) = *Dom t* ⌊⋆⌋ *Dom u*
⟨*proof*⟩

lemma *Cod-HcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows *Cod* (*t* ⌊⋆⌋ *u*) = *Cod t* ⌊⋆⌋ *Cod u*
⟨*proof*⟩

lemma *is-Hcomp-HcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
and ¬ *is-Prim*$_0$ *t* **and** ¬ *is-Prim*$_0$ *u*
shows *is-Hcomp* (*t* ⌊⋆⌋ *u*)
⟨*proof*⟩

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* ⟨*μ*⟩$_0$ = *0*
| *dim* ⟨*μ*⟩ = *1*
| *dim* (*t* ⋆ *u*) = (*dim t* + *dim u*)
| *dim* (*t* ⋅ *u*) = *dim t*
| *dim* **l**[*t*] = *dim t*
| *dim* **l**$^{-1}$[*t*] = *dim t*
| *dim* **r**[*t*] = *dim t*
| *dim* **r**$^{-1}$[*t*] = *dim t*
| *dim* **a**[*t*, *u*, *v*] = *dim t* + *dim u* + *dim v*
| *dim* **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*⟩

lemma *HcompNml-Trg-Nml*:
assumes *Nml t*
shows *Trg t* ⌊⋆⌋ *t* = *t*
⟨*proof*⟩

lemma *HcompNml-Nml-Src*:
assumes *Nml t*
shows *t* ⌊⋆⌋ *Src t* = *t*
⟨*proof*⟩

lemma *HcompNml-Obj-Nml*:
assumes *Obj t* **and** *Nml u* **and** *Src t* = *Trg u*

shows $t \lfloor\star\rfloor\ u = u$
⟨*proof*⟩

lemma *HcompNml-Nml-Obj*:
assumes *Nml t* **and** *Obj u* **and** *Src t* = *Trg u*
shows $t \lfloor\star\rfloor\ u = t$
⟨*proof*⟩

lemma *Ide-HcompNml*:
assumes *Ide t* **and** *Ide u* **and** *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows *Ide* $(t \lfloor\star\rfloor\ u)$
⟨*proof*⟩

lemma *Can-HcompNml*:
assumes *Can t* **and** *Can u* **and** *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows *Can* $(t \lfloor\star\rfloor\ u)$
⟨*proof*⟩

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*⟩

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** ‹$\lfloor\cdot\rfloor$› *55*)
where $\langle\nu\rangle_0\ \lfloor\cdot\rfloor\ u = u$
| $\langle\nu\rangle\ \lfloor\cdot\rfloor\ \langle\mu\rangle = \langle\nu \cdot \mu\rangle$
| $(u \star v)\ \lfloor\cdot\rfloor\ (w \star x) = (u \lfloor\cdot\rfloor\ w \star v \lfloor\cdot\rfloor\ x)$
| $t \lfloor\cdot\rfloor\ \langle\mu\rangle_0 = t$
| $t \lfloor\cdot\rfloor$ - = *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* $(t \lfloor\cdot\rfloor\ u)$
and *Dom* $(t \lfloor\cdot\rfloor\ u)$ = *Dom u*
and *Cod* $(t \lfloor\cdot\rfloor\ u)$ = *Cod t*
⟨*proof*⟩

lemma *VcompNml-in-Hom* [*intro*]:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows $t \lfloor\cdot\rfloor\ u \in$ *HHom* (*Src u*) (*Trg u*) **and** $t \lfloor\cdot\rfloor\ u \in$ *VHom* (*Dom u*) (*Cod t*)
⟨*proof*⟩

lemma *Src-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Src* $(t \lfloor\cdot\rfloor\ u)$ = *Src u*

⟨*proof*⟩

lemma *Trg-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Trg* (*t* ⌊·⌋ *u*) = *Trg u*
⟨*proof*⟩

lemma *Dom-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Dom* (*t* ⌊·⌋ *u*) = *Dom u*
⟨*proof*⟩

lemma *Cod-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *Dom t* = *Cod u*
shows *Cod* (*t* ⌊·⌋ *u*) = *Cod t*
⟨*proof*⟩

lemma *VcompNml-Cod-Nml* [*simp*]:
assumes *Nml t*
shows *VcompNml* (*Cod t*) *t* = *t*
⟨*proof*⟩

lemma *VcompNml-Nml-Dom* [*simp*]:
assumes *Nml t*
shows *t* ⌊·⌋ (*Dom t*) = *t*
⟨*proof*⟩

lemma *VcompNml-Ide-Nml* [*simp*]:
assumes *Nml t* **and** *Ide a* **and** *Dom a* = *Cod t*
shows *a* ⌊·⌋ *t* = *t*
⟨*proof*⟩

lemma *VcompNml-Nml-Ide* [*simp*]:
assumes *Nml t* **and** *Ide a* **and** *Dom t* = *Cod a*
shows *t* ⌊·⌋ *a* = *t*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *Can-and-Nml-implies-Ide*:
assumes *Can t* **and** *Nml t*
shows *Ide t*
⟨*proof*⟩

lemma *VcompNml-Can-Inv* [*simp*]:
assumes *Can t* **and** *Nml t*
shows *t* ⌊·⌋ *Inv t* = *Cod t*
⟨*proof*⟩

lemma *VcompNml-Inv-Can* [*simp*]:
assumes *Can t* **and** *Nml t*
shows *Inv t* ⌊·⌋ *t* = *Dom t*
⟨*proof*⟩

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*⟩

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*⟩

lemma *Nmlize-in-Hom* [*intro*]:
assumes *Arr t*
shows $\lfloor t \rfloor \in HHom\ (Src\ t)\ (Trg\ t)$ **and** $\lfloor t \rfloor \in VHom\ \lfloor Dom\ t \rfloor\ \lfloor Cod\ t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Src*:
assumes *Arr t*
shows $\lfloor Src\ t \rfloor = Src\ \lfloor t \rfloor$ **and** $\lfloor Src\ t \rfloor = Src\ t$
⟨*proof*⟩

lemma *Nmlize-Trg*:
assumes *Arr t*
shows $\lfloor Trg\ t \rfloor = Trg\ \lfloor t \rfloor$ **and** $\lfloor Trg\ t \rfloor = Trg\ t$
⟨*proof*⟩

lemma *Nmlize-Dom*:
assumes *Arr t*
shows $\lfloor Dom\ t \rfloor = Dom\ \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Cod*:
assumes *Arr t*
shows $\lfloor Cod\ t \rfloor = Cod\ \lfloor t \rfloor$
⟨*proof*⟩

lemma *Ide-Nmlize-Ide*:
assumes *Ide t*
shows $Ide\ \lfloor t \rfloor$
⟨*proof*⟩

lemma *Ide-Nmlize-Can*:
assumes *Can t*
shows $Ide\ \lfloor t \rfloor$
⟨*proof*⟩

lemma *Can-Nmlize-Can*:
assumes *Can t*
shows $Can\ \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Nml* [*simp*]:
assumes *Nml t*
shows $\lfloor t \rfloor = t$
⟨*proof*⟩

lemma *Nmlize-Nmlize* [*simp*]:
assumes *Arr t*
shows $\lfloor \lfloor t \rfloor \rfloor = \lfloor t \rfloor$

⟨*proof*⟩

lemma *Nmlize-Hcomp*:
assumes *Arr t* **and** *Arr u*
shows $\lfloor t \star u \rfloor = \lfloor \lfloor t \rfloor \star \lfloor u \rfloor \rfloor$
⟨*proof*⟩

lemma *Nmlize-Hcomp-Obj-Arr* [*simp*]:
assumes *Arr u*
shows $\lfloor \langle b \rangle_0 \star u \rfloor = \lfloor u \rfloor$
⟨*proof*⟩

lemma *Nmlize-Hcomp-Arr-Obj* [*simp*]:
assumes *Arr t* **and** *Src t* = $\langle a \rangle_0$
shows $\lfloor t \star \langle a \rangle_0 \rfloor = \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Hcomp-Prim-Arr* [*simp*]:
assumes *Arr u* **and** ¬ *is-Prim*$_0$ $\lfloor u \rfloor$
shows $\lfloor \langle \mu \rangle \star u \rfloor = \langle \mu \rangle \star \lfloor u \rfloor$
⟨*proof*⟩

lemma *Nmlize-Hcomp-Hcomp*:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\lfloor (t \star u) \star v \rfloor = \lfloor \lfloor t \rfloor \star (\lfloor u \rfloor \star \lfloor v \rfloor) \rfloor$
⟨*proof*⟩

lemma *Nmlize-Hcomp-Hcomp′*:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows $\lfloor t \star u \star v \rfloor = \lfloor \lfloor t \rfloor \star \lfloor u \rfloor \star \lfloor v \rfloor \rfloor$
⟨*proof*⟩

lemma *Nmlize-Vcomp-Cod-Arr*:
assumes *Arr t*
shows $\lfloor Cod\ t \cdot t \rfloor = \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Vcomp-Arr-Dom*:
assumes *Arr t*
shows $\lfloor t \cdot Dom\ t \rfloor = \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-Inv*:
assumes *Can t*
shows $\lfloor Inv\ t \rfloor = Inv\ \lfloor t \rfloor$
⟨*proof*⟩

1.6.3 Reductions

Function *red* defined below takes a formal identity t to a canonical arrow $f\!\downarrow \in \mathit{Hom}\ f\ \lfloor f \rfloor$. The auxiliary function *red2* takes a pair $(f,\ g)$ of normalized formal identities and produces a canonical arrow $f \Downarrow g \in \mathit{Hom}\ (f \star g)\ \lfloor f \star g \rfloor$.

fun *red2* (**infixr** ‹⇓› *53*)
where $\langle b \rangle_0 \Downarrow u = \mathbf{l}[u]$
| $\langle f \rangle \Downarrow \langle a \rangle_0 = \mathbf{r}[\langle f \rangle]$
| $\langle f \rangle \Downarrow u = \langle f \rangle \star u$
| $(t \star u) \Downarrow \langle a \rangle_0 = \mathbf{r}[t \star u]$
| $(t \star u) \Downarrow v = (t \Downarrow \lfloor u \star v \rfloor) \cdot (t \star (u \Downarrow v)) \cdot \mathbf{a}[t,\ u,\ v]$
| $t \Downarrow u = \mathit{undefined}$

fun *red* (‹-↓› [*56*] *56*)
where $\langle f \rangle_0\!\downarrow = \langle f \rangle_0$
| $\langle f \rangle\!\downarrow = \langle f \rangle$
| $(t \star u)\!\downarrow = (\mathit{if}\ \mathit{Nml}\ (t \star u)\ \mathit{then}\ t \star u\ \mathit{else}\ (\lfloor t \rfloor \Downarrow \lfloor u \rfloor) \cdot (t\!\downarrow \star u\!\downarrow))$
| $t\!\downarrow = \mathit{undefined}$

lemma *red-Nml* [*simp*]:
assumes *Nml a*
shows $a\!\downarrow = a$
⟨*proof*⟩

lemma *red2-Nml*:
assumes *Nml* $(a \star b)$
shows $a \Downarrow b = a \star b$
⟨*proof*⟩

lemma *Can-red2*:
assumes *Ide a* **and** *Nml a* **and** *Ide b* **and** *Nml b* **and** *Src a* = *Trg b*
shows *Can* $(a \Downarrow b)$
and $a \Downarrow b \in \mathit{VHom}\ (a \star b)\ \lfloor a \star b \rfloor$
⟨*proof*⟩

lemma *red2-in-Hom* [*intro*]:
assumes *Ide u* **and** *Nml u* **and** *Ide v* **and** *Nml v* **and** *Src u* = *Trg v*
shows $u \Downarrow v \in \mathit{HHom}\ (\mathit{Src}\ v)\ (\mathit{Trg}\ u)$ **and** $u \Downarrow v \in \mathit{VHom}\ (u \star v)\ \lfloor u \star v \rfloor$
⟨*proof*⟩

lemma *red2-simps* [*simp*]:
assumes *Ide u* **and** *Nml u* **and** *Ide v* **and** *Nml v* **and** *Src u* = *Trg v*
shows $\mathit{Src}\ (u \Downarrow v) = \mathit{Src}\ v$ **and** $\mathit{Trg}\ (u \Downarrow v) = \mathit{Trg}\ u$
and $\mathit{Dom}\ (u \Downarrow v) = u \star v$ **and** $\mathit{Cod}\ (u \Downarrow v) = \lfloor u \star v \rfloor$
⟨*proof*⟩

lemma *Can-red*:
assumes *Ide u*
shows *Can* $(u\!\downarrow)$ **and** $u\!\downarrow \in \mathit{VHom}\ u\ \lfloor u \rfloor$

⟨*proof*⟩

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*⟩

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* ($t\!\downarrow$) = $\lfloor t \rfloor$
⟨*proof*⟩

lemma *red-Src*:
assumes *Ide t*
shows *Src* $t\!\downarrow$ = *Src t*
⟨*proof*⟩

lemma *red-Trg*:
assumes *Ide t*
shows *Trg* $t\!\downarrow$ = *Trg t*
⟨*proof*⟩

lemma *Nmlize-red* [*simp*]:
assumes *Ide t*
shows $\lfloor t\!\downarrow \rfloor = \lfloor t \rfloor$
⟨*proof*⟩

lemma *Nmlize-red2* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Nml t* **and** *Nml u* **and** *Src t* = *Trg u*
shows $\lfloor t \Downarrow u \rfloor = \lfloor t \star u \rfloor$
⟨*proof*⟩

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** $\langle\cdot_C\rangle$ *55*)
and src_C :: $'c \Rightarrow 'c$
and trg_C :: $'c \Rightarrow 'c$
and *V* :: *'b comp* (**infixr** $\langle\cdot\rangle$ *55*)

and $H :: 'b\ comp$ (**infixr** ‹⋆› *53*)
and a :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a[-, -, -]›)
and i :: $'b \Rightarrow 'b$ (‹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*⟩

lemma *eval-in-hom'*:
shows $Arr\ t \Longrightarrow «⦇t⦈ : ⦇Src\ t⦈ \rightarrow ⦇Trg\ t⦈» \wedge «⦇t⦈ : ⦇Dom\ t⦈ \Rightarrow ⦇Cod\ t⦈»$
⟨*proof*⟩

lemma *eval-in-hom* [*intro*]:
assumes $Arr\ t$
shows $«⦇t⦈ : ⦇Src\ t⦈ \rightarrow ⦇Trg\ t⦈»$ **and** $«⦇t⦈ : ⦇Dom\ t⦈ \Rightarrow ⦇Cod\ t⦈»$
⟨*proof*⟩

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** $⦇Cod\ f⦈ = cod\ ⦇f⦈$
⟨*proof*⟩

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*⦈
⟨*proof*⟩

lemma *obj-eval-Obj*:
shows *Obj t* ⟹ *obj* ⦇*t*⦈
⟨*proof*⟩

lemma *ide-eval-Ide*:
shows *Ide t* ⟹ *ide* ⦇*t*⦈
⟨*proof*⟩

lemma *arr-eval-Arr* [*simp*]:
assumes *Arr t*
shows *arr* ⦇*t*⦈
⟨*proof*⟩

lemma *eval-Lunit* [*simp*]:
assumes *Arr t*
shows ⦇**l**[*t*]⦈ = l[⦇*Cod t*⦈] · (*trg* ⦇*t*⦈ ⋆ ⦇*t*⦈)
⟨*proof*⟩

lemma *eval-Lunit'* [*simp*]:
assumes *Arr t*
shows ⦇**l**⁻¹[*t*]⦈ = l⁻¹[⦇*Cod t*⦈] · ⦇*t*⦈
⟨*proof*⟩

lemma *eval-Runit* [*simp*]:
assumes *Arr t*
shows ⦇**r**[*t*]⦈ = r[⦇*Cod t*⦈] · (⦇*t*⦈ ⋆ *src* ⦇*t*⦈)
⟨*proof*⟩

lemma *eval-Runit'* [*simp*]:
assumes *Arr t*
shows ⦇**r**⁻¹[*t*]⦈ = r⁻¹[⦇*Cod t*⦈] · ⦇*t*⦈
⟨*proof*⟩

lemma *eval-Assoc* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Arr v* **and** *Src t* = *Trg u* **and** *Src u* = *Trg v*
shows ⦇**a**[*t*, *u*, *v*]⦈ = α (*cod* ⦇*t*⦈, *cod* ⦇*u*⦈, *cod* ⦇*v*⦈) · ((⦇*t*⦈ ⋆ ⦇*u*⦈) ⋆ ⦇*v*⦈)
⟨*proof*⟩

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*⟩

lemma *eval-Lunit-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{l}[t]|\!\} = \mathrm{l}[\{\!|t|\!\}]$
⟨*proof*⟩

lemma *eval-Lunit′-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{l}^{-1}[t]|\!\} = \mathrm{l}^{-1}[\{\!|t|\!\}]$
⟨*proof*⟩

lemma *eval-Runit-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{r}[t]|\!\} = \mathrm{r}[\{\!|t|\!\}]$
⟨*proof*⟩

lemma *eval-Runit′-Ide* [*simp*]:
assumes *Ide t*
shows $\{\!|\mathbf{r}^{-1}[t]|\!\} = \mathrm{r}^{-1}[\{\!|t|\!\}]$
⟨*proof*⟩

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 $\{\!|\mathbf{a}[t,\ u,\ v]|\!\} = \alpha\ (\{\!|t|\!\},\ \{\!|u|\!\},\ \{\!|v|\!\})$
⟨*proof*⟩

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 $\{\!|\mathbf{a}^{-1}[t,\ u,\ v]|\!\} = \mathrm{a}^{-1}[\{\!|t|\!\},\ \{\!|u|\!\},\ \{\!|v|\!\}]$
⟨*proof*⟩

lemma *iso-eval-Can*:
shows *Can t* ⟹ *iso* $\{\!|t|\!\}$
⟨*proof*⟩

lemma *eval-Inv-Can*:
shows *Can t* ⟹ $\{\!|Inv\ t|\!\} = inv\ \{\!|t|\!\}$
⟨*proof*⟩

lemma *eval-VcompNml*:
assumes *Nml t* **and** *Nml u* **and** *VSeq t u*
shows $\{\!|t\ \lfloor\cdot\rfloor\ u|\!\} = \{\!|t|\!\} \cdot \{\!|u|\!\}$
⟨*proof*⟩

lemma *eval-red-Hcomp*:
assumes *Ide a* **and** *Ide b*
shows $\{\!|(a \star b)\downarrow|\!\} = \{\!|\lfloor a\rfloor \Downarrow \lfloor b\rfloor|\!\} \cdot (\{\!|a\downarrow|\!\} \star \{\!|b\downarrow|\!\})$

⟨*proof*⟩

lemma *eval-red2-Nml-Prim*$_0$:
assumes *Ide t* **and** *Nml t* **and** *Src t* = ⟨*a*⟩$_0$
shows ⦇*t* ⇓ ⟨*a*⟩$_0$⦈ = r[⦇*t*⦈]
⟨*proof*⟩

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* ⟨*proof*⟩

sublocale *evaluation-map V src trg V H* a i *src trg* ‹λμ. *if arr* μ *then* μ *else null*›
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *canonical-factorization*:
assumes *Arr t*
shows *coherent t* ⟷ ⦇*t*⦈ = *inv* ⦇*Cod t*↓⦈ · ⦇⌊*t*⌋⦈ · ⦇*Dom t*↓⦈

⟨*proof*⟩

lemma *coherent-iff-coherent-Inv*:
assumes *Can t*
shows *coherent t* ⟷ *coherent* (*Inv t*)
⟨*proof*⟩

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* ⟨*a*⟩$_0$
⟨*proof*⟩

lemma *coherent-Prim*:
assumes *Arr* ⟨*f*⟩
shows *coherent* ⟨*f*⟩
⟨*proof*⟩

lemma *coherent-Lunit-Ide*:
assumes *Ide t*
shows *coherent* **l**[*t*]
⟨*proof*⟩

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* **r**[*t*]
⟨*proof*⟩

lemma *coherent-Lunit'-Ide*:
assumes *Ide a*
shows *coherent* **l**$^{-1}$[*a*]
⟨*proof*⟩

lemma *coherent-Runit'-Ide*:
assumes *Ide a*
shows *coherent* **r**$^{-1}$[*a*]
⟨*proof*⟩

lemma *red2-Nml-Src*:
assumes *Ide t* **and** *Nml t*
shows ⦇*t* ⇓ *Src t*⦈ = r[⦇*t*⦈]
⟨*proof*⟩

lemma *red2-Trg-Nml*:
assumes *Ide t* **and** *Nml t*

shows $\{\!|Trg\ t \Downarrow t|\!\} = \mathrm{l}[\{\!|t|\!\}]$
⟨*proof*⟩

lemma *coherence-key-fact*:
assumes *Ide a* ∧ *Nml a* **and** *Ide b* ∧ *Nml b* **and** *Ide c* ∧ *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*⟩

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*⟩

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* $\mathbf{a}^{-1}[a, b, c]$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

The main result: "Every formal arrow is coherent."

theorem *coherence*:
assumes *Arr t*
shows *coherent t*
⟨*proof*⟩

corollary *eval-eqI*:
assumes *VPar t u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows $\{\!|t|\!\} = \{\!|u|\!\}$
⟨*proof*⟩

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*⟩

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* ⟨*proof*⟩
interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
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* ≡ $\{\!|Inv\ (u\!\downarrow) \cdot t\!\downarrow|\!\}$

1.7.1 Basic Properties

The following develop basic properties of *can*.

lemma *can-in-hom* [*intro*]:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows «*can u t* : $\{\!|t|\!\} \Rightarrow \{\!|u|\!\}$»
⟨*proof*⟩

lemma *can-simps* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows *arr* (*can u t*) **and** *dom* (*can u t*) = $\{\!|t|\!\}$ **and** *cod* (*can u t*) = $\{\!|u|\!\}$
⟨*proof*⟩

lemma *inverse-arrows-can*:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows *iso* (*can u t*) **and** *inverse-arrows* (*can u t*) (*can t u*)
⟨*proof*⟩

lemma *inv-can* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows *inv* (*can u t*) = *can t u*
⟨*proof*⟩

lemma *vcomp-can* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Ide v* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** $\lfloor u \rfloor = \lfloor v \rfloor$
shows *can v u* · *can u t* = *can v t*
⟨*proof*⟩

lemma *hcomp-can* [*simp*]:
assumes *Ide t* **and** *Ide u* **and** *Ide v* **and** *Ide w* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** $\lfloor v \rfloor = \lfloor w \rfloor$
and *Src t* = *Trg v* **and** *Src u* = *Trg w*
shows *can u t* ⋆ *can w v* = *can* (*u* ⋆ *w*) (*t* ⋆ *v*)
⟨*proof*⟩

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 l[*f*] = *can* ⟨*f*⟩ (⟨*trg f*⟩$_0$ ⋆ ⟨*f*⟩)
and r[*f*] = *can* ⟨*f*⟩ (⟨*f*⟩ ⋆ ⟨*src f*⟩$_0$)
⟨*proof*⟩

lemma *canI-unitor-1*:
assumes *obj a*
shows l[*a*] = *can* ⟨*a*⟩$_0$ (⟨*a*⟩$_0$ ⋆ ⟨*a*⟩$_0$)
and r[*a*] = *can* ⟨*a*⟩$_0$ (⟨*a*⟩$_0$ ⋆ ⟨*a*⟩$_0$)
⟨*proof*⟩

lemma *canI-associator-0*:
assumes *ide f* **and** *ide g* **and** *ide h* **and** *src f* = *trg g* **and** *src g* = *trg h*

shows $\mathrm{a}[f,\ g,\ h] = \mathit{can}\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle)$
⟨*proof*⟩

lemma *canI-associator-1*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows $\mathrm{a}[\mathit{trg}\ f,\ f,\ g] = \mathit{can}\ (\langle \mathit{trg}\ f\rangle_0 \star \langle f\rangle \star \langle g\rangle)\ ((\langle \mathit{trg}\ f\rangle_0 \star \langle f\rangle) \star \langle g\rangle)$
and $\mathrm{a}[f,\ \mathit{src}\ f,\ g] = \mathit{can}\ (\langle f\rangle \star \langle \mathit{src}\ f\rangle_0 \star \langle g\rangle)\ ((\langle f\rangle \star \langle \mathit{src}\ f\rangle_0) \star \langle g\rangle)$
and $\mathrm{a}[f,\ g,\ \mathit{src}\ g] = \mathit{can}\ (\langle f\rangle \star \langle g\rangle \star \langle \mathit{src}\ g\rangle_0)\ ((\langle f\rangle \star \langle g\rangle) \star \langle \mathit{src}\ g\rangle_0)$
⟨*proof*⟩

lemma *canI-associator-2*:
assumes *ide f*
shows $\mathrm{a}[\mathit{trg}\ f,\ \mathit{trg}\ f,\ f] = \mathit{can}\ (\langle \mathit{trg}\ f\rangle_0 \star \langle \mathit{trg}\ f\rangle_0 \star \langle f\rangle)\ ((\langle \mathit{trg}\ f\rangle_0 \star \langle \mathit{trg}\ f\rangle_0) \star \langle f\rangle)$
and $\mathrm{a}[\mathit{trg}\ f,\ f,\ \mathit{src}\ f] = \mathit{can}\ (\langle \mathit{trg}\ f\rangle_0 \star \langle f\rangle \star \langle \mathit{src}\ f\rangle_0)\ ((\langle \mathit{trg}\ f\rangle_0 \star \langle f\rangle) \star \langle \mathit{src}\ f\rangle_0)$
and $\mathrm{a}[f,\ \mathit{src}\ f,\ \mathit{src}\ f] = \mathit{can}\ (\langle f\rangle \star \langle \mathit{src}\ f\rangle_0 \star \langle \mathit{src}\ f\rangle_0)\ ((\langle f\rangle \star \langle \mathit{src}\ f\rangle_0) \star \langle \mathit{src}\ f\rangle_0)$
⟨*proof*⟩

lemma *canI-associator-3*:
assumes *obj a*
shows $\mathrm{a}[a,\ a,\ a] = \mathit{can}\ (\langle a\rangle_0 \star \langle a\rangle_0 \star \langle a\rangle_0)\ ((\langle a\rangle_0 \star \langle a\rangle_0) \star \langle a\rangle_0)$
⟨*proof*⟩

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 $\mathrm{a}[f \star g,\ h,\ k] = \mathit{can}\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)\ (((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle) \star \langle k\rangle)$
and $\mathrm{a}[f,\ g \star h,\ k] = \mathit{can}\ (\langle f\rangle \star (\langle g\rangle \star \langle h\rangle) \star \langle k\rangle)\ ((\langle f\rangle \star (\langle g\rangle \star \langle h\rangle)) \star \langle k\rangle)$
and $\mathrm{a}[f,\ g,\ h \star k] = \mathit{can}\ (\langle f\rangle \star \langle g\rangle \star \langle h\rangle \star \langle k\rangle)\ ((\langle f\rangle \star \langle g\rangle) \star \langle h\rangle \star \langle k\rangle)$
⟨*proof*⟩

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 $\mathit{can}\ f\ (f \star \mathit{Src}\ f) = \mathrm{r}[\{\!|f|\!\}]$
and $\mathit{can}\ f\ (\mathit{Trg}\ f \star f) = \mathrm{l}[\{\!|f|\!\}]$
and $\mathit{can}\ (f \star \mathit{Src}\ f)\ f = \mathrm{r}^{-1}[\{\!|f|\!\}]$
and $\mathit{can}\ (\mathit{Trg}\ f \star f)\ f = \mathrm{l}^{-1}[\{\!|f|\!\}]$
⟨*proof*⟩

lemma *canE-associator*:
assumes *Ide f* **and** *Ide g* **and** *Ide h* **and** *Src f* = *Trg g* **and** *Src g* = *Trg h*
shows $\mathit{can}\ (f \star g \star h)\ ((f \star g) \star h) = \mathrm{a}[\{\!|f|\!\},\ \{\!|g|\!\},\ \{\!|h|\!\}]$
and $\mathit{can}\ ((f \star g) \star h)\ (f \star g \star h) = \mathrm{a}^{-1}[\{\!|f|\!\},\ \{\!|g|\!\},\ \{\!|h|\!\}]$
⟨*proof*⟩

lemma *can-Ide-self*:
assumes *Ide t*
shows *can t t* = $\{\!|t|\!\}$
⟨*proof*⟩

1.7.4 Rules for Whiskering

lemma *whisker-can-right-0*:
assumes *Ide t* **and** *Ide u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$ **and** *ide f* **and** *Src t* = $\langle trg\ f \rangle_0$
shows *can u t* ⋆ *f* = *can* (*u* ⋆ ⟨*f*⟩) (*t* ⋆ ⟨*f*⟩)
⟨*proof*⟩

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* ⋆ *a* = *can* (*u* ⋆ $\langle a \rangle_0$) (*t* ⋆ $\langle a \rangle_0$)
⟨*proof*⟩

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* ⋆ *can u t* = *can* (⟨*g*⟩ ⋆ *u*) (⟨*g*⟩ ⋆ *t*)
⟨*proof*⟩

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* ⋆ *can u t* = *can* ($\langle b \rangle_0$ ⋆ *u*) ($\langle b \rangle_0$ ⋆ *t*)
⟨*proof*⟩

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** ‹·$_B$› *55*)
and *H* :: *'a comp* (**infixr** ‹⋆$_B$› *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 *Arr* :: *'a* ⇒ *bool* +
assumes *src-closed*: *Arr f* ⟹ *Arr* (*src*$_B$ *f*)
and *trg-closed*: *Arr f* ⟹ *Arr* (*trg*$_B$ *f*)
and *hcomp-closed*: ⟦ *Arr f*; *Arr g*; *trg*$_B$ *f* = *src*$_B$ *g* ⟧ ⟹ *Arr* (*g* ⋆$_B$ *f*)
and *assoc-closed*: ⟦ *Arr f* ∧ *B.ide f*; *Arr g* ∧ *B.ide g*; *Arr h* ∧ *B.ide h*;
  src$_B$ *f* = *trg*$_B$ *g*; *src*$_B$ *g* = *trg*$_B$ *h* ⟧ ⟹ *Arr* (a_B *f g h*)
and *assoc'-closed*: ⟦ *Arr f* ∧ *B.ide f*; *Arr g* ∧ *B.ide g*; *Arr h* ∧ *B.ide h*;
  src$_B$ *f* = *trg*$_B$ *g*; *src*$_B$ *g* = *trg*$_B$ *h* ⟧ ⟹ *Arr* (*B.inv* (a_B *f g h*))
and *lunit-closed*: ⟦ *Arr f*; *B.ide f* ⟧ ⟹ *Arr* (*B.*𝔩 *f*)
and *lunit'-closed*: ⟦ *Arr f*; *B.ide f* ⟧ ⟹ *Arr* (*B.inv* (*B.*𝔩 *f*))
and *runit-closed*: ⟦ *Arr f*; *B.ide f* ⟧ ⟹ *Arr* (*B.*𝔯 *f*)
and *runit'-closed*: ⟦ *Arr f*; *B.ide f* ⟧ ⟹ *Arr* (*B.inv* (*B.*𝔯 *f*))
begin

  notation *B.in-hom* (‹«- : - ⇒$_B$ -»›)

  notation *comp* (**infixr** ‹·› *55*)

  definition *hcomp* (**infixr** ‹⋆› *53*)
  where *g* ⋆ *f* = (*if arr f* ∧ *arr g* ∧ *trg*$_B$ *f* = *src*$_B$ *g then g* ⋆$_B$ *f else null*)

  definition *src*
  where *src* *μ* = (*if arr* *μ* *then src*$_B$ *μ* *else null*)

  definition *trg*
  where *trg* *μ* = (*if arr* *μ* *then trg*$_B$ *μ* *else null*)

  interpretation *src*: *endofunctor* ‹(·)› *src*
    ⟨*proof*⟩

  interpretation *trg*: *endofunctor* ‹(·)› *trg*
    ⟨*proof*⟩

  interpretation *horizontal-homs* ‹(·)› *src trg*
    ⟨*proof*⟩

  interpretation *functor VV.comp* ‹(·)› ‹λ*μν*. *fst* *μν* ⋆ *snd* *μν*›
    ⟨*proof*⟩

interpretation *horizontal-composition* ‹(·)› ‹(⋆)› *src trg*
⟨*proof*⟩

abbreviation a
where a $\mu\ \nu\ \tau \equiv$ *if VVV.arr* (μ, ν, τ) *then* $\mathrm{a}_B\ \mu\ \nu\ \tau$ *else null*

abbreviation (*input*) α_{SB}
where $\alpha_{SB}\ \mu\nu\tau \equiv$ 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* ($\alpha_{SB}\ \mu\nu\tau$)
⟨*proof*⟩

lemma *lunit-closed′*:
assumes *Arr f*
shows *Arr* ($B.\mathfrak{l}\ f$)
⟨*proof*⟩

lemma *runit-closed′*:
assumes *Arr f*
shows *Arr* ($B.\mathfrak{r}\ f$)
⟨*proof*⟩

interpretation *natural-isomorphism VVV.comp* ‹(·)› *HoHV HoVH* α_{SB}
⟨*proof*⟩

interpretation *L*: *endofunctor* ‹(·)› *L*
⟨*proof*⟩
interpretation *R*: *endofunctor* ‹(·)› *R*
⟨*proof*⟩

interpretation *L*: *faithful-functor* ‹(·)› ‹(·)› *L*
⟨*proof*⟩
interpretation *L*: *full-functor* ‹(·)› ‹(·)› *L*
⟨*proof*⟩

interpretation *R*: *faithful-functor* ‹(·)› ‹(·)› *R*
⟨*proof*⟩
interpretation *R*: *full-functor* ‹(·)› ‹(·)› *R*
⟨*proof*⟩

interpretation *bicategory* ‹(·)› ‹(⋆)› a i *src trg*
⟨*proof*⟩

proposition *is-bicategory*:
shows *bicategory* (·) (⋆) a i *src trg*
⟨*proof*⟩

lemma *obj-char*:
shows *obj a* $\longleftrightarrow$ *arr a* $\wedge$ *B.obj a*
⟨*proof*⟩

lemma *hcomp-char*:
shows *hcomp* = (λ*f g*. *if arr f* $\wedge$ *arr g* $\wedge$ *src f* = *trg g then f* $\star_B$ *g else null*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *lunit-simp*:
assumes *ide f*
shows *lunit f* = *B.lunit f*
⟨*proof*⟩

lemma *lunit′-simp*:
assumes *ide f*
shows *lunit′ f* = *B.lunit′ f*
⟨*proof*⟩

lemma *runit-simp*:
assumes *ide f*
shows *runit f* = *B.runit f*
⟨*proof*⟩

lemma *runit′-simp*:
assumes *ide f*
shows *runit′ f* = *B.runit′ f*
⟨*proof*⟩

lemma *comp-eqI* [*intro*]:
assumes *seq f g* **and** *f* = *f′* **and** *g* = *g′*
shows *f* · *g* = *f′* ·$_B$ *g′*
⟨*proof*⟩

lemma *comp-eqI′* [*intro*]:
assumes *seq f g* **and** *f* = *f′* **and** *g* = *g′*
shows *f* ·$_B$ *g* = *f′* · *g′*
⟨*proof*⟩

lemma *hcomp-eqI* [*intro*]:

assumes *hseq f g* **and** $f = f'$ **and** $g = g'$
shows $f \star g = f' \star_B g'$
⟨*proof*⟩

lemma *hcomp-eqI'* [*intro*]:
assumes *hseq f g* **and** $f = f'$ **and** $g = g'$
shows $f \star_B g = f' \star g'$
⟨*proof*⟩

lemma *arr-compI*:
assumes *seq f g*
shows *arr* $(f \cdot_B g)$
⟨*proof*⟩

lemma *arr-hcompI*:
assumes *hseq f g*
shows *arr* $(f \star_B g)$
⟨*proof*⟩

end

sublocale *subbicategory* ⊆ *bicategory* ‹(·)› ‹(⋆)› a i *src trg*
⟨*proof*⟩

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 «$\omega : w \star_B w \Rightarrow w$» 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*$_B$ ‹λμ. *B.arr* μ ∧ *src*$_B$ μ = *a* ∧ *trg*$_B$ μ = *a*›
for *V* :: *'a comp* (**infixr** ‹·$_B$› *55*)
and *H* :: *'a comp* (**infixr** ‹⋆$_B$› *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* ⋆$_B$ *w* ⇒ *w*»
and ω*-is-iso*: *B.iso* ω

begin

notation *hcomp* (**infixr** ‹⋆› *53*)

lemma *arr-simps*:
assumes *arr* μ
shows *src* $\mu = a$ **and** *trg* $\mu = a$
⟨*proof*⟩

lemma ω*-simps* [*simp*]:
shows *arr* ω
and *src* $\omega = a$ **and** *trg* $\omega = a$
and *dom* $\omega = w \star_B w$ **and** *cod* $\omega = w$
⟨*proof*⟩

lemma *ide-w*:
shows *B.ide w*
⟨*proof*⟩

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*⟩

lemma *VxV-arr-eq-VV-arr*:
shows *VxV.arr f* $\longleftrightarrow$ *VV.arr f*
⟨*proof*⟩

lemma *VxV-comp-eq-VV-comp*:
shows *VxV.comp* = *VV.comp*
⟨*proof*⟩

lemma *VxVxV-arr-eq-VVV-arr*:
shows *VxVxV.arr f* $\longleftrightarrow$ *VVV.arr f*
⟨*proof*⟩

lemma *VxVxV-comp-eq-VVV-comp*:
shows *VxVxV.comp* = *VVV.comp*
⟨*proof*⟩

interpretation *H*: *functor VxV.comp* ‹(·)› ‹$\lambda\mu\nu$. *fst* $\mu\nu \star$ *snd* $\mu\nu$›
⟨*proof*⟩

interpretation *H*: *binary-endofunctor* ‹(·)› ‹$\lambda\mu\nu$. *fst* $\mu\nu \star$ *snd* $\mu\nu$› ⟨*proof*⟩

lemma *HoHV-eq-ToTC*:
shows *HoHV* = *H.ToTC*
⟨*proof*⟩

lemma *HoVH-eq-ToCT*:
shows *HoVH* = *H.ToCT*
⟨*proof*⟩

interpretation *ToTC*: *functor VxVxV.comp* ‹(·)› *H.ToTC*
⟨*proof*⟩
interpretation *ToCT*: *functor VxVxV.comp* ‹(·)› *H.ToCT*
⟨*proof*⟩

interpretation α: *natural-isomorphism VxVxV.comp* ‹(·)› *H.ToTC H.ToCT* α
⟨*proof*⟩

interpretation *L*: *endofunctor* ‹(·)› ‹λ*f*. *fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)›
⟨*proof*⟩

interpretation *L′*: *equivalence-functor* ‹(·)› ‹(·)› ‹λ*f*. *fst* (*w*, *f*) ⋆ *snd* (*w*, *f*)›
⟨*proof*⟩
interpretation *L*: *equivalence-functor* ‹(·)› ‹(·)› ‹λ*f*. *fst* (*cod* ω, *f*) ⋆ *snd* (*cod* ω, *f*)›
⟨*proof*⟩

interpretation *R*: *endofunctor* ‹(·)› ‹λ*f*. *fst* (*f*, *w*) ⋆ *snd* (*f*, *w*)›
⟨*proof*⟩

interpretation *R′*: *equivalence-functor* ‹(·)› ‹(·)› ‹λ*f*. *fst* (*f*, *w*) ⋆ *snd* (*f*, *w*)›
⟨*proof*⟩
interpretation *R*: *equivalence-functor* ‹(·)› ‹(·)› ‹λ*f*. *fst* (*f*, *cod* ω) ⋆ *snd* (*f*, *cod* ω)›
⟨*proof*⟩

interpretation *M*: *monoidal-category* ‹(·)› ‹λμν. *fst* μν ⋆ *snd* μν› α ω
⟨*proof*⟩

proposition *is-monoidal-category*:
shows *monoidal-category* (·) (λμν. *fst* μν ⋆ *snd* μν) α ω
⟨*proof*⟩

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, ι) and (a', ι'), 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 «ω : $w \star w \Rightarrow w$»
and *iso* ω
shows $\exists!\psi$. «ψ : $a \Rightarrow w$» $\wedge$ *iso* $\psi \wedge \psi \cdot \mathrm{i}[a] = \omega \cdot (\psi \star \psi)$
⟨*proof*⟩

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 $\eta :: 'a$
and $\varepsilon :: 'a$
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*

⟨*proof*⟩

lemma *counit-in-hom* [*intro*]:
shows «ε : *trg f* → *trg f*» **and** «ε : $f \star g$ ⇒ *trg f*»
⟨*proof*⟩

lemma *unit-in-hom* [*intro*]:
shows «η : *src f* → *src f*» **and** «η : *src f* ⇒ $g \star f$»
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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 «$\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] : g \Rightarrow g$»
⟨*proof*⟩

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*⟩

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)
⟨*proof*⟩

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* ≡ ∃ *φ ψ*. *internal-equivalence f g φ ψ*

lemma *quasi-inversesI*:
assumes *ide f* **and** *ide g*
and *src f* ≅ *g* ⋆ *f* **and** *f* ⋆ *g* ≅ *trg f*
shows *quasi-inverses f g*
⟨*proof*⟩

lemma *quasi-inversesE*:
assumes *quasi-inverses f g*
and ⟦*ide f*; *ide g*; *src f* ≅ *g* ⋆ *f*; *f* ⋆ *g* ≅ *trg f*⟧ ⟹ *T*
shows *T*
⟨*proof*⟩

lemma *quasi-inverse-unique*:
assumes *quasi-inverses f g* **and** *quasi-inverses f g′*
shows *isomorphic g g′*
⟨*proof*⟩

lemma *quasi-inverses-symmetric*:
assumes *quasi-inverses f g*
shows *quasi-inverses g f*
⟨*proof*⟩

definition *equivalence-map*
where *equivalence-map f* ≡ ∃ *g η ε*. *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*
⟨*proof*⟩

lemma *equivalence-mapE*:
assumes *equivalence-map f*

obtains g **where** *quasi-inverses f g*
⟨*proof*⟩

lemma *equivalence-map-is-ide*:
assumes *equivalence-map f*
shows *ide f*
⟨*proof*⟩

lemma *obj-is-equivalence-map*:
assumes *obj a*
shows *equivalence-map a*
⟨*proof*⟩

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′* $((g' \star \varphi) \cdot (\psi \star f) \cdot \eta)$ $(\varepsilon \cdot (inv\ \varphi \star g) \cdot (f' \star inv\ \psi))$
⟨*proof*⟩

lemma *equivalence-map-preserved-by-iso*:
assumes *equivalence-map f* **and** $f \cong f'$
shows *equivalence-map f′*
⟨*proof*⟩

lemma *equivalence-preserved-by-iso-right*:
assumes *equivalence-in-bicategory V H* a i *src trg f g η ε*
and «$\varphi : g \Rightarrow g'$» **and** *iso* φ
shows *equivalence-in-bicategory V H* a i *src trg f g′* $((\varphi \star f) \cdot \eta)$ $(\varepsilon \cdot (f \star inv\ \varphi))$
⟨*proof*⟩

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*⟩

definition *some-quasi-inverse*
where *some-quasi-inverse f* = (*SOME g. quasi-inverses f g*)

notation *some-quasi-inverse* (‹-~› [*1000*] *1000*)

lemma *quasi-inverses-some-quasi-inverse*:
assumes *equivalence-map f*
shows *quasi-inverses f* $f^{\sim}$
and *quasi-inverses* $f^{\sim}$ *f*
⟨*proof*⟩

lemma *quasi-inverse-antipar*:
assumes *equivalence-map f*

shows $src\ f^{\sim} = trg\ f$ **and** $trg\ f^{\sim} = src\ f$
⟨*proof*⟩

lemma *quasi-inverse-in-hom* [*intro*]:
assumes *equivalence-map f*
shows «$f^{\sim} : trg\ f \to src\ f$»
and «$f^{\sim} : f^{\sim} \Rightarrow f^{\sim}$»
⟨*proof*⟩

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** $cod\ f^{\sim} = f^{\sim}$
⟨*proof*⟩

lemma *quasi-inverse-quasi-inverse*:
assumes *equivalence-map f*
shows $(f^{\sim})^{\sim} \cong f$
⟨*proof*⟩

lemma *comp-quasi-inverse*:
assumes *equivalence-map f*
shows $f^{\sim} \star f \cong src\ f$ **and** $f \star f^{\sim} \cong trg\ f$
⟨*proof*⟩

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*⟩

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 $src\ trg\ \langle h \star f \rangle\ \langle g \star k \rangle\ \eta\ \varepsilon$
⟨*proof*⟩

interpretation *equivalence-in-bicategory* $V\ H$ a i $src\ trg\ \langle h \star f \rangle\ \langle g \star k \rangle\ \eta\ \varepsilon$
⟨*proof*⟩

lemma *is-equivalence*:
shows *equivalence-in-bicategory* $V\ H$ a i $src\ trg\ (h \star f)\ (g \star k)\ \eta\ \varepsilon$
⟨*proof*⟩

sublocale *equivalence-in-bicategory* $V\ H$ a i $src\ trg\ \langle h \star f \rangle\ \langle g \star k \rangle\ \eta\ \varepsilon$
⟨*proof*⟩

end

context *bicategory*
begin

lemma *equivalence-maps-compose*:
assumes *equivalence-map* f **and** *equivalence-map* f' **and** $src\ f' = trg\ f$
shows *equivalence-map* $(f' \star f)$
⟨*proof*⟩

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*⟩

lemma *quasi-inverse-hcomp*:
assumes *equivalence-map* f **and** *equivalence-map* f' **and** *equivalence-map* $(f \star f')$
shows $(f \star f')^{\sim} \cong f'^{\sim} \star f^{\sim}$
⟨*proof*⟩

lemma *quasi-inverse-respects-isomorphic*:

assumes *equivalence-map f* **and** *equivalence-map f'* **and** $f \cong f'$
shows $f^{\sim} \cong f'^{\sim}$
⟨*proof*⟩

end

1.9.4 Equivalent Objects

context *bicategory*
begin

definition *equivalent-objects*
where *equivalent-objects a b* ≡ ∃*f*. «*f* : *a* → *b*» ∧ *equivalence-map f*

lemma *equivalent-objects-reflexive*:
assumes *obj a*
shows *equivalent-objects a a*
⟨*proof*⟩

lemma *equivalent-objects-symmetric*:
assumes *equivalent-objects a b*
shows *equivalent-objects b a*
⟨*proof*⟩

lemma *equivalent-objects-transitive* [*trans*]:
assumes *equivalent-objects a b* **and** *equivalent-objects b c*
shows *equivalent-objects a c*
⟨*proof*⟩

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* ⇒ *'a* ⇒ *'a* (**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 e_0 :: *'a*
and d_0 :: *'a*
and η_0 :: *'a*

and $\varepsilon_0 :: 'a$
and $e_1 :: 'a$
and $d_1 :: 'a$
and $\eta_1 :: 'a$
and $\varepsilon_1 :: 'a$
begin

interpretation *hom*: *subcategory V* ‹$\lambda\mu$. «μ : *src* e_0 → *src* e_1»›
⟨*proof*⟩

no-notation *in-hom* (‹«- : - → -»›)

interpretation *hom'*: *subcategory V* ‹$\lambda\mu$. «μ : *trg* e_0 → *trg* e_1»›
⟨*proof*⟩

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*⟩

abbreviation (*input*) *G*
where $G \equiv \lambda\mu'.\ d_1 \star \mu' \star e_0$

interpretation *G*: *functor hom'.comp hom.comp G*
⟨*proof*⟩

interpretation *GF*: *composite-functor hom.comp hom'.comp hom.comp F G* ⟨*proof*⟩
interpretation *FG*: *composite-functor hom'.comp hom.comp hom'.comp G F* ⟨*proof*⟩

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 → *src* e_1» **and** *ide f*
shows «$\varphi_0\ f$: *src* e_0 → *src* e_1»
and «$\varphi_0\ f$: $f \Rightarrow d_1 \star (e_1 \star f \star d_0) \star e_0$»
⟨*proof*⟩

lemma *iso*-φ_0:
assumes «f : *src* e_0 → *src* e_1» **and** *ide f*
shows *iso* ($\varphi_0\ f$)
⟨*proof*⟩

interpretation φ: *transformation-by-components hom.comp hom.comp hom.map* ‹*G o F*› φ_0

⟨*proof*⟩

lemma *transformation-by-components-*φ_0:
shows *transformation-by-components hom.comp hom.comp hom.map* (*G o F*) φ_0
⟨*proof*⟩

interpretation φ: *natural-isomorphism hom.comp hom.comp hom.map* ‹*G o F*› *φ.map*
⟨*proof*⟩

lemma *natural-isomorphism-*φ:
shows *natural-isomorphism hom.comp hom.comp hom.map* (*G o F*) *φ.map*
⟨*proof*⟩

definition φ
where $\varphi \equiv \varphi.map$

lemma *φ-ide-simp*:
assumes «*f* : *src* e_0 → *src* e_1» **and** *ide f*
shows $\varphi\ f = \varphi_0\ f$
⟨*proof*⟩

lemma *φ-components-are-iso*:
assumes «*f* : *src* e_0 → *src* e_1» **and** *ide f*
shows *iso* ($\varphi\ f$)
⟨*proof*⟩

lemma *φ-eq*:
shows $\varphi = (\lambda\mu.$ *if* «μ : *src* e_0 → *src* e_1» *then* φ_0 (*cod* μ) · μ *else null*)
⟨*proof*⟩

lemma *φ-in-hom* [*intro*]:
assumes «μ : *src* e_0 → *src* e_1»
shows «$\varphi\ \mu$: *src* e_0 → *src* e_1»
and «$\varphi\ \mu$: *dom* μ ⇒ d_1 ⋆ (e_1 ⋆ *cod* μ ⋆ d_0) ⋆ e_0»
⟨*proof*⟩

lemma *φ-simps* [*simp*]:
assumes «μ : *src* e_0 → *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* μ **and** *cod* ($\varphi\ \mu$) = d_1 ⋆ (e_1 ⋆ *cod* μ ⋆ d_0) ⋆ e_0
⟨*proof*⟩

interpretation d_0: *equivalence-in-bicategory V H* a i *src trg* d_0 e_0 ‹*inv* ε_0› ‹*inv* η_0›
⟨*proof*⟩
interpretation d_1: *equivalence-in-bicategory V H* a i *src trg* d_1 e_1 ‹*inv* ε_1› ‹*inv* η_1›
⟨*proof*⟩
interpretation $d_0 e_0$: *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›

⟨*proof*⟩

interpretation ψ: *inverse-transformation hom'.comp hom'.comp hom'.map* ‹*F o G*› $d_0\, e_0.\varphi$
⟨*proof*⟩

definition ψ
where $\psi \equiv \psi.map$

lemma ψ*-ide-simp*:
assumes «f': *trg* $e_0 \to$ *trg* e_1» **and** *ide* f'
shows $\psi\ f' = \mathrm{r}[f'] \cdot \mathrm{l}[f' \star trg\ e_0] \cdot (\varepsilon_1 \star f' \star \varepsilon_0) \cdot ((e_1 \star d_1) \star \mathrm{a}[f',\ e_0,\ d_0]) \cdot$
$\mathrm{a}^{-1}[e_1,\ d_1,\ (f' \star e_0) \star d_0] \cdot (e_1 \star \mathrm{a}[d_1,\ f' \star e_0,\ d_0])$
⟨*proof*⟩

lemma ψ*-components-are-iso*:
assumes «f' : *trg* $e_0 \to$ *trg* e_1» **and** *ide* f'
shows *iso* $(\psi\ f')$
⟨*proof*⟩

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*⟩

lemma ψ*-in-hom* [*intro*]:
assumes «μ' : *trg* $e_0 \to$ *trg* e_1»
shows «$\psi\ \mu'$: *trg* $e_0 \to$ *trg* e_1»
and «$\psi\ \mu'$: $e_1 \star (d_1 \star dom\ \mu' \star e_0) \star d_0 \Rightarrow cod\ \mu'$»
⟨*proof*⟩

lemma ψ*-simps* [*simp*]:
assumes «μ' : *trg* $e_0 \to$ *trg* e_1»
shows *arr* $(\psi\ \mu')$
and *src* $(\psi\ \mu') =$ *trg* e_0 **and** *trg* $(\psi\ \mu') =$ *trg* e_1
and *dom* $(\psi\ \mu') = e_1 \star (d_1 \star dom\ \mu' \star e_0) \star d_0$ **and** *cod* $(\psi\ \mu') =$ *cod* μ'
⟨*proof*⟩

interpretation *equivalence-of-categories hom'.comp hom.comp F G* φ ψ
⟨*proof*⟩

lemma *induces-equivalence-of-hom-categories*:
shows *equivalence-of-categories hom'.comp hom.comp F G* φ ψ
⟨*proof*⟩

lemma *equivalence-functor-F*:
shows *equivalence-functor hom.comp hom'.comp F*

⟨*proof*⟩

lemma *equivalence-functor-G*:
shows *equivalence-functor hom'.comp hom.comp G*
⟨*proof*⟩

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 ⋆ μ = e ⋆ μ'*
shows *μ = μ'*
⟨*proof*⟩

lemma *equivalence-cancel-right*:
assumes *equivalence-map e*
and *par μ μ'* **and** *src μ = trg e* **and** *μ ⋆ e = μ' ⋆ e*
shows *μ = μ'*
⟨*proof*⟩

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 ⋆ f ≅ e ⋆ f'*
shows *f ≅ f'*
⟨*proof*⟩

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 ⋆ e ≅ f' ⋆ e*
shows *f ≅ f'*
⟨*proof*⟩

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 $D.isomorphic\ (src_D\ (F\ a))\ (F\ a)$
⟨*proof*⟩

lemma *isomorphic-trg*:
assumes $C.obj\ a$
shows $D.isomorphic\ (trg_D\ (F\ a))\ (F\ a)$
⟨*proof*⟩

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 $hseq_D\ (F\ \mu)\ (F\ \nu)$
⟨*proof*⟩

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** $D.cod\ (map_0\ a) = map_0\ a$
⟨*proof*⟩

lemma *preserves-src* [*simp*]:
assumes $C.arr\ \mu$
shows $src_D\ (F\ \mu) = map_0\ (src_C\ \mu)$
⟨*proof*⟩

lemma *preserves-trg* [*simp*]:
assumes $C.arr\ \mu$
shows $trg_D\ (F\ \mu) = map_0\ (trg_C\ \mu)$
⟨*proof*⟩

lemma *preserves-hhom* [*intro*]:
assumes $C.arr\ \mu$
shows $D.in\text{-}hhom\ (F\ \mu)\ (map_0\ (src_C\ \mu))\ (map_0\ (trg_C\ \mu))$
⟨*proof*⟩

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 $FF \equiv \lambda\mu\nu.\ if\ C.VV.arr\ \mu\nu\ then\ (F\ (fst\ \mu\nu),\ F\ (snd\ \mu\nu))\ else\ D.VV.null$

sublocale *FF*: *functor C.VV.comp D.VV.comp FF*
⟨*proof*⟩

lemma *functor-FF*:
shows *functor C.VV.comp D.VV.comp FF*
⟨*proof*⟩

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* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
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* $C.VV.comp\ V_D\ H_D oFF.map\ FoH_C.map\ \Phi$
for $V_C :: 'c\ comp$ (**infixr** $‹\cdot_C›$ *55*)
and $H_C :: 'c\ comp$ (**infixr** $‹\star_C›$ *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ ($‹\mathrm{a}_C[\text{-}, \text{-}, \text{-}]›$)

and $\mathrm{i}_C :: 'c \Rightarrow 'c$ ($\langle \mathrm{i}_C[\text{-}] \rangle$)
and $src_C :: 'c \Rightarrow 'c$
and $trg_C :: 'c \Rightarrow 'c$
and $V_D :: 'd\ comp$ (**infixr** $\langle \cdot_D \rangle$ *55*)
and $H_D :: 'd\ comp$ (**infixr** $\langle \star_D \rangle$ *53*)
and $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ ($\langle \mathrm{a}_D[\text{-}, \text{-}, \text{-}] \rangle$)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ ($\langle \mathrm{i}_D[\text{-}] \rangle$)
and $src_D :: 'd \Rightarrow 'd$
and $trg_D :: 'd \Rightarrow 'd$
and $F :: 'c \Rightarrow 'd$
and $\Phi :: 'c * 'c \Rightarrow 'd +$
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-hom* ($\langle\langle \text{-} : \text{-} \to_C \text{-} \rangle\rangle$)
no-notation *D.in-hom* ($\langle\langle \text{-} : \text{-} \to_D \text{-} \rangle\rangle$)
notation *C.in-hhom* ($\langle\langle \text{-} : \text{-} \to_C \text{-} \rangle\rangle$)
notation *C.in-hom* ($\langle\langle \text{-} : \text{-} \Rightarrow_C \text{-} \rangle\rangle$)
notation *D.in-hhom* ($\langle\langle \text{-} : \text{-} \to_D \text{-} \rangle\rangle$)
notation *D.in-hom* ($\langle\langle \text{-} : \text{-} \Rightarrow_D \text{-} \rangle\rangle$)

notation *C.lunit* ($\langle \mathrm{l}_C[\text{-}] \rangle$)
notation *C.runit* ($\langle \mathrm{r}_C[\text{-}] \rangle$)
notation *C.lunit'* ($\langle \mathrm{l}_C^{-1}[\text{-}] \rangle$)
notation *C.runit'* ($\langle \mathrm{r}_C^{-1}[\text{-}] \rangle$)
notation *C*.a′ ($\langle \mathrm{a}_C^{-1}[\text{-}, \text{-}, \text{-}] \rangle$)
notation *D.lunit* ($\langle \mathrm{l}_D[\text{-}] \rangle$)
notation *D.runit* ($\langle \mathrm{r}_D[\text{-}] \rangle$)
notation *D.lunit'* ($\langle \mathrm{l}_D^{-1}[\text{-}] \rangle$)
notation *D.runit'* ($\langle \mathrm{r}_D^{-1}[\text{-}] \rangle$)
notation *D*.a′ ($\langle \mathrm{a}_D^{-1}[\text{-}, \text{-}, \text{-}] \rangle$)

lemma *weakly-preserves-objects*:
assumes *C.obj a*
shows *D.isomorphic* (*map*$_0$ *a*) (*F a*)
⟨*proof*⟩

lemma *cmp-in-hom* [*intro*]:
assumes *C.ide a* **and** *C.ide b* **and** $src_C\ a = trg_C\ b$
shows «$\Phi\ (a, b) : map_0\ (src_C\ b) \to_D map_0\ (trg_C\ a)$»
and «$\Phi\ (a, b) : F\ a \star_D F\ b \Rightarrow_D F\ (a \star_C b)$»
⟨*proof*⟩

lemma *cmp-simps* [*simp*]:
assumes *C.ide f* **and** *C.ide g* **and** $src_C\ f = trg_C\ g$
shows *D.arr* ($\Phi\ (f, g)$)

and $src_D\ (\Phi\ (f,\ g)) = src_D\ (F\ g)$ **and** $trg_D\ (\Phi\ (f,\ g)) = trg_D\ (F\ f)$
and $D.dom\ (\Phi\ (f,\ g)) = F\ f \star_D F\ g$ **and** $D.cod\ (\Phi\ (f,\ g)) = F\ (f \star_C g)$
⟨*proof*⟩

lemma *cmp-in-hom′*:
assumes $C.arr\ \mu$ **and** $C.arr\ \nu$ **and** $src_C\ \mu = trg_C\ \nu$
shows «$\Phi\ (\mu,\ \nu) : map_0\ (src_C\ \nu) \to_D map_0\ (trg_C\ \mu)$»
and «$\Phi\ (\mu,\ \nu) : F\ (C.dom\ \mu) \star_D F\ (C.dom\ \nu) \Rightarrow_D F\ (C.cod\ \mu \star_C C.cod\ \nu)$»
⟨*proof*⟩

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 $D.cod\ (\Phi\ (\mu,\ \nu)) = F\ (C.cod\ \mu \star_C C.cod\ \nu)$
⟨*proof*⟩

lemma *cmp-components-are-iso* [*simp*]:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $src_C\ f = trg_C\ g$
shows $D.iso\ (\Phi\ (f,\ g))$
⟨*proof*⟩

lemma *weakly-preserves-hcomp*:
assumes $C.ide\ f$ **and** $C.ide\ g$ **and** $src_C\ f = trg_C\ g$
shows $D.isomorphic\ (F\ f \star_D F\ g)\ (F\ (f \star_C g))$
⟨*proof*⟩

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 «$F\ \mathrm{i}_C[a] \cdot_D \Phi\ (a,\ a) : map_0\ a \to_D map_0\ a$»
and «$\mathrm{i}[a] : F\ a \star_D F\ a \Rightarrow_D F\ a$»
⟨*proof*⟩

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** $D.cod\ \mathrm{i}[a] = F\ a$

⟨*proof*⟩

lemma *iso*-i:
assumes *C.obj a*
shows *D.iso* i[*a*]
⟨*proof*⟩

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* ≡ *THE* ψ. «$\psi : map_0\ a \Rightarrow_D F\ a$» ∧ *D.iso* ψ ∧
$\psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$

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 ∃ !ψ. «$\psi : map_0\ a \Rightarrow_D F\ a$» ∧ *D.iso* ψ ∧ $\psi \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}[a] \cdot_D (\psi \star_D \psi)$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *unit-in-hom* [*intro*]:
assumes *C.obj a*
shows «*unit a* : $map_0\ a \to_D map_0\ a$»
and «*unit a* : $map_0\ a \Rightarrow_D F\ a$»
⟨*proof*⟩

lemma *unit-eqI*:
assumes *C.obj a* **and** «μ: $map_0\ a \Rightarrow_D F\ a$» **and** *D.iso* μ
and $\mu \cdot_D \mathrm{i}_D[map_0\ a] = \mathrm{i}\ a \cdot_D (\mu \star_D \mu)$
shows μ = *unit a*
⟨*proof*⟩

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* ≡ *THE* μ. «$\mu : F\ (trg_C\ f) \star_D F\ f \Rightarrow_D F\ f$» ∧
$F\ (trg_C\ f) \star_D \mu = (\mathrm{i}[trg_C\ f] \star_D F\ f) \cdot_D \mathrm{a}_D{}^{-1}[F\ (trg_C\ f),\ F\ (trg_C\ f),\ F\ f]$

lemma *lF-char*:

assumes *C.ide f*
shows «*lF f* : *F* (*trg*$_C$ *f*) ⋆$_D$ *F f* ⇒$_D$ *F f*»
and *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*]
and ∃!*μ*. «*μ* : *F* (*trg*$_C$ *f*) ⋆$_D$ *F f* ⇒$_D$ *F f*» ∧
F (*trg*$_C$ *f*) ⋆$_D$ *μ* = (i[*trg*$_C$ *f*] ⋆$_D$ *F f*) ·$_D$ a$_D{}^{-1}$[*F* (*trg*$_C$ *f*), *F* (*trg*$_C$ *f*), *F f*]
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *lunit-coherence2*:
assumes *C.ide f*
shows *lF f* = *F* l$_C$[*f*] ·$_D$ Φ (*trg*$_C$ *f*, *f*)
⟨*proof*⟩

lemma *lunit-coherence*:
assumes *C.ide f*
shows l$_D$[*F f*] = *F* l$_C$[*f*] ·$_D$ Φ (*trg*$_C$ *f*, *f*) ·$_D$ (*unit* (*trg*$_C$ *f*) ⋆$_D$ *F f*)
⟨*proof*⟩

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*$_B$ a$_B$ i$_B$ *src*$_B$ *trg*$_B$
for *V* :: *'a comp* (**infixr** ‹·› *55*)

and $H_B :: 'a\ comp$ (**infixr** ‹$\star_B$› *53*)
and $\mathrm{a}_B :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (‹a_B[-, -, -]›)
and $\mathrm{i}_B :: 'a \Rightarrow 'a$ (‹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*
⟨*proof*⟩

interpretation *H*: *functor VV.comp V* ‹$\lambda\mu\nu.$ *fst* $\mu\nu \star$ *snd* $\mu\nu$›
⟨*proof*⟩

interpretation *horizontal-composition V H src trg*
⟨*proof*⟩

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* $\mu\nu\tau$)
⟨*proof*⟩

lemma *VVV-ide-char*:
shows *VVV.ide* $\mu\nu\tau \longleftrightarrow$ *B.VVV.ide* (*DN* $\mu\nu\tau$)
⟨*proof*⟩

lemma *VVV-dom-char*:
shows *VVV.dom* $\mu\nu\tau =$ *UP* (*B.VVV.dom* (*DN* $\mu\nu\tau$))
⟨*proof*⟩

lemma *VVV-cod-char*:
shows $VVV.cod\ \mu\nu\tau = UP\ (B.VVV.cod\ (DN\ \mu\nu\tau))$
⟨*proof*⟩

lemma *HoHV-char*:
shows $HoHV\ \mu\nu\tau = B.HoVH\ (DN\ \mu\nu\tau)$
⟨*proof*⟩

lemma *HoVH-char*:
shows $HoVH\ \mu\nu\tau = B.HoHV\ (DN\ \mu\nu\tau)$
⟨*proof*⟩

definition a (‹a[-, -, -]›)
where $\mathrm{a}[\mu, \nu, \tau] \equiv B.\alpha'\ (DN\ (\mu, \nu, \tau))$

interpretation *natural-isomorphism VVV.comp* ‹(·)› *HoHV HoVH*
‹λμντ. a[*fst* μντ, *fst* (*snd* μντ), *snd* (*snd* μντ)]›
⟨*proof*⟩

sublocale *bicategory V H* a i *src trg*
⟨*proof*⟩

proposition *is-bicategory*:
shows *bicategory V H* a i *src trg*
⟨*proof*⟩

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 $\mathrm{a}[f, g, h] = B.\mathrm{a}'\ h\ g\ f$
⟨*proof*⟩

lemma *lunit-ide-simp*:
assumes *B.ide f*
shows *lunit f* = *B.runit f*
⟨*proof*⟩

lemma *runit-ide-simp*:
assumes *B.ide f*
shows *runit f* = *B.lunit f*
⟨*proof*⟩

end

context *pseudofunctor*
begin

interpretation *C'*: *op-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ ⟨*proof*⟩

interpretation D': *op-bicategory* V_D H_D a_D i_D src_D trg_D ⟨*proof*⟩

notation $C'.H$ (**infixr** ‹$\star_C{}^{op}$› 53)
notation $D'.H$ (**infixr** ‹$\star_D{}^{op}$› 53)

interpretation F': *weak-arrow-of-homs* V_C $C'.src$ $C'.trg$ V_D $D'.src$ $D'.trg$ F
⟨*proof*⟩
interpretation $H_D{}'oFF$: *composite-functor* $C'.VV.comp$ $D'.VV.comp$ V_D $F'.FF$
‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_D{}^{op}$ *snd* $\mu\nu$› ⟨*proof*⟩
interpretation $FoH_C{}'$: *composite-functor* $C'.VV.comp$ V_C V_D
‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_C{}^{op}$ *snd* $\mu\nu$› F
⟨*proof*⟩
interpretation Φ': *natural-isomorphism* $C'.VV.comp$ V_D $H_D{}'oFF.map$ $FoH_C{}'.map$
‹λf. Φ (*snd* f, *fst* f)›
⟨*proof*⟩
interpretation F': *pseudofunctor* V_C $C'.H$ $C'.\mathrm{a}$ i_C $C'.src$ $C'.trg$
V_D $D'.H$ $D'.\mathrm{a}$ i_D $D'.src$ $D'.trg$
F ‹λf. Φ (*snd* f, *fst* f)›
⟨*proof*⟩

lemma *induces-pseudofunctor-between-opposites*:
shows *pseudofunctor* $(\cdot_C)$ $(\star_C{}^{op})$ $C'.\mathrm{a}$ i_C $C'.src$ $C'.trg$
$(\cdot_D)$ $(\star_D{}^{op})$ $D'.\mathrm{a}$ i_D $D'.src$ $D'.trg$
F $(\lambda f.\ \Phi\ (snd\ f,\ fst\ f))$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *preserves-adjunction-data*:
assumes *adjunction-data-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
shows *adjunction-data-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$(F\ f)\ (F\ g)\ (D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g))$
⟨*proof*⟩

lemma *preserves-equivalence*:
assumes *equivalence-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
shows *equivalence-in-bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$(F\ f)\ (F\ g)\ (D.inv\ (\Phi\ (g,\ f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f,\ g))$
⟨*proof*⟩

lemma *preserves-equivalence-maps*:
assumes $C.equivalence\text{-}map\ f$
shows $D.equivalence\text{-}map\ (F\ f)$
⟨*proof*⟩

lemma *preserves-equivalent-objects*:
assumes $C.equivalent\text{-}objects\ a\ b$
shows $D.equivalent\text{-}objects\ (map_0\ a)\ (map_0\ b)$
⟨*proof*⟩

lemma *preserves-isomorphic*:
assumes $C.isomorphic\ f\ g$
shows $D.isomorphic\ (F\ f)\ (F\ g)$
⟨*proof*⟩

lemma *preserves-quasi-inverses*:
assumes $C.quasi\text{-}inverses\ f\ g$
shows $D.quasi\text{-}inverses\ (F\ f)\ (F\ g)$
⟨*proof*⟩

lemma *preserves-quasi-inverse*:
assumes $C.equivalence\text{-}map\ f$
shows $D.isomorphic\ (F\ (C.some\text{-}quasi\text{-}inverse\ f))\ (D.some\text{-}quasi\text{-}inverse\ (F\ f))$
⟨*proof*⟩

end

1.10.5 Identity Pseudofunctors

locale *identity-pseudofunctor* =
B: *bicategory* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
for $V_B :: 'b\ comp$ (**infixr** ‹$\cdot_B$› *55*)
and $H_B :: 'b\ comp$ (**infixr** ‹$\star_B$› *53*)
and $\mathrm{a}_B :: 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and $\mathrm{i}_B :: 'b \Rightarrow 'b$ (‹i_B[-]›)
and $src_B :: 'b \Rightarrow 'b$
and $trg_B :: 'b \Rightarrow 'b$
begin

The underlying vertical functor is just the identity functor on the vertical category, which is already available as *B.map*.

abbreviation *map*
where $map \equiv B.map$

interpretation *I*: *weak-arrow-of-homs* $V_B\ src_B\ trg_B\ V_B\ src_B\ trg_B\ map$
⟨*proof*⟩

interpretation *II*: *functor* $B.VV.comp\ B.VV.comp\ I.FF$
⟨*proof*⟩

interpretation $H_B o II$: *composite-functor* $B.VV.comp\ B.VV.comp\ V_B\ I.FF$
‹$\lambda\mu\nu.\ fst\ \mu\nu \star_B snd\ \mu\nu$›
⟨*proof*⟩
interpretation IoH_B: *composite-functor* $B.VV.comp\ V_B\ V_B$ ‹$\lambda\mu\nu.\ fst\ \mu\nu \star_B snd\ \mu\nu$› *map*
⟨*proof*⟩

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* $B.VV.comp\ V_B\ H_B o II.map\ IoH_B.map\ cmp$
⟨*proof*⟩

interpretation *cmp*: *natural-isomorphism* $B.VV.comp\ V_B\ H_B o II.map\ IoH_B.map\ cmp$

⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *unit-char'*:
assumes *B.obj a*
shows *unit a* = *a*
⟨*proof*⟩

end

lemma (**in** *identity-pseudofunctor*) map_0*-simp* [*simp*]:
assumes *B.obj a*
shows map_0 *a* = *a*
⟨*proof*⟩

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*⟩

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)
⟨*proof*⟩

interpretation *functor S.comp V map*
⟨*proof*⟩

interpretation *weak-arrow-of-homs S.comp S.src S.trg V* src_B trg_B *map*
⟨*proof*⟩

interpretation *HoFF*: *composite-functor S.VV.comp B.VV.comp V FF*
‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$›
⟨*proof*⟩
interpretation *FoH*: *composite-functor S.VV.comp S.comp V* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$› *map*
⟨*proof*⟩

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*⟩

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* $\mu\nu$))
⟨*proof*⟩

lemma *iso-cmp*:
assumes *S.VV.ide* $\mu\nu$
shows *B.iso* (*cmp* $\mu\nu$)
⟨*proof*⟩

interpretation Φ_E: *natural-isomorphism S.VV.comp V HoFF.map FoH.map cmp*
⟨*proof*⟩

sublocale *pseudofunctor S.comp S.hcomp S.*a i *S.src S.trg V H* a_B i src_B trg_B *map cmp*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma map_0-*simp* [*simp*]:
assumes *S.obj a*

shows $map_0\ a = a$
⟨*proof*⟩

lemma *unit-char'*:
assumes $S.obj\ a$
shows $unit\ a = a$
⟨*proof*⟩

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 $\mathrm{a}_B :: 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and $\mathrm{i}_B :: 'b \Rightarrow 'b$ (‹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 $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹i_D[-]›)
and $src_D :: 'd \Rightarrow 'd$
and $trg_D :: 'd \Rightarrow 'd$
and $F :: 'b \Rightarrow 'c$
and $\Phi_F :: 'b * 'b \Rightarrow 'c$
and $G :: 'c \Rightarrow 'd$
and $\Phi_G :: 'c * 'c \Rightarrow 'd$
begin

sublocale *composite-functor* $V_B\ V_C\ V_D\ F\ G$ ⟨*proof*⟩

sublocale *weak-arrow-of-homs* $V_B\ src_B\ trg_B\ V_D\ src_D\ trg_D$ ‹$G\ o\ F$›
⟨*proof*⟩

interpretation $H_D o GF\text{-}GF$: *composite-functor* $B.VV.comp$ $D.VV.comp$ V_D FF
$\langle \lambda\mu\nu.\ fst\ \mu\nu \star_D snd\ \mu\nu \rangle$
⟨*proof*⟩
interpretation $GFoH_B$: *composite-functor* $B.VV.comp$ V_B V_D $\langle \lambda\mu\nu.\ fst\ \mu\nu \star_B snd\ \mu\nu \rangle$
$\langle G\ o\ F \rangle$
⟨*proof*⟩

definition *cmp*
where $cmp\ \mu\nu = (if\ B.VV.arr\ \mu\nu\ then$
$G\ (F\ (H_B\ (fst\ \mu\nu)\ (snd\ \mu\nu))) \cdot_D G\ (\Phi_F\ (B.VV.dom\ \mu\nu)) \cdot_D$
$\Phi_G\ (F\ (B.dom\ (fst\ \mu\nu)),\ F\ (B.dom\ (snd\ \mu\nu)))$
$else\ D.null)$

lemma *cmp-in-hom* [*intro*]:
assumes $B.VV.arr\ \mu\nu$
shows «$cmp\ \mu\nu : H_D o GF\text{-}GF.map\ (B.VV.dom\ \mu\nu) \Rightarrow_D GFoH_B.map\ (B.VV.cod\ \mu\nu)$»
⟨*proof*⟩

lemma *cmp-simps* [*simp*]:
assumes $B.VV.arr\ \mu\nu$
shows $D.arr\ (cmp\ \mu\nu)$
and $D.dom\ (cmp\ \mu\nu) = H_D o GF\text{-}GF.map\ (B.VV.dom\ \mu\nu)$
and $D.cod\ (cmp\ \mu\nu) = GFoH_B.map\ (B.VV.cod\ \mu\nu)$
⟨*proof*⟩

interpretation Φ: *natural-transformation*
$B.VV.comp\ V_D\ H_D o GF\text{-}GF.map\ GFoH_B.map\ cmp$
⟨*proof*⟩

interpretation Φ: *natural-isomorphism* $B.VV.comp\ V_D\ H_D o GF\text{-}GF.map\ GFoH_B.map\ cmp$
⟨*proof*⟩

sublocale *pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ \langle G\ o\ F \rangle\ cmp$
⟨*proof*⟩

lemma *is-pseudofunctor*:
shows $pseudofunctor\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (G\ o\ F)\ cmp$
⟨*proof*⟩

lemma *map*$_0$*-simp* [*simp*]:
assumes $B.obj\ a$
shows $map_0\ a = G.map_0\ (F.map_0\ a)$
⟨*proof*⟩

lemma *unit-char*$'$:
assumes $B.obj\ a$
shows $unit\ a = G\ (F.unit\ a) \cdot_D G.unit\ (F.map_0\ a)$
⟨*proof*⟩

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 ⇒ '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 :: *'c ⇒ 'd*
and Φ :: *'c ∗ 'c ⇒ 'd*
and *Arr* :: *'c ⇒ bool*
begin

  abbreviation *map*
  where *map* ≡ $\lambda\mu$. *if* $C'.arr$ μ *then* F μ *else* $D.null$

  abbreviation *cmp*
  where *cmp* ≡ $\lambda\mu\nu$. *if* $C'.VV.arr$ $\mu\nu$ *then* Φ $\mu\nu$ *else* $D.null$

  interpretation *functor* $C'.comp$ V_D *map*
    ⟨*proof*⟩

  interpretation *weak-arrow-of-homs* $C'.comp$ $C'.src$ $C'.trg$ V_D src_D trg_D *map*
    ⟨*proof*⟩

  interpretation H_D‘*oFF*: *composite-functor* $C'.VV.comp$ $D.VV.comp$ V_D *FF*
                  ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_D$ *snd* $\mu\nu$›
    ⟨*proof*⟩
  interpretation *FoH*$_{C'}$: *composite-functor* $C'.VV.comp$ $C'.comp$ V_D
                  ‹$\lambda\mu\nu$. $C'.hcomp$ (*fst* $\mu\nu$) (*snd* $\mu\nu$)› *map*
    ⟨*proof*⟩

  interpretation Φ: *natural-transformation* $C'.VV.comp$ V_D H_D‘*oFF.map* *FoH*$_{C'}$*.map* *cmp*
    ⟨*proof*⟩

interpretation Φ: *natural-isomorphism C′.VV.comp V_D H_D‹oFF.map FoH_C′.map cmp*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *map₀-simp* [*simp*]:
assumes *C′.obj a*
shows *map₀ a* = *F.map₀ a*
⟨*proof*⟩

lemma *unit-char′*:
assumes *C′.obj a*
shows *F.unit a* = *unit a*
⟨*proof*⟩

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* ⇒ *′c*
and *trg_C* :: *′c* ⇒ *′c*
and *V_D* :: *′d comp* (**infixr** ‹·_D› *55*)
and *H_D* :: *′d comp* (**infixr** ‹⋆_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* :: *′c* ⇒ *′d*
and Φ :: *′c* ∗ *′c* ⇒ *′d*
and *Obj* :: *′d* ⇒ *bool* +
assumes *preserves-arr*: ⋀μ. *C.arr* μ ⟹ *D′.arr* (*F* μ)

begin

abbreviation *map*
where $map \equiv F$

abbreviation *cmp*
where $cmp \equiv \Phi$

interpretation *functor* V_C $D'.comp$ F
⟨*proof*⟩

interpretation *weak-arrow-of-homs* V_C src_C trg_C $D'.comp$ $D'.src$ $D'.trg$ F
⟨*proof*⟩

interpretation $H_{D'}oFF$: *composite-functor* $C.VV.comp$ $D'.VV.comp$ $D'.comp$ FF
$\langle\lambda\mu\nu.\ D'.hcomp\ (fst\ \mu\nu)\ (snd\ \mu\nu)\rangle$
⟨*proof*⟩
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
⟨*proof*⟩

interpretation *natural-transformation* $C.VV.comp$ $D'.comp$ $H_{D'}oFF.map$ $FoH_C.map$ Φ
⟨*proof*⟩

interpretation *natural-isomorphism* $C.VV.comp$ $D'.comp$ $H_{D'}oFF.map$ $FoH_C.map$ Φ
⟨*proof*⟩

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*⟩

lemma *is-pseudofunctor*:
shows *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*⟩

lemma map_0*-simp* [*simp*]:
assumes $C.obj\ a$
shows $map_0\ a = F.map_0\ a$
⟨*proof*⟩

lemma *unit-char'*:
assumes $C.obj\ a$
shows $F.unit\ a = unit\ a$
⟨*proof*⟩

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* $\equiv$
$\forall f\ g\ \mu\ \mu'.\ \langle\!\langle \mu : f \Rightarrow_C g \rangle\!\rangle \wedge \langle\!\langle \mu' : f \Rightarrow_C g \rangle\!\rangle \wedge F\ \mu = F\ \mu' \longrightarrow \mu = \mu'$

lemma *locally-faithful-iff-faithful*:
shows *locally-faithful* $\longleftrightarrow$ *faithful-functor* $V_C\ V_D\ F$
⟨*proof*⟩

end

In contrast, it is not true that a pseudofunctor that is locally full is also globally full, because we can have $\langle\!\langle \nu : F\ h \Rightarrow_D F\ k \rangle\!\rangle$ 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*:
$[\![\ C.obj\ a;\ C.obj\ b;\ \langle\!\langle g : map_0\ a \rightarrow_D map_0\ b \rangle\!\rangle;\ D.ide\ g\ ]\!] \Longrightarrow$
$\exists f.\ \langle\!\langle f : a \rightarrow_C b \rangle\!\rangle \wedge C.ide\ f \wedge D.isomorphic\ (F\ f)\ g$
and *locally-full*:
$[\![\ C.ide\ f;\ C.ide\ f';\ src_C\ f = src_C\ f';\ trg_C\ f = trg_C\ f';\ \langle\!\langle \nu : F\ f \Rightarrow_D F\ f' \rangle\!\rangle\ ]\!] \Longrightarrow$
$\exists \mu.\ \langle\!\langle \mu : f \Rightarrow_C f' \rangle\!\rangle \wedge F\ \mu = \nu$
begin

lemma *reflects-ide*:
assumes $C.endo\ \mu$ **and** $D.ide\ (F\ \mu)$
shows $C.ide\ \mu$
⟨*proof*⟩

lemma *reflects-iso*:
assumes $C.arr\ \mu$ **and** $D.iso\ (F\ \mu)$
shows $C.iso\ \mu$

⟨*proof*⟩

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′*
⟨*proof*⟩

lemma *reflects-equivalence*:
assumes *C.ide f* **and** *C.ide g*
and «η : $src_C\ f \Rightarrow_C g \star_C f$» **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* (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*))
shows *equivalence-in-bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ *f g* η ε
⟨*proof*⟩

lemma *reflects-equivalence-map*:
assumes *C.ide f* **and** *D.equivalence-map* (*F f*)
shows *C.equivalence-map f*
⟨*proof*⟩

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*⟩

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 ‹$\lambda\mu$. «μ : *a* $\rightarrow_C$ *a′*»›
⟨*proof*⟩
sublocale $hhom_D$: *subcategory* V_D ‹$\lambda\mu$. «μ : map_0 *a* $\rightarrow_D$ map_0 *a′*»›
⟨*proof*⟩

definition F_1
where F_1 = ($\lambda\mu$. *if* $hhom_C.arr\ \mu$ *then F* μ *else D.null*)

interpretation F_1: *functor* $hhom_C.comp\ hhom_D.comp\ F_1$
⟨*proof*⟩

interpretation F_1: *fully-faithful-and-essentially-surjective-functor*
hhom$_C$.comp hhom$_D$.comp F_1
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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* $G_1\ F_1\ \eta\ \varepsilon$
⟨*proof*⟩

sublocale $\eta\varepsilon$: *adjoint-equivalence hhom$_C$.comp hhom$_D$.comp* $G_1\ F_1\ \eta\ \varepsilon$
⟨*proof*⟩

sublocale $\eta\varepsilon$: *meta-adjunction hhom$_C$.comp hhom$_D$.comp* $G_1\ F_1\ \eta\varepsilon.\varphi\ \eta\varepsilon.\psi$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *is-equivalence-pseudofunctor*:
shows *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
⟨*proof*⟩

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›
⟨*proof*⟩

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*⟩

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* ⟨*proof*⟩

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*
⟨*proof*⟩

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* $\mathrm{i}[a]$
    ⟨*proof*⟩

  lemma *strict-lunit'*:
  assumes *ide f*
  shows $\mathrm{l}^{-1}[f] = f$
    ⟨*proof*⟩

  lemma *strict-runit'*:
  assumes *ide f*
  shows $\mathrm{r}^{-1}[f] = f$
    ⟨*proof*⟩

  lemma *hcomp-obj-arr*:
  assumes *obj b* **and** *arr f* **and** *b* = *trg f*
  shows $b \star f = f$
    ⟨*proof*⟩

  lemma *hcomp-arr-obj*:
  assumes *arr f* **and** *obj a* **and** *src f* = *a*
  shows $f \star a = f$
    ⟨*proof*⟩

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* a^{-1}[*f*, *g*, *h*]
⟨*proof*⟩

lemma *hcomp-assoc*:
shows (*μ* ⋆ *ν*) ⋆ *τ* = *μ* ⋆ *ν* ⋆ *τ*
⟨*proof*⟩

In a strict bicategory, every canonical isomorphism is an identity.

interpretation *bicategorical-language* ⟨*proof*⟩
interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
notation *E.eval* (‹⦇-⦈›)

lemma *ide-eval-Can*:
assumes *Can t*
shows *ide* ⦇*t*⦈
⟨*proof*⟩

lemma *ide-can*:
assumes *Ide f* **and** *Ide g* **and** ⌊*f*⌋ = ⌊*g*⌋
shows *ide* (*can g f*)
⟨*proof*⟩

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*⟩

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 $(\langle dom\ \mu\rangle, \langle cod\ \mu\rangle, \mu)$, 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* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
for V_B :: *'a comp* (**infixr** ‹$\cdot_B$› *55*)
and H_B :: *'a ⇒ 'a ⇒ 'a* (**infixr** ‹$\star_B$› *53*)
and a_B :: *'a ⇒ 'a ⇒ 'a ⇒ 'a* (‹a_B[-, -, -]›)
and i_B :: *'a ⇒ 'a* (‹i_B[-]›)
and src_B :: *'a ⇒ 'a*
and trg_B :: *'a ⇒ 'a*
begin

  sublocale E: *self-evaluation-map* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$ ⟨*proof*⟩

  notation *B.in-hhom* (‹«- : - →$_B$ -»›)
  notation *B.in-hom* (‹«- : - ⇒$_B$ -»›)

  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 $\langle\!\langle \mu : \{\!|A|\!\} \Rightarrow_B \{\!|B|\!\} \rangle\!\rangle$.

abbreviation *HOM*
where $HOM\ A\ B \equiv \{\mu.\ E.Src\ A = E.Src\ B \wedge E.Trg\ A = E.Trg\ B \wedge \langle\!\langle \mu : \{\!|A|\!\} \Rightarrow_B \{\!|B|\!\} \rangle\!\rangle\}$

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* $\langle \lambda\text{- - - } \mu\ \nu.\ \mu \cdot_B \nu \rangle$
⟨*proof*⟩

lemma *is-concrete-category*:
shows *concrete-category IDE HOM EVAL* $(\lambda\text{- - - } \mu\ \nu.\ \mu \cdot_B \nu)$
⟨*proof*⟩

abbreviation *vcomp* (**infixr** $\langle\cdot\rangle$ 55)
where $vcomp \equiv COMP$

lemma *arr-char*:
shows $arr\ F \longleftrightarrow$
$E.Nml\ (Dom\ F) \wedge E.Ide\ (Dom\ F) \wedge E.Nml\ (Cod\ F) \wedge E.Ide\ (Cod\ F) \wedge$
$E.Src\ (Dom\ F) = E.Src\ (Cod\ F) \wedge E.Trg\ (Dom\ F) = E.Trg\ (Cod\ F) \wedge$
$\langle\!\langle Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\} \rangle\!\rangle \wedge F \neq Null$
⟨*proof*⟩

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 $\langle\!\langle Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\} \rangle\!\rangle$ **and** $F \neq Null$
shows $arr\ F$
⟨*proof*⟩

lemma *arrE* [*elim*]:
assumes $arr\ F$
shows $(\llbracket E.Nml\ (Dom\ F); E.Ide\ (Dom\ F); E.Nml\ (Cod\ F); E.Ide\ (Cod\ F);$
$E.Src\ (Dom\ F) = E.Src\ (Cod\ F); E.Trg\ (Dom\ F) = E.Trg\ (Cod\ F);$
$\langle\!\langle Map\ F : \{\!|Dom\ F|\!\} \Rightarrow_B \{\!|Cod\ F|\!\} \rangle\!\rangle; F \neq Null \rrbracket \Longrightarrow T) \Longrightarrow T$
⟨*proof*⟩

lemma *ide-char*:

shows *ide F* $\longleftrightarrow$ *endo F* $\wedge$ *B.ide* (*Map F*)
⟨*proof*⟩

lemma *ideI* [*intro*]:
assumes *arr F* **and** *Dom F* = *Cod F* **and** *B.ide* (*Map F*)
shows *ide F*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)⦈
⟨*proof*⟩

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*)⦈
⟨*proof*⟩

interpretation *src*: *endofunctor vcomp src*
⟨*proof*⟩

interpretation *trg*: *endofunctor vcomp trg*
⟨*proof*⟩

interpretation *horizontal-homs vcomp src trg*
⟨*proof*⟩

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*) ·$_B$

$B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu))$
else null

lemma *arr-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *arr* $(\mu \star \nu)$
⟨*proof*⟩

lemma *src-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *src* $(\mu \star \nu)$ = *src* ν
⟨*proof*⟩

lemma *trg-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *trg* (*hcomp* μ ν) = *trg* μ
⟨*proof*⟩

lemma *hseq-char*:
shows *arr* $(\mu \star \nu) \longleftrightarrow$ *arr* $\mu \wedge$ *arr* $\nu \wedge$ *src* μ = *trg* ν
⟨*proof*⟩

lemma *Dom-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Dom* $(\mu \star \nu)$ = *Dom* $\mu\ \lfloor\star\rfloor$ *Dom* ν
⟨*proof*⟩

lemma *Cod-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Cod* $(\mu \star \nu)$ = *Cod* $\mu\ \lfloor\star\rfloor$ *Cod* ν
⟨*proof*⟩

lemma *Map-hcomp* [*simp*]:
assumes *arr* μ **and** *arr* ν **and** *src* μ = *trg* ν
shows *Map* $(\mu \star \nu)$ = $B.can\ (Cod\ \mu\ \lfloor\star\rfloor\ Cod\ \nu)\ (Cod\ \mu \star Cod\ \nu) \cdot_B$
$(Map\ \mu \star_B Map\ \nu) \cdot_B$
$B.can\ (Dom\ \mu \star Dom\ \nu)\ (Dom\ \mu\ \lfloor\star\rfloor\ Dom\ \nu)$
⟨*proof*⟩

interpretation *functor VV.comp vcomp* ‹$\lambda\mu\nu.$ *hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
⟨*proof*⟩

interpretation *horizontal-composition vcomp hcomp src trg*
⟨*proof*⟩

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*⟩

lemma *obj-char*:
shows *obj a* ⟷ *endo a* ∧ *E.Obj* (*Dom a*) ∧ *Map a* = ⦇*Dom a*⦈
⟨*proof*⟩

lemma *hcomp-obj-self*:
assumes *obj a*
shows *a* ⋆ *a* = *a*
⟨*proof*⟩

lemma *hcomp-ide-src*:
assumes *ide f*
shows *f* ⋆ *src f* = *f*
⟨*proof*⟩

lemma *hcomp-trg-ide*:
assumes *ide f*
shows *trg f* ⋆ *f* = *f*
⟨*proof*⟩

interpretation *L*: *full-functor vcomp vcomp L*
⟨*proof*⟩

interpretation *R*: *full-functor vcomp vcomp R*
⟨*proof*⟩

interpretation *L*: *faithful-functor vcomp vcomp L*
⟨*proof*⟩

interpretation *R*: *faithful-functor vcomp vcomp R*
⟨*proof*⟩

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*
‹λ*τμν*. a (*fst τμν*) (*fst* (*snd τμν*)) (*snd* (*snd τμν*))›
⟨*proof*⟩

definition i
where i ≡ λ*a*. *a*

sublocale *bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

lemma *is-bicategory*:
shows *bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

sublocale *strict-bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

theorem *is-strict-bicategory*:
shows *strict-bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

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*⟩

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* ($UP_0\ a$)
⟨*proof*⟩

lemma UP_0*-in-hom* [*intro*]:
assumes *B.obj a*
shows «$UP_0\ a$: $UP_0\ a \to UP_0\ a$»
and «$UP_0\ a$: $UP_0\ a \Rightarrow UP_0\ a$»
⟨*proof*⟩

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* ($UP_0\ a$) = $UP_0\ a$
⟨*proof*⟩

definition *UP*
where *UP* $\mu \equiv$ *if B.arr* μ *then MkArr* ⟨*B.dom* μ⟩ ⟨*B.cod* μ⟩ μ *else null*

lemma *Dom-UP* [*simp*]:
assumes $B.arr\ \mu$
shows $Dom\ (UP\ \mu) = \langle B.dom\ \mu \rangle$
⟨*proof*⟩

lemma *Cod-UP* [*simp*]:
assumes $B.arr\ \mu$
shows $Cod\ (UP\ \mu) = \langle B.cod\ \mu \rangle$
⟨*proof*⟩

lemma *Map-UP* [*simp*]:
assumes $B.arr\ \mu$
shows $Map\ (UP\ \mu) = \mu$
⟨*proof*⟩

lemma *arr-UP*:
assumes $B.arr\ \mu$
shows $arr\ (UP\ \mu)$
⟨*proof*⟩

lemma *UP-in-hom* [*intro*]:
assumes $B.arr\ \mu$
shows $«UP\ \mu : UP_0\ (src_B\ \mu) \to UP_0\ (trg_B\ \mu)»$
and $«UP\ \mu : UP\ (B.dom\ \mu) \Rightarrow UP\ (B.cod\ \mu)»$
⟨*proof*⟩

lemma *UP-simps* [*simp*]:
assumes $B.arr\ \mu$
shows $arr\ (UP\ \mu)$
and $src\ (UP\ \mu) = UP_0\ (src_B\ \mu)$ **and** $trg\ (UP\ \mu) = UP_0\ (trg_B\ \mu)$
and $dom\ (UP\ \mu) = UP\ (B.dom\ \mu)$ **and** $cod\ (UP\ \mu) = UP\ (B.cod\ \mu)$
⟨*proof*⟩

interpretation *UP*: *functor* V_B *vcomp UP*
⟨*proof*⟩

interpretation *UP*: *weak-arrow-of-homs* V_B src_B trg_B *vcomp src trg UP*
⟨*proof*⟩

interpretation *HoUP-UP*: *composite-functor B.VV.comp VV.comp vcomp*
UP.FF ‹$\lambda\mu\nu.$ *hcomp* (*fst* $\mu\nu$) (*snd* $\mu\nu$)› ⟨*proof*⟩
interpretation *UPoH*: *composite-functor B.VV.comp* V_B *vcomp*
‹$\lambda\mu\nu.$ *fst* $\mu\nu$ $\star_B$ *snd* $\mu\nu$› *UP* ⟨*proof*⟩

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*⟩

abbreviation cmp_{UP}
where $cmp_{UP} \equiv \Phi.map$

lemma cmp_{UP}*-in-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 «$cmp_{UP}\ \mu\nu : UP_0\ (src_B\ (snd\ \mu\nu)) \to UP_0\ (trg_B\ (fst\ \mu\nu))$»
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*⟩

lemma cmp_{UP}*-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 $cod\ (cmp_{UP}\ \mu\nu) = UP\ (B.cod\ (fst\ \mu\nu) \star_B B.cod\ (snd\ \mu\nu))$
⟨*proof*⟩

lemma cmp_{UP}*-ide-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 $Map\ (cmp_{UP}\ fg) = fst\ fg \star_B snd\ fg$
⟨*proof*⟩

interpretation Φ: *natural-isomorphism*
$B.VV.comp\ vcomp\ HoUP\text{-}UP.map\ UPoH.map\ cmp_{UP}$
⟨*proof*⟩

lemma cmp_{UP}*-ide-simp*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $cmp_{UP}\ (f,\ g) = MkArr\ (\langle f\rangle \star \langle g\rangle)\ \langle f \star_B g\rangle\ (f \star_B g)$
⟨*proof*⟩

lemma cmp_{UP}'*-ide-simp*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $inv\ (cmp_{UP}\ (f,\ g)) = MkArr\ \langle f \star_B g\rangle\ (\langle f\rangle \star \langle g\rangle)\ (f \star_B g)$
⟨*proof*⟩

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*⟩

lemma *UP-is-pseudofunctor*:
shows $pseudofunctor\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ vcomp\ hcomp\ \mathrm{a}\ \mathrm{i}\ src\ trg\ UP\ cmp_{UP}\ \langle proof\rangle$

lemma $UP\text{-}map_0\text{-}obj$ [*simp*]:

assumes *B.obj a*
shows *UP.map*$_0$ *a* = *UP*$_0$ *a*
⟨*proof*⟩

interpretation *UP*: *full-functor V*$_B$ *vcomp UP*
⟨*proof*⟩

interpretation *UP*: *faithful-functor V*$_B$ *vcomp UP*
⟨*proof*⟩

interpretation *UP*: *fully-faithful-functor V*$_B$ *vcomp UP* ⟨*proof*⟩

lemma *UP-is-fully-faithful-functor*:
shows *fully-faithful-functor V*$_B$ *vcomp UP*
⟨*proof*⟩

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*⟩

lemma *eval-Dom-ide* [*simp*]:
assumes *ide g*
shows ⦇*Dom g*⦈ = *Map g*
⟨*proof*⟩

lemma *Cod-ide*:
assumes *ide f*
shows *Cod f* = *Dom f*
⟨*proof*⟩

lemma *Map-preserves-objects*:
assumes *obj a*
shows *B.obj* (*Map a*)
⟨*proof*⟩

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*⟩

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$_{UP}$
⟨*proof*⟩

Next, we work out the details of the equivalence pseudofunctor *DN* in the converse direction.

definition *DN*
where $DN\ \mu \equiv if\ arr\ \mu\ then\ Map\ \mu\ else\ B.null$

lemma *DN-in-hom* [*intro*]:
assumes $arr\ \mu$
shows «$DN\ \mu : DN\ (src\ \mu) \rightarrow_B DN\ (trg\ \mu)$»
and «$DN\ \mu : DN\ (dom\ \mu) \Rightarrow_B DN\ (cod\ \mu)$»
⟨*proof*⟩

lemma *DN-simps* [*simp*]:
assumes $arr\ \mu$
shows $B.arr\ (DN\ \mu)$
and $src_B\ (DN\ \mu) = DN\ (src\ \mu)$ **and** $trg_B\ (DN\ \mu) = DN\ (trg\ \mu)$
and $B.dom\ (DN\ \mu) = DN\ (dom\ \mu)$ **and** $B.cod\ (DN\ \mu) = DN\ (cod\ \mu)$
⟨*proof*⟩

interpretation *functor vcomp* V_B *DN*
⟨*proof*⟩

interpretation *DN*: *weak-arrow-of-homs vcomp src trg* V_B src_B trg_B *DN*
⟨*proof*⟩

interpretation *functor VV.comp B.VV.comp DN.FF*
⟨*proof*⟩
interpretation *HoDN-DN*: *composite-functor VV.comp B.VV.comp* V_B
DN.FF ‹$\lambda\mu\nu.\ H_B\ (fst\ \mu\nu)\ (snd\ \mu\nu)$› ⟨*proof*⟩
interpretation *DNoH*: *composite-functor VV.comp vcomp* V_B
‹$\lambda\mu\nu.\ fst\ \mu\nu \star snd\ \mu\nu$› *DN* ⟨*proof*⟩

abbreviation Ψ_o
where $\Psi_o\ fg \equiv B.can\ (Dom\ (fst\ fg)\ \lfloor\star\rfloor\ Dom\ (snd\ fg))\ (Dom\ (fst\ fg) \star Dom\ (snd\ fg))$

abbreviation $\Psi_o{}'$
where $\Psi_o{}'\ fg \equiv B.can\ (Dom\ (fst\ fg) \star Dom\ (snd\ fg))\ (Dom\ (fst\ fg)\ \lfloor\star\rfloor\ Dom\ (snd\ fg))$

lemma Ψ_o*-in-hom*:
assumes *VV.ide fg*
shows «$\Psi_o\ fg : Map\ (fst\ fg) \star_B Map\ (snd\ fg) \Rightarrow_B \{\!|Dom\ (fst\ fg)\ \lfloor\star\rfloor\ Dom\ (snd\ fg)|\!\}$»
and «$\Psi_o{}'\ fg : \{\!|Dom\ (fst\ fg)\ \lfloor\star\rfloor\ Dom\ (snd\ fg)|\!\} \Rightarrow_B Map\ (fst\ fg) \star_B Map\ (snd\ fg)$»
and *B.inverse-arrows* $(\Psi_o\ fg)\ (\Psi_o{}'\ fg)$
⟨*proof*⟩

interpretation Ψ: *transformation-by-components*
VV.comp V_B *HoDN-DN.map DNoH.map* Ψ_o
⟨*proof*⟩

abbreviation cmp_{DN}
where $cmp_{DN} \equiv \Psi.map$

interpretation Ψ: *natural-isomorphism VV.comp* V_B *HoDN-DN.map DNoH.map* cmp_{DN}
⟨*proof*⟩

no-notation *B.in-hom* (‹«- : - →$_B$ -»›)

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* $\mu\nu$)) →$_B$ *DN* (*trg* (*fst* $\mu\nu$))»
and «cmp_{DN} $\mu\nu$: *DN* (*dom* (*fst* $\mu\nu$)) ⋆$_B$ *DN* (*dom* (*snd* $\mu\nu$))
⇒$_B$ *DN* (*cod* (*fst* $\mu\nu$) ⋆ *cod* (*snd* $\mu\nu$))»
⟨*proof*⟩

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* $\mu\nu$)) ⋆$_B$ *DN* (*dom* (*snd* $\mu\nu$))
and *B.cod* (cmp_{DN} $\mu\nu$) = *DN* (*cod* (*fst* $\mu\nu$) ⋆ *cod* (*snd* $\mu\nu$))
⟨*proof*⟩

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*⟩

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}
⟨*proof*⟩

interpretation *faithful-functor vcomp* V_B *DN*
⟨*proof*⟩

no-notation *B.in-hom* (‹«- : - →$_B$ -»›)

lemma *DN-UP*:
assumes *B.arr* μ
shows *DN* (*UP* μ) = μ
⟨*proof*⟩

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*⟩

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}
⟨*proof*⟩

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}
⟨*proof*⟩

lemma *UP-unit-char*:
assumes *B.obj a*
shows *UP.unit a* = *MkArr* ⟨*a*⟩$_0$ ⟨*a*⟩ *a*
⟨*proof*⟩

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*)) ⋆$_D$ *F g*) ⋅$_D$ *D.inv* (Φ (trg_C *g*, *g*))
and *F* $\mathrm{r}_C[g]$ = (*F g* ⋆$_D$ *D.inv* (*unit* (src_C *g*))) ⋅$_D$ *D.inv* (Φ (*g*, src_C *g*))
and *F* (*C.lunit′ g*) = Φ (trg_C *g*, *g*) ⋅$_D$ (*unit* (trg_C *g*) ⋆$_D$ *F g*)
and *F* (*C.runit′ g*) = Φ (*g*, src_C *g*) ⋅$_D$ (*F g* ⋆$_D$ *unit* (src_C *g*))
⟨*proof*⟩

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* $\mathrm{a}_C[f, g, h]$ = 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*))
and *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*⟩

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 $(g \star \varepsilon) \cdot (\eta \star g) = g \Longrightarrow (\varepsilon \star f) \cdot (f \star \eta) = f$
⟨*proof*⟩

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* ⟨*proof*⟩

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.cmp*$_{UP}$
⟨*proof*⟩
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*$_{UP}$
⟨*proof*⟩
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*)) ·$_S$ *S.UP* η ·$_S$ *UP.unit* (*src f*)›
‹*S.inv* (*UP.unit* (*trg f*)) ·$_S$ *S.UP* ε ·$_S$ *S.cmp*$_{UP}$ (*f*, *g*)›
⟨*proof*⟩
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*)) ·$_S$ *S.UP* η ·$_S$ *UP.unit* (*src f*)›
‹*S.inv* (*UP.unit* (*trg f*)) ·$_S$ *S.UP* ε ·$_S$ *S.cmp*$_{UP}$ (*f*, *g*)›
⟨*proof*⟩

lemma *UP-triangle*:
shows *S.UP* $((g \star \varepsilon) \cdot \mathrm{a}[g, f, g] \cdot (\eta \star g)) =$
S.cmp$_{UP}$ (*g*, *src g*) ·$_S$ (*S.UP g* ⋆$_S$ *S.UP* ε ·$_S$ *S.cmp*$_{UP}$ (*f*, *g*)) ·$_S$
(*S.inv* (*S.cmp*$_{UP}$ (*g*, *f*)) ·$_S$ *S.UP* η ⋆$_S$ *S.UP g*) ·$_S$ *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*) ·$_S$ (*S.UP g* ⋆$_S$ *UP.unit* (*src g*))) ·$_S$
(*S.inv* (*UP.unit* (*trg g*)) ⋆$_S$ *S.UP g*) ·$_S$ *S.inv* (*S.cmp*$_{UP}$ (*trg g*, *g*))

and $S.UP\ ((\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta)) =$
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
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 $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g] \Longrightarrow$
$S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (S.UP\ \varepsilon \cdot_S S.cmp_{UP}\ (f,\ g) \star_S S.UP\ f) \cdot_S$
$(S.UP\ f \star_S S.inv\ (S.cmp_{UP}\ (g,\ f)) \cdot_S S.UP\ \eta) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f)) =$
$(S.cmp_{UP}\ (trg\ f,\ f) \cdot_S (UP.unit\ (trg\ f) \star_S S.UP\ f)) \cdot_S$
$(S.UP\ f \star_S S.inv\ (UP.unit\ (src\ f))) \cdot_S S.inv\ (S.cmp_{UP}\ (f,\ src\ f))$
⟨*proof*⟩

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*⟩

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*⟩

lemma *triangle-left-iff-right*:
shows $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = \mathrm{l}^{-1}[f] \cdot \mathrm{r}[f] \longleftrightarrow$
$(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = \mathrm{r}^{-1}[g] \cdot \mathrm{l}[g]$
⟨*proof*⟩

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*⟩

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*⟩

lemma *is-concrete-bicategory*:
shows *concrete-bicategory OBJ HOM ID COMP ID ASSOC*
⟨*proof*⟩

lemma *unit-simp*:
assumes *obj a*
shows i *a* = *a*
⟨*proof*⟩

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*⟩

lemma *is-strict-bicategory*:
shows *strict-bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

sublocale *strict-bicategory vcomp hcomp* a i *src trg*
⟨*proof*⟩

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 «$\varepsilon : f \star g \Rightarrow$ *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 $\mathrm{l}[f] \cdot (\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) \cdot \mathrm{r}^{-1}[f] = f$
    ⟨*proof*⟩

  lemma *triangle-right′*:
  shows $\mathrm{r}[g] \cdot (g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) \cdot \mathrm{l}^{-1}[g] = g$
    ⟨*proof*⟩

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*⟩

end

1.13.1 Adjoint Transpose

context *adjunction-in-bicategory*
begin

  interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
  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 $trnl_\eta\ v \in hom\ (f \star v)\ u \to hom\ v\ (g \star u)$
and $trnl_\varepsilon\ u \in hom\ v\ (g \star u) \to hom\ (f \star v)\ u$
and $\langle\!\langle \mu : f \star v \Rightarrow u \rangle\!\rangle \Longrightarrow trnl_\varepsilon\ u\ (trnl_\eta\ v\ \mu) = \mu$
and $\langle\!\langle \nu : v \Rightarrow g \star u \rangle\!\rangle \Longrightarrow trnl_\eta\ v\ (trnl_\varepsilon\ u\ \nu) = \nu$
and *bij-betw* $(trnl_\eta\ v)\ (hom\ (f \star v)\ u)\ (hom\ v\ (g \star u))$
and *bij-betw* $(trnl_\varepsilon\ u)\ (hom\ v\ (g \star u))\ (hom\ (f \star v)\ u)$
⟨*proof*⟩

lemma $trnl_\varepsilon$*-comp*:
assumes *ide u* **and** *seq* $\mu\ \nu$ **and** *src f* = *trg* μ
shows $trnl_\varepsilon\ u\ (\mu \cdot \nu) = trnl_\varepsilon\ u\ \mu \cdot (f \star \nu)$
⟨*proof*⟩

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 $trnr_\eta\ v \in hom\ (v \star g)\ u \to hom\ v\ (u \star f)$
and $trnr_\varepsilon\ u \in hom\ v\ (u \star f) \to hom\ (v \star g)\ u$
and $\langle\!\langle \mu : v \star g \Rightarrow u \rangle\!\rangle \Longrightarrow trnr_\varepsilon\ u\ (trnr_\eta\ v\ \mu) = \mu$
and $\langle\!\langle \nu : v \Rightarrow u \star f \rangle\!\rangle \Longrightarrow trnr_\eta\ v\ (trnr_\varepsilon\ u\ \nu) = \nu$
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*⟩

lemma $trnr_\eta$*-comp*:
assumes *ide v* **and** *seq* $\mu\ \nu$ **and** *src* μ = *trg f*
shows $trnr_\eta\ v\ (\mu \cdot \nu) = (\mu \star f) \cdot trnr_\eta\ v\ \nu$
⟨*proof*⟩

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 $(\varepsilon \star f) \cdot \mathrm{a}^{-1}[f,\ g,\ f] \cdot (f \star \eta) = f$
⟨*proof*⟩

lemma *triangle-right*:
shows $(g \star \varepsilon) \cdot \mathrm{a}[g,\ f,\ g] \cdot (\eta \star g) = g$
⟨*proof*⟩

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 $trnr_\eta\ v\ \mu = (\mu \star f) \cdot \mathrm{a}^{-1}[v,\ g,\ f] \cdot (v \star \eta)$
⟨*proof*⟩

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 $trnr_\varepsilon\ u\ \nu = (u \star \varepsilon) \cdot \mathrm{a}[u,\ f,\ g] \cdot (\nu \star g)$
⟨*proof*⟩

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 $trnl_\eta\ v\ \mu = (g \star \mu) \cdot \mathrm{a}[g,\ f,\ v] \cdot (\eta \star v)$
⟨*proof*⟩

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 $trnl_\varepsilon\ u\ \nu = (\varepsilon \star u) \cdot \mathrm{a}^{-1}[f,\ g,\ u] \cdot (f \star \nu)$
⟨*proof*⟩

end

locale *adjunction-in-strict-bicategory* =
strict-bicategory +
adjunction-in-normal-bicategory
begin

lemma *triangle-left*:
shows $(\varepsilon \star f) \cdot (f \star \eta) = f$
⟨*proof*⟩

lemma *triangle-right*:
shows $(g \star \varepsilon) \cdot (\eta \star g) = g$
⟨*proof*⟩

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 $trnr_\eta\ v\ \mu = (\mu \star f) \cdot (v \star \eta)$
⟨*proof*⟩

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 $trnr_\varepsilon\ u\ \nu = (u \star \varepsilon) \cdot (\nu \star g)$
⟨*proof*⟩

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 $trnl_\eta\ v\ \mu = (g \star \mu) \cdot (\eta \star v)$
⟨*proof*⟩

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 $trnl_\varepsilon\ u\ \nu = (\varepsilon \star u) \cdot (f \star \nu)$
⟨*proof*⟩

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* ⟨*proof*⟩
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*⟩

lemma *left-adjoint-is-ide*:
assumes *is-left-adjoint f*
shows *ide f*
⟨*proof*⟩

lemma *right-adjoint-is-ide*:
assumes *is-right-adjoint f*
shows *ide f*
⟨*proof*⟩

lemma *adjunction-preserved-by-iso-right*:
assumes *adjunction-in-bicategory V H* a i *src trg f g η ε*
and «*φ* : *g* ⇒ *g*′» **and** *iso φ*
shows *adjunction-in-bicategory V H* a i *src trg f g*′ ((*φ* ⋆ *f*) · *η*) (*ε* · (*f* ⋆ *inv φ*))
⟨*proof*⟩

lemma *adjunction-preserved-by-iso-left*:
assumes *adjunction-in-bicategory V H* a i *src trg f g η ε*
and «*φ* : *f* ⇒ *f*′» **and** *iso φ*
shows *adjunction-in-bicategory V H* a i *src trg f*′ *g* ((*g* ⋆ *φ*) · *η*) (*ε* · (*inv φ* ⋆ *g*))
⟨*proof*⟩

lemma *adjoint-pair-preserved-by-iso*:
assumes *adjoint-pair f g*
and «*φ* : *f* ⇒ *f*′» **and** *iso φ*
and «*ψ* : *g* ⇒ *g*′» **and** *iso ψ*
shows *adjoint-pair f*′ *g*′
⟨*proof*⟩

lemma *left-adjoint-preserved-by-iso*:
assumes *is-left-adjoint f*
and «*φ* : *f* ⇒ *f*′» **and** *iso φ*
shows *is-left-adjoint f*′
⟨*proof*⟩

lemma *right-adjoint-preserved-by-iso*:
assumes *is-right-adjoint g*
and «*φ* : *g* ⇒ *g*′» **and** *iso φ*
shows *is-right-adjoint g*′
⟨*proof*⟩

lemma *left-adjoint-preserved-by-iso*′:

assumes *is-left-adjoint f* **and** $f \cong f'$
shows *is-left-adjoint f'*
⟨*proof*⟩

lemma *right-adjoint-preserved-by-iso'*:
assumes *is-right-adjoint g* **and** $g \cong g'$
shows *is-right-adjoint g'*
⟨*proof*⟩

lemma *obj-self-adjunction*:
assumes *obj a*
shows *adjunction-in-bicategory V H* a i *src trg a a* $\mathrm{l}^{-1}[a]$ r[*a*]
⟨*proof*⟩

lemma *obj-is-self-adjoint*:
assumes *obj a*
shows *adjoint-pair a a* **and** *is-left-adjoint a* **and** *is-right-adjoint a*
⟨*proof*⟩

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* (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*⟩

lemma *preserves-adjoint-pair*:
assumes *C.adjoint-pair f g*
shows *D.adjoint-pair* (*F f*) (*F g*)
⟨*proof*⟩

lemma *preserves-left-adjoint*:
assumes *C.is-left-adjoint f*
shows *D.is-left-adjoint* (*F f*)
⟨*proof*⟩

lemma *preserves-right-adjoint*:
assumes *C.is-right-adjoint g*
shows *D.is-right-adjoint* (*F g*)
⟨*proof*⟩

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 $adjunction\text{-}in\text{-}bicategory\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ f)\ (F\ g)$
$(D.inv\ (\Phi\ (g, f)) \cdot_D F\ \eta \cdot_D unit\ (src_C\ f))$
$(D.inv\ (unit\ (trg_C\ f)) \cdot_D F\ \varepsilon \cdot_D \Phi\ (f, g))$
shows $adjunction\text{-}in\text{-}bicategory\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ f\ g\ \eta\ \varepsilon$
⟨*proof*⟩

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*⟩

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*⟩

lemma *reflects-right-adjoint*:
assumes $C.ide\ g$ **and** $D.is\text{-}right\text{-}adjoint\ (F\ g)$
shows $C.is\text{-}right\text{-}adjoint\ g$
⟨*proof*⟩

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 $src :: 'a \Rightarrow 'a$
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 $\eta \equiv (g \star \sigma \star f) \cdot \zeta$

abbreviation ε
where $\varepsilon \equiv \tau \cdot (h \star \xi \star k)$

interpretation *adjunction-data-in-bicategory* V H a i *src trg* $\langle h \star f \rangle$ $\langle g \star k \rangle$ η ε
⟨*proof*⟩

sublocale *adjunction-in-strict-bicategory* V H a i *src trg* $\langle h \star f \rangle$ $\langle g \star k \rangle$ η ε
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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*⟩

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* ⟨*proof*⟩

notation $S.vcomp$ (**infixr** $\langle \cdot_S \rangle$ 55)

notation *S.hcomp* (**infixr** ‹⋆$_S$› *53*)
notation *S.in-hom* (‹«- : - ⇒$_S$ -»›)
notation *S.in-hhom* (‹«- : - →$_S$ -»›)

interpretation *UP*: *fully-faithful-functor V S.vcomp S.UP*
⟨*proof*⟩
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*$_{UP}$
⟨*proof*⟩

lemma *left-adjoints-compose*:
assumes *is-left-adjoint f* **and** *is-left-adjoint f′* **and** *src f* = *trg f′*
shows *is-left-adjoint* (*f* ⋆ *f′*)
⟨*proof*⟩

lemma *right-adjoints-compose*:
assumes *is-right-adjoint g* **and** *is-right-adjoint g′* **and** *src g′* = *trg g*
shows *is-right-adjoint* (*g′* ⋆ *g*)
⟨*proof*⟩

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** ≡ *SOME g. adjoint-pair f g*

definition *some-unit*
where *some-unit f* ≡ *SOME η*. ∃*ε*. *adjunction-in-bicategory V H* a i *src trg f f** *η ε*

definition *some-counit*
where *some-counit f* ≡
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*)
⟨*proof*⟩

lemma *left-adjoint-extends-to-adjoint-pair*:
assumes *is-left-adjoint f*

shows *adjoint-pair f f**
⟨*proof*⟩

lemma *right-adjoint-in-hom* [*intro*]:
assumes *is-left-adjoint f*
shows «*f** : *trg f* → *src f*»
and «*f** : *f** ⇒ *f**»
⟨*proof*⟩

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**
⟨*proof*⟩

end

locale *map-in-bicategory* =
bicategory + *chosen-right-adjoints* +
fixes *f* :: *'a*
assumes *is-map*: *is-left-adjoint f*
begin

abbreviation *η*
where *η* ≡ *some-unit f*

abbreviation *ε*
where *ε* ≡ *some-counit f*

sublocale *adjunction-in-bicategory V H* a i *src trg f* ‹*f**› *η ε*
⟨*proof*⟩

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*⟩

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 : $g \Rightarrow g'$» **and** *iso* φ
shows *adjoint-equivalence-in-bicategory V H* a i *src trg f g'* (($\varphi \star f$) $\cdot$ η) ($\varepsilon \cdot (f \star inv\ \varphi)$)
⟨*proof*⟩

lemma *adjoint-equivalence-preserved-by-iso-left*:
assumes *adjoint-equivalence-in-bicategory V H* a i *src trg f g η ε*
and «φ : $f \Rightarrow f'$» **and** *iso* φ
shows *adjoint-equivalence-in-bicategory V H* a i *src trg f' g* (($g \star \varphi$) $\cdot$ η) ($\varepsilon \cdot (inv\ \varphi \star g)$)
⟨*proof*⟩

end

context *strict-bicategory*
begin

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

lemma *equivalence-refines-to-adjoint-equivalence*:
assumes *equivalence-map f* **and** «g : $trg\ f \to src\ f$» **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*⟩

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* ⟨*proof*⟩

notation *S.vcomp* (**infixr** ‹$\cdot_S$› *55*)
notation *S.hcomp* (**infixr** ‹$\star_S$› *53*)
notation *S.in-hom* (‹«- : - ⇒$_S$ -»›)
notation *S.in-hhom* (‹«- : - →$_S$ -»›)

interpretation *UP*: *fully-faithful-functor V S.vcomp S.UP*
⟨*proof*⟩

interpretation *UP*: *equivalence-pseudofunctor V H* a i *src trg*
*S.vcomp S.hcomp S.*a *S.*i *S.src S.trg S.UP* $S.cmp_{U\,P}$
⟨*proof*⟩
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* $S.cmp_{U\,P}$
⟨*proof*⟩

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*⟩

lemma *equivalence-map-extends-to-adjoint-equivalence*:
assumes *equivalence-map f*
shows ∃ *g η ε*. *adjoint-equivalence-in-bicategory V H* a i *src trg f g η ε*
⟨*proof*⟩

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* ≅ *g′*
⟨*proof*⟩

end

We now use strictification to extend to arbitrary bicategories.

context *bicategory*
begin

interpretation *S*: *strictified-bicategory V H* a i *src trg* ⟨*proof*⟩

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* $S.cmp_{U\,P}$
⟨*proof*⟩
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* $S.cmp_{U\,P}$

⟨*proof*⟩
interpretation *UP*: *fully-faithful-functor V S.vcomp S.UP*
⟨*proof*⟩

lemma *left-adjoint-determines-right-up-to-iso*:
assumes *adjoint-pair f g* **and** *adjoint-pair f g′*
shows $g \cong g'$
⟨*proof*⟩

lemma *right-adjoint-determines-left-up-to-iso*:
assumes *adjoint-pair f g* **and** *adjoint-pair f′ g*
shows $f \cong f'$
⟨*proof*⟩

end

context *chosen-right-adjoints*
begin

lemma *isomorphic-to-left-adjoint-implies-isomorphic-right-adjoint*:
assumes *is-left-adjoint f* **and** $f \cong h$
shows $f^* \cong h^*$
⟨*proof*⟩

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*⟩

lemma *right-adjoint-to-equivalence-is-equivalence*:
assumes *equivalence-map f* **and** *adjoint-pair f g*
shows *equivalence-map g*
⟨*proof*⟩

lemma *left-adjoint-to-equivalence-is-equivalence*:
assumes *equivalence-map f* **and** *adjoint-pair g f*
shows *equivalence-map g*
⟨*proof*⟩

lemma *quasi-inverses-are-adjoint-pair*:
assumes *quasi-inverses f g*
shows *adjoint-pair f g*
⟨*proof*⟩

lemma *quasi-inverses-isomorphic-right*:
assumes *quasi-inverses f g*
shows *quasi-inverses f g'* $\longleftrightarrow$ $g \cong g'$
⟨*proof*⟩

lemma *quasi-inverses-isomorphic-left*:
assumes *quasi-inverses f g*
shows *quasi-inverses f' g* $\longleftrightarrow$ $f \cong f'$
⟨*proof*⟩

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 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
for $V_C :: 'c\ comp$ (**infixr** ‹$\cdot_C$› *55*)
and $H_C :: 'c\ comp$ (**infixr** ‹$\star_C$› *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹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 $\tau_1 :: 'c \Rightarrow 'd$ +
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 «$\tau_0\ a : \tau_0\ a \Rightarrow_D \tau_0\ a$»
    ⟨*proof*⟩

lemma *map₀-simps* [*simp*]:
assumes *C.obj a*
shows *D.ide* (τ_0 *a*)
and src_D (τ_0 *a*) = *F.map₀ a* **and** trg_D (τ_0 *a*) = *G.map₀ a*
⟨*proof*⟩

lemma *map₁-in-hom* [*intro*]:
assumes *C.ide f*
shows «τ_1 *f* : *F.map₀* (src_C *f*) $\to_D$ *G.map₀* (trg_C *f*)»
and «τ_1 *f* : *G f* $\star_D$ τ_0 (src_C *f*) $\Rightarrow_D$ τ_0 (trg_C *f*) $\star_D$ *F f*»
⟨*proof*⟩

lemma *map₁-simps* [*simp*]:
assumes *C.ide f*
shows *D.arr* (τ_1 *f*)
and src_D (τ_1 *f*) = *F.map₀* (src_C *f*)
and trg_D (τ_1 *f*) = *G.map₀* (trg_C *f*)
and *D.dom* (τ_1 *f*) = *G f* $\star_D$ τ_0 (src_C *f*)
and *D.cod* (τ_1 *f*) = τ_0 (trg_C *f*) $\star_D$ *F f*
⟨*proof*⟩

lemma *inv-naturality*:
assumes *C.arr* μ
shows (*G* μ $\star_D$ τ_0 (src_C μ)) $\cdot_D$ *D.inv* (τ_1 (*C.dom* μ)) =
D.inv (τ_1 (*C.cod* μ)) $\cdot_D$ (τ_0 (trg_C μ) $\star_D$ *F* μ)
⟨*proof*⟩

end

1.14.2 Identity Pseudonatural Transformation

locale *identity-pseudonatural-transformation* =
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
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* :: *'c* ⇒ *'d*

and $\Phi_F :: 'c * 'c \Rightarrow 'd$
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*⟩

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F\ map_0\ map_1$
⟨*proof*⟩

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 $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)

and $\mathrm{i}_D :: 'd \Rightarrow 'd$ ($\langle \mathrm{i}_D[\text{-}] \rangle$)
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 $H :: 'c \Rightarrow 'd$
and $\Phi_H :: 'c * 'c \Rightarrow 'd$
and $\sigma_0 :: 'c \Rightarrow 'd$
and $\sigma_1 :: 'c \Rightarrow 'd$
and $\varrho_0 :: 'c \Rightarrow 'd$
and $\varrho_1 :: 'c \Rightarrow 'd$
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*⟩

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ H\ \Phi_H\ map_0\ map_1$
⟨*proof*⟩

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* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
τ.*F*: *pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$ +
τ.*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** $\langle\cdot_B\rangle$ *55*)
and $H_B :: 'b\ comp$ (**infixr** $\langle\star_B\rangle$ *53*)
and $\mathrm{a}_B :: 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ ($\langle\mathrm{a}_B[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_B :: 'b \Rightarrow 'b$ ($\langle\mathrm{i}_B[\text{-}]\rangle$)
and $src_B :: 'b \Rightarrow 'b$
and $trg_B :: 'b \Rightarrow 'b$
and $V_C :: 'c\ comp$ (**infixr** $\langle\cdot_C\rangle$ *55*)
and $H_C :: 'c\ comp$ (**infixr** $\langle\star_C\rangle$ *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ ($\langle\mathrm{a}_C[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ ($\langle\mathrm{i}_C[\text{-}]\rangle$)
and $src_C :: 'c \Rightarrow 'c$
and $trg_C :: 'c \Rightarrow 'c$
and $V_D :: 'd\ comp$ (**infixr** $\langle\cdot_D\rangle$ *55*)
and $H_D :: 'd\ comp$ (**infixr** $\langle\star_D\rangle$ *53*)
and $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ ($\langle\mathrm{a}_D[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ ($\langle\mathrm{i}_D[\text{-}]\rangle$)
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 $H :: 'b \Rightarrow 'c$
and $\Phi_H :: 'b * 'b \Rightarrow 'c$
and $\tau_0 :: 'c \Rightarrow 'd$
and $\tau_1 :: 'c \Rightarrow 'd$
begin

interpretation *FoH*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ H\ \Phi_H\ F\ \Phi_F$
⟨*proof*⟩
interpretation *GoH*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ H\ \Phi_H\ G\ \Phi_G$
⟨*proof*⟩

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* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$\langle F\ o\ H\rangle\ FoH.cmp\ \langle G\ o\ H\rangle\ GoH.cmp\ map_0\ map_1$
⟨*proof*⟩

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$(F\ o\ H)\ FoH.cmp\ (G\ o\ H)\ GoH.cmp\ map_0\ map_1$
⟨*proof*⟩

end

locale *pseudonatural-transformation-whisker-left* =
  B: *bicategory* V_B H_B a_B i_B src_B trg_B +
  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 +
  $\tau.F$: *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* ⇒ *'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* ⇒ *'d* ⇒ *'d* ⇒ *'d* (‹a_D[-, -, -]›)
and i_D :: *'d* ⇒ *'d* (‹i_D[-]›)
and src_D :: *'d* ⇒ *'d*
and trg_D :: *'d* ⇒ *'d*
and F :: *'b* ⇒ *'c*
and Φ_F :: *'b* ∗ *'b* ⇒ *'c*
and G :: *'b* ⇒ *'c*
and Φ_G :: *'b* ∗ *'b* ⇒ *'c*
and H :: *'c* ⇒ *'d*
and Φ_H :: *'c* ∗ *'c* ⇒ *'d*
and τ_0 :: *'b* ⇒ *'c*
and τ_1 :: *'b* ⇒ *'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
    ⟨*proof*⟩
  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
    ⟨*proof*⟩

  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* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
HoF.map HoF.cmp HoG.map HoG.cmp $map_0\ map_1$
⟨*proof*⟩

lemma *is-pseudonatural-transformation*:
shows *pseudonatural-transformation* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
HoF.map HoF.cmp HoG.map HoG.cmp $map_0\ map_1$
⟨*proof*⟩

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*: $C.obj\ a \Longrightarrow D.equivalence\text{-}map\ (\tau_0\ a)$

Identity Transformations are Pseudonatural Equivalences

sublocale *identity-pseudonatural-transformation* ⊆
pseudonatural-equivalence $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi_F\ F\ \Phi_F\ map_0\ map_1$
⟨*proof*⟩

Composition of Pseudonatural Equivalences

locale *composite-pseudonatural-equivalence* =
composite-pseudonatural-transformation +
σ: *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi_F\ G\ \Phi_G\ \sigma_0\ \sigma_1$ +
ϱ: *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$G\ \Phi_G\ H\ \Phi_H\ \varrho_0\ \varrho_1$
begin

sublocale *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi_F\ H\ \Phi_H\ map_0\ map_1$
⟨*proof*⟩

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$F\ \Phi_F\ H\ \Phi_H\ map_0\ map_1$
⟨*proof*⟩

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
⟨*proof*⟩
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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩
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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (\tau_0\ a)\ (\tau_0'\ a)\ (\eta\ a)\ (\varepsilon\ a)$
⟨*proof*⟩

lemma *map*$_0$*-in-hhom* [*intro*]:
assumes $C.obj\ a$
shows «$\tau_0'\ a : G_0\ a \to_D F_0\ a$»
⟨*proof*⟩

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*⟩

lemma *equivalence-map-map*$_0$ [*simp*]:
assumes $C.obj\ a$
shows $D.equivalence\text{-}map\ (\tau_0'\ a)$
⟨*proof*⟩

lemma *unit-in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\eta\ a : F_0\ a \to_D F_0\ a$»
and «$\eta\ a : F_0\ a \Rightarrow_D \tau_0'\ a \star_D \tau_0\ a$»
⟨*proof*⟩

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*⟩

lemma *iso-unit*:
assumes $C.obj\ a$
shows $D.iso\ (\eta\ a)$
⟨*proof*⟩

lemma *counit-in-hom* [*intro*]:
assumes $C.obj\ a$
shows «$\varepsilon\ a : G_0\ a \to_D G_0\ a$»
and «$\varepsilon\ a : \tau_0\ a \star_D \tau_0'\ a \Rightarrow_D G_0\ a$»
⟨*proof*⟩

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*⟩

lemma *iso-counit*:
assumes *C.obj a*
shows *D.iso* (*ε a*)
⟨*proof*⟩

lemma *quasi-inverts-components*:
assumes *C.obj a*
shows *D.isomorphic* (*τ₀′ a ⋆D τ₀ a*) (*F₀ a*)
and *D.isomorphic* (*τ₀ a ⋆D τ₀′ a*) (*G₀ a*)
and *D.quasi-inverses* (*τ₀ a*) (*τ₀′ a*)
⟨*proof*⟩

definition *map₁*
where *map₁ f* = (*τ₀′* (*trgC f*) *⋆D* r*D*[*G f*]) *·D*
(*τ₀′* (*trgC f*) *⋆D G f ⋆D ε* (*srcC f*)) *·D*
(*τ₀′* (*trgC f*) *⋆D* a*D*[*G f*, *τ₀* (*srcC f*), *τ₀′* (*srcC f*)]) *·D*
a*D*[*τ₀′* (*trgC f*), *G f ⋆D τ₀* (*srcC f*), *τ₀′* (*srcC f*)] *·D*
((*τ₀′* (*trgC f*) *⋆D D.inv* (*τ₁ f*)) *⋆D τ₀′* (*srcC f*)) *·D*
(a*D*[*τ₀′* (*trgC f*), *τ₀* (*trgC f*), *F f*] *⋆D τ₀′* (*srcC f*)) *·D*
((*η* (*trgC f*) *⋆D F f*) *⋆D τ₀′* (*srcC f*)) *·D*
(l*D*⁻¹[*F f*] *⋆D τ₀′* (*srcC f*))

abbreviation (*input*) *τ₁′*
where *τ₁′* ≡ *map₁*

lemma *map₁-in-hom* [*intro*]:
assumes *C.ide f*
shows «*τ₁′ f* : *G₀* (*srcC f*) →*D F₀* (*trgC f*)»
and «*τ₁′ f* : *F f ⋆D τ₀′* (*srcC f*) ⇒*D τ₀′* (*trgC f*) *⋆D G f*»
⟨*proof*⟩

lemma *map₁-simps* [*simp*]:
assumes *C.ide f*
shows *D.arr* (*τ₁′ f*)
and *srcD* (*τ₁′ f*) = *G₀* (*srcC f*) **and** *trgD* (*τ₁′ f*) = *F₀* (*trgC f*)
and *D.dom* (*τ₁′ f*) = *F f ⋆D τ₀′* (*srcC f*) **and** *D.cod* (*τ₁′ f*) = *τ₀′* (*trgC f*) *⋆D G f*
⟨*proof*⟩

lemma *iso-map₁-ide*:
assumes *C.ide f*
shows *D.iso* (*τ₁′ f*)
⟨*proof*⟩

interpretation *EV*: *self-evaluation-map V_D H_D* a*D* i*D src_D trg_D* ⟨*proof*⟩
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*⟩

lemma *is-pseudonatural-equivalence*:
shows *pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ F\ \Phi_F\ \tau_0{}'\ \tau_1{}'$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $\sigma_0\ (trg_C\ \mu) \star_D G\ \mu \star_D \tau_0\ (src_C\ \mu)$ 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*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
⟨*proof*⟩

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 «$\varphi\ f : G\ f \Rightarrow_D \tau_0\ (trg_C\ f) \star_D F\ f \star_D \sigma_0\ (src_C\ f)$»
⟨*proof*⟩

lemma *iso-*φ:
assumes $C.ide\ f$
shows $D.iso\ (\varphi\ f)$
⟨*proof*⟩

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 «$\psi\ f : F\ f \Rightarrow_D \sigma_0\ (trg_C\ f) \star_D G\ f \star_D \tau_0\ (src_C\ f)$»
⟨*proof*⟩

lemma *iso-*ψ:
assumes $C.ide\ f$
shows $D.iso\ (\psi\ f)$
⟨*proof*⟩

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*⟩

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*⟩

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 $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹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*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$
⟨*proof*⟩

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*⟩

lemma *is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
⟨*proof*⟩

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*⟩

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* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
τ: *pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1$ +
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 $V_C :: 'c\ comp$ (**infixr** $\langle\cdot_C\rangle$ *55*)
and $H_C :: 'c\ comp$ (**infixr** $\langle\star_C\rangle$ *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ ($\langle\mathrm{a}_C[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ ($\langle\mathrm{i}_C[\text{-}]\rangle$)
and $src_C :: 'c \Rightarrow 'c$
and $trg_C :: 'c \Rightarrow 'c$
and $V_D :: 'd\ comp$ (**infixr** $\langle\cdot_D\rangle$ *55*)
and $H_D :: 'd\ comp$ (**infixr** $\langle\star_D\rangle$ *53*)
and $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ ($\langle\mathrm{a}_D[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ ($\langle\mathrm{i}_D[\text{-}]\rangle$)
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 $\tau_1 :: 'c \Rightarrow 'd$
and $\tau_0' :: 'c \Rightarrow 'd$
and $\tau_1' :: 'c \Rightarrow 'd$
and $\Gamma :: 'c \Rightarrow 'd$ +
assumes Γ-*in-hom*: $C.obj\ a \Longrightarrow \langle\!\langle \Gamma\ a : \tau_0\ a \Rightarrow_D \tau_0'\ a \rangle\!\rangle$
and *naturality*: $[\![\langle\!\langle f : a \rightarrow_C b \rangle\!\rangle;\ C.ide\ f\ ]\!] \Longrightarrow \tau_1'\ f \cdot_D (G\ f \star_D \Gamma\ a) = (\Gamma\ b \star_D F\ f) \cdot_D \tau_1\ f$

locale *invertible-modification* =
  modification +
  assumes *components-are-iso*: $C.obj\ a \Longrightarrow D.iso\ (\Gamma\ a)$

locale *identity-modification* =
  C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +
  D: *bicategory* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ +
  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 $V_C :: 'c\ comp$ (**infixr** $\langle\cdot_C\rangle$ *55*)
and $H_C :: 'c\ comp$ (**infixr** $\langle\star_C\rangle$ *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ ($\langle \mathrm{a}_C[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ ($\langle \mathrm{i}_C[\text{-}]\rangle$)
and $src_C :: 'c \Rightarrow 'c$
and $trg_C :: 'c \Rightarrow 'c$
and $V_D :: 'd\ comp$ (**infixr** $\langle\cdot_D\rangle$ *55*)
and $H_D :: 'd\ comp$ (**infixr** $\langle\star_D\rangle$ *53*)
and $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ ($\langle \mathrm{a}_D[\text{-}, \text{-}, \text{-}]\rangle$)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ ($\langle \mathrm{i}_D[\text{-}]\rangle$)
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

  abbreviation *map*
  where $map \equiv \tau_0$

  sublocale *invertible-modification*
        $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ \tau_0\ \tau_1\ \tau_0\ \tau_1\ map$
    ⟨*proof*⟩

end

locale *composite-modification* =
  C: *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ +

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 $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹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 $\varrho_0 :: 'c \Rightarrow 'd$
and $\varrho_1 :: 'c \Rightarrow 'd$
and $\sigma_0 :: 'c \Rightarrow 'd$
and $\sigma_1 :: 'c \Rightarrow 'd$
and $\tau_0 :: 'c \Rightarrow 'd$
and $\tau_1 :: 'c \Rightarrow 'd$
and $\Gamma :: 'c \Rightarrow 'd$
and $\Delta :: 'c \Rightarrow 'd$
begin

abbreviation *map*
where *map* $a \equiv \Delta\ a\ \cdot_D\ \Gamma\ a$

sublocale *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 τ_0 τ_1 *map*
⟨*proof*⟩

end

context *converse-pseudonatural-equivalence*
begin

interpretation EV: *self-evaluation-map* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ \langle proof\rangle$
notation $EV.eval\ (\langle\{\!|\text{-}|\!\}\rangle)$

interpretation ι_F: *identity-pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$
$\langle proof\rangle$
interpretation ι_G: *identity-pseudonatural-transformation*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G$
$\langle proof\rangle$

interpretation $\tau'\tau$: *composite-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G\ \Phi_G\ F\ \Phi_F$
$\tau_0\ \tau_1\ map_0\ map_1$
$\langle proof\rangle$
interpretation $\tau\tau'$: *composite-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ F\ \Phi_F\ G\ \Phi_G$
$map_0\ map_1\ \tau_0\ \tau_1$
$\langle proof\rangle$

interpretation η: *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F$
$\iota_F.map_0\ \iota_F.map_1\ \tau'\tau.map_0\ \tau'\tau.map_1\ \eta$
$\langle proof\rangle$

lemma *unit-is-invertible-modification*:
shows *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ F\ \Phi_F$
$\iota_F.map_0\ \iota_F.map_1\ \tau'\tau.map_0\ \tau'\tau.map_1\ \eta$
$\langle proof\rangle$

interpretation ε: *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G\ \Phi_G$
$\tau\tau'.map_0\ \tau\tau'.map_1\ \iota_G.map_0\ \iota_G.map_1\ \varepsilon$
$\langle proof\rangle$

lemma *counit-is-invertible-modification*:
shows *invertible-modification*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G\ \Phi_G$
$\tau\tau'.map_0\ \tau\tau'.map_1\ \iota_G.map_0\ \iota_G.map_1\ \varepsilon$
$\langle proof\rangle$

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 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_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 +
  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 +
  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 Φ_G F Φ_F +
  GF: *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 Φ_G +
  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 +
  η: *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 +
  ε: *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
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 $\eta_0 :: 'd \Rightarrow 'd$
and $\eta_1 :: 'd \Rightarrow 'd$
and $\varepsilon_0 :: 'c \Rightarrow 'c$
and $\varepsilon_1 :: 'c \Rightarrow 'c$
begin

notation *C.isomorphic* (**infix** $\langle\cong_C\rangle$ *50*)
notation *D.isomorphic* (**infix** $\langle\cong_D\rangle$ *50*)

notation *C.iso-in-hom* ($\langle\langle\text{- : - }\cong_C\text{ -}\rangle\rangle$)
notation *D.iso-in-hom* ($\langle\langle\text{- : - }\cong_D\text{ -}\rangle\rangle$)

notation *C.some-quasi-inverse* ($\langle\text{-}^{\sim_C}\rangle$ [*1000*] *1000*)
notation *D.some-quasi-inverse* ($\langle\text{-}^{\sim_D}\rangle$ [*1000*] *1000*)

interpretation η': *converse-pseudonatural-equivalence*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$I_D.map\ I_D.cmp\ GF.map\ GF.cmp\ \eta_0\ \eta_1$
⟨*proof*⟩
interpretation ε': *converse-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$FG.map\ FG.cmp\ I_C.map\ I_C.cmp\ \varepsilon_0\ \varepsilon_1$
⟨*proof*⟩

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*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ GF.map\ GF.cmp$
⟨*proof*⟩

interpretation *FG*: *equivalence-pseudofunctor*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ FG.map\ FG.cmp$
⟨*proof*⟩

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*⟩

interpretation *G*: *faithful-functor* $V_C\ V_D\ G$
⟨*proof*⟩

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 $\Phi\ \psi\ f\ f' \equiv \mathrm{r}_C[f'] \cdot_C$

$(f' \star_C \varepsilon'.counit\ (src_C\ f)) \cdot_C$
$\mathrm{a}_C[f', \varepsilon_0\ (src_C\ f), \varepsilon'.map_0\ (src_C\ f)] \cdot_C$
$(C.inv\ (\varepsilon_1\ f') \star_C \varepsilon'.map_0\ (src_C\ f)) \cdot_C$
$\mathrm{a}_C{}^{-1}[\varepsilon_0\ (trg_C\ f), F\ (G\ f'), \varepsilon'.map_0\ (src_C\ f)] \cdot_C$
$(\varepsilon_0\ (trg_C\ f) \star_C F\ \psi \star_C \varepsilon'.map_0\ (src_C\ f)) \cdot_C$
$(\varepsilon_0\ (trg_C\ f) \star_C C.inv\ (\varepsilon'.map_1\ f)) \cdot_C$
$\mathrm{a}_C[\varepsilon_0\ (trg_C\ f), \varepsilon'.map_0\ (trg_C\ f), f] \cdot_C$
$(C.inv\ (\varepsilon'.counit\ (trg_C\ f)) \star_C f) \cdot_C$
$\mathrm{l}_C{}^{-1}[f]$

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 «$\psi : G\ f \cong_D G\ f'$»
shows «$\Phi\ \psi\ f\ f' : f \cong_C f'$»
⟨*proof*⟩

abbreviation Ψ
where $\Psi\ f\ b\ b' \equiv \mathrm{r}_D[G\ (F\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b))] \cdot_D$
$(G\ (F\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b)) \star_D \eta'.\varepsilon\ b) \cdot_D$
$\mathrm{a}_D[G\ (F\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b)), \eta_0\ b, \eta'.map_0\ b] \cdot_D$
$(D.inv\ (\eta_1\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b)) \star_D \eta'.map_0\ b) \cdot_D$
$(\mathrm{a}_D[\eta_0\ b', \eta'.map_0\ b', G\ f \star_D \eta_0\ b] \star_D \eta'.map_0\ b) \cdot_D$
$\mathrm{a}_D{}^{-1}[\eta_0\ b' \star_D \eta'.map_0\ b', G\ f \star_D \eta_0\ b, \eta'.map_0\ b] \cdot_D$
$((\eta_0\ b' \star_D \eta'.map_0\ b') \star_D \mathrm{a}_D{}^{-1}[G\ f, \eta_0\ b, \eta'.map_0\ b]) \cdot_D$
$(D.inv\ (\eta'.counit\ b') \star_D G\ f \star_D D.inv\ (\eta'.counit\ b)) \cdot_D$
$\mathrm{l}_D{}^{-1}[G\ f \star_D G.map_0\ (F.map_0\ b)] \cdot_D$
$\mathrm{r}_D{}^{-1}[G\ f]$

lemma *F-is-locally-essentially-surjective*:
assumes $D.obj\ b$ **and** $D.obj\ b'$ **and** «$f : F.map_0\ b \to_C F.map_0\ b'$» **and** $C.ide\ f$
shows «$\Phi\ (\Psi\ f\ b\ b')\ f\ (F\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b)) :$
$f \cong_C F\ (\eta'.map_0\ b' \star_D G\ f \star_D \eta_0\ b)$»
⟨*proof*⟩

interpretation *F*: *equivalence-pseudofunctor*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
⟨*proof*⟩

lemma *equivalence-pseudofunctor-left*:
shows *equivalence-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi_F$
⟨*proof*⟩

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 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_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
for F'
and $\Phi_F{}'$
and τ_0
and τ_1

begin

  interpretation GF': *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' $\Phi_F{}'$ G Φ_G
    ⟨*proof*⟩
  interpretation $F'G$: *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 Φ_G F' $\Phi_F{}'$
    ⟨*proof*⟩
  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
    ⟨*proof*⟩
  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 $\tau'.map_0$ $\tau'.map_1$
    ⟨*proof*⟩
  interpretation $G o \tau$: *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
    ⟨*proof*⟩
  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 $G o \tau.map_0$ $G o \tau.map_1$
    ⟨*proof*⟩
  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
                    $F'G.map$ $F'G.cmp$ $E.FG.map$ $E.FG.cmp$ $E.I_C.map$ $E.I_C.cmp$
                    $\tau' o G.map_0$ $\tau' o G.map_1$ ε_0 ε_1
    ⟨*proof*⟩

  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*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F'\ \Phi_F{}'\ G\ \Phi_G$
$unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

lemma *induces-equivalence-of-bicategories*:
shows *equivalence-of-bicategories*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F'\ \Phi_F{}'\ G\ \Phi_G$
$unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

end

locale *equivalence-of-bicategories-and-pseudonatural-equivalence-right* =
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*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G'\ \Phi_G{}'\ \tau_0\ \tau_1$
for G'
and $\Phi_G{}'$
and τ_0
and τ_1
begin

interpretation $G'F$: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F\ G'\ \Phi_G{}'$
⟨*proof*⟩
interpretation FG': *composite-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G'\ \Phi_G{}'\ F\ \Phi_F$
⟨*proof*⟩
interpretation τ': *converse-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ G'\ \Phi_G{}'\ \tau_0\ \tau_1$
⟨*proof*⟩
interpretation τoF: *pseudonatural-equivalence-whisker-right* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$G\ \Phi_G\ G'\ \Phi_G{}'\ F\ \Phi_F\ \tau_0\ \tau_1$
⟨*proof*⟩
interpretation $Fo\tau'$: *pseudonatural-equivalence-whisker-left*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$G'\ \Phi_G{}'\ G\ \Phi_G\ F\ \Phi_F\ \tau'.map_0\ \tau'.map_1$
⟨*proof*⟩
sublocale *unit*: *composite-pseudonatural-equivalence*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$E.I_D.map\ E.I_D.cmp\ E.GF.map\ E.GF.cmp\ G'F.map\ G'F.cmp$

η_0 η_1 $\tau oF.map_0$ $\tau oF.map_1$
⟨*proof*⟩
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
⟨*proof*⟩

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' $\Phi_{G'}$
$unit_0$ $unit_1$ $counit_0$ $counit_1$
⟨*proof*⟩

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' $\Phi_{G'}$
$unit_0$ $unit_1$ $counit_0$ $counit_1$
⟨*proof*⟩

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*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$I_D.map\ I_D.cmp\ E.GF.map\ E.GF.cmp\ \eta_0\ \eta_1$
⟨*proof*⟩
sublocale *unit*: *converse-pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$E.FG.map\ E.FG.cmp\ I_C.map\ I_C.cmp\ \varepsilon_0\ \varepsilon_1$
⟨*proof*⟩

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*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$
$unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

lemma *is-equivalence-of-bicategories*:
shows *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ G\ \Phi_G\ F\ \Phi_F$
$unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

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 a_C i_C src_C trg_C V_D H_D a_D i_D src_D trg_D K Φ_K +
FG: *equivalence-of-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 F Φ_F G Φ_G ϱ_0 ϱ_1 σ_0 σ_1 +
HK: *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 H Φ_H K Φ_K ζ_0 ζ_1 ξ_0 ξ_1
for $V_B :: 'b\ comp$ (**infixr** ‹$\cdot_B$› *55*)
and $H_B :: 'b\ comp$ (**infixr** ‹$\star_B$› *53*)
and $\mathrm{a}_B :: 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (‹a_B[-, -, -]›)
and $\mathrm{i}_B :: 'b \Rightarrow 'b$ (‹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 $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹i_D[-]›)
and $src_D :: 'd \Rightarrow 'd$
and $trg_D :: 'd \Rightarrow 'd$
and $F :: 'c \Rightarrow 'b$
and $\Phi_F :: 'c * 'c \Rightarrow 'b$
and $G :: 'b \Rightarrow 'c$
and $\Phi_G :: 'b * 'b \Rightarrow 'c$
and $H :: 'd \Rightarrow 'c$
and $\Phi_H :: 'd * 'd \Rightarrow 'c$
and $K :: 'c \Rightarrow 'd$
and $\Phi_K :: 'c * 'c \Rightarrow 'd$
and $\varrho_0 :: 'c \Rightarrow 'c$
and $\varrho_1 :: 'c \Rightarrow 'c$
and $\sigma_0 :: 'b \Rightarrow 'b$
and $\sigma_1 :: 'b \Rightarrow 'b$
and $\zeta_0 :: 'd \Rightarrow 'd$
and $\zeta_1 :: 'd \Rightarrow 'd$
and $\xi_0 :: 'c \Rightarrow 'c$
and $\xi_1 :: 'c \Rightarrow 'c$
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* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ H\ \Phi_H\ F\ \Phi_F$
⟨*proof*⟩
sublocale *KG*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G\ K\ \Phi_K$
⟨*proof*⟩
interpretation *IG*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$G\ \Phi_G\ I_C.map\ I_C.cmp$
⟨*proof*⟩
interpretation *IH*: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$H\ \Phi_H\ I_C.map\ I_C.cmp$
⟨*proof*⟩
interpretation *HKG*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$G\ \Phi_G\ HK.FG.map\ HK.FG.cmp$
⟨*proof*⟩
interpretation *GFH*: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$H\ \Phi_H\ FG.GF.map\ FG.GF.cmp$
⟨*proof*⟩
interpretation *KGFH*: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$GFH.map\ GFH.cmp\ K\ \Phi_K$
⟨*proof*⟩
interpretation *FHKG*: *composite-pseudofunctor* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B$
$HKG.map\ HKG.cmp\ F\ \Phi_F$
⟨*proof*⟩
interpretation *ϱoH*: *pseudonatural-equivalence-whisker-right* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$I_C.map\ I_C.cmp\ FG.GF.map\ FG.GF.cmp\ H\ \Phi_H\ \varrho_0\ \varrho_1$
⟨*proof*⟩
interpretation *KoϱoH*: *pseudonatural-equivalence-whisker-left* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$

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*⟩
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
⟨*proof*⟩
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*⟩

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 $Ko\varrho oH.map_0$ $Ko\varrho oH.map_1$
⟨*proof*⟩
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
⟨*proof*⟩

abbreviation *left-map*
where *left-map* ≡ $FH.map$

abbreviation *right-map*
where *right-map* ≡ $KG.map$

abbreviation *left-cmp*
where *left-cmp* ≡ $FH.cmp$

abbreviation *right-cmp*
where *right-cmp* ≡ $KG.cmp$

definition $unit_0$
where $unit_0$ ≡ $unit.map_0$

definition $unit_1$
where $unit_1$ ≡ $unit.map_1$

definition $counit_0$
where $counit_0$ ≡ $counit.map_0$

definition $counit_1$
where $counit_1$ == $counit.map_1$

lemma $unit_0$-*simp*:
assumes $D.obj$ a

shows $unit_0\ a = K\ (\varrho_0\ (H.map_0\ a))\ \star_D\ \zeta_0\ a$
⟨*proof*⟩

lemma $unit_1$*-simp*:
assumes $D.ide\ f$
shows $unit_1\ f = \mathrm{a}_D{}^{-1}[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))),\ \zeta_0\ (trg_D\ f),\ f]\ \cdot_D$
$(K\ (\varrho_0\ (H.map_0\ (trg_D\ f)))\ \star_D\ \zeta_1\ f)\ \cdot_D$
$\mathrm{a}_D[K\ (\varrho_0\ (H.map_0\ (trg_D\ f))),\ K\ (H\ f),\ \zeta_0\ (src_D\ f)]\ \cdot_D$
$(D.inv\ (\Phi_K\ (\varrho_0\ (H.map_0\ (trg_D\ f)),\ H\ f))\ \cdot_D$
$K\ (\varrho_1\ (H\ f))\ \cdot_D$
$\Phi_K\ (G\ (F\ (H\ f)),\ \varrho_0\ (H.map_0\ (src_D\ f)))\ \star_D\ \zeta_0\ (src_D\ f))\ \cdot_D$
$\mathrm{a}_D{}^{-1}[K\ (G\ (F\ (H\ f))),\ K\ (\varrho_0\ (H.map_0\ (src_D\ f))),\ \zeta_0\ (src_D\ f)]$
⟨*proof*⟩

lemma $counit_0$*-simp*:
assumes $B.obj\ a$
shows $counit_0\ a = \sigma_0\ a\ \star_B\ F\ (\xi_0\ (G\text{-}.map_0\ a))$
⟨*proof*⟩

lemma $counit_1$*-simp*:
assumes $B.ide\ f$
shows $counit_1\ f = \mathrm{a}_B{}^{-1}[\sigma_0\ (trg_B\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f))),\ F\ (H\ (K\ (G\ f)))]\ \cdot_B$
$(\sigma_0\ (trg_B\ f)\ \star_B$
$B.inv\ (\Phi_F\ (\xi_0\ (G\text{-}.map_0\ (trg_B\ f)),\ H\ (K\ (G\ f))))\ \cdot_B$
$F\ (\xi_1\ (G\ f))\ \cdot_B$
$\Phi_F\ (G\ f,\ \xi_0\ (G\text{-}.map_0\ (src_B\ f))))\ \cdot_B$
$\mathrm{a}_B[\sigma_0\ (trg_B\ f),\ F\ (G\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))]\ \cdot_B$
$(\sigma_1\ f\ \star_B\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f))))\ \cdot_B$
$\mathrm{a}_B{}^{-1}[f,\ \sigma_0\ (src_B\ f),\ F\ (\xi_0\ (G\text{-}.map_0\ (src_B\ f)))]$
⟨*proof*⟩

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*⟩

lemma *is-equivalence-of-bicategories*:
shows *equivalence-of-bicategories* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
left-map left-cmp right-map right-cmp $unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

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** ‹·› *55*)
notation *hcomp* (**infixr** ‹⋆› *53*)
notation *in-hom* (‹«- : - ⇒ -»›)
notation *in-hhom* (‹«- : - → -»›)
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*
⟨*proof*⟩

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 $B.equivalent\text{-}objects\ (P_0\ a)\ a$
and $B.equivalent\text{-}objects\ a\ a' \Longrightarrow P_0\ a = P_0\ a'$
and $P_0\ (P_0\ a) = P_0\ a$
and $B.obj\ (P_0\ a)$
and $P_0\ a \in Obj$
⟨*proof*⟩

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 $e\ a = (SOME\ e.\ «e : a \rightarrow_B P_0\ a» \wedge$
$(\exists d\ \eta\ \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ e\ d\ \eta\ \varepsilon))$

definition d
where $d\ a = (SOME\ d.\ \exists \eta\ \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ d\ \eta\ \varepsilon)$

definition η
where $\eta\ a = (SOME\ \eta.\ \exists \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ (d\ a)\ \eta\ \varepsilon)$

definition ε
where $\varepsilon\ a = (SOME\ \varepsilon.\ adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ (d\ a)\ (\eta\ a)\ \varepsilon)$

lemma *chosen-adjoint-equivalence*:
assumes $B.obj\ a$
shows $adjoint\text{-}equivalence\text{-}in\text{-}bicategory$
$V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ (e\ a)\ (d\ a)\ (\eta\ a)\ (\varepsilon\ a)$
and $«e\ a : a \rightarrow_B P_0\ a»$ **and** $B.ide\ (d\ a)$ **and** $B.ide\ (e\ a)$ **and** $B.iso\ (\eta\ a)$ **and** $B.iso\ (\varepsilon\ a)$
⟨*proof*⟩

lemma $equivalence\text{-}data\text{-}in\text{-}hom_B$ [*intro*]:
assumes $B.obj\ a$
shows $«e\ a : a \rightarrow_B P_0\ a»$ **and** $«d\ a : P_0\ a \rightarrow_B a»$
and $«e\ a : e\ a \Rightarrow_B e\ a»$ **and** $«d\ a : d\ a \Rightarrow_B d\ a»$
and $«\eta\ a : a \rightarrow_B a»$ **and** $«\varepsilon\ a : P_0\ a \rightarrow_B P_0\ a»$
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*⟩

lemma $equivalence\text{-}data\text{-}simps_B$ [*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** $B.cod\ (\varepsilon\ a) = P_0\ a$
⟨*proof*⟩

lemma *equivalence-data-in-hom* [*intro*]:
assumes *obj a*
shows «$e\ a : a \to P_0\ a$» **and** «$d\ a : P_0\ a \to a$»
and «$e\ a : e\ a \Rightarrow e\ a$» **and** «$d\ a : d\ a \Rightarrow d\ a$»
and «$\eta\ a : a \to a$» **and** «$\varepsilon\ a : P_0\ a \to P_0\ a$»
and «$\eta\ a : a \Rightarrow d\ a \star e\ a$» **and** «$\varepsilon\ a : e\ a \star d\ a \Rightarrow P_0\ a$»
⟨*proof*⟩

lemma *equivalence-data-simps* [*simp*]:
assumes *obj a*
shows *ide* $(d\ a)$ **and** *ide* $(e\ a)$ **and** *iso* $(\eta\ a)$ **and** *iso* $(\varepsilon\ a)$
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 $src\ (\eta\ a) = a$ **and** $trg\ (\eta\ a) = a$ **and** $src\ (\varepsilon\ a) = P_0\ a$ **and** $trg\ (\varepsilon\ a) = P_0\ a$
and $dom\ (\eta\ a) = a$ **and** $cod\ (\eta\ a) = d\ a \star_B\ e\ a$
and $dom\ (\varepsilon\ a) = e\ a \star_B\ d\ a$ **and** $cod\ (\varepsilon\ a) = P_0\ a$

⟨*proof*⟩

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 $B.arr\ \mu$
shows «$P\ \mu : P_0\ (src_B\ \mu) \to_B P_0\ (trg_B\ \mu)$»
and «$P\ \mu : P\ (B.dom\ \mu) \Rightarrow_B P\ (B.cod\ \mu)$»
⟨*proof*⟩

lemma *P-simps*$_B$ [*simp*]:
assumes $B.arr\ \mu$
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** $B.cod\ (P\ \mu) = P\ (B.cod\ \mu)$
⟨*proof*⟩

lemma *P-in-hom* [*intro*]:
assumes $B.arr\ \mu$
shows «$P\ \mu : P_0\ (src_B\ \mu) \to P_0\ (trg_B\ \mu)$»
and «$P\ \mu : P\ (B.dom\ \mu) \Rightarrow P\ (B.cod\ \mu)$»
⟨*proof*⟩

lemma *P-simps* [*simp*]:
assumes $B.arr\ \mu$
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** $cod\ (P\ \mu) = P\ (B.cod\ \mu)$
⟨*proof*⟩

interpretation P: *functor* V_B *comp* P
⟨*proof*⟩

interpretation *faithful-functor* V_B *comp* P
⟨*proof*⟩

interpretation P: *weak-arrow-of-homs* V_B src_B trg_B *comp src trg* P
⟨*proof*⟩

The following seems to be needed to avoid non-confluent simplifications, *e.g.* of $S.src\ (P\ \mu)$ to $P.map_0\ a$ and to $P_0\ a$.

lemma *P-map₀-simp* [*simp*]:
assumes *B.obj a*
shows $P.map_0\ a = P_0\ a$
⟨*proof*⟩

interpretation *HoPP*: *composite-functor B.VV.comp VV.comp comp*
P.FF ‹*λμν. hcomp* (*fst μν*) (*snd μν*)›
⟨*proof*⟩
interpretation *PoH*: *composite-functor B.VV.comp* V_B *comp* ‹(*λμν. fst μν ⋆B snd μν*)› *P*
⟨*proof*⟩

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

definition *CMP*
where $CMP\ f\ g \equiv (e\ (trg_B\ f) \star_B \mathrm{a}_B{}^{-1}[f,\ g,\ d\ (src_B\ g)])\ \cdot_B$
$(e\ (trg_B\ f) \star_B f \star_B$
$\mathrm{l}_B[g \star_B d\ (src_B\ g)]\ \cdot_B\ (B.inv\ (\eta\ (trg_B\ g)) \star_B g \star_B d\ (src_B\ g)))\ \cdot_B$
$(e\ (trg_B\ f) \star_B f \star_B \mathrm{a}_B{}^{-1}[d\ (src_B\ f),\ e\ (trg_B\ g),\ g \star_B d\ (src_B\ g)])\ \cdot_B$
$\mathrm{a}_B[e\ (trg_B\ f),\ f,\ d\ (src_B\ f) \star_B P\ g]\ \cdot_B$
$\mathrm{a}_B[e\ (trg_B\ f) \star_B f,\ d\ (src_B\ f),\ P\ g]\ \cdot_B$
$(\mathrm{a}_B{}^{-1}[e\ (trg_B\ f),\ f,\ d\ (src_B\ f)] \star_B P\ g)$

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 «$CMP\ f\ g : P_0\ (src_B\ g) \to P_0\ (trg_B\ f)$»
and «$CMP\ f\ g : P\ f \star P\ g \Rightarrow P\ (f \star_B g)$»
and «$CMP\ f\ g : P_0\ (src_B\ g) \to_B P_0\ (trg_B\ f)$»
and «$CMP\ f\ g : P\ f \star_B P\ g \Rightarrow_B P\ (f \star_B g)$»
⟨*proof*⟩

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* ⋆ *P g* **and** *cod* (*CMP f g*) = *P* (*f* ⋆$_B$ *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* ⋆$_B$ *P g* **and** *B.cod* (*CMP f g*) = *P* (*f* ⋆$_B$ *g*)
⟨*proof*⟩

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*⟩

abbreviation (*input*) *SRC*
where *SRC* $\mu \equiv d$ (*src*$_B$ μ) ⋆$_B$ *e* (*src*$_B$ μ)

abbreviation (*input*) *TRG*
where *TRG* $\mu \equiv d$ (*trg*$_B$ μ) ⋆$_B$ *e* (*trg*$_B$ μ)

definition *LUNIT*
where *LUNIT f* ≡ $\mathrm{l}_B[f]$ ⋅$_B$ (*B.inv* (*η* (*trg*$_B$ *f*)) ⋆$_B$ *f*)

definition *RUNIT*
where *RUNIT f* ≡ $\mathrm{r}_B[f]$ ⋅$_B$ (*f* ⋆$_B$ *B.inv* (*η* (*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* →$_B$ *trg*$_B$ *f*»
and «*LUNIT f* : *TRG f* ⋆$_B$ *f* ⇒$_B$ *f*»
⟨*proof*⟩

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* ⋆$_B$ *f*
and *B.cod* (*LUNIT f*) = *f*
⟨*proof*⟩

lemma *RUNIT-in-hom* [*intro*]:
assumes *B.ide f*
shows «*RUNIT f* : *src*$_B$ *f* →$_B$ *trg*$_B$ *f*»
and «*RUNIT f* : *f* ⋆$_B$ *SRC f* ⇒$_B$ *f*»
⟨*proof*⟩

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 $B.cod\ (RUNIT\ f) = f$
⟨*proof*⟩

lemma *iso-LUNIT*:
assumes $B.ide\ f$
shows $B.iso\ (LUNIT\ f)$
⟨*proof*⟩

lemma *iso-RUNIT*:
assumes $B.ide\ f$
shows $B.iso\ (RUNIT\ f)$
⟨*proof*⟩

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*⟩

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*⟩

lemma *LUNIT-hcomp*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $LUNIT\ (f \star_B g) \cdot_B \mathrm{a}_B[d\ (trg_B\ f) \star_B e\ (trg_B\ f), f, g] = LUNIT\ f \star_B g$
⟨*proof*⟩

lemma *RUNIT-hcomp*:
assumes $B.ide\ f$ **and** $B.ide\ g$ **and** $src_B\ f = trg_B\ g$
shows $RUNIT\ (f \star_B g) = (f \star_B RUNIT\ g) \cdot_B \mathrm{a}_B[f, g, SRC\ g]$
⟨*proof*⟩

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*⟩

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 $CMP\ (B.cod\ \mu)\ (B.cod\ \nu) \cdot_B (P\ \mu \star_B P\ \nu)$
$\quad = P\ (\mu \star_B \nu) \cdot_B CMP\ (B.dom\ \mu)\ (B.dom\ \nu)$
⟨*proof*⟩

interpretation *EV*: *self-evaluation-map* V_B H_B a_B i_B src_B trg_B ⟨*proof*⟩
notation *EV.eval* (‹⦇-⦈›)

abbreviation (*input*) *SRCt* (‹**SRC**›)
where **SRC** $\mu \equiv \langle d\ (src_B\ \mu)\rangle \star \langle e\ (src_B\ \mu)\rangle$

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 $P\ \mathrm{a}_B[f,\ g,\ h] \cdot_B CMP\ (f \star_B g)\ h \cdot_B (CMP\ f\ g \star_B P\ h)$
$\quad = CMP\ f\ (g \star_B h) \cdot_B (P\ f \star_B CMP\ g\ h) \cdot_B \mathrm{a}_B[P\ f,\ P\ g,\ P\ h]$
⟨*proof*⟩

interpretation *CMP*: *transformation-by-components B.VV.comp comp HoPP.map PoH.map*
‹$\lambda\mu\nu.$ *CMP* (*fst* $\mu\nu$) (*snd* $\mu\nu$)›
⟨*proof*⟩

interpretation *CMP*: *natural-isomorphism B.VV.comp comp HoPP.map PoH.map CMP.map*
⟨*proof*⟩

definition Φ_P
where $\Phi_P = CMP.map$

interpretation Φ_P: *natural-isomorphism B.VV.comp comp HoPP.map PoH.map* Φ_P
⟨*proof*⟩

no-notation *B.in-hom* (‹«- : - $\to_B$ -»›)

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*) → *trg* (*P f*)»
and «$\Phi_P\ (f,\ g)$: *P f* ⋆ *P g* ⇒ *P* (*f* $\star_B$ *g*)»
⟨*proof*⟩

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* ⋆ *P g*
and *cod* ($\Phi_P\ (f,\ g)$) = *P* (*f* $\star_B$ *g*)
⟨*proof*⟩

sublocale *prj*: *pseudofunctor* V_B H_B a_B i_B src_B trg_B *comp hcomp* a i_B *src trg* P Φ_P
⟨*proof*⟩

lemma *pseudofunctor-prj*:
shows *pseudofunctor* V_B H_B a_B i_B src_B trg_B *comp hcomp* a i_B *src trg* P Φ_P
⟨*proof*⟩

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* (*ε a*)
⟨*proof*⟩

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 Φ_E *P* Φ_P
⟨*proof*⟩
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 Φ_P *E* Φ_E
⟨*proof*⟩
sublocale I_C: *identity-pseudofunctor* V_B H_B a_B i_B src_B trg_B ⟨*proof*⟩
sublocale I_S: *identity-pseudofunctor comp hcomp* a i_B *src trg* ⟨*proof*⟩

no-notation *B.in-hom* (‹«- : - →$_B$ -»›)

abbreviation (*input*) $unit_0$
where $unit_0 \equiv e$

definition $unit_1$
where $unit_1\ f$ = (*e* (*trg*$_B$ *f*) $\star_B$ r_B[*f*]) $\cdot_B$
a_B[*e* (*trg*$_B$ *f*), *f*, *src*$_B$ *f*] $\cdot_B$
((*e* (*trg*$_B$ *f*) $\star_B$ *f*) $\star_B$ *B.inv* (*η* (*src*$_B$ *f*))) $\cdot_B$
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 «η_1 *f* : *src*$_B$ *f* →$_B$ *P*$_0$ (*trg*$_B$ *f*)»
and «η_1 *f* : (*e* (*trg*$_B$ *f*) $\star_B$ *f* $\star_B$ *d* (*src*$_B$ *f*)) $\star_B$ *e* (*src*$_B$ *f*) ⇒$_B$ *e* (*trg*$_B$ *f*) $\star_B$ *f*»
⟨*proof*⟩

lemma *unit*$_1$*-simps* [*simp*]:
assumes *B.ide f*
shows *B.arr* (η_1 *f*)
and src_B (η_1 *f*) = src_B *f* **and** trg_B (η_1 *f*) = P_0 (trg_B *f*)
and *B.dom* (η_1 *f*) = (*e* (trg_B *f*) $\star_B$ *f* $\star_B$ *d* (src_B *f*)) $\star_B$ *e* (src_B *f*)
and *B.cod* (η_1 *f*) = *e* (trg_B *f*) $\star_B$ *f*
and *B.iso* (η_1 *f*)
⟨*proof*⟩

lemma *unit*$_1$*-in-hom*$_S$ [*intro*]:
assumes *ide f*
shows «η_1 *f* : *src f* → P_0 (*trg f*)»
and «η_1 *f* : *PoE.map f* ⋆ *e* (*src f*) ⇒ *e* (*trg f*) ⋆ I_S*.map f*»
⟨*proof*⟩

lemma *unit*$_1$*-simps*$_S$ [*simp*]:
assumes *ide f*
shows *arr* (η_1 *f*)
and *src* (η_1 *f*) = *src f* **and** *trg* (η_1 *f*) = P_0 (*trg f*)
and *dom* (η_1 *f*) = *PoE.map f* ⋆ *e* (*src f*)
and *cod* (η_1 *f*) = *e* (*trg f*) ⋆ I_S*.map f*
and *iso* (η_1 *f*)
⟨*proof*⟩

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* ∘ *E*› *PoE.cmp* $unit_0$ $unit_1$
⟨*proof*⟩

abbreviation (*input*) $counit_0$
where $counit_0$ ≡ *d*

definition $counit_1$
where $counit_1$ *f* = a_B[*d* (trg_B *f*), *e* (trg_B *f*), *f* $\star_B$ *d* (src_B *f*)] $\cdot_B$
(η (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 ε_0 ≡ $counit_0$

abbreviation (*input*) ε_1
where ε_1 ≡ $counit_1$

lemma *counit*$_1$*-in-hom* [*intro*]:
assumes *B.ide f*
shows «ε_1 *f* : *f* $\star_B$ *d* (src_B *f*) ⇒$_B$ *d* (trg_B *f*) $\star_B$ *e* (trg_B *f*) $\star_B$ *f* $\star_B$ *d* (src_B *f*)»
⟨*proof*⟩

lemma *counit*$_1$*-simps* [*simp*]:
assumes *B.ide f*
shows *B.arr* (ε_1 *f*)
and src_B (ε_1 *f*) = P_0 (src_B *f*) **and** trg_B (ε_1 *f*) = trg_B *f*
and *B.dom* (ε_1 *f*) = *f* $\star_B$ *d* (src_B *f*)
and *B.cod* (ε_1 *f*) = *d* (trg_B *f*) $\star_B$ *e* (trg_B *f*) $\star_B$ *f* $\star_B$ *d* (src_B *f*)
and *B.iso* (ε_1 *f*)
⟨*proof*⟩

lemma *technical*:
assumes *B.ide f* **and** *B.ide g* **and** src_B *g* = trg_B *f*
shows (ε_1 *g* $\star_B$ *P f*) $\cdot_B$ $\mathrm{a}_B{}^{-1}$[*g*, *d* (src_B *g*), *P f*] $\cdot_B$ (*g* $\star_B$ ε_1 *f*)
= (a_B[*d* (trg_B *g*), *e* (trg_B *g*), *g* $\star_B$ *d* (src_B *g*)] $\star_B$ *P f*) $\cdot_B$
(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*)
$\star_B$ a_B[*d* (src_B *g*), *e* (src_B *g*), *f* $\star_B$ *d* (src_B *f*)]) $\cdot_B$
((*η* (trg_B *g*) $\star_B$ *g*) $\cdot_B$ $\mathrm{l}_B{}^{-1}$[*g*] $\star_B$ (*η* (src_B *g*) $\star_B$ *f* $\star_B$ *d* (src_B *f*))) $\cdot_B$
(*g* $\star_B$ a_B[src_B *g*, *f*, *d* (src_B *f*)] $\cdot_B$ ($\mathrm{l}_B{}^{-1}$[*f*] $\star_B$ *d* (src_B *f*)))
⟨*proof*⟩

sublocale *counit*: *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
‹*E* ∘ *P*› *EoP.cmp* I_C*.map* I_C*.cmp* *counit*$_0$ *counit*$_1$
⟨*proof*⟩

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*$_1$
⟨*proof*⟩

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*$_1$
⟨*proof*⟩

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 $G_0 = \mathit{inv\text{-}into}\ (\mathit{Collect}\ C.\mathit{obj})\ F_0$

lemma G_0*-props*:
shows $D.\mathit{obj}\ b \Longrightarrow C.\mathit{obj}\ (G_0\ b) \wedge F_0\ (G_0\ b) = b$
and $C.\mathit{obj}\ a \Longrightarrow D.\mathit{obj}\ (F_0\ a) \wedge G_0\ (F_0\ a) = a$
⟨*proof*⟩

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 $G\ \nu = (\mathit{if}\ D.\mathit{arr}\ \nu\ \mathit{then}$
$\mathit{equivalence\text{-}pseudofunctor\text{-}at\text{-}hom.G_1}\ V_C\ \mathit{src}_C\ \mathit{trg}_C\ V_D\ \mathit{src}_D\ \mathit{trg}_D\ F$
$(G_0\ (\mathit{src}_D\ \nu))\ (G_0\ (\mathit{trg}_D\ \nu))\ \nu$
$\mathit{else}\ C.\mathit{null})$

lemma *G-in-hom* [*intro*]:
assumes $D.\mathit{arr}\ \nu$
shows «$G\ \nu : G_0\ (\mathit{src}_D\ \nu) \rightarrow_C G_0\ (\mathit{trg}_D\ \nu)$»
and «$G\ \nu : G\ (D.\mathit{dom}\ \nu) \Rightarrow_C G\ (D.\mathit{cod}\ \nu)$»
⟨*proof*⟩

lemma *G-simps* [*simp*]:
assumes $D.\mathit{arr}\ \nu$
shows $C.\mathit{arr}\ (G\ \nu)$
and $\mathit{src}_C\ (G\ \nu) = G_0\ (\mathit{src}_D\ \nu)$ **and** $\mathit{trg}_C\ (G\ \nu) = G_0\ (\mathit{trg}_D\ \nu)$
and $C.\mathit{dom}\ (G\ \nu) = G\ (D.\mathit{dom}\ \nu)$ **and** $C.\mathit{cod}\ (G\ \nu) = G\ (D.\mathit{cod}\ \nu)$
⟨*proof*⟩

interpretation *G*: *functor* $V_D\ V_C\ G$
⟨*proof*⟩

lemma *functor-G*:
shows *functor* $V_D\ V_C\ G$
⟨*proof*⟩

interpretation *G*: *faithful-functor* $V_D\ V_C\ G$
⟨*proof*⟩

interpretation *FG*: *composite-functor* $V_D\ V_C\ V_D\ G\ F$ ⟨*proof*⟩
interpretation *FG*: *faithful-functor* $V_D\ V_D\ F\ o\ G$
⟨*proof*⟩

interpretation *GF*: *composite-functor* V_C V_D V_C F G ⟨*proof*⟩
interpretation *GF*: *faithful-functor* V_C V_C G *o* F
⟨*proof*⟩

interpretation *G*: *weak-arrow-of-homs* V_D src_D trg_D V_C src_C trg_C G
⟨*proof*⟩

lemma *weak-arrow-of-homs-G*:
shows *weak-arrow-of-homs* V_D src_D trg_D V_C src_C trg_C G
⟨*proof*⟩

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* $\mu\nu$›
⟨*proof*⟩
sublocale GoH_D: *composite-functor D.VV.comp* V_D V_C ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star_D$ *snd* $\mu\nu$› G
⟨*proof*⟩

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* ν *then*
equivalence-pseudofunctor-at-hom.η V_C src_C trg_C V_D src_D trg_D F
(G_0 (src_D ν)) (G_0 (trg_D ν)) ν
else D.null)

lemma *η-in-hom*:
assumes *D.arr* ν
shows [*intro*]: «$\eta\ \nu$: src_D ν $\rightarrow_D$ trg_D ν»
and [*intro*]: «$\eta\ \nu$: *D.dom* ν $\Rightarrow_D$ F (G (*D.cod* ν))»
and *D.ide* ν $\Longrightarrow$ *D.iso* ($\eta\ \nu$)
⟨*proof*⟩

lemma *η-simps* [*simp*]:
assumes *D.arr* ν
shows *D.arr* ($\eta\ \nu$)
and src_D ($\eta\ \nu$) = src_D ν **and** trg_D ($\eta\ \nu$) = trg_D ν
and *D.dom* ($\eta\ \nu$) = *D.dom* ν **and** *D.cod* ($\eta\ \nu$) = F (G (*D.cod* ν))
and *D.ide* ν $\Longrightarrow$ *D.iso* ($\eta\ \nu$)
⟨*proof*⟩

lemma *η-naturality*:
assumes *D.arr* ν
shows η (*D.cod* ν) $\cdot_D$ ν = $\eta\ \nu$ **and** F ($G\ \nu$) $\cdot_D$ η (*D.dom* ν) = $\eta\ \nu$
and ν $\cdot_D$ *D.inv* (η (*D.dom* ν)) = *D.inv* (η (*D.cod* ν)) $\cdot_D$ F ($G\ \nu$)
⟨*proof*⟩

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

⟨*proof*⟩

In view of the bijectivity assumption, we can obtain the counit ε the same way.

definition ε
where $\varepsilon\ \mu = (\mathit{if}\ C.\mathit{arr}\ \mu\ \mathit{then}$
$\quad \mathit{equivalence\text{-}pseudofunctor\text{-}at\text{-}hom.}\varepsilon\ V_C\ \mathit{src}_C\ \mathit{trg}_C\ V_D\ \mathit{src}_D\ \mathit{trg}_D\ F$
$\quad\ (\mathit{src}_C\ \mu)\ (\mathit{trg}_C\ \mu)\ \mu$
$\quad \mathit{else}\ C.\mathit{null})$

lemma *ε-in-hom*:
assumes $C.\mathit{arr}\ \mu$
shows [*intro*]: «$\varepsilon\ \mu : \mathit{src}_C\ \mu \rightarrow_C \mathit{trg}_C\ \mu$»
and [*intro*]: «$\varepsilon\ \mu : G\ (F\ (C.\mathit{dom}\ \mu)) \Rightarrow_C C.\mathit{cod}\ \mu$»
and $C.\mathit{ide}\ \mu \Longrightarrow C.\mathit{iso}\ (\varepsilon\ \mu)$
⟨*proof*⟩

lemma *ε-simps* [*simp*]:
assumes $C.\mathit{arr}\ \mu$
shows $C.\mathit{arr}\ (\varepsilon\ \mu)$
and $\mathit{src}_C\ (\varepsilon\ \mu) = \mathit{src}_C\ \mu$ **and** $\mathit{trg}_C\ (\varepsilon\ \mu) = \mathit{trg}_C\ \mu$
and $C.\mathit{dom}\ (\varepsilon\ \mu) = G\ (F\ (C.\mathit{dom}\ \mu))$ **and** $C.\mathit{cod}\ (\varepsilon\ \mu) = C.\mathit{cod}\ \mu$
and $C.\mathit{ide}\ \mu \Longrightarrow C.\mathit{iso}\ (\varepsilon\ \mu)$
⟨*proof*⟩

lemma *ε-naturality*:
assumes $C.\mathit{arr}\ \mu$
shows $\varepsilon\ (C.\mathit{cod}\ \mu) \cdot_C G\ (F\ \mu) = \varepsilon\ \mu$ **and** $\mu \cdot_C \varepsilon\ (C.\mathit{dom}\ \mu) = \varepsilon\ \mu$
and $G\ (F\ \mu) \cdot_C C.\mathit{inv}\ (\varepsilon\ (C.\mathit{dom}\ \mu)) = C.\mathit{inv}\ (\varepsilon\ (C.\mathit{cod}\ \mu)) \cdot_C \mu$
⟨*proof*⟩

interpretation *ε*: $\mathit{natural\text{-}isomorphism}\ V_C\ V_C\ \mathit{GF.map}\ C.\mathit{map}\ \varepsilon$
⟨*proof*⟩

interpretation *GFG*: $\mathit{composite\text{-}functor}\ V_D\ V_C\ V_C\ G\ ‹\mathit{GF.map}›$ ⟨*proof*⟩
interpretation *FGF*: $\mathit{composite\text{-}functor}\ V_C\ V_D\ V_D\ F\ ‹\mathit{FG.map}›$ ⟨*proof*⟩

interpretation *Goη*: $\mathit{natural\text{-}transformation}\ V_D\ V_C\ G\ \mathit{GFG.map}\ ‹G \circ \eta›$
⟨*proof*⟩

interpretation *ηoF*: $\mathit{natural\text{-}transformation}\ V_C\ V_D\ F\ \mathit{FGF.map}\ ‹\eta \circ F›$
⟨*proof*⟩

interpretation *εoG*: $\mathit{natural\text{-}transformation}\ V_D\ V_C\ \mathit{GFG.map}\ G\ ‹\varepsilon \circ G›$
⟨*proof*⟩

interpretation *Foε*: $\mathit{natural\text{-}transformation}\ V_C\ V_D\ \mathit{FGF.map}\ F\ ‹F \circ \varepsilon›$
⟨*proof*⟩

interpretation *εoG-Goη*: $\mathit{vertical\text{-}composite}\ V_D\ V_C\ G\ \mathit{GFG.map}\ G\ ‹G \circ \eta›\ ‹\varepsilon \circ G›$ ⟨*proof*⟩

interpretation *Foε-ηoF*: *vertical-composite* V_C V_D F *FGF.map* F ‹$\eta \circ F$› ‹$F \circ \varepsilon$› ⟨*proof*⟩

Bijectivity results in an ordinary adjunction between the vertical categories.

lemma *adjunction-ηε*:
shows *unit-counit-adjunction* V_C V_D G F η ε
⟨*proof*⟩

interpretation *ηε*: *unit-counit-adjunction* V_C V_D G F η ε
⟨*proof*⟩

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 $\Phi_{G0}\ f\ f' = C.inv\ (\varepsilon\ (G\ f \star_C G\ f') \cdot_C G\ (\Phi\ (G\ f,\ G\ f') \cdot_D (\eta\ f \star_D \eta\ f')))$

lemma Φ_{G0}*-in-hom*:
assumes *D.ide* f **and** *D.ide* f' **and** $src_D\ f = trg_D\ f'$
shows «$\Phi_{G0}\ f\ f' : G\ f \star_C G\ f' \Rightarrow_C G\ (f \star_D f')$»
⟨*proof*⟩

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*⟩

lemma Φ_{G0}*-naturality*:
assumes *D.arr* ν **and** *D.arr* ν' **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*⟩

interpretation Φ_G: *transformation-by-components D.VV.comp* V_C H_D*oGG.map GoH*$_D$*.map*
‹$\lambda fg.\ \Phi_{G0}\ (fst\ fg)\ (snd\ fg)$›
⟨*proof*⟩

interpretation Φ_G: *natural-isomorphism D.VV.comp* V_C H_D*oGG.map GoH*$_D$*.map* Φ_G*.map*
⟨*proof*⟩

abbreviation Φ_G
where $\Phi_G \equiv \Phi_G.map$

lemma Φ_G*-in-hom* [*intro*]:
assumes *D.arr* ν **and** *D.arr* ν' **and** $src_D\ \nu = trg_D\ \nu'$
shows *C.in-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*⟩

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 $C.cod\ (\Phi_G.map\ (\nu,\ \nu')) = G\ (D.cod\ \nu \star_D D.cod\ \nu')$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

interpretation GF: *composite-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi\ G\ \Phi_G.map$
⟨*proof*⟩
interpretation FG: *composite-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ G\ \Phi_G.map\ F\ \Phi$
⟨*proof*⟩
interpretation I_C: *identity-pseudofunctor* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$ ⟨*proof*⟩
interpretation I_D: *identity-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$ ⟨*proof*⟩

lemma η-*equals-FG-unit*:
assumes $D.obj\ a$
shows $\eta\ a = FG.unit\ a$
⟨*proof*⟩

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*⟩

lemma ε-*inverts-GF-unit*:
assumes $C.obj\ a$
shows $\varepsilon\ a \cdot_C GF.unit\ a = a$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

interpretation *EQ*: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi\ G\ \Phi_G.map\ {unit_0}'\ {unit_1}'\ {counit_0}'\ {counit_1}'$
⟨*proof*⟩

lemma *extends-to-equivalence-of-bicategories*:
shows $equivalence\text{-}of\text{-}bicategories\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi\ G\ \Phi_G.map\ {unit_0}'\ {unit_1}'\ {counit_0}'\ {counit_1}'$
⟨*proof*⟩

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*⟩

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 $\exists\, b'.\ b' \in map_0\ `\ Collect\ C.obj \wedge D.equivalent\text{-}objects\ b\ b'$
⟨*proof*⟩

lemma *bij-betw-U-V*:
shows $bij\text{-}betw\ map_0\ U\ V$
⟨*proof*⟩

abbreviation (*input*) Arr_U
where $Arr_U \equiv \lambda\mu.\ C.arr\ \mu \wedge src_C\ \mu \in U \wedge trg_C\ \mu \in U$

interpretation C_U: *subbicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ Arr_U$
⟨*proof*⟩

interpretation C_U: *dense-subbicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ U$
⟨*proof*⟩

abbreviation (*input*) Arr_V
where $Arr_V \equiv \lambda\mu.\ D.arr\ \mu \wedge src_D\ \mu \in V \wedge trg_D\ \mu \in V$

interpretation D_V: *subbicategory* V_D H_D a_D i_D src_D trg_D Arr_V
⟨*proof*⟩

interpretation D_V: *dense-subbicategory* V_D H_D a_D i_D src_D trg_D V
⟨*proof*⟩

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
$\langle\lambda\mu.\ C.arr\ \mu \wedge src_C\ \mu \in U \wedge trg_C\ \mu \in U\rangle$
⟨*proof*⟩

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 $F_U.map$ $F_U.cmp$ $\langle\lambda a.\ a \in V\rangle$
⟨*proof*⟩

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*⟩

interpretation F_{UV}: *equivalence-pseudofunctor-bij-on-obj*
$C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ 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*⟩

interpretation EQ_{UV}: *equivalence-of-bicategories*
$D_V.comp$ $D_V.hcomp$ $D_V.\mathrm{a}$ i_D $D_V.src$ $D_V.trg$
$C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ i_C $C_U.src$ $C_U.trg$
$F_{UV}.map$ $F_{UV}.cmp$ $F_{UV}.G$ $F_{UV}.\Phi_G$
$F_{UV}.unit_0{}'$ $F_{UV}.unit_1{}'$ $F_{UV}.counit_0{}'$ $F_{UV}.counit_1{}'$
⟨*proof*⟩

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 a_C i_C src_C trg_C $C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ i_C $C_U.src$ $C_U.trg$
$C_U.E$ $C_U.\Phi_E$ $C_U.P$ $C_U.\Phi_P$
$C_U.unit_0$ $C_U.unit_1$ $C_U.counit_0$ $C_U.counit_1$
⟨*proof*⟩

interpretation $EQ_C{}'$: *converse-equivalence-of-bicategories*
V_C H_C a_C i_C src_C trg_C $C_U.comp$ $C_U.hcomp$ $C_U.\mathrm{a}$ i_C $C_U.src$ $C_U.trg$
$C_U.E$ $C_U.\Phi_E$ $C_U.P$ $C_U.\Phi_P$
$C_U.unit_0$ $C_U.unit_1$ $C_U.counit_0$ $C_U.counit_1$
⟨*proof*⟩

interpretation EQ_D: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$D_V.E\ D_V.\Phi_E\ D_V.P\ D_V.\Phi_P$
$D_V.unit_0\ D_V.unit_1\ D_V.counit_0\ D_V.counit_1$
⟨*proof*⟩

interpretation $EQ_{UV}\,oEQ_C'$: *composite-equivalence-of-bicategories*
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$C_U.comp\ C_U.hcomp\ C_U.\mathrm{a}\ \mathrm{i}_C\ C_U.src\ C_U.trg$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F_{UV}.map\ F_{UV}.cmp\ F_{UV}.G\ F_{UV}.\Phi_G$
$C_U.P\ C_U.\Phi_P\ C_U.E\ C_U.\Phi_E$
$F_{UV}.unit_0'\ F_{UV}.unit_1'\ F_{UV}.counit_0'\ F_{UV}.counit_1'$
$EQ_C'.unit_0\ EQ_C'.unit_1\ EQ_C'.counit_0\ EQ_C'.counit_1$
⟨*proof*⟩

interpretation $EQ_D\,oEQ_{UV}\,oEQ_C'$: *composite-equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
$D_V.comp\ D_V.hcomp\ D_V.\mathrm{a}\ \mathrm{i}_D\ D_V.src\ D_V.trg$
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$D_V.E\ D_V.\Phi_E\ D_V.P\ D_V.\Phi_P$
$EQ_{UV}\,oEQ_C'.left\text{-}map\ EQ_{UV}\,oEQ_C'.left\text{-}cmp$
$EQ_{UV}\,oEQ_C'.right\text{-}map\ EQ_{UV}\,oEQ_C'.right\text{-}cmp$
$D_V.unit_0\ D_V.unit_1\ D_V.counit_0\ D_V.counit_1$
$EQ_{UV}\,oEQ_C'.unit_0\ EQ_{UV}\,oEQ_C'.unit_1$
$EQ_{UV}\,oEQ_C'.counit_0\ EQ_{UV}\,oEQ_C'.counit_1$
⟨*proof*⟩

lemma *induces-equivalence-of-bicategories*:
shows *equivalence-of-bicategories* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$EQ_D\,oEQ_{UV}\,oEQ_C'.left\text{-}map\ EQ_D\,oEQ_{UV}\,oEQ_C'.left\text{-}cmp$
$EQ_D\,oEQ_{UV}\,oEQ_C'.right\text{-}map\ EQ_D\,oEQ_{UV}\,oEQ_C'.right\text{-}cmp$
$EQ_D\,oEQ_{UV}\,oEQ_C'.unit_0\ EQ_D\,oEQ_{UV}\,oEQ_C'.unit_1$
$EQ_D\,oEQ_{UV}\,oEQ_C'.counit_0\ EQ_D\,oEQ_{UV}\,oEQ_C'.counit_1$
⟨*proof*⟩

lemma *left-map-simp*:
assumes $C.arr\ \mu$
shows $EQ_D\,oEQ_{UV}\,oEQ_C'.left\text{-}map\ \mu = D_V.E\ (F\ (C_U.P\ \mu))$
⟨*proof*⟩

lemma *right-map-simp*:
assumes $D.arr\ \nu$
shows $EQ_D\,oEQ_{UV}\,oEQ_C'.right\text{-}map\ \nu = C_U.E\ (F_{UV}.G\ (D_V.P\ \nu))$
⟨*proof*⟩

lemma $unit_0$*-simp*:
assumes $C.obj\ a$
shows $EQ_D\,oEQ_{UV}\,oEQ_C'.unit_0\ a =$

$C_U.E\ (F_{UV}.G\ (D_V.\eta_0\ (D_V.src\ (F\ (C_U.P\ a)))))\ \star_C\ C_U.E\ (C_U.P_0\ (src_C\ a))$
$\star_C\ EQ_C'.unit_0\ a$
⟨*proof*⟩

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 «$\tau_0\ a : map_0\ (C_U.P_0\ a) \to_D map_0\ a$»
and «$\tau_0\ a : \tau_0\ a \Rightarrow_D \tau_0\ a$»
⟨*proof*⟩

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** $trg_D\ (\tau_0\ a) = map_0\ a$
⟨*proof*⟩

lemma τ_1-*in-hom* [*intro*]:
assumes $C.ide\ f$
shows «$\tau_1\ f : map_0\ (C_U.P_0\ (src_C\ f)) \to_D map_0\ (trg_C\ f)$»
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*⟩

lemma τ_1-*simps* [*simp*]:
assumes $C.ide\ f$
shows $D.arr\ (\tau_1\ f)$
and $src_D\ (\tau_1\ f) = map_0\ (C_U.P_0\ (src_C\ f))$ **and** $trg_D\ (\tau_1\ f) = map_0\ (trg_C\ f)$
and $D.dom\ (\tau_1\ f) = F\ f \star_D \tau_0\ (src_C\ f)$ **and** $D.cod\ (\tau_1\ f) = \tau_0\ (trg_C\ f) \star_D F\ (C_U.P\ f)$
⟨*proof*⟩

lemma *iso-*τ_1:
assumes $C.ide\ f$
shows $D.iso\ (\tau_1\ f)$
⟨*proof*⟩

interpretation τ: *pseudonatural-equivalence*
$V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$

$EQ_D oEQ_{UV} oEQ_C'.left\text{-}map\ EQ_D oEQ_{UV} oEQ_C'.left\text{-}cmp\ F\ \Phi$
$\tau_0\ \tau_1$
⟨*proof*⟩

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*⟩

definition *right-map*
where $right\text{-}map \equiv EQ_D oEQ_{UV} oEQ_C'.right\text{-}map$

definition *right-cmp*
where $right\text{-}cmp \equiv EQ_D oEQ_{UV} oEQ_C'.right\text{-}cmp$

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* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi$ *right-map right-cmp* $unit_0\ unit_1\ counit_0\ counit_1$
⟨*proof*⟩

end

locale *converse-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* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ F\ \Phi_F$
for $V_C :: 'c\ comp$ (**infixr** ‹$\cdot_C$› *55*)
and $H_C :: 'c\ comp$ (**infixr** ‹$\star_C$› *53*)
and $\mathrm{a}_C :: 'c \Rightarrow 'c \Rightarrow 'c \Rightarrow 'c$ (‹a_C[-, -, -]›)
and $\mathrm{i}_C :: 'c \Rightarrow 'c$ (‹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 $\mathrm{a}_D :: 'd \Rightarrow 'd \Rightarrow 'd \Rightarrow 'd$ (‹a_D[-, -, -]›)
and $\mathrm{i}_D :: 'd \Rightarrow 'd$ (‹i_D[-]›)
and $src_D :: 'd \Rightarrow 'd$
and $trg_D :: 'd \Rightarrow 'd$
and $F :: 'c \Rightarrow 'd$
and $\Phi_F :: 'c * 'c \Rightarrow 'd$
begin

interpretation *E*: *equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi_F\ F.right\text{-}map\ F.right\text{-}cmp\ F.unit_0\ F.unit_1\ F.counit_0\ F.counit_1$
⟨*proof*⟩
interpretation *E'*: *converse-equivalence-of-bicategories*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F\ \Phi_F\ F.right\text{-}map\ F.right\text{-}cmp\ F.unit_0\ F.unit_1\ F.counit_0\ F.counit_1$
⟨*proof*⟩

sublocale *equivalence-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F.right\text{-}map\ F.right\text{-}cmp$
⟨*proof*⟩

lemma *is-equivalence-pseudofunctor*:
shows *equivalence-pseudofunctor* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
$F.right\text{-}map\ F.right\text{-}cmp$
⟨*proof*⟩

end

definition *equivalent-bicategories*
where *equivalent-bicategories* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C \equiv$
$\exists F\ \Phi.$ *equivalence-pseudofunctor*
$V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ F\ \Phi$

lemma *equivalent-bicategories-reflexive*:
assumes *bicategory* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
shows *equivalent-bicategories* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
⟨*proof*⟩

lemma *equivalent-bicategories-symmetric*:
assumes *equivalent-bicategories* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$
shows *equivalent-bicategories* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
⟨*proof*⟩

lemma *equivalent-bicategories-transitive*:
assumes *equivalent-bicategories* $V_B\ H_B\ \mathrm{a}_B\ \mathrm{i}_B\ src_B\ trg_B\ V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C$
and *equivalent-bicategories* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D$

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*⟩

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
    ⟨proof⟩

  lemma leg-in-hom [intro]:
  shows «leg0 : apex → src»
  and «leg1 : apex → trg»
    ⟨proof⟩

  lemma leg-simps [simp]:
  shows C.arr leg0 and C.dom leg0 = apex
  and C.arr leg1 and C.dom leg1 = apex
    ⟨proof⟩

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
    ⟨proof⟩

  lemma cod-src-eq-dom-src [simp]:
  shows cod.src = dom.src
    ⟨proof⟩

  lemma cod-trg-eq-dom-trg [simp]:
  shows cod.trg = dom.trg
    ⟨proof⟩

  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
    ⟨proof⟩

  lemma cod-simps [simp]:
  shows cod.apex = apex and cod.leg0 = leg0 and cod.leg1 = leg1
    ⟨proof⟩

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*
    ⟨*proof*⟩

  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*
    ⟨*proof*⟩

  lemma *is-partial-composition*:
  shows *partial-magma vcomp*
    ⟨*proof*⟩

  lemma *null-char*:
  shows *V.null* = *Null*
    ⟨*proof*⟩

  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*⟩

  lemma *has-domain-char*:
  shows *V.domains* μ ≠ {} ⟷ *arrow-of-spans C* μ
  ⟨*proof*⟩

  lemma *has-codomain-char*:
  shows *V.codomains* μ ≠ {} ⟷ *arrow-of-spans C* μ
  ⟨*proof*⟩

  lemma *arr-char*:

shows $V.arr\ \mu \longleftrightarrow arrow\text{-}of\text{-}spans\ C\ \mu$
⟨*proof*⟩

lemma *seq-char*:
shows $V.seq\ \nu\ \mu \longleftrightarrow arrow\text{-}of\text{-}spans\ C\ \mu \wedge arrow\text{-}of\text{-}spans\ C\ \nu \wedge Dom\ \nu = Cod\ \mu$
⟨*proof*⟩

interpretation V: *category vcomp*
⟨*proof*⟩

lemma *is-category*:
shows *category vcomp*
⟨*proof*⟩

lemma *dom-char*:
shows $V.dom = (\lambda\mu.\ if\ V.arr\ \mu\ then$
$(\!|Chn = span\text{-}in\text{-}category.apex\ C\ (Dom\ \mu),\ Dom = Dom\ \mu,\ Cod = Dom\ \mu|\!)$
$else\ V.null)$
⟨*proof*⟩

lemma *cod-char*:
shows $V.cod = (\lambda\mu.\ if\ V.arr\ \mu\ then$
$(\!|Chn = span\text{-}in\text{-}category.apex\ C\ (Cod\ \mu),\ Dom = Cod\ \mu,\ Cod = Cod\ \mu|\!)$
$else\ V.null)$
⟨*proof*⟩

lemma *vcomp-char*:
shows $vcomp = (\lambda\nu\ \mu.\ if\ V.seq\ \nu\ \mu\ then$
$(\!|Chn = Chn\ \nu \cdot Chn\ \mu,\ Dom = Dom\ \mu,\ Cod = Cod\ \nu|\!)$
$else\ V.null)$
⟨*proof*⟩

lemma *vcomp-eq*:
assumes $V.seq\ \nu\ \mu$
shows $\nu \cdot \mu = (\!|Chn = Chn\ \nu \cdot Chn\ \mu,\ Dom = Dom\ \mu,\ Cod = Cod\ \nu|\!)$
⟨*proof*⟩

lemma *Chn-vcomp*:
assumes $V.seq\ \nu\ \mu$
shows $Chn\ (\nu \cdot \mu) = Chn\ \nu \cdot Chn\ \mu$
⟨*proof*⟩

lemma *ide-char'*:
shows $V.ide\ \mu \longleftrightarrow identity\text{-}arrow\text{-}of\text{-}spans\ C\ \mu$
⟨*proof*⟩

lemma *Chn-in-hom*:
assumes $V.in\text{-}hom\ \tau\ f\ g$
shows $C.in\text{-}hom\ (Chn\ \tau)\ (Chn\ f)\ (Chn\ g)$

⟨*proof*⟩

abbreviation *mkIde*
where *mkIde f0 f1* ≡
(|*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*⟩

abbreviation *mkObj*
where *mkObj a* ≡ *mkIde a a*

lemma *ide-mkObj*:
assumes *C.ide a*
shows *V.ide* (*mkObj a*)
⟨*proof*⟩

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*⟩

lemma *iso-char*:
shows *V.iso* μ ⟷ *V.arr* μ ∧ *C.iso* (*Chn* μ)
⟨*proof*⟩

lemma *inv-eq*:
assumes *V.iso* μ
shows *V.inv* μ = (|*Chn* = *C.inv* (*Chn* μ), *Dom* = *Cod* μ, *Cod* = *Dom* μ|)
⟨*proof*⟩

end

2.1.3 Putting Spans in Homs

context *span-vertical-category*
begin

interpretation *V*: *category vcomp*
⟨*proof*⟩

definition *src*
where *src* μ ≡ *if V.arr* μ *then mkObj* (*C.cod* (*Leg0* (*Dom* μ))) *else V.null*

lemma *ide-src* [*simp*]:
assumes *V.arr* μ
shows *V.ide* (*src* μ)

⟨*proof*⟩

interpretation *src*: *endofunctor vcomp src*
⟨*proof*⟩

lemma *src-is-endofunctor*:
shows *endofunctor vcomp src*
⟨*proof*⟩

lemma *src-vcomp*:
assumes *V.seq ν μ*
shows *src* (*ν* ⋆ *μ*) = *src ν* ⋆ *src μ*
⟨*proof*⟩

definition *trg*
where *trg μ* ≡ *if V.arr μ then mkObj* (*C.cod* (*Leg1* (*Dom μ*))) *else V.null*

lemma *ide-trg* [*simp*]:
assumes *V.arr μ*
shows *V.ide* (*trg μ*)
⟨*proof*⟩

interpretation *trg*: *endofunctor vcomp trg*
⟨*proof*⟩

lemma *trg-is-endofunctor*:
shows *endofunctor vcomp trg*
⟨*proof*⟩

lemma *trg-vcomp*:
assumes *V.seq ν μ*
shows *trg* (*ν* ⋆ *μ*) = *trg ν* ⋆ *trg μ*
⟨*proof*⟩

lemma *src-trg-simps* [*simp*]:
assumes *V.arr μ*
shows *src* (*src μ*) = *src μ*
and *src* (*trg μ*) = *trg μ*
and *trg* (*src μ*) = *src μ*
and *trg* (*trg μ*) = *trg μ*
⟨*proof*⟩

sublocale *horizontal-homs vcomp src trg*
⟨*proof*⟩

lemma *has-horizontal-homs*:
shows *horizontal-homs vcomp src trg*
⟨*proof*⟩

lemma *obj-char*:
shows *obj a* $\longleftrightarrow$ *V.ide a* $\wedge$ *a* = *mkObj* (*Chn a*)
⟨*proof*⟩

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* ≡ ⟪*Leg0* = *T.leg0* · p_0[*S.leg0*, *T.leg1*], *Leg1* = *S.leg1* · p_1[*S.leg0*, *T.leg1*]⟫

lemma *leg0-prj-in-hom*:
shows «*T.leg0* · p_0[*S.leg0*, *T.leg1*] : *S.leg0* ↓↓ *T.leg1* → *C.cod* (*Leg0 T*)»
⟨*proof*⟩

lemma *leg1-prj-in-hom*:
shows «*S.leg1* · p_1[*S.leg0*, *T.leg1*] : *S.leg0* ↓↓ *T.leg1* → *C.cod* (*Leg1 S*)»
⟨*proof*⟩

lemma *is-span* [*simp*]:
shows *span-in-category C this*
⟨*proof*⟩

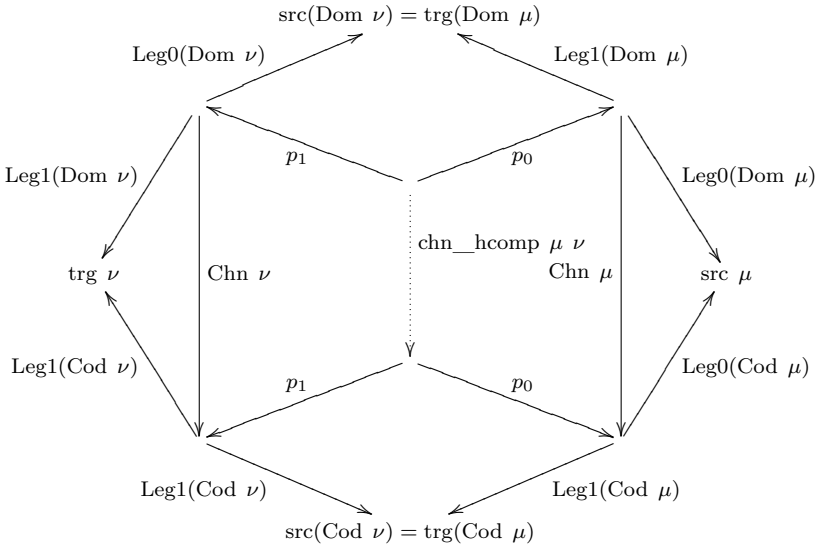
sublocale *span-in-category C this*
⟨*proof*⟩

end

locale *span-bicategory* =
C: *elementary-category-with-pullbacks* +
span-vertical-category
begin

definition *chine-hcomp*

where *chine-hcomp* ν $\mu \equiv$
$\langle Chn\ \nu \cdot \mathrm{p}_1[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$
$[\![Leg0\ (Cod\ \nu),\ Leg1\ (Cod\ \mu)]\!]$
$Chn\ \mu \cdot \mathrm{p}_0[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]\rangle$



definition *hcomp*
where *hcomp* ν $\mu \equiv$ *if arr* $\mu \wedge$ *arr* $\nu \wedge$ *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** ‹⋆› *53*)

lemma *chine-hcomp-props*:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows «*chine-hcomp* ν μ : *Leg0* (*Dom* ν) ↓↓ *Leg1* (*Dom* μ) →$_C$ *Leg0* (*Cod* ν) ↓↓ *Leg1* (*Cod* μ)»
and *C.commutative-square* (*Leg0* (*Cod* ν)) (*Leg1* (*Cod* μ))
(*Chn* $\nu \cdot \mathrm{p}_1[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$)
(*Chn* $\mu \cdot \mathrm{p}_0[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$)
and *C.commutative-square* $\mathrm{p}_1[Leg0\ (Cod\ \nu),\ Leg1\ (Cod\ \mu)]$ (*Chn* ν)
(*chine-hcomp* ν μ) $\mathrm{p}_1[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$
and *C.commutative-square* $\mathrm{p}_0[Leg0\ (Cod\ \nu),\ Leg1\ (Cod\ \mu)]$ (*Chn* μ)
(*chine-hcomp* ν μ) $\mathrm{p}_0[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$
and $\mathrm{p}_0[Leg0\ (Cod\ \nu),\ Leg1\ (Cod\ \mu)] \cdot$ *chine-hcomp* ν μ =
Chn $\mu \cdot \mathrm{p}_0[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$
and $\mathrm{p}_1[Leg0\ (Cod\ \nu),\ Leg1\ (Cod\ \mu)] \cdot$ *chine-hcomp* ν μ =
Chn $\nu \cdot \mathrm{p}_1[Leg0\ (Dom\ \nu),\ Leg1\ (Dom\ \mu)]$
⟨*proof*⟩

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)»
⟨*proof*⟩

lemma *arrow-of-spans-hcomp*:
assumes *arr* μ **and** *arr* ν **and** *src* ν = *trg* μ
shows *arrow-of-spans C* (ν ⋆ μ)
⟨*proof*⟩

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*⟩

lemma *chine-hcomp-arr-ide*:
assumes *arr* μ **and** *ide f* **and** *src* μ = *trg f*
shows *chine-hcomp* μ *f* =
⟨*Chn* μ · p_1[*Leg0* (*Dom* μ), *Leg1* (*Dom f*)]
[[*Leg0* (*Cod* μ), *Leg1* (*Cod f*)]]
p_0[*Leg0* (*Dom* μ), *Leg1* (*Dom f*)]⟩
⟨*proof*⟩

lemma *chine-hcomp-ide-ide*:
assumes *ide g* **and** *ide f* **and** *src g* = *trg f*
shows *chine-hcomp g f* = *Leg0* (*Dom g*) $\downarrow\downarrow$ *Leg1* (*Dom f*)
⟨*proof*⟩

lemma *chine-hcomp-trg-arr*:
assumes *arr* μ
shows *chine-hcomp* (*trg* μ) μ =
⟨p_1[*C.cod* (*Leg1* (*Dom* μ)), *Leg1* (*Dom* μ)]
[[*C.cod* (*Leg1* (*Dom* μ)), *Leg1* (*Cod* μ)]]
Chn μ · p_0[*C.cod* (*Leg1* (*Dom* μ)), *Leg1* (*Dom* μ)]⟩
⟨*proof*⟩

lemma *chine-hcomp-trg-ide*:
assumes *ide f*
shows *chine-hcomp* (*trg f*) *f* = *C.cod* (*Leg1* (*Dom f*)) $\downarrow\downarrow$ *Leg1* (*Dom f*)
⟨*proof*⟩

lemma *chine-hcomp-arr-src*:
assumes *arr* μ
shows *chine-hcomp* μ (*src* μ) =

$\langle Chn\ \mu \cdot \mathrm{p}_1[Leg0\ (Dom\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]$
$[\![Leg0\ (Cod\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]\!]$
$\mathrm{p}_0[Leg0\ (Dom\ \mu),\ C.cod\ (Leg0\ (Dom\ \mu))]\rangle$
⟨*proof*⟩

lemma *chine-hcomp-ide-src*:
assumes *ide f*
shows *chine-hcomp f (src f)* = *Leg0 (Dom f)* ↓↓ *C.cod (Leg0 (Dom f))*
⟨*proof*⟩

lemma *src-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src ν* = *trg μ*
shows *src (ν* ⋆ *μ)* = *src μ*
⟨*proof*⟩

lemma *trg-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src ν* = *trg μ*
shows *trg (ν* ⋆ *μ)* = *trg ν*
⟨*proof*⟩

lemma *dom-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src ν* = *trg μ*
shows *dom (ν* ⋆ *μ)* = *dom ν* ⋆ *dom μ*
⟨*proof*⟩

lemma *cod-hcomp* [*simp*]:
assumes *arr μ* **and** *arr ν* **and** *src ν* = *trg μ*
shows *cod (ν* ⋆ *μ)* = *cod ν* ⋆ *cod μ*
⟨*proof*⟩

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 *(ν′* ⋅ *ν)* ⋆ *(μ′* ⋅ *μ)* = *(ν′* ⋆ *μ′)* ⋅ *(ν* ⋆ *μ)*
⟨*proof*⟩

interpretation *H*: *functor VV.comp vcomp* ‹*λνμ. fst νμ* ⋆ *snd νμ*›
⟨*proof*⟩

lemma *hcomp-is-functor*:
shows *functor VV.comp vcomp (λνμ. fst νμ* ⋆ *snd νμ)*
⟨*proof*⟩

lemma *ide-hcomp*:
assumes *ide f* **and** *ide g* **and** *src f* = *trg g*
shows *ide (f* ⋆ *g)*
⟨*proof*⟩

sublocale *horizontal-composition vcomp hcomp src trg*
⟨*proof*⟩

lemma *has-horizontal-composition*:
shows *horizontal-composition vcomp hcomp src trg*
⟨*proof*⟩

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 $\mu = \mu'$
⟨*proof*⟩

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*⟩

interpretation ℓ: *natural-isomorphism vcomp vcomp L map* ℓ.*map*
⟨*proof*⟩

lemma ℓ-*is-natural-isomorphism*:
shows *natural-isomorphism vcomp vcomp L map* ℓ.*map*
⟨*proof*⟩

sublocale *L*: *equivalence-functor vcomp vcomp L*
⟨*proof*⟩

lemma *equivalence-functor-L*:
shows *equivalence-functor vcomp vcomp L*
⟨*proof*⟩

abbreviation r
where r $f \equiv$ ⟦*Chn* = p_1[*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*⟩

interpretation ϱ: *natural-isomorphism vcomp vcomp R map* ϱ.*map*
⟨*proof*⟩

lemma *ϱ-is-natural-isomorphism*:
shows *natural-isomorphism vcomp vcomp R map ϱ.map*
⟨*proof*⟩

sublocale *R*: *equivalence-functor vcomp vcomp R*
⟨*proof*⟩

lemma *equivalence-functor-R*:
shows *equivalence-functor vcomp vcomp R*
⟨*proof*⟩

definition *unit* (‹i[-]›)
where i[*a*] ≡ ⦇*Chn* = p_0[*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*⟩

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*
⟨*proof*⟩

lemma *iso-unit*:
assumes *obj a*
shows *iso* i[*a*]
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *legs-form-cospan*:
shows *C.cospan μ.dom.leg0 ν.dom.leg1* **and** *C.cospan μ.cod.leg0 ν.cod.leg1*

⟨*proof*⟩

interpretation *μν*: *arrow-of-spans C* ‹*μ ⋆ ν*›
⟨*proof*⟩

lemma *composite-is-arrow* [*simp*]:
shows *arr* (*μ ⋆ ν*)
⟨*proof*⟩

lemma *composite-in-hom* [*intro*]:
shows «*μ ⋆ ν* : *dom μ ⋆ dom ν* ⇒ *cod μ ⋆ cod ν*»
⟨*proof*⟩

lemma *composite-simps* [*simp*]:
shows *src* (*μ ⋆ ν*) = *src ν* **and** *trg* (*μ ⋆ ν*) = *trg μ*
and *dom* (*μ ⋆ ν*) = *dom μ ⋆ dom ν* **and** *cod* (*μ ⋆ ν*) = *cod μ ⋆ cod ν*
⟨*proof*⟩

lemma *chine-composite*:
shows *Chn* (*μ ⋆ ν*) = ⟨*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*]⟩
⟨*proof*⟩

lemma *chine-composite-in-hom* [*intro*]:
shows «*Chn* (*μ ⋆ ν*) : *μ.dom.leg0* ↓↓ *ν.dom.leg1* →$_C$ *μ.cod.leg0* ↓↓ *ν.cod.leg1*»
⟨*proof*⟩

end

sublocale *two-composable-arrows-of-spans* ⊆ *arrow-of-spans C* ‹*μ ⋆ ν*›
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation *H*: *functor VV.comp vcomp* ‹*λνμ. fst νμ ⋆ snd νμ*›
⟨*proof*⟩

interpretation *μν*: *identity-arrow-of-spans C* ‹*μ ⋆ ν*›
⟨*proof*⟩

lemma *ide-composite* [*simp*]:
shows *ide* ($\mu \star \nu$)
⟨*proof*⟩

lemma *apex-composite*:
shows $\mu\nu.apex = \mu.leg0 \downarrow\downarrow \nu.leg1$
⟨*proof*⟩

lemma *leg0-composite*:
shows $\mu\nu.leg0 = \nu.leg0 \cdot \mathrm{p}_0[\mu.leg0, \nu.leg1]$
⟨*proof*⟩

lemma *leg1-composite*:
shows $\mu\nu.leg1 = \mu.leg1 \cdot \mathrm{p}_1[\mu.leg0, \nu.leg1]$
⟨*proof*⟩

lemma *chine-composite*:
shows *Chn* ($\mu \star \nu$) = $\mu.leg0 \downarrow\downarrow \nu.leg1$
⟨*proof*⟩

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 «$prj_0 : \mu.leg0 \downarrow\downarrow \nu.leg1 \rightarrow_C \nu.apex$»
⟨*proof*⟩

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* $prj_0 = \nu.apex$
⟨*proof*⟩

sublocale *identity-arrow-of-spans C* ‹$\mu \star \nu$›
⟨*proof*⟩

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 ‹$\mu \star \nu \star \pi$›
⟨*proof*⟩

interpretation $\mu\nu$-π: *arrow-of-spans* C ‹$(\mu \star \nu) \star \pi$›
⟨*proof*⟩

lemma *composites-are-arrows* [*simp*]:
shows *arr* $(\mu \star \nu \star \pi)$ **and** *arr* $((\mu \star \nu) \star \pi)$
⟨*proof*⟩

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»
⟨*proof*⟩

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
⟨*proof*⟩

lemma *chine-composite*:
shows $\mu\nu\pi$.*chine* =
⟨μ.*chine* · p_1[μ.*dom.leg0*, ν.*dom.leg1* · p_1[ν.*dom.leg0*, π.*dom.leg1*]]
⟦μ.*cod.leg0*, ν.*cod.leg1* · p_1[ν.*cod.leg0*, π.*cod.leg1*]⟧
⟨ν.*chine* · p_1[ν.*dom.leg0*, π.*dom.leg1*]
⟦ν.*cod.leg0*, π.*cod.leg1*⟧
π.*chine* · p_0[ν.*dom.leg0*, π.*dom.leg1*]⟩ ·
p_0[μ.*dom.leg0*, ν.*dom.leg1* · p_1[ν.*dom.leg0*, π.*dom.leg1*]]⟩
and $\mu\nu$-π.*chine* =
⟨⟨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*]⟩ ·
p_1[ν.*dom.leg0* · p_0[μ.*dom.leg0*, ν.*dom.leg1*], π.*dom.leg1*]
⟦ν.*cod.leg0* · p_0[μ.*cod.leg0*, ν.*cod.leg1*], π.*cod.leg1*⟧
π.*chine* · p_0[ν.*dom.leg0* · p_0[μ.*dom.leg0*, ν.*dom.leg1*], π.*dom.leg1*]⟩
⟨*proof*⟩

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* $((\mu \star \nu) \star \pi)$
    ⟨*proof*⟩

  interpretation $\mu\nu\pi$: *identity-arrow-of-spans C* ‹$\mu \star \nu \star \pi$›
    ⟨*proof*⟩
  interpretation $\mu\nu$-π: *identity-arrow-of-spans C* ‹$(\mu \star \nu) \star \pi$›
    ⟨*proof*⟩

  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 $\mu\nu$-$\pi.leg0 = \pi.leg0 \cdot Prj_0$
  ⟨*proof*⟩

  lemma *leg1-composite*:
  shows $\mu\nu\pi.leg1 = \mu.leg1 \cdot Prj_1$
  and $\mu\nu$-$\pi.leg1 = \mu.leg1 \cdot Prj_{11}$
  ⟨*proof*⟩

  definition *chine-assoc*
  where *chine-assoc* ≡
      $\langle Prj_{11}\ [\![\mu.leg0,\ \nu.leg1 \cdot \mathrm{p}_1[\nu.leg0,\ \pi.leg1]]\!]\ \langle Prj_{01}\ [\![\nu.leg0,\ \pi.leg1]\!]\ Prj_0\rangle\rangle$

  definition *chine-assoc′*
  where *chine-assoc′* ≡
      $\langle\langle Prj_1\ [\![\mu.leg0,\ \nu.leg1]\!]\ Prj_{10}\rangle\ [\![\nu.leg0 \cdot \mathrm{p}_0[\mu.leg0,\ \nu.leg1],\ \pi.leg1]\!]\ Prj_{00}\rangle$

lemma *chine-composite*:
shows $\mu\nu\text{-}\pi.chine = \nu.leg0 \cdot \mu\nu.prj_0 \downdownarrows \pi.leg1$
and $\mu\nu\pi.chine = \mu.leg0 \downdownarrows \nu.leg1 \cdot \nu\pi.prj_1$
⟨*proof*⟩

lemma *prj-in-hom* [*intro*]:
shows «$Prj_{11} : \mu\nu\text{-}\pi.chine \to_C \mu.apex$»
and «$Prj_{01} : \mu\nu\text{-}\pi.chine \to_C \nu.apex$»
and «$Prj_0 : \mu\nu\text{-}\pi.chine \to_C \pi.apex$»
and «$Prj_1 : \mu\nu\pi.chine \to_C \mu.apex$»
and «$Prj_{10} : \mu\nu\pi.chine \to_C \nu.apex$»
and «$Prj_{00} : \mu\nu\pi.chine \to_C \pi.apex$»
⟨*proof*⟩

lemma *prj-simps* [*simp*]:
shows $C.arr\ Prj_{11}$
and $C.arr\ Prj_{01}$
and $C.arr\ Prj_0$
and $C.dom\ Prj_{11} = \mu\nu\text{-}\pi.chine$
and $C.dom\ Prj_{01} = \mu\nu\text{-}\pi.chine$
and $C.dom\ Prj_0 = \mu\nu\text{-}\pi.chine$
and $C.cod\ Prj_{11} = \mu.apex$
and $C.cod\ Prj_{01} = \nu.apex$
and $C.cod\ Prj_0 = \pi.apex$
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 $C.cod\ Prj_1 = \mu.apex$
and $C.cod\ Prj_{10} = \nu.apex$
and $C.cod\ Prj_{00} = \pi.apex$
⟨*proof*⟩

lemma *chine-assoc-props*:
shows «$chine\text{-}assoc : \mu\nu\text{-}\pi.chine \to_C \mu\nu\pi.chine$»
and $Prj_1 \cdot chine\text{-}assoc = Prj_{11}$
and $Prj_{10} \cdot chine\text{-}assoc = Prj_{01}$
and $Prj_{00} \cdot chine\text{-}assoc = Prj_0$
⟨*proof*⟩

lemma *chine-assoc-in-hom* [*intro*]:
shows «$chine\text{-}assoc : \mu\nu\text{-}\pi.chine \to_C \mu\nu\pi.chine$»
⟨*proof*⟩

lemma *prj-chine-assoc* [*simp*]:

shows $Prj_1 \cdot \mathit{chine\text{-}assoc} = Prj_{11}$
and $Prj_{10} \cdot \mathit{chine\text{-}assoc} = Prj_{01}$
and $Prj_{00} \cdot \mathit{chine\text{-}assoc} = Prj_0$
⟨*proof*⟩

lemma *chine-assoc′-props*:
shows «$\mathit{chine\text{-}assoc}'$: $\mu\nu\pi.\mathit{chine} \to_C \mu\nu\text{-}\pi.\mathit{chine}$»
and $Prj_{11} \cdot \mathit{chine\text{-}assoc}' = Prj_1$
and $Prj_{01} \cdot \mathit{chine\text{-}assoc}' = Prj_{10}$
and $Prj_0 \cdot \mathit{chine\text{-}assoc}' = Prj_{00}$
⟨*proof*⟩

lemma *chine-assoc′-in-hom* [*intro*]:
shows «$\mathit{chine\text{-}assoc}'$: $\mu\nu\pi.\mathit{chine} \to_C \mu\nu\text{-}\pi.\mathit{chine}$»
⟨*proof*⟩

lemma *prj-chine-assoc′* [*simp*]:
shows $Prj_{11} \cdot \mathit{chine\text{-}assoc}' = Prj_1$
and $Prj_{01} \cdot \mathit{chine\text{-}assoc}' = Prj_{10}$
and $Prj_0 \cdot \mathit{chine\text{-}assoc}' = Prj_{00}$
⟨*proof*⟩

lemma *prj-joint-monic*:
assumes «h: $a \to_C \mu\nu\text{-}\pi.\mathit{chine}$» **and** «$h'$: $a \to_C \mu\nu\text{-}\pi.\mathit{chine}$»
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*⟩

lemma *prj′-joint-monic*:
assumes «h: $a \to_C \mu\nu\pi.\mathit{chine}$» **and** «$h'$: $a \to_C \mu\nu\pi.\mathit{chine}$»
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*⟩

lemma *chine-assoc-inverse*:
shows $C.\mathit{inverse\text{-}arrows}\ \mathit{chine\text{-}assoc}\ \mathit{chine\text{-}assoc}'$
⟨*proof*⟩

end

context *three-composable-arrows-of-spans*
begin

interpretation V: *category vcomp*
⟨*proof*⟩
interpretation H: *horizontal-homs vcomp src trg*
⟨*proof*⟩

interpretation *dom-μ*: *arrow-of-spans C* ‹*dom μ*›

⟨*proof*⟩
interpretation *dom-ν*: *arrow-of-spans C ‹dom ν›*
⟨*proof*⟩
interpretation *dom-π*: *arrow-of-spans C ‹dom π›*
⟨*proof*⟩
interpretation *doms*: *three-composable-identity-arrows-of-spans C prj0 prj1*
‹dom μ› ‹dom ν› ‹dom π›
⟨*proof*⟩

interpretation *cod-μ*: *arrow-of-spans C ‹cod μ›*
⟨*proof*⟩
interpretation *cod-ν*: *arrow-of-spans C ‹cod ν›*
⟨*proof*⟩
interpretation *cod-π*: *arrow-of-spans C ‹cod π›*
⟨*proof*⟩
interpretation *cods*: *three-composable-identity-arrows-of-spans C prj0 prj1*
‹cod μ› ‹cod ν› ‹cod π›
⟨*proof*⟩

interpretation *μνπ*: *arrow-of-spans C ‹μ ⋆ ν ⋆ π›*
⟨*proof*⟩

interpretation *μν-π*: *arrow-of-spans C ‹(μ ⋆ ν) ⋆ π›*
⟨*proof*⟩

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*⟩

lemma *chine-composite-in-hom* [*intro*]:
shows «*μν-π.chine* : *Chn* ((*dom μ* ⋆ *dom ν*) ⋆ *dom π*) →$_C$ *Chn* ((*cod μ* ⋆ *cod ν*) ⋆ *cod π*)»
and «*μνπ.chine* : *Chn* (*dom μ* ⋆ *dom ν* ⋆ *dom π*) →$_C$ *Chn* (*cod μ* ⋆ *cod ν* ⋆ *cod π*)»
⟨*proof*⟩

lemma *cospan-μν*:
shows *C.cospan μ.dom.leg0 ν.dom.leg1*
⟨*proof*⟩

lemma *cospan-νπ*:
shows *C.cospan ν.dom.leg0 π.dom.leg1*
⟨*proof*⟩

lemma *commutativities*:
shows $\mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_{11} = \nu.cod.leg1 \cdot \nu.chine \cdot doms.Prj_{01}$

and $\pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_0 =$
$(\nu.cod.leg0 \cdot \mathrm{p}_0[\mu.cod.leg0,\ \nu.cod.leg1]) \cdot$
$\langle \mu.chine \cdot doms.Prj_{11}\ [\![\mu.cod.leg0,\ \nu.cod.leg1]\!]\ \nu.chine \cdot doms.Prj_{01} \rangle$
⟨*proof*⟩

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 $cods.Prj_0 \cdot \mu\nu\text{-}\pi.chine = \pi.chine \cdot doms.Prj_0$
⟨*proof*⟩

lemma *commutativities'*:
shows $\nu.cod.leg0 \cdot \nu.chine \cdot doms.Prj_{10} = \pi.cod.leg1 \cdot \pi.chine \cdot doms.Prj_{00}$
and $\mu.cod.leg0 \cdot \mu.chine \cdot doms.Prj_1 =$
$(\nu.cod.leg1 \cdot \mathrm{p}_1[\nu.cod.leg0,\ \pi.cod.leg1]) \cdot$
$\langle \nu.chine \cdot doms.Prj_{10}\ [\![\nu.cod.leg0,\ \pi.cod.leg1]\!]\ \pi.chine \cdot doms.Prj_{00} \rangle$
⟨*proof*⟩

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 $cods.Prj_{00} \cdot \mu\nu\pi.chine = \pi.chine \cdot doms.Prj_{00}$
⟨*proof*⟩

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*⟩

end

context *span-bicategory*
begin

abbreviation (*input*) $assoc_{SB}$
where $assoc_{SB}\ f\ g\ h \equiv (\!|Chn = three\text{-}composable\text{-}identity\text{-}arrows\text{-}of\text{-}spans.chine\text{-}assoc$
$C\ prj0\ prj1\ f\ g\ h,$
$Dom = Dom\ ((f \star g) \star h),\ Cod = Cod\ (f \star g \star h)|\!)$

abbreviation (*input*) $assoc'_{SB}$
where $assoc'_{SB}\ f\ g\ h \equiv (\!|Chn = three\text{-}composable\text{-}identity\text{-}arrows\text{-}of\text{-}spans.chine\text{-}assoc'$
$C\ prj0\ prj1\ f\ g\ h,$
$Dom = Cod\ (f \star g \star h),\ Cod = Dom\ ((f \star g) \star h)|\!)$

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 «$assoc_{SB}\ f\ g\ h : (f \star g) \star h \Rightarrow f \star g \star h$»
and *src* $(assoc'_{SB}\ f\ g\ h)$ = *src h* **and** *trg* $(assoc'_{SB}\ f\ g\ h)$ = *trg f*
and «$assoc'_{SB}\ f\ g\ h : f \star g \star h \Rightarrow (f \star g) \star h$»

⟨*proof*⟩

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*»
⟨*proof*⟩

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*»
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

interpretation α: *transformation-by-components VVV.comp vcomp HoHV HoVH*
‹λ*fgh*. $assoc_{SB}$ (*fst fgh*) (*fst* (*snd fgh*)) (*snd* (*snd fgh*))›
⟨*proof*⟩

definition *assoc* (‹a[-, -, -]›)
where *assoc* ≡ λμ ν π. α.*map* (μ, ν, π)

abbreviation (*input*) α_{SB}
where α_{SB} ≡ λμνπ. *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*
⟨*proof*⟩

lemma *natural-transformation-α*:
shows *natural-transformation VVV.comp vcomp HoHV HoVH* α_{SB}

⟨*proof*⟩

interpretation α: *natural-transformation VVV.comp vcomp HoHV HoVH* α_{SB}
⟨*proof*⟩

sublocale α: *natural-isomorphism VVV.comp vcomp HoHV HoVH* α_{SB}
⟨*proof*⟩

lemma *natural-isomorphism-*α:
shows *natural-isomorphism VVV.comp vcomp HoHV HoVH* α_{SB}
⟨*proof*⟩

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*
⟨*proof*⟩

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* ‹$\mu \star \nu \star \pi$›

⟨*proof*⟩
interpretation *HHμνπ*: *identity-arrow-of-spans C ‹(μ ⋆ ν) ⋆ π›*
⟨*proof*⟩
interpretation *HνHπϱ*: *identity-arrow-of-spans C ‹ν ⋆ π ⋆ ϱ›*
⟨*proof*⟩
interpretation *HHνπϱ*: *identity-arrow-of-spans C ‹(ν ⋆ π) ⋆ ϱ›*
⟨*proof*⟩

interpretation *HμHνHπϱ*: *arrow-of-spans C ‹μ ⋆ ν ⋆ π ⋆ ϱ›*
⟨*proof*⟩
interpretation *HμHHνπϱ*: *arrow-of-spans C ‹μ ⋆ (ν ⋆ π) ⋆ ϱ›*
⟨*proof*⟩
interpretation *HHμνHπϱ*: *arrow-of-spans C ‹(μ ⋆ ν) ⋆ π ⋆ ϱ›*
⟨*proof*⟩
interpretation *HHμHνπϱ*: *arrow-of-spans C ‹(μ ⋆ ν ⋆ π) ⋆ ϱ›*
⟨*proof*⟩
interpretation *HHHμνπϱ*: *arrow-of-spans C ‹((μ ⋆ ν) ⋆ π) ⋆ ϱ›*
⟨*proof*⟩

interpretation *assocμνπ*: *arrow-of-spans C ‹assoc$_{SB}$ μ ν π›*
⟨*proof*⟩
interpretation *assocνπϱ*: *arrow-of-spans C ‹assoc$_{SB}$ ν π ϱ›*
⟨*proof*⟩

interpretation *μ-νπ*: *two-composable-identity-arrows-of-spans C prj0 prj1 μ ‹ν ⋆ π›*
⟨*proof*⟩
interpretation *μν-π*: *two-composable-identity-arrows-of-spans C prj0 prj1 ‹μ ⋆ ν› π*
⟨*proof*⟩
interpretation *ν-πϱ*: *two-composable-identity-arrows-of-spans C prj0 prj1 ν ‹π ⋆ ϱ›*
⟨*proof*⟩
interpretation *νπ-ϱ*: *two-composable-identity-arrows-of-spans C prj0 prj1 ‹ν ⋆ π› ϱ*
⟨*proof*⟩

interpretation *μ-νπ-ϱ*: *three-composable-identity-arrows-of-spans C prj0 prj1 μ ‹ν ⋆ π› ϱ*
⟨*proof*⟩
interpretation *μ-ν-πϱ*: *three-composable-identity-arrows-of-spans C prj0 prj1 μ ν ‹π ⋆ ϱ›*
⟨*proof*⟩
interpretation *μν-π-ϱ*: *three-composable-identity-arrows-of-spans C prj0 prj1 ‹μ ⋆ ν› π ϱ*
⟨*proof*⟩

lemma *chines-eq*:
shows *HμHHνπϱ.chine = μ.leg0 ↓↓ HHνπϱ.leg1*
and *HHμHνπϱ.chine = assocμνπ.cod.leg0 ↓↓ ϱ.leg1*
and *HμHνHπϱ.chine = μ.leg0 ↓↓ HνHπϱ.leg1*
⟨*proof*⟩

lemma *cospan-μ0-HνHπϱ1*:
shows *C.cospan μ.leg0 HνHπϱ.leg1*

⟨*proof*⟩

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*⟩

lemma *chine-composites*:
shows $Chn\ (\mu \star assoc_{SB}\ \nu\ \pi\ \varrho) = chine\text{-}hcomp\ \mu\ (assoc_{SB}\ \nu\ \pi\ \varrho)$
and $Chn\ (assoc_{SB}\ \mu\ (\nu \star \pi)\ \varrho) = \mu\text{-}\nu\pi\text{-}\varrho.chine\text{-}assoc$
and $Chn\ (assoc_{SB}\ \mu\ \nu\ \pi \star \varrho) = chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho$
and $Chn\ (assoc_{SB}\ \mu\ \nu\ (\pi \star \varrho)) = \mu\text{-}\nu\text{-}\pi\varrho.chine\text{-}assoc$
and $Chn\ (assoc_{SB}\ (\mu \star \nu)\ \pi\ \varrho) = \mu\nu\text{-}\pi\text{-}\varrho.chine\text{-}assoc$
⟨*proof*⟩

lemma *prj-in-homs* [*intro*, *simp*]:
shows «$\mathrm{p}_0[\mu.leg0,\ HH\nu\pi\varrho.leg1] : H\mu HH\nu\pi\varrho.chine \to_C HH\nu\pi\varrho.chine$»
and «$\mathrm{p}_1[\mu.leg0,\ H\nu H\pi\varrho.leg1] : H\mu H\nu H\pi\varrho.chine \to_C \mu.apex$»
and «$\mathrm{p}_1[assoc\mu\nu\pi.cod.leg0,\ \varrho.cod.leg1] : HH\mu H\nu\pi\varrho.chine \to_C H\mu H\nu\pi.chine$»
and «$\mathrm{p}_0[HH\mu\nu\pi.leg0,\ \varrho.leg1] : HHH\mu\nu\pi\varrho.chine \to_C \varrho.chine$»
and «$\mathrm{p}_1[HH\mu\nu\pi.leg0,\ \varrho.leg1] : HHH\mu\nu\pi\varrho.chine \to_C HH\mu\nu\pi.chine$»
and «$\mathrm{p}_1[\nu\pi.leg0 \cdot \mu\text{-}\nu\pi.prj_0,\ \varrho.leg1] : HH\mu H\nu\pi\varrho.chine \to_C H\mu H\nu\pi.chine$»
and «$\mathrm{p}_1[assoc\mu\nu\pi.dom.leg0,\ \varrho.leg1] : HHH\mu\nu\pi\varrho.chine \to_C HH\mu\nu\pi.chine$»
and «$\mu\text{-}\nu\pi.prj_0 : H\mu H\nu\pi.chine \to_C \nu\pi.apex$»
⟨*proof*⟩

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 «$chine\text{-}hcomp\ \mu\ (assoc_{SB}\ \nu\ \pi\ \varrho) : H\mu HH\nu\pi\varrho.chine \to_C H\mu H\nu H\pi\varrho.chine$»
and «$chine\text{-}hcomp\ (assoc_{SB}\ \mu\ \nu\ \pi)\ \varrho : HHH\mu\nu\pi\varrho.chine \to_C HH\mu H\nu\pi\varrho.chine$»
⟨*proof*⟩

lemma *commutative-squares* [*intro*, *simp*]:
shows $C.commutative\text{-}square\ \nu\pi.leg0\ \varrho.leg1\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01}\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_0$
and $C.commutative\text{-}square\ \nu.leg0\ \pi\varrho.leg1\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_{01}\ \mu\text{-}\nu\text{-}\pi\varrho.Prj_0$
and $C.commutative\text{-}square\ \mathrm{p}_0[\mu.cod.leg0,\ assoc\nu\pi\varrho.cod.leg1]\ assoc\nu\pi\varrho.chine$
$(chine\text{-}hcomp\ \mu\ (assoc_{SB}\ \nu\ \pi\ \varrho))\ \mathrm{p}_0[\mu.leg0,\ assoc\nu\pi\varrho.dom.leg1]$
and $C.commutative\text{-}square\ \mathrm{p}_1[\mu.cod.leg0,\ assoc\nu\pi\varrho.cod.leg1]\ \mu.chine$
$(chine\text{-}hcomp\ \mu\ (assoc_{SB}\ \nu\ \pi\ \varrho))\ \mathrm{p}_1[\mu.leg0,\ assoc\nu\pi\varrho.dom.leg1]$
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 $C.commutative\text{-}square\ \mu.leg0\ (\nu\pi.leg1 \cdot \nu\pi\text{-}\varrho.prj_1)\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_{11}$
$\langle \mu\text{-}\nu\pi\text{-}\varrho.Prj_{01}\ [\![\nu\pi.leg0,\ \varrho.leg1]\!]\ \mu\text{-}\nu\pi\text{-}\varrho.Prj_0 \rangle$

and *C.commutative-square* μ.*leg0* (ν.*leg1* $\cdot$ ν-$\pi\varrho$.*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*⟩

lemma *chine-pentagon*:
shows *Chn* ((μ $\star$ *assoc*$_{SB}$ ν π ϱ) $\cdot$ *assoc*$_{SB}$ μ (ν $\star$ π) ϱ $\cdot$ (*assoc*$_{SB}$ μ ν π $\star$ ϱ)) =
Chn (*assoc*$_{SB}$ μ ν (π $\star$ ϱ) $\cdot$ *assoc*$_{SB}$ (μ $\star$ ν) π ϱ)
⟨*proof*⟩

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* $\star$ α_{SB} (*g*, *h*, *k*)) $\cdot$ α_{SB} (*f*, *g* $\star$ *h*, *k*) $\cdot$ (α_{SB} (*f*, *g*, *h*) $\star$ *k*) =
α_{SB} (*f*, *g*, *h* $\star$ *k*) $\cdot$ α_{SB} (*f* $\star$ *g*, *h*, *k*)
⟨*proof*⟩

lemma *extends-to-bicategory*:
shows *bicategory vcomp hcomp assoc unit src trg*
⟨*proof*⟩

sublocale *bicategory vcomp hcomp assoc unit src trg*
⟨*proof*⟩

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*⟩

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 $\mathrm{r}[f] = \langle\!|\mathit{Chn} = \mathrm{p}_1[\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))],$
$\mathit{Dom} = \langle\!|\mathit{Leg0} = \mathrm{p}_0[\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))],$
$\mathit{Leg1} = \mathit{Leg1}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_1[\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))]|\rangle,$
$\mathit{Cod} = \mathit{Cod}\ f|\rangle$
⟨*proof*⟩

lemma *lunit-ide-eq*:
assumes *ide f*
shows $\mathrm{l}[f] = \langle\!|\mathit{Chn} = \mathrm{p}_0[\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)],$
$\mathit{Dom} = \langle\!|\mathit{Leg0} = \mathit{Leg0}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_0[\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)],$
$\mathit{Leg1} = \mathrm{p}_1[\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)]|\rangle,$
$\mathit{Cod} = \mathit{Cod}\ f|\rangle$
⟨*proof*⟩

lemma *runit′-ide-eq*:
assumes *ide f*
shows $\mathrm{r}^{-1}[f] = \langle\!|\mathit{Chn} = \langle \mathit{Chn}\ f\ [\![\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))]\!]\ \mathit{Leg0}\ (\mathit{Dom}\ f)\rangle,$
$\mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = \langle\!|\mathit{Leg0} = \mathrm{p}_0[\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))],$
$\mathit{Leg1} = \mathit{Leg1}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_1[\mathit{Leg0}\ (\mathit{Dom}\ f),\ \mathit{C.cod}\ (\mathit{Leg0}\ (\mathit{Dom}\ f))]|\rangle|\rangle$
⟨*proof*⟩

lemma *lunit′-ide-eq*:
assumes *ide f*
shows $\mathrm{l}^{-1}[f] = \langle\!|\mathit{Chn} = \langle \mathit{Leg1}\ (\mathit{Dom}\ f)\ [\![\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)]\!]\ \mathit{Chn}\ f\rangle,$
$\mathit{Dom} = \mathit{Cod}\ f,$
$\mathit{Cod} = \langle\!|\mathit{Leg0} = \mathit{Leg0}\ (\mathit{Dom}\ f) \cdot \mathrm{p}_0[\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)],$
$\mathit{Leg1} = \mathrm{p}_1[\mathit{C.cod}\ (\mathit{Leg1}\ (\mathit{Dom}\ f)),\ \mathit{Leg1}\ (\mathit{Dom}\ f)]|\rangle|\rangle$
⟨*proof*⟩

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*
⟨*proof*⟩
interpretation *g*: *identity-arrow-of-spans C g*
⟨*proof*⟩

interpretation *gf*: *two-composable-identity-arrows-of-spans C prj0 prj1 g f*
⟨*proof*⟩
interpretation *fg*: *two-composable-identity-arrows-of-spans C prj0 prj1 f g*
⟨*proof*⟩

interpretation *fgf*: *three-composable-identity-arrows-of-spans C prj0 prj1 f g f* ⟨*proof*⟩
interpretation *gfg*: *three-composable-identity-arrows-of-spans C prj0 prj1 g f g* ⟨*proof*⟩

interpretation η: *arrow-of-spans C* η
⟨*proof*⟩
interpretation ε: *arrow-of-spans C* ε
⟨*proof*⟩

lemma *chine-unit-in-hom*:
shows «$\eta.chine : f.dsrc \rightarrow_C g.leg0 \downarrow\downarrow f.leg1$»
⟨*proof*⟩

lemma *chine-counit-in-hom*:
shows «$\varepsilon.chine : f.leg0 \downarrow\downarrow g.leg1 \rightarrow_C f.dtrg$»
⟨*proof*⟩

lemma η*-leg-simps*:
shows $\eta.dom.leg0 = f.dsrc$ **and** $\eta.dom.leg1 = f.dsrc$
and $\eta.cod.leg0 = gf.leg0$ **and** $\eta.cod.leg1 = gf.leg1$
⟨*proof*⟩

lemma ε*-leg-simps*:
shows $\varepsilon.cod.leg0 = f.dtrg$ **and** $\varepsilon.cod.leg1 = f.dtrg$
and $\varepsilon.dom.leg0 = fg.leg0$ **and** $\varepsilon.dom.leg1 = fg.leg1$
⟨*proof*⟩

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*⟩

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 $unit\text{-}for\text{-}map\ f \equiv (\!|Chn = \langle C.inv\ (Leg0\ (Dom\ f))$
$[\![Leg1\ (Dom\ f),\ Leg1\ (Dom\ f)]\!]$
$C.inv\ (Leg0\ (Dom\ f))\rangle,$
$Dom = Dom\ (src\ f),$
$Cod = Dom\ (hcomp\ (adjoint\text{-}of\text{-}map\ f)\ f)|\!)$

abbreviation *counit-for-map*
where $counit\text{-}for\text{-}map\ f \equiv (\!|Chn = Leg1\ (Dom\ f) \cdot \mathrm{p}_0[Leg0\ (Dom\ f),\ Leg0\ (Dom\ f)],$
$Dom = Dom\ (hcomp\ f\ (adjoint\text{-}of\text{-}map\ f)),$
$Cod = Dom\ (trg\ f)|\!)$

lemma *is-left-adjoint-char*:
shows $is\text{-}left\text{-}adjoint\ f \longleftrightarrow ide\ f \wedge C.iso\ (Leg0\ (Dom\ f))$
and $is\text{-}left\text{-}adjoint\ f \Longrightarrow$
$adjunction\text{-}in\text{-}bicategory\ vcomp\ hcomp\ assoc\ unit\ src\ trg\ f$
$(adjoint\text{-}of\text{-}map\ f)\ (unit\text{-}for\text{-}map\ f)\ (counit\text{-}for\text{-}map\ f)$
⟨*proof*⟩

end

end

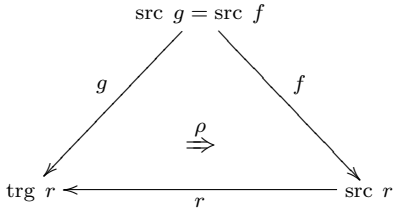
2.2 Tabulations

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.



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 ⇒ r ⋆ f»
begin

  lemma *base-in-hom* [*intro*]:
  shows «r : *src r* → *trg r*» **and** «r : r ⇒ r»
    ⟨*proof*⟩

  lemma *base-simps* [*simp*]:
  shows *ide r* **and** *arr r*
  and *dom r* = r **and** *cod r* = r
    ⟨*proof*⟩

  lemma *tab-in-hom* [*intro*]:
  shows «ϱ : *src f* → *trg r*» **and** «ϱ : g ⇒ r ⋆ f»
    ⟨*proof*⟩

  lemma *ide-leg1*:
  shows *ide g*
    ⟨*proof*⟩

  lemma *leg1-in-hom* [*intro*]:
  shows «g : *src f* → *trg r*» **and** «g : g ⇒ g»
    ⟨*proof*⟩

  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*
⟨*proof*⟩

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* ⋆ *f*
⟨*proof*⟩

lemma *leg0-in-hom* [*intro*]:
shows «*f* : *src f* → *src r*» **and** «*f* : *f* ⇒ *f*»
⟨*proof*⟩

lemma *leg0-simps* [*simp*]:
shows *ide f* **and** *arr f*
and *trg f* = *src r*
and *dom f* = *f* **and** *cod f* = *f*
⟨*proof*⟩

The following function, which composes ϱ with a 2-cell «$\vartheta : f \star w \Rightarrow u$» 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 ϑ* ≡ (*r* ⋆ *ϑ*) · a[*r*, *f*, *w*] · (*ϱ* ⋆ *w*)

lemma *composite-cell-in-hom*:
assumes *ide w* **and** «*w* : *src u* → *src f*» **and** «*ϑ* : *f* ⋆ *w* ⇒ *u*»
shows «*composite-cell w ϑ* : *g* ⋆ *w* ⇒ *r* ⋆ *u*»
⟨*proof*⟩

We define some abbreviations for various combinations of conditions that occur in the hypotheses and conclusions of the tabulation axioms.

abbreviation (*input*) *uwϑω*
where *uwϑω u w ϑ ω* ≡ *ide w* ∧ «*ϑ* : *f* ⋆ *w* ⇒ *u*» ∧ «*ω* : *dom ω* ⇒ *r* ⋆ *u*»

abbreviation (*input*) *uwϑων*
where *uwϑων u w ϑ ω ν* ≡
ide w ∧ «*ϑ* : *f* ⋆ *w* ⇒ *u*» ∧ «*ν* : *dom ω* ⇒ *g* ⋆ *w*» ∧ *iso ν* ∧
(*r* ⋆ *ϑ*) · a[*r*, *f*, *w*] · (*ϱ* ⋆ *w*) · *ν* = *ω*

abbreviation (*input*) *uwϑw′ϑ′β*
where *uwϑw′ϑ′β u w ϑ w′ ϑ′ β* ≡
ide u ∧ *ide w* ∧ *ide w′* ∧
«*ϑ* : *f* ⋆ *w* ⇒ *u*» ∧ «*ϑ′* : *f* ⋆ *w′* ⇒ *u*» ∧ «*β* : *g* ⋆ *w* ⇒ *g* ⋆ *w′*» ∧
(*r* ⋆ *ϑ*) · a[*r*, *f*, *w*] · (*ϱ* ⋆ *w*) = (*r* ⋆ *ϑ′*) · a[*r*, *f*, *w′*] · (*ϱ* ⋆ *w′*) · *β*

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

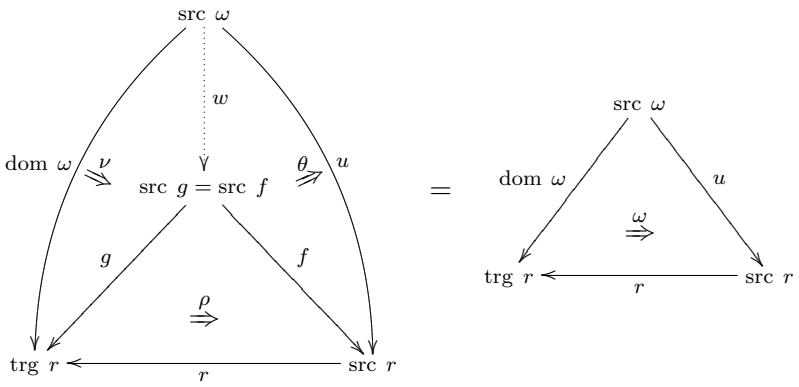
weak form of the biuniversal property that only applies to $(u,\ \omega,\ v)$ for which u is a map. The second notion, which they call "wide tabulation", concerns arbitrary $(f,\ \varrho,\ g)$, and assumes a strong form of the biuniversal property that applies to all $(u,\ \omega,\ v)$. On its face, 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*: $\bigwedge u\ \omega.$
  $[\![\ ide\ u;\ «\omega : dom\ \omega \Rightarrow r \star u»\ ]\!] \Longrightarrow$
  $\exists w\ \vartheta\ \nu.\ ide\ w \wedge «\vartheta : f \star w \Rightarrow u» \wedge «\nu : dom\ \omega \Rightarrow g \star w» \wedge iso\ \nu \wedge$
    $composite\text{-}cell\ w\ \vartheta \cdot \nu = \omega$
and *T2*: $\bigwedge u\ w\ w'\ \vartheta\ \vartheta'\ \beta.$
  $[\![\ ide\ w;\ ide\ w';\ «\vartheta : f \star w \Rightarrow u»;\ «\vartheta' : f \star w' \Rightarrow u»;\ «\beta : g \star w \Rightarrow g \star w'»;$
    $composite\text{-}cell\ w\ \vartheta = composite\text{-}cell\ w'\ \vartheta' \cdot \beta\ ]\!] \Longrightarrow$
  $\exists!\gamma.\ «\gamma : w \Rightarrow w'» \wedge \beta = g \star \gamma \wedge \vartheta = \vartheta' \cdot (f \star \gamma)$

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*⟩

lemma *ϱ-in-terms-of-rep*:
shows $\varrho = (T0.trnr_\varepsilon\ r\ \varrho \star f) \cdot T0.trnr_\eta\ g\ (g \star f^*)$
⟨*proof*⟩

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 ϑ = *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* ∧ «*ϑ* : *f* ⋆ *w* ⇒ *u*»
begin

  lemma *ide-u*:
  shows *ide u*
    ⟨*proof*⟩

  lemma *u-in-hom* [*intro*]:
  shows «*u* : *src u* → *src r*»
    ⟨*proof*⟩

  lemma *u-simps* [*simp*]:
  shows *ide u* **and** *arr u*
  and *trg u* = *src r*
  and *dom u* = *u* **and** *cod u* = *u*
    ⟨*proof*⟩

  lemma *ide-w*:
  shows *ide w*
    ⟨*proof*⟩

  lemma *w-in-hom* [*intro*]:
  shows «*w* : *src u* → *src f*» **and** «*w* : *w* ⇒ *w*»
  ⟨*proof*⟩

  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*
    ⟨*proof*⟩

  lemma *ϑ-in-hom* [*intro*]:
  shows «*ϑ* : *src u* → *src r*» **and** «*ϑ* : *f* ⋆ *w* ⇒ *u*»
  ⟨*proof*⟩

  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*
    ⟨*proof*⟩

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*⟩

  lemma *ω-simps* [*simp*]:
  shows *arr ω* **and** *src ω* = *src w* **and** *trg ω* = *trg r*
  and *cod ω* = *r* ⋆ *u*
    ⟨*proof*⟩

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*⟩

  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*
    ⟨*proof*⟩

  sublocale *uwϑω*
  ⟨*proof*⟩

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 $\varrho :: 'a$
and $f :: 'a$
and $g :: 'a$
and $u :: 'a$
and $w :: 'a$
and $\vartheta :: 'a$
and $w' :: 'a$
and $\vartheta' :: 'a$

locale $uw\vartheta w'\vartheta'\gamma =$
$uw\vartheta w'\vartheta' +$
fixes $\gamma :: 'a$
assumes γ*-in-vhom*: «$\gamma : w \Rightarrow w'$»
and $\vartheta = \vartheta' \cdot (f \star \gamma)$
begin

lemma γ*-in-hom* [*intro*]:
shows «$\gamma : src\ u \rightarrow src\ f$» **and** «$\gamma : w \Rightarrow w'$»
⟨*proof*⟩

lemma γ*-simps* [*simp*]:
shows *arr* γ
and *src* $\gamma = src\ u$ **and** *trg* $\gamma = src\ f$
and *dom* $\gamma = w$ **and** *cod* $\gamma = w'$
⟨*proof*⟩

end

locale $uw\vartheta w'\vartheta'\beta =$
$uw\vartheta w'\vartheta' +$
fixes $\beta :: 'a$
assumes $uw\vartheta w'\vartheta'\beta$: $uw\vartheta w'\vartheta'\beta\ u\ w\ \vartheta\ w'\ \vartheta'\ \beta$
begin

lemma β*-in-hom* [*intro*]:
shows «$\beta : src\ u \rightarrow trg\ r$» **and** «$\beta : g \star w \Rightarrow g \star w'$»
⟨*proof*⟩

lemma β*-simps* [*simp*]:
shows *arr* β **and** *src* $\beta = src\ u$ **and** *trg* $\beta = trg\ r$
and *dom* $\beta = g \star w$ **and** *cod* $\beta = g \star w'$
⟨*proof*⟩

end

2.2.2 Tabulations yield Factorizations

If $(f,\ \varrho,\ g)$ 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* ⟨*proof*⟩
notation *E.eval* (‹⦃-⦄›)

lemma *satisfies-T0*:
shows *is-left-adjoint f*
⟨*proof*⟩

sublocale *tabulation-data-with-T0*
⟨*proof*⟩
sublocale *narrow-tabulation*
⟨*proof*⟩

end

A tabulation $(f,\ \varrho,\ g)$ 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* ⟨*proof*⟩
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*$_\varepsilon$ *r* ϱ : $g \star f^* \Rightarrow r$» **and** *iso* (*T0.trnr*$_\varepsilon$ *r* ϱ)
⟨*proof*⟩

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** «ψ : $g \star f^* \Rightarrow r$» **and** *iso* ψ
⟨*proof*⟩

end

It is natural to ask whether if «ψ : $g \star f^* \Rightarrow r$» 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 «ψ : $g \star f^* \Rightarrow r$» 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*⟩

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 «φ : $f \Rightarrow f'$» **and** *iso* φ
shows *tabulation V H* a i *src trg r* ((*r* ⋆ φ) · ϱ) f' *g*
⟨*proof*⟩

Similarly, tabulations are preserved under composition of an isomorphism with the "output leg".

lemma *preserved-by-output-iso*:
assumes «φ : $g' \Rightarrow g$» **and** *iso* φ
shows *tabulation V H* a i *src trg r* ($\varrho \cdot \varphi$) *f g*′
⟨*proof*⟩

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*⟩

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 $\exists \varrho$. *tabulation* V H a i *src trg* $(g \star f^*)$ ϱ f g
shows *tabulation* V H a i *src trg* $(g \star f^*)$ $(trnr_\eta\ g\ (g \star f^*))$ f g
⟨*proof*⟩

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' → *src* f», «w' : *src* f → *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*⟩

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* $r\ \varrho'\ f'\ g'$
shows $\exists\, w\ w'\ \varphi\ \psi\ \vartheta\ \nu\ \vartheta'\ \nu'.$
$\quad$ *equivalence-in-bicategory* $V\ H$ a i *src trg* $w'\ w\ \psi\ \varphi \wedge$
$\quad$ «w : *src* $f \rightarrow$ *src* f'» $\wedge$ «w' : *src* $f' \rightarrow$ *src* f» $\wedge$
$\quad$ «ϑ : $f' \star w \Rightarrow f$» $\wedge$ «ν : $g \Rightarrow g' \star w$» $\wedge$ *iso* $\nu \wedge$
$\quad$ $\varrho = (r \star \vartheta) \cdot \mathrm{a}[r, f', w] \cdot (\varrho' \star w) \cdot \nu \wedge$
$\quad$ «ϑ' : $f \star w' \Rightarrow f'$» $\wedge$ «ν' : $g' \Rightarrow g \star w'$» $\wedge$ *iso* $\nu' \wedge$
$\quad$ $\varrho' = (r \star \vartheta') \cdot \mathrm{a}[r, f, w'] \cdot (\varrho \star w') \cdot \nu'$
⟨*proof*⟩

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* $(\cdot_C)\ (\star_C)\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ r\ \varrho\ f\ g$
shows *tabulation* $(\cdot_D)\ (\star_D)\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ r)\ (D.inv\ (\Phi\ (r, f)) \cdot_D F\ \varrho)\ (F\ f)\ (F\ g)$
⟨*proof*⟩

lemma *reflects-tabulation*:
assumes $C.ide\ r$ **and** $C.ide\ f$ **and** «ϱ : $g \Rightarrow_C r \star_C f$»
assumes *tabulation* $V_D\ H_D\ \mathrm{a}_D\ \mathrm{i}_D\ src_D\ trg_D\ (F\ r)\ (D.inv\ (\Phi\ (r, f)) \cdot_D F\ \varrho)\ (F\ f)\ (F\ g)$
shows *tabulation* $V_C\ H_C\ \mathrm{a}_C\ \mathrm{i}_C\ src_C\ trg_C\ r\ \varrho\ f\ g$
⟨*proof*⟩

end

end

2.3 Bicategories of Spans

theory *BicategoryOfSpans*
imports *Category3.ConcreteCategory IsomorphismClass CanonicalIsos EquivalenceOfBicategories*
$\quad$ *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* μ $\wedge$ *iso* μ' $\wedge$ $\mu = \mu'$

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*⟩

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** *ϱ* ≡ ⦇*Chn* = ⟨*ra* ⟦*r0*, *r0*⟧ *ra*⟩,
Dom = ⦇*Leg0* = *ra*, *Leg1* = *r1*⦈,
Cod = ⦇*Leg0* = *k0*, *Leg1* = *r1* · *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*⟩

end

context *span-bicategory*
begin

interpretation *chosen-right-adjoints vcomp hcomp assoc unit src trg* ⟨*proof*⟩
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*⟩

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
⟨*proof*⟩

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* ⋆ *f**) *r*
shows ∃ ϱ. *tabulation V H* a i *src trg r* ϱ *f g*
⟨*proof*⟩

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* ⋆ *w*)
shows *is-left-adjoint w*
⟨*proof*⟩

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* ⋆ *f** ≅ *r*

definition tab_0
where $tab_0\ r \equiv$ *SOME f*. ∃ *g*. *isomorphic-rep* (*iso-class-rep* ⟦*r*⟧) *f g*

definition tab_1

where $tab_1\ r \equiv SOME\ g.\ isomorphic\text{-}rep\ (iso\text{-}class\text{-}rep\ [\![r]\!])\ (tab_0\ r)\ g$

definition *rep*
where $rep\ r \equiv SOME\ \varphi.\ «\varphi : tab_1\ r \star (tab_0\ r)^* \Rightarrow r» \wedge iso\ \varphi$

lemma *rep-props*:
assumes *ide r*
shows $«rep\ r : tab_1\ r \star (tab_0\ r)^* \Rightarrow r»$ **and** $iso\ (rep\ r)$
and $r \cong iso\text{-}class\text{-}rep\ [\![r]\!]$
and $isomorphic\text{-}rep\ r\ (tab_0\ r)\ (tab_1\ r)$
and $tab_1\ r \star (tab_0\ r)^* \cong r$
⟨*proof*⟩

lemma tab_0*-in-hom* [*intro*]:
assumes *ide r*
shows $«tab_0\ r : src\ (tab_0\ r) \rightarrow src\ r»$
and $«tab_0\ r : tab_0\ r \Rightarrow tab_0\ r»$
⟨*proof*⟩

lemma tab_0*-simps* [*simp*]:
assumes *ide r*
shows $ide\ (tab_0\ r)$
and $is\text{-}left\text{-}adjoint\ (tab_0\ r)$
and $trg\ (tab_0\ r) = src\ r$
and $dom\ (tab_0\ r) = tab_0\ r$ **and** $cod\ (tab_0\ r) = tab_0\ r$
⟨*proof*⟩

lemma tab_1*-in-hom* [*intro*]:
assumes *ide r*
shows $«tab_1\ r : src\ (tab_0\ r) \rightarrow trg\ r»$
and $«tab_1\ r : tab_1\ r \Rightarrow tab_1\ r»$
⟨*proof*⟩

lemma tab_1*-simps* [*simp*]:
assumes *ide r*
shows $ide\ (tab_1\ r)$
and $is\text{-}left\text{-}adjoint\ (tab_1\ r)$
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** $cod\ (tab_1\ r) = tab_1\ r$
⟨*proof*⟩

lemma *rep-in-hom* [*intro*]:
assumes *ide r*
shows $«rep\ r : src\ r \rightarrow trg\ r»$
and $«rep\ r : tab_1\ r \star (tab_0\ r)^* \Rightarrow r»$
⟨*proof*⟩

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*) = $tab_1\ r \star (tab_0\ r)^*$ **and** *cod* (*rep r*) = *r*
⟨*proof*⟩

lemma *iso-rep*:
assumes *ide r*
shows *iso* (*rep r*)
⟨*proof*⟩

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 $tab_1\ r \star (tab_0\ r)^*$ 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* : $tab_1\ r \star (tab_0\ r)^* \Rightarrow r$».

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* ‹$tab_0\ r$›
⟨*proof*⟩
interpretation tab_1: *map-in-bicategory V H* a i *src trg* ‹$tab_1\ r$›
⟨*proof*⟩

A tabulation ($tab_0\ r$, *tab*, $tab_1\ r$) of *r* can be obtained as the adjoint transpose of the isomorphism «*rep r* : $(tab_1\ r) \star (tab_0\ r)^* \Rightarrow r$». 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 $tab \equiv tab_0.trnr_\eta\ (tab_1\ r)\ (rep\ r)$

In view of *BS2′*, the 1-cell $(tab_1\ r) \star (tab_0\ r)^*$ 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* : $(tab_1\ r) \star (tab_0\ r)^* \Rightarrow r$». 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*
$((tab_1\ r) \star (tab_0\ r)^*)\ (tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*))\ (tab_0\ r)\ (tab_1\ r)$
⟨*proof*⟩

lemma *tab-def-alt*:
shows $tab = (rep\ r \star tab_0\ r) \cdot tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*)$
and $(inv\ (rep\ r) \star tab_0\ r) \cdot tab = tab_0.trnr_\eta\ (tab_1\ r)\ ((tab_1\ r) \star (tab_0\ r)^*)$

⟨*proof*⟩

lemma *tab-is-tabulation*:
shows *tabulation V H* a i *src trg r tab* (*tab*$_0$ *r*) (*tab*$_1$ *r*)
⟨*proof*⟩

lemma *tab-in-hom* [*intro*]:
shows «*tab* : *src* (*tab*$_0$ *r*) → *trg r*»
and «*tab* : *tab*$_1$ *r* ⇒ *r* ⋆ *tab*$_0$ *r*»
⟨*proof*⟩

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* ⋆ *tab*$_0$ *r*
⟨*proof*⟩

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*›
⟨*proof*⟩

context *identity-in-bicategory-of-spans*
begin

interpretation *tab*$_0$: *map-in-bicategory V H* a i *src trg* ‹*tab*$_0$ *r*›
⟨*proof*⟩
interpretation *tab*$_1$: *map-in-bicategory V H* a i *src trg* ‹*tab*$_1$ *r*›
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *isomorphic-implies-same-tab*:
assumes *isomorphic r r*′
shows *tab*$_0$ *r* = *tab*$_0$ *r*′ **and** *tab*$_1$ *r* = *tab*$_1$ *r*′
⟨*proof*⟩

"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 $\exists \varrho\ f\ g.$ *is-left-adjoint* $f \wedge$ *is-left-adjoint* $g \wedge$ *tabulation* $V\ H$ a i *src trg* $r\ \varrho\ f\ g$
⟨*proof*⟩

end

2.3.5 Tabulations in a Bicategory of Spans

context *bicategory-of-spans*
begin

abbreviation *tab-of-ide*
where *tab-of-ide* $r \equiv$ *identity-in-bicategory-of-spans.tab* $V\ H$ a i *src trg* r

abbreviation prj_0
where $prj_0\ h\ k \equiv tab_0\ (k^* \star h)$

abbreviation prj_1
where $prj_1\ h\ k \equiv tab_1\ (k^* \star h)$

lemma *prj-in-hom* [*intro*]:
assumes *ide* h **and** *is-left-adjoint* k **and** *trg* $h =$ *trg* k
shows «$prj_0\ h\ k$: *src* $(prj_0\ h\ k) \rightarrow$ *src* h»
and «$prj_1\ h\ k$: *src* $(prj_0\ h\ k) \rightarrow$ *src* k»
and «$prj_0\ h\ k$: $prj_0\ h\ k \Rightarrow prj_0\ h\ k$»
and «$prj_1\ h\ k$: $prj_1\ h\ k \Rightarrow prj_1\ h\ k$»
⟨*proof*⟩

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* $(prj_1\ h\ k)$
⟨*proof*⟩

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* $(s_1 \star w)$
    ⟨*proof*⟩

  lemma *is-map*:
  shows *is-left-adjoint* w
    ⟨*proof*⟩

  lemma *hseq-leg*$_0$:
  shows *hseq* s_0 w
    ⟨*proof*⟩

  lemma *composite-with-leg0-is-map*:
  shows *is-left-adjoint* $(s_0 \star w)$
    ⟨*proof*⟩

  lemma *leg0-uniquely-isomorphic*:
  shows $s_0 \star w \cong r_0$
  and $\exists!\vartheta.\ «\vartheta : s_0 \star w \Rightarrow r_0»$
  ⟨*proof*⟩

  lemma *leg1-uniquely-isomorphic*:
  shows $r_1 \cong s_1 \star w$
  and $\exists!\nu.\ «\nu : r_1 \Rightarrow s_1 \star w»$

⟨*proof*⟩

definition *the-ϑ*
where *the-ϑ* ≡ *THE ϑ*. «*ϑ* : s_0 ⋆ *w* ⇒ r_0»

definition *the-ν*
where *the-ν* ≡ *THE ν*. «*ν* : r_1 ⇒ s_1 ⋆ *w*»

lemma *the-ϑ-props*:
shows «*the-ϑ* : s_0 ⋆ *w* ⇒ r_0» **and** *iso the-ϑ*
⟨*proof*⟩

lemma *the-ϑ-in-hom* [*intro*]:
shows «*the-ϑ* : *src* r_0 → *trg* r_0»
and «*the-ϑ* : s_0 ⋆ *w* ⇒ r_0»
⟨*proof*⟩

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 ⋆ *w* **and** *cod the-ϑ* = r_0
⟨*proof*⟩

lemma *the-ν-props*:
shows «*the-ν* : r_1 ⇒ s_1 ⋆ *w*» **and** *iso the-ν*
⟨*proof*⟩

lemma *the-ν-in-hom* [*intro*]:
shows «*the-ν* : *src* r_1 → *trg* r_1»
and «*the-ν* : r_1 ⇒ s_1 ⋆ *w*»
⟨*proof*⟩

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 ⋆ *w*
⟨*proof*⟩

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* ⇒ *'a* ⇒ *'a* (**infixr** ‹⋆› *53*)
and a :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›)

and i :: $'a \Rightarrow 'a$ ($\langle$i[-]$\rangle$)
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
$\langle$*proof*$\rangle$

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** $\langle\cdot\rangle$ *55*)
and H :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\langle\star\rangle$ *53*)
and a :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ ($\langle$a[-, -, -]$\rangle$)
and i :: $'a \Rightarrow 'a$ ($\langle$i[-]$\rangle$)
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 $\langle$*proof*$\rangle$

sublocale *identity-in-bicategory-of-spans* $\subseteq$
tabulation-in-maps V H a i *src trg* r *tab* $\langle tab_0\ r\rangle$ $\langle tab_1\ r\rangle$
$\langle$*proof*$\rangle$

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* ‹$k^* \star h$›
⟨*proof*⟩

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

interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
notation *E.eval* (‹⦃-⦄›)

interpretation *h*: *map-in-bicategory V H* a i *src trg h*
⟨*proof*⟩
interpretation *k*: *map-in-bicategory V H* a i *src trg k*
⟨*proof*⟩

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* $p_0 \to$ *src h*»
⟨*proof*⟩

lemma p_1*-in-hom* [*intro*]:
shows «p_1 : *src* $p_0 \to$ *src k*»
⟨*proof*⟩

lemma p_0*-simps* [*simp*]:
shows *trg* p_0 = *src h*
⟨*proof*⟩

lemma p_1*-simps* [*simp*]:
shows *trg* p_1 = *src k*
⟨*proof*⟩

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 «φ : *src* $p_0 \to$ *trg h*»
and «φ : $k \star p_1 \Rightarrow h \star p_0$»
⟨*proof*⟩

lemma *φ-simps* [*simp*]:
shows *arr φ*
and *src φ* = *src* p_0 **and** *trg φ* = *trg h*
and *dom φ* = $k \star p_1$ **and** *cod φ* = $h \star p_0$
⟨*proof*⟩

lemma *transpose-φ*:
shows *tab* = $\mathrm{a}^{-1}[k^*,\ h,\ p_0] \cdot k.trnl_\eta\ p_1\ \varphi$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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 $\langle\!\langle \omega : dom\ \omega \Rightarrow s \star r_0 \rangle\!\rangle$: 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) ·
(*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 ω **and** *is-induced-by-cell* w' r_0 ω
shows *isomorphic w* w'
⟨*proof*⟩

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*⟩

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 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 $\varrho :: 'a$
and $r_0 :: 'a$
and $r_1 :: 'a$
and $s :: 'a$
and $\sigma :: 'a$
and $s_0 :: 'a$
and $s_1 :: 'a$ +
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 $r_0 s_1$: *cospan-of-maps-in-bicategory-of-spans* V H a i *src* *trg* s_1 r_0
⟨*proof*⟩

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.

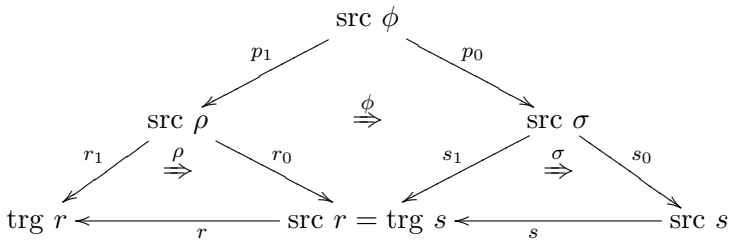
lemma $r_0 s_1$-*is-cospan*:
shows *cospan-of-maps-in-bicategory-of-spans* V H a i *src* *trg* s_1 r_0
⟨*proof*⟩

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 $p_1 \equiv r_0 s_1.p_1$



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 «$tab : r_1 \star p_1 \Rightarrow (r \star s) \star s_0 \star p_0$»
⟨*proof*⟩

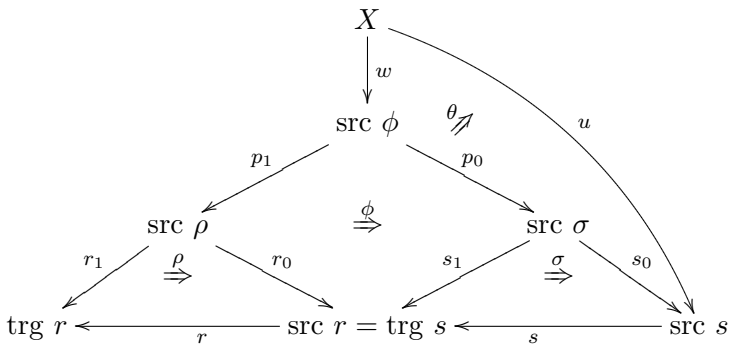
interpretation *tabulation-data V H* a i *src trg* ‹$r \star s$› *tab* ‹$s_0 \star p_0$› ‹$r_1 \star p_1$›
⟨*proof*⟩

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* ⟨*proof*⟩
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* $w\ \vartheta \cdot \mathrm{a}^{-1}[r_1,\ p_1,\ w]$



⟨*proof*⟩

lemma *composite-is-tabulation*:
shows *tabulation V H* a i *src trg* (*r* ⋆ *s*) *tab* (*s*$_0$ ⋆ *p*$_0$) (*r*$_1$ ⋆ *p*$_1$)
⟨*proof*⟩

end

sublocale *composite-tabulation-in-maps* ⊆
tabulation V H a i *src trg* ‹*r* ⋆ *s*› *tab* ‹*s*$_0$ ⋆ *p*$_0$› ‹*r*$_1$ ⋆ *p*$_1$›
⟨*proof*⟩

sublocale *composite-tabulation-in-maps* ⊆
tabulation-in-maps V H a i *src trg* ‹*r* ⋆ *s*› *tab* ‹*s*$_0$ ⋆ *p*$_0$› ‹*r*$_1$ ⋆ *p*$_1$›
⟨*proof*⟩

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* ≡ {*F*. ∃*f*. «*f* : *a* →$_B$ *b*» ∧ *B.is-left-adjoint f* ∧ *F* = ⟦*f*⟧$_B$}

lemma *in-HomD*:
assumes *F* ∈ *Hom a b*

shows $F \neq \{\}$
and *B.is-iso-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-left-adjoint f*
and $f \in F \Longrightarrow F = [\![f]\!]_B$
⟨*proof*⟩

definition *Comp*
where *Comp G F* ≡ {*h*. *B.is-iso-class F* ∧ *B.is-iso-class G* ∧
(∃ *f g*. *f* ∈ *F* ∧ *g* ∈ *G* ∧ *g* ⋆ *f* ≅$_B$ *h*)}

lemma *in-CompI* [*intro*]:
assumes *B.is-iso-class F* **and** *B.is-iso-class G*
and *f* ∈ *F* **and** *g* ∈ *G* **and** *g* ⋆ *f* ≅$_B$ *h*
shows *h* ∈ *Comp G F*
⟨*proof*⟩

lemma *in-CompE* [*elim*]:
assumes *h* ∈ *Comp G F*
and ⋀*f g*. ⟦ *B.is-iso-class F*; *B.is-iso-class G*; *f* ∈ *F*; *g* ∈ *G*; *g* ⋆ *f* ≅$_B$ *h* ⟧ ⟹ *T*
shows *T*
⟨*proof*⟩

lemma *is-iso-class-Comp*:
assumes *Comp G F* ≠ {}
shows *B.is-iso-class* (*Comp G F*)
⟨*proof*⟩

lemma *Comp-extensionality*:
assumes *Comp G F* ≠ {}
shows *B.is-iso-class F* **and** *B.is-iso-class G*
and *F* ≠ {} **and** *G* ≠ {}
⟨*proof*⟩

lemma *Comp-eqI* [*intro*]:
assumes *h* ∈ *Comp G F* **and** *h'* ∈ *Comp G' F'* **and** *h* ≅$_B$ *h'*
shows *Comp G F* = *Comp G' F'*
⟨*proof*⟩

lemma *Comp-eq-iso-class-memb*:
assumes *h* ∈ *Comp G F*
shows *Comp G F* = $[\![h]\!]_B$
⟨*proof*⟩

interpretation *concrete-category* ‹*Collect B.obj*› *Hom B.iso-class* ‹λ- - -. *Comp*›
⟨*proof*⟩

lemma *is-concrete-category*:

shows *concrete-category* (*Collect B.obj*) *Hom B.iso-class* (λ- - -. *Comp*)
⟨*proof*⟩

sublocale *concrete-category* ‹*Collect B.obj*› *Hom B.iso-class* ‹λ- - -. *Comp*›
⟨*proof*⟩

abbreviation *comp* (**infixr** ‹⊙› *55*)
where *comp* ≡ *COMP*
notation *in-hom* (‹«- : - → -»›)
no-notation *B.in-hom* (‹«- : - →$_B$ -»›)

lemma *Map-memb-in-hhom*:
assumes «*F* : *A* → *B*» **and** *f* ∈ *Map F*
shows «*f* : *Dom A* →$_B$ *Dom B*»
⟨*proof*⟩

lemma *MkArr-in-hom′*:
assumes *B.is-left-adjoint f* **and** «*f* : *a* →$_B$ *b*»
shows «*MkArr a b* ⟦*f*⟧$_B$: *MkIde a* → *MkIde b*»
⟨*proof*⟩

The isomorphisms in $Maps(B)$ are the isomorphism classes of equivalence maps in B.

lemma *iso-char*:
shows *iso F* ⟷ *arr F* ∧ (∀*f*. *f* ∈ *Map F* ⟶ *B.equivalence-map f*)
⟨*proof*⟩

lemma *iso-char′*:
shows *iso F* ⟷ *arr F* ∧ (∃*f*. *f* ∈ *Map F* ∧ *B.equivalence-map f*)
⟨*proof*⟩

The following mapping takes a map in B to the corresponding arrow of $Maps(B)$.

abbreviation *CLS* (‹[[[-]]]›)
where [[[*f*]]] ≡ *MkArr* (*src f*) (*trg f*) ⟦*f*⟧$_B$

lemma *ide-CLS-obj*:
assumes *B.obj a*
shows *ide* [[[*a*]]]
⟨*proof*⟩

lemma *CLS-in-hom*:
assumes *B.is-left-adjoint f*
shows «[[[*f*]]] : [[[*src f*]]] → [[[*trg f*]]]»
⟨*proof*⟩

lemma *CLS-eqI*:
assumes *B.ide f*
shows [[[*f*]]] = [[[*g*]]] ⟷ *f* ≅$_B$ *g*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *ide-REP*:
assumes *arr A*
shows *B.ide* (*REP A*)
⟨*proof*⟩

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*⟩

lemma *isomorphic-REP-src*:
assumes *ide A*
shows *REP A* ≅$_B$ *src* (*REP A*)
⟨*proof*⟩

lemma *isomorphic-REP-trg*:
assumes *ide A*
shows *REP A* ≅$_B$ *trg* (*REP A*)
⟨*proof*⟩

lemma *CLS-REP*:

assumes *arr F*
shows $[\![[\![REP\ F]\!]]\!] = F$
⟨*proof*⟩

lemma *REP-CLS*:
assumes *B.is-left-adjoint f*
shows $REP\ [\![[\![f]\!]]\!] \cong_B f$
⟨*proof*⟩

lemma *isomorphic-hcomp-REP*:
assumes *seq F G*
shows $REP\ F \star REP\ G \cong_B REP\ (F \odot G)$
⟨*proof*⟩

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*⟩

lemma *is-elementary-category-with-pullbacks*:
shows *elementary-category-with-pullbacks comp* $PRJ_0\ PRJ_1$
⟨*proof*⟩

lemma *is-category-with-pullbacks*:
shows *category-with-pullbacks comp*
⟨*proof*⟩

sublocale *elementary-category-with-pullbacks comp* $PRJ_0\ PRJ_1$
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation $r_0 s_1$: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* s_1 r_0
⟨*proof*⟩

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*⟩

end

context *identity-in-bicategory-of-spans*
begin

interpretation *Maps*: *maps-category V H* a i *src trg* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp* $Maps.PRJ_0$ $Maps.PRJ_1$ ⟨*proof*⟩

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* ⟷ *equivalence-map* (tab_0 *r*)
⟨*proof*⟩

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*⟩

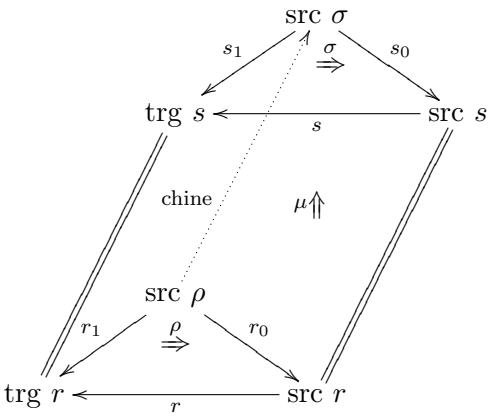
The chosen tabulation of r determines a span in $Maps(B)$.

lemma *determines-span*:
assumes *ide r*
shows *span-in-category Maps.comp* (⦇*Leg0* = $[\![tab_0\ r]\!]$, *Leg1* = $[\![tab_1\ r]\!]$⦈)
⟨*proof*⟩

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.



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* ⇒ *'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 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 «Δ : $r_1 \Rightarrow s \star r_0$»
⟨*proof*⟩

lemma Δ-*simps* [*simp*]:
shows *arr* Δ
and *src* Δ = *src* ϱ **and** *trg* Δ = *trg* σ
and *dom* Δ = r_1 **and** *cod* Δ = $s \star r_0$
⟨*proof*⟩

abbreviation *is-induced-map*
where *is-induced-map* $w \equiv \sigma$.*is-induced-by-cell* w r_0 Δ

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 $w \wedge$
$(\exists \nu\ \vartheta.$ «ϑ : $s_0 \star w \Rightarrow r_0$» $\wedge$ «ν : $r_1 \Rightarrow s_1 \star w$» $\wedge$ *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*⟩

lemma *exists-induced-map*:
shows $\exists w.$ *is-induced-map* w
⟨*proof*⟩

lemma *induced-map-unique*:
assumes *is-induced-map* w **and** *is-induced-map* w'
shows $w \cong w'$
⟨*proof*⟩

definition *chine*
where *chine* ≡ *SOME* $w.$ *is-induced-map* w

lemma *chine-is-induced-map*:
shows *is-induced-map chine*
⟨*proof*⟩

lemma *chine-in-hom* [*intro*]:
shows «*chine* : *src* $r_0 \to$ *src* s_0»
and «*chine*: *chine* ⇒ *chine*»
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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 $(\mu \star r_0) \cdot \varrho = (s \star \textit{the-}\vartheta) \cdot \mathrm{a}[s, s_0, \textit{chine}] \cdot (\sigma \star \textit{chine}) \cdot \textit{the-}\nu$
⟨*proof*⟩

lemma *induced-map-preserved-by-iso*:
assumes *is-induced-map w* **and** *isomorphic w w′*
shows *is-induced-map w′*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma Δ-*eq-*ϱ:
shows $\Delta = \varrho$
⟨*proof*⟩

lemma *chine-is-equivalence*:
shows *equivalence-map chine*
⟨*proof*⟩

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 $\mu :: 'a$ +
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
⟨*proof*⟩

end

sublocale *identity-in-bicategory-of-spans* ⊆ *arrow-in-bicategory-of-spans* V H a i *src* *trg* r r r
⟨*proof*⟩

context *arrow-in-bicategory-of-spans*
begin

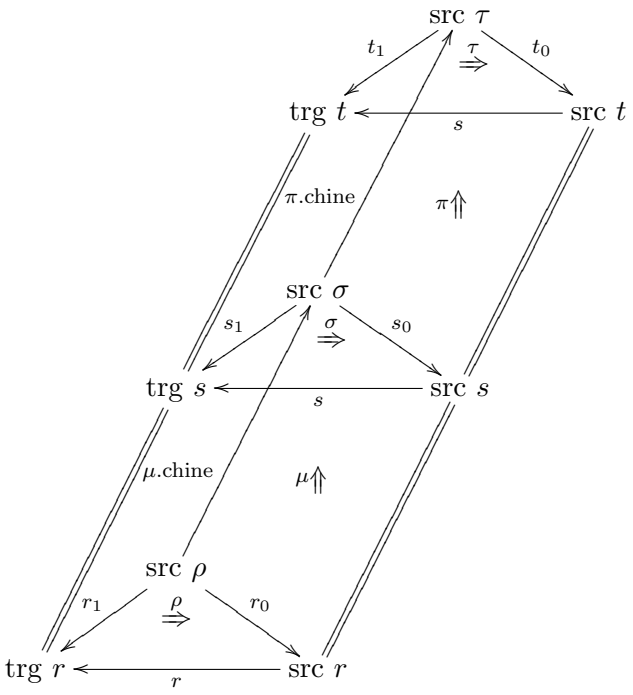
interpretation *arrow-of-tabulations-in-maps* V H a i *src* *trg* r *r.tab* r_0 r_1 s *s.tab* s_0 s_1 μ
⟨*proof*⟩
interpretation r: *arrow-of-tabulations-in-maps* V H a i *src* *trg* r *r.tab* r_0 r_1 r *r.tab* r_0 r_1 r
⟨*proof*⟩

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*⟩

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 $\varrho :: 'a$
and $r_0 :: 'a$
and $r_1 :: 'a$
and $s :: 'a$
and $\sigma :: 'a$
and $s_0 :: 'a$
and $s_1 :: 'a$
and $t :: 'a$
and $\tau :: 'a$
and $t_0 :: 'a$
and $t_1 :: 'a$
and $\mu :: 'a$
and $\pi :: 'a$
begin



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

interpretation *arrow-of-tabulations-in-maps V H* a i *src trg r ϱ r₀ r₁ t τ t₀ t₁ ‹π · μ›*
⟨*proof*⟩

lemma *is-arrow-of-tabulations-in-maps*:
shows *arrow-of-tabulations-in-maps V H* a i *src trg r ϱ r₀ r₁ t τ t₀ t₁* (*π · μ*)
⟨*proof*⟩

lemma *chine-char*:
shows *chine* ≅ *π.chine ⋆ μ.chine*
⟨*proof*⟩

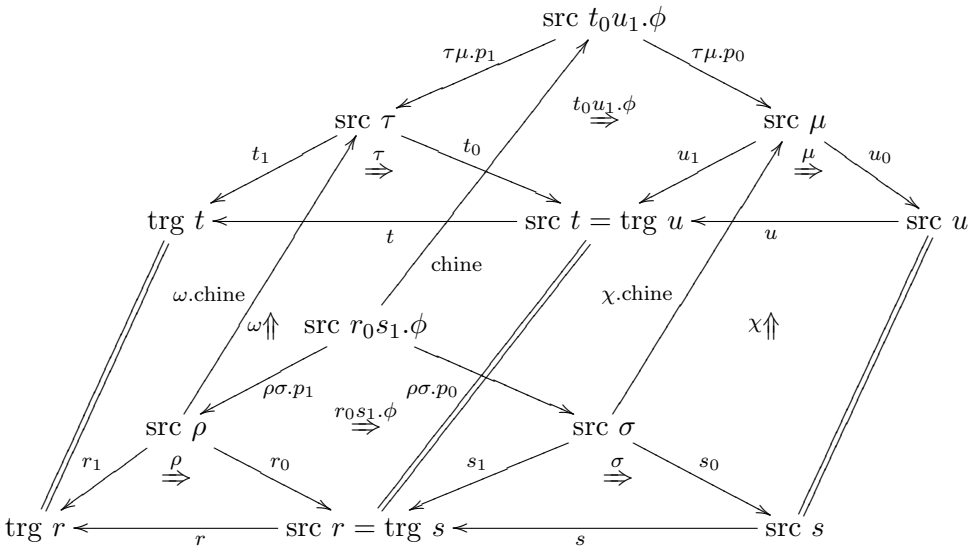
end

sublocale *vertical-composite-of-arrows-of-tabulations-in-maps* ⊆
arrow-of-tabulations-in-maps V H a i *src trg r ϱ r₀ r₁ t τ t₀ t₁ π · μ*
⟨*proof*⟩

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 ϱ r₀ r₁* +
σ: *tabulation-in-maps V H* a i *src trg s σ s₀ s₁* +
τ: *tabulation-in-maps V H* a i *src trg t τ t₀ t₁* +

μ: *tabulation-in-maps V H* a i *src trg u* μ u_0 u_1 +
$\varrho\sigma$: *composite-tabulation-in-maps V H* a i *src trg r* ϱ r_0 r_1 *s* σ s_0 s_1 +
$\tau\mu$: *composite-tabulation-in-maps V H* a i *src trg t* τ t_0 t_1 *u* μ u_0 u_1 +
ω: *arrow-of-tabulations-in-maps V H* a i *src trg r* ϱ r_0 r_1 *t* τ t_0 t_1 ω +
χ: *arrow-of-tabulations-in-maps V H* a i *src trg s* σ s_0 s_1 *u* μ u_0 u_1 χ
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 $\varrho :: 'a$
and $r_0 :: 'a$
and $r_1 :: 'a$
and $s :: 'a$
and $\sigma :: 'a$
and $s_0 :: 'a$
and $s_1 :: 'a$
and $t :: 'a$
and $\tau :: 'a$
and $t_0 :: 'a$
and $t_1 :: 'a$
and $u :: 'a$
and $\mu :: 'a$
and $u_0 :: 'a$
and $u_1 :: 'a$
and $\omega :: 'a$
and $\chi :: 'a$
begin



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

interpretation *arrow-of-tabulations-in-maps V H* a i *src trg*
‹*r* ⋆ *s*› ϱσ.*tab* ‹s_0 ⋆ ϱσ.p_0› ‹r_1 ⋆ ϱσ.p_1›
‹*t* ⋆ *u*› τμ.*tab* ‹u_0 ⋆ τμ.p_0› ‹t_1 ⋆ τμ.p_1› ‹ω ⋆ χ›
⟨*proof*⟩

lemma *is-arrow-of-tabulations-in-maps*:
shows *arrow-of-tabulations-in-maps V H* a i *src trg*
(*r* ⋆ *s*) ϱσ.*tab* (s_0 ⋆ ϱσ.p_0) (r_1 ⋆ ϱσ.p_1)
(*t* ⋆ *u*) τμ.*tab* (u_0 ⋆ τμ.p_0) (t_1 ⋆ τμ.p_1) (ω ⋆ χ)
⟨*proof*⟩

sublocale *arrow-of-tabulations-in-maps V H* a i *src trg*
‹*r* ⋆ *s*› ϱσ.*tab* ‹s_0 ⋆ ϱσ.p_0› ‹r_1 ⋆ ϱσ.p_1›
‹*t* ⋆ *u*› τμ.*tab* ‹u_0 ⋆ τμ.p_0› ‹t_1 ⋆ τμ.p_1› ‹ω ⋆ χ›
⟨*proof*⟩

interpretation *Maps*: *maps-category V H* a i *src trg* ⟨*proof*⟩
notation *Maps.comp* (**infixr** ‹⊙› *55*)

interpretation r_0s_1: *cospan-of-maps-in-bicategory-of-spans* ‹(·)› ‹(⋆)› a i *src trg* s_1 r_0
⟨*proof*⟩
interpretation r_0s_1: *arrow-of-tabulations-in-maps* ‹(·)› ‹(⋆)› a i *src trg*
‹$r_0{}^*$ ⋆ s_1› r_0s_1.*tab* r_0s_1.p_0 r_0s_1.p_1
‹$r_0{}^*$ ⋆ s_1› r_0s_1.*tab* r_0s_1.p_0 r_0s_1.p_1
‹$r_0{}^*$ ⋆ s_1›
⟨*proof*⟩
interpretation t_0u_1: *cospan-of-maps-in-bicategory-of-spans* ‹(·)› ‹(⋆)› a i *src trg* u_1 t_0
⟨*proof*⟩
interpretation t_0u_1: *arrow-of-tabulations-in-maps* ‹(·)› ‹(⋆)› a i *src trg*
‹$t_0{}^*$ ⋆ u_1› t_0u_1.*tab* t_0u_1.p_0 t_0u_1.p_1
‹$t_0{}^*$ ⋆ u_1› t_0u_1.*tab* t_0u_1.p_0 t_0u_1.p_1
‹$t_0{}^*$ ⋆ u_1›
⟨*proof*⟩

interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
notation *E.eval* (‹⦃-⦄›)

no-notation *in-hom* (‹«- : - → -»›)

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*⟩

lemma *comp-L*:
shows *Maps.seq* $[\![t_0]\!]$ $[\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!]$
and $[\![t_0]\!] \odot [\![[\![\omega.chine \star \varrho\sigma.p_1]\!]]\!] =$
Maps.MkArr (*src* $(\omega.chine \star \varrho\sigma.p_1)$) (*src* t) (*Maps.Comp* $[\![t_0]\!]$ $[\![\omega.chine \star \varrho\sigma.p_1]\!]$)
⟨*proof*⟩

lemma *comp-R*:
shows *Maps.seq* $[\![u_1]\!]$ $[\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!]$
and $[\![u_1]\!] \odot [\![[\![\chi.chine \star \varrho\sigma.p_0]\!]]\!] =$
Maps.MkArr (*src* $r_0 s_1.p_0$) (*trg* u) (*Maps.Comp* $[\![u_1]\!]$ $[\![\chi.chine \star r_0 s_1.p_0]\!]$)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp Maps.PRJ*$_0$ *Maps.PRJ*$_1$ ⟨*proof*⟩

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*⟩

lemma *spn-props*:
assumes *arr* μ
shows «*spn* μ : *src* (*tab*$_0$ (*dom* μ)) → *src* (*tab*$_0$ (*cod* μ))»
and *is-left-adjoint* (*spn* μ)
and *tab*$_0$ (*cod* μ) $\star$ *spn* $\mu \cong$ *tab*$_0$ (*dom* μ)
and *tab*$_1$ (*cod* μ) $\star$ *spn* $\mu \cong$ *tab*$_1$ (*dom* μ)
⟨*proof*⟩

lemma *spn-in-hom* [*intro*]:
assumes *arr* μ
shows «*spn* μ : *src* (*tab*$_0$ (*dom* μ)) → *src* (*tab*$_0$ (*cod* μ))»
and «*spn* μ : *spn* μ ⇒ *spn* μ»
  ⟨*proof*⟩

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))
  ⟨*proof*⟩

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*⟩

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*⟩

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* = $[\![[\![tab_0\ (dom\ \mu)]\!]]\!]$, *Leg1* = $[\![[\![tab_1\ (dom\ \mu)]\!]]\!]$⦈
⟨*proof*⟩

lemma *Cod-SPN* [*simp*]:
assumes *arr* μ
shows *Cod* (*SPN* μ) = ⦇*Leg0* = $[\![[\![tab_0\ (cod\ \mu)]\!]]\!]$, *Leg1* = $[\![[\![tab_1\ (cod\ \mu)]\!]]\!]$⦈
⟨*proof*⟩

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*⟩

interpretation *SPN*: *functor V Span.vcomp SPN*
⟨*proof*⟩

lemma *SPN-is-functor*:
shows *functor V Span.vcomp SPN*

⟨*proof*⟩

interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩

lemma *SPN-is-weak-arrow-of-homs*:
shows *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩

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* ∘ *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* ∘ *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*
⟨*proof*⟩
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
⟨*proof*⟩
interpretation *s*: *arrow-in-bicategory-of-spans* V H a i *src trg* s s s
⟨*proof*⟩
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
⟨*proof*⟩

sublocale *identity-in-bicategory-of-spans* V H a i *src trg* ‹$r \star s$›
⟨*proof*⟩
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
⟨*proof*⟩

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 $(r \star s)$›
‹$r \star s$›
⟨*proof*⟩

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 $(tab_0\ (r \star s))$ $(tab_1\ (r \star s))$
$(r \star s)$
⟨*proof*⟩

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 $tab_0\ (r \star s) \star cmp \cong tab_0\ s \star \varrho\sigma.p_0$
and $tab_1\ (r \star s) \star cmp \cong tab_1\ r \star \varrho\sigma.p_1$
⟨*proof*⟩

lemma *cmp-in-hom* [*intro*]:
shows «*cmp* : *src* $\varrho\sigma$.*tab* $\rightarrow$ *src tab*»
and «*cmp* : *cmp* $\Rightarrow$ *cmp*»
⟨*proof*⟩

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*
⟨*proof*⟩

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* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp Maps.PRJ*$_0$ *Maps.PRJ*$_1$ ⟨*proof*⟩

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*
⟨*proof*⟩
interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩

interpretation *SPN-r-SPN-s*: *arrow-of-spans Maps.comp* ‹*SPN r* ○ *SPN s*›
⟨*proof*⟩
interpretation *SPN-r-SPN-s*: *identity-arrow-of-spans Maps.comp* ‹*SPN r* ○ *SPN s*›
⟨*proof*⟩
interpretation *SPN-rs*: *arrow-of-spans Maps.comp* ‹*SPN* (*r* ⋆ *s*)›
⟨*proof*⟩
interpretation *SPN-rs*: *identity-arrow-of-spans Maps.comp* ‹*SPN* (*r* ⋆ *s*)›
⟨*proof*⟩

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
⟨*proof*⟩

lemma R_0-*in-hom* [*intro*]:

shows *Maps.in-hom* R_0 (*Maps.MkIde* (*src* $r.s_0$)) (*Maps.MkIde* (*src* r))
⟨*proof*⟩

lemma R_1*-in-hom* [*intro*]:
shows *Maps.in-hom* R_1 (*Maps.MkIde* (*src* $r.s_0$)) (*Maps.MkIde* (*trg* r))
⟨*proof*⟩

lemma S_0*-in-hom* [*intro*]:
shows *Maps.in-hom* S_0 (*Maps.MkIde* (*src* $s.s_0$)) (*Maps.MkIde* (*src* s))
⟨*proof*⟩

lemma S_1*-in-hom* [*intro*]:
shows *Maps.in-hom* S_1 (*Maps.MkIde* (*src* $s.s_0$)) (*Maps.MkIde* (*trg* s))
⟨*proof*⟩

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
⟨*proof*⟩

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)* ⋆ *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* $\varrho\sigma$.*tab*)
⟨*proof*⟩

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* $\varrho\sigma$.*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*⟩

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*⟩

interpretation $R_0{'}S_1$: *identity-in-bicategory-of-spans V H* a i *src trg* ‹(tab_0 *r*)* ⋆ tab_1 *s*›
⟨*proof*⟩

lemma *prj-tab-agreement*:
shows (tab_0 *r*)* ⋆ tab_1 *s* ≅ (*Maps.REP* R_0)* ⋆ *Maps.REP* S_1
and $\varrho\sigma.p_0$ ≅ prj_0 (*Maps.REP* S_1) (*Maps.REP* R_0)
and $\varrho\sigma.p_1$ ≅ prj_1 (*Maps.REP* S_1) (*Maps.REP* R_0)
⟨*proof*⟩

lemma *chine-hcomp-SPN-SPN*:
shows *Span.chine-hcomp* (*SPN r*) (*SPN s*) = *Maps.MkIde* (*src* $\varrho\sigma.p_0$)
⟨*proof*⟩

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* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp* $Maps.PRJ_0$ $Maps.PRJ_1$ ⟨*proof*⟩

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*
⟨*proof*⟩
interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩

interpretation *HoSPN-SPN*: *composite-functor VV.comp Span.VV.comp Span.vcomp*
SPN.FF ‹λμν. *fst* μν ∘ *snd* μν›
⟨*proof*⟩
interpretation *SPNoH*: *composite-functor VV.comp V*
Span.vcomp ‹λμν. *fst* μν ⋆ *snd* μν› *SPN*
⟨*proof*⟩

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* ≡

$(\!|\ Chn = [\![[\![two\text{-}composable\text{-}identities\text{-}in\text{-}bicategory\text{-}of\text{-}spans.cmp\ V\ H\ \mathrm{a}\ \mathrm{i}\ src\ trg\ r\ s]\!]\!],$
$Dom = Dom\ (SPN\ r \circ SPN\ s),\ Cod = Dom\ (SPN\ (r \star s))\ |\!)$

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*⟩

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*)
⟨*proof*⟩

lemma *compositor-is-iso*:
assumes *ide r* **and** *ide s* **and** *src r* = *trg s*
shows *Span.iso* (*CMP r s*)
⟨*proof*⟩

interpretation Ξ: *transformation-by-components VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map ‹λ*rs*. *CMP* (*fst rs*) (*snd rs*)›
⟨*proof*⟩

interpretation Ξ: *natural-isomorphism VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map Ξ.*map*
⟨*proof*⟩

lemma *compositor-naturalitytransformation*:
shows *transformation-by-components VV.comp Span.vcomp HoSPN-SPN.map SPNoH.map*
(λ*rs*. *CMP* (*fst rs*) (*snd rs*))
⟨*proof*⟩

lemma *compositor-naturalityisomorphism*:
shows *natural-isomorphism VV.comp Span.vcomp HoSPN-SPN.map SPNoH.map* Ξ.*map*
⟨*proof*⟩

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 $H :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹$\star$› *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 *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*
⟨*proof*⟩
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*
⟨*proof*⟩

interpretation *E*: *self-evaluation-map V H* a i *src trg* ⟨*proof*⟩
notation *E.eval* (‹⦃-⦄›)

interpretation *Maps*: *maps-category V H* a i *src trg* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp Maps.PRJ*$_0$ *Maps.PRJ*$_1$ ⟨*proof*⟩

no-notation *Fun.comp* (**infixl** ‹$\circ$› *55*)
notation *Span.vcomp* (**infixr** ‹$\cdot$› *55*)
notation *Span.hcomp* (**infixr** ‹$\circ$› *53*)
notation *Maps.comp* (**infixr** ‹$\odot$› *55*)
notation *isomorphic* (**infix** ‹$\cong$› *50*)

interpretation *SPN*: *functor V Span.vcomp SPN*
⟨*proof*⟩
interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩
interpretation *SPN-SPN*: *functor VV.comp Span.VV.comp SPN.FF*
⟨*proof*⟩
interpretation *HoSPN-SPN*: *composite-functor VV.comp Span.VV.comp Span.vcomp*
SPN.FF ‹$\lambda\mu\nu$. *fst* $\mu\nu$ ∘ *snd* $\mu\nu$›
⟨*proof*⟩
interpretation *SPNoH*: *composite-functor VV.comp V Span.vcomp* ‹$\lambda\mu\nu$. *fst* $\mu\nu$ $\star$ *snd* $\mu\nu$› *SPN*
⟨*proof*⟩

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* ‹$(f \star g) \star h$›
⟨*proof*⟩
interpretation *HfHgh*: *identity-in-bicategory-of-spans* V H a i *src trg* ‹$f \star g \star h$›
⟨*proof*⟩
interpretation *Tfg*: *two-composable-identities-in-bicategory-of-spans* V H a i *src trg* f g
⟨*proof*⟩
interpretation *Tgh*: *two-composable-identities-in-bicategory-of-spans* V H a i *src trg* g h
⟨*proof*⟩
interpretation *THfgh*: *two-composable-identities-in-bicategory-of-spans* V H a i *src trg*
‹$f \star g$› h
⟨*proof*⟩
interpretation *THfgh*: *tabulation* V H a i *src trg* ‹$(f \star g) \star h$› *THfgh.ϱσ.tab*
‹$tab_0\ h \star$ *THfgh.ϱσ.*p_0› ‹$tab_1\ (f \star g) \star$ *THfgh.ϱσ.*p_1›
⟨*proof*⟩
interpretation *TfHgh*: *two-composable-identities-in-bicategory-of-spans* V H a i *src trg*
f ‹$g \star h$›
⟨*proof*⟩
interpretation *TfHgh*: *tabulation* 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›
⟨*proof*⟩

interpretation *Tfg-Hfg*: *arrow-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›
‹$f \star g$› ‹*tab-of-ide* $(f \star g)$› ‹$tab_0\ (f \star g)$› ‹$tab_1\ (f \star g)$›
‹$f \star g$›
⟨*proof*⟩
interpretation *Tgh-Hgh*: *arrow-of-tabulations-in-maps* V H a i *src trg*
‹$g \star h$› *Tgh.ϱσ.tab* ‹$tab_0\ h \star$ *Tgh.ϱσ.*p_0› ‹$tab_1\ g \star$ *Tgh.ϱσ.*p_1›
‹$g \star h$› ‹*tab-of-ide* $(g \star h)$› ‹$tab_0\ (g \star h)$› ‹$tab_1\ (g \star h)$›
‹$g \star h$›
⟨*proof*⟩
interpretation *THfgh-HHfgh*:
arrow-of-tabulations-in-maps V H a i *src trg*
‹$(f \star g) \star h$› *THfgh.ϱσ.tab* ‹$tab_0\ h \star$ *THfgh.ϱσ.*p_0› ‹$tab_1\ (f \star g) \star$ *THfgh.ϱσ.*p_1›
‹$(f \star g) \star h$› ‹*tab-of-ide* $((f \star g) \star h)$› ‹$tab_0\ ((f \star g) \star h)$› ‹$tab_1\ ((f \star g) \star h)$›
‹$(f \star g) \star h$›
⟨*proof*⟩
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* $(f \star g \star h)$› ‹$tab_0\ (f \star g \star h)$› ‹$tab_1\ (f \star g \star h)$›
‹$f \star g \star h$›
⟨*proof*⟩
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› ‹$tab_0\ h$› ‹$tab_1\ h$›
⟨*proof*⟩
interpretation *TTfgh-THfgh*:

horizontal-composite-of-arrows-of-tabulations-in-maps V H a i *src trg*
‹*f* ⋆ *g*› *Tfg.ϱσ.tab* ‹*tab*$_0$ *g* ⋆ *Tfg.ϱσ.p*$_0$› ‹*tab*$_1$ *f* ⋆ *Tfg.ϱσ.p*$_1$›
h ‹*tab-of-ide h*› ‹*tab*$_0$ *h*› ‹*tab*$_1$ *h*›
‹*f* ⋆ *g*› ‹*tab-of-ide* (*f* ⋆ *g*)› ‹*tab*$_0$ (*f* ⋆ *g*)› ‹*tab*$_1$ (*f* ⋆ *g*)›
h ‹*tab-of-ide h*› ‹*tab*$_0$ *h*› ‹*tab*$_1$ *h*›
‹*f* ⋆ *g*› *h*
⟨*proof*⟩

interpretation *TfTgh*: *composite-tabulation-in-maps V H* a i *src trg*
f ‹*tab-of-ide f*› ‹*tab*$_0$ *f*› ‹*tab*$_1$ *f*›
‹*g* ⋆ *h*› *Tgh.ϱσ.tab* ‹*tab*$_0$ *h* ⋆ *Tgh.ϱσ.p*$_0$› ‹*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$›
⟨*proof*⟩

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* ⋆ *h*› *Tgh.ϱσ.tab* ‹*tab*$_0$ *h* ⋆ *Tgh.ϱσ.p*$_0$› ‹*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$›
f ‹*tab-of-ide f*› ‹*tab*$_0$ *f*› ‹*tab*$_1$ *f*›
‹*g* ⋆ *h*› ‹*tab-of-ide* (*g* ⋆ *h*)› ‹*tab*$_0$ (*g* ⋆ *h*)› ‹*tab*$_1$ (*g* ⋆ *h*)›
f ‹*g* ⋆ *h*›
⟨*proof*⟩

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* ⋆ *h*› *Tgh.ϱσ.tab* ‹*tab*$_0$ *h* ⋆ *Tgh.ϱσ.p*$_0$› ‹*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$›
f ‹*tab-of-ide f*› ‹*tab*$_0$ *f*› ‹*tab*$_1$ *f*›
‹*g* ⋆ *h*› *Tgh.ϱσ.tab* ‹*tab*$_0$ *h* ⋆ *Tgh.ϱσ.p*$_0$› ‹*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$›
f ‹*g* ⋆ *h*›
⟨*proof*⟩

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* ⋆ *g*) ⋆ *h*› *TTfgh.tab* ‹*tab*$_0$ *h* ⋆ *TTfgh.p*$_0$› ‹(*tab*$_1$ *f* ⋆ *Tfg.ϱσ.p*$_1$) ⋆ *TTfgh.p*$_1$›
‹*f* ⋆ *g* ⋆ *h*› *TfTgh.tab* ‹(*tab*$_0$ *h* ⋆ *Tgh.ϱσ.p*$_0$) ⋆ *TfTgh.p*$_0$› ‹*tab*$_1$ *f* ⋆ *TfTgh.p*$_1$›
‹a[*f*, *g*, *h*]›
⟨*proof*⟩

This interpretation defines the map, from the apex of the tabulation associated with the horizontal composite (*f* ⋆ *g*) ⋆ *h* to the apex of the tabulation associated with the horizontal composite *f* ⋆ *g* ⋆ *h*, induced by the associativity isomorphism a[*f*, *g*, *h*] from (*f* ⋆ *g*) ⋆ *h* to *f* ⋆ *g* ⋆ *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*⟩

interpretation *SPN-f*: *arrow-of-spans Maps.comp ‹SPN f›*
⟨*proof*⟩
interpretation *SPN-g*: *arrow-of-spans Maps.comp ‹SPN g›*
⟨*proof*⟩
interpretation *SPN-h*: *arrow-of-spans Maps.comp ‹SPN h›*
⟨*proof*⟩
interpretation *SPN-fgh*: *three-composable-identity-arrows-of-spans Maps.comp*
Maps.PRJ$_0$ *Maps.PRJ*$_1$ *‹SPN f› ‹SPN g› ‹SPN h›*
⟨*proof*⟩

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.ϱσ.p*$_1$ ⋆ *TTfgh.p*$_1$⟧⟧
and *SPN-fgh.Prj*$_{01}$ = ⟦⟦*Tfg.ϱσ.p*$_0$ ⋆ *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.ϱσ.p*$_1$ ⋆ *TfTgh.p*$_0$⟧⟧
and *SPN-fgh.Prj*$_{00}$ = ⟦⟦*Tgh.ϱσ.p*$_0$ ⋆ *TfTgh.p*$_0$⟧⟧
⟨*proof*⟩

interpretation Φ: *transformation-by-components VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map ‹λrs. CMP (fst rs) (snd rs)›
⟨*proof*⟩
interpretation Φ: *natural-isomorphism VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map Φ.*map*
⟨*proof*⟩

interpretation *VVV′*: *subcategory VxVxV.comp*
‹λτμν. arr (fst τμν) ∧ arr (fst (snd τμν)) ∧ arr (snd (snd τμν)) ∧
src (fst (snd τμν)) = trg (snd (snd τμν)) ∧
src (fst τμν) = trg (fst (snd τμν))›
⟨*proof*⟩

We define abbreviations for the left and right-hand sides of the equation for associativity coherence.

abbreviation *LHS*
where *LHS* ≡ *SPN* a[*f*, *g*, *h*] ⋅ Φ.*map* (*f* ⋆ *g*, *h*) ⋅ (Φ.*map* (*f*, *g*) ∘ *SPN h*)

abbreviation *RHS*
where *RHS* ≡ Φ.*map* (*f*, *g* ⋆ *h*) ⋅ (*SPN f* ∘ Φ.*map* (*g*, *h*)) ⋅
Span.assoc (*SPN f*) (*SPN g*) (*SPN h*)

lemma *arr-LHS*:
shows *Span.arr LHS*
⟨*proof*⟩

lemma *arr-RHS*:

shows *Span.arr RHS*
⟨*proof*⟩

lemma *par-LHS-RHS*:
shows *Span.par LHS RHS*
⟨*proof*⟩

lemma *Chn-LHS-eq*:
shows *Chn LHS* =
$[\![[\![HHfgh\text{-}HfHgh.chine]\!]\!] \odot [\![[\![THfgh\text{-}HHfgh.chine]\!]\!] \odot [\![[\![TTfgh\text{-}THfgh.chine]\!]\!]$
⟨*proof*⟩

abbreviation *tuple-BC*
where *tuple-BC* ≡ *Maps.tuple SPN-fgh.Prj*$_{01}$ *SPN-fgh.ν.leg0 SPN-fgh.π.leg1 SPN-fgh.Prj*$_0$

abbreviation *tuple-ABC*
where *tuple-ABC* ≡ *Maps.tuple SPN-fgh.Prj*$_{11}$
SPN-fgh.μ.leg0
(*SPN-fgh.ν.leg1* ⊙ *SPN-fgh.νπ.prj*$_1$)
tuple-BC

abbreviation *tuple-BC′*
where *tuple-BC′* ≡ *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′* ≡ *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* ⊙ *SPN-fgh.νπ.prj*$_1$)
SPN-fgh.Prj$_{11}$ *tuple-BC*
⟨*proof*⟩

lemma *tuple-ABC-eq-ABC′*:
shows *tuple-BC* = *tuple-BC′*
and *tuple-ABC* = *tuple-ABC′*
⟨*proof*⟩

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*⟩

lemma *tuple-ABC-in-hom*:
shows *Maps.in-hom tuple-ABC* (*Maps.MkIde* (*src TTfgh.p*$_0$)) (*Maps.MkIde* (*src TfTgh.p*$_0$))
⟨*proof*⟩

lemma *Chn-RHS-eq*:
shows *Chn RHS* = ⟦⟦*TfHgh-HfHgh.chine*⟧⟧ ⊙ ⟦⟦*TfTgh-TfHgh.chine*⟧⟧ ⊙ *tuple-ABC′*
⟨*proof*⟩

interpretation g_0h_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* ‹*tab*$_1$ *h*› ‹*tab*$_0$ *g*›
⟨*proof*⟩
interpretation f_0g_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg* ‹*tab*$_1$ *g*› ‹*tab*$_0$ *f*›
⟨*proof*⟩
interpretation f_0gh_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg*
‹*tab*$_1$ *g* ⋆ *Tgh.ϱσ.p*$_1$› ‹*tab*$_0$ *f*›
⟨*proof*⟩
interpretation fg_0h_1: *cospan-of-maps-in-bicategory-of-spans V H* a i *src trg*
‹*tab*$_1$ *h*› ‹*tab*$_0$ *g* ⋆ *Tfg.p*$_0$›
⟨*proof*⟩

lemma *src-tab-eq*:
shows (a^{-1}[*f*, *g*, *h*] ⋆ *tab*$_0$ *h* ⋆ *TTfgh.p*$_0$) ·
TfTgh.composite-cell TTfgh-TfTgh.chine TTfgh-TfTgh.the-ϑ · *TTfgh-TfTgh.the-ν* =
TTfgh.tab
⟨*proof*⟩

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 ⟦⟦*TfTgh.p*$_1$ ⋆ *TTfgh-TfTgh.chine*⟧⟧ = ⟦⟦*Tfg.p*$_1$ ⋆ *TTfgh.p*$_1$⟧⟧
and ⟦⟦*Tgh.p*$_1$ ⋆ *TfTgh.p*$_0$ ⋆ *TTfgh-TfTgh.chine*⟧⟧ = ⟦⟦*Tfg.p*$_0$ ⋆ *TTfgh.p*$_1$⟧⟧
and ⟦⟦*Tgh.p*$_0$ ⋆ *TfTgh.p*$_0$ ⋆ *TTfgh-TfTgh.chine*⟧⟧ = ⟦⟦*TTfgh.p*$_0$⟧⟧
⟨*proof*⟩

Finally, we can show that *TTfgh-TfTgh.chine* is given by tupling.

lemma *CLS-chine*:
shows ⟦⟦*TTfgh-TfTgh.chine*⟧⟧ = *tuple-ABC*
⟨*proof*⟩

At long last, we can show associativity coherence for *SPN*.

lemma *assoc-coherence*:
shows *LHS* = *RHS*
⟨*proof*⟩

end

SPN is an Equivalence Pseudofunctor

context *bicategory-of-spans*
begin

interpretation *Maps*: *maps-category V H* a i *src trg* ⟨*proof*⟩
interpretation *Span*: *span-bicategory Maps.comp* $Maps.PRJ_0$ $Maps.PRJ_1$ ⟨*proof*⟩

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*
⟨*proof*⟩
interpretation *SPN*: *weak-arrow-of-homs V src trg Span.vcomp Span.src Span.trg SPN*
⟨*proof*⟩
interpretation *HoSPN-SPN*: *composite-functor VV.comp Span.VV.comp Span.vcomp*
SPN.FF ‹*λμν. Span.hcomp* (*fst μν*) (*snd μν*)›
⟨*proof*⟩
interpretation *SPNoH*: *composite-functor VV.comp V Span.vcomp*
‹*λμν. fst μν ⋆ snd μν*› *SPN*
⟨*proof*⟩

interpretation Φ: *transformation-by-components VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map ‹*λrs. CMP* (*fst rs*) (*snd rs*)›
⟨*proof*⟩
interpretation Φ: *natural-isomorphism VV.comp Span.vcomp*
HoSPN-SPN.map SPNoH.map Φ.*map*
⟨*proof*⟩

abbreviation Φ
where Φ ≡ Φ.*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*⟩

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
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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] 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.

[13] E. W. Stark. Monoidal categories. *Archive of Formal Proofs*, May 2017. http://isa-afp.org/entries/MonoidalCategory.shtml, Formal proof development.

[14] R. Street. Fibrations in bicategories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, XXI(2):111–159, 1980.

[15] R. H. Street. Fibrations and Yoneda's lemma in a 2-category. In *Lecture Notes in Mathematics 420*, pages 104–133. Springer-Verlag, 1974.