

# Monoidal Categories

Eugene W. Stark

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

December 14, 2021

## Abstract

Building on the formalization of basic category theory set out in the author's previous AFP article [6], the present article formalizes some basic aspects of the theory of monoidal categories. Among the notions defined here are monoidal category, monoidal functor, and equivalence of monoidal categories. The main theorems formalized are MacLane's coherence theorem and the constructions of the free monoidal category and free strict monoidal category generated by a given category. The coherence theorem is proved syntactically, using a structurally recursive approach to reduction of terms that might have some novel aspects. We also give proofs of some results given by Etingof *et al* [2], which may prove useful in a formal setting. In particular, we show that the left and right unitors need not be taken as given data in the definition of monoidal category, nor does the definition of monoidal functor need to take as given a specific isomorphism expressing the preservation of the unit object. Our definitions of monoidal category and monoidal functor are stated so as to take advantage of the economy afforded by these facts.

Revisions made subsequent to the first version of this article added material on cartesian monoidal categories; showing that the underlying category of a cartesian monoidal category is a cartesian category, and that every cartesian category extends to a cartesian monoidal category.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Monoidal Category</b>	<b>6</b>
2.1	Monoidal Category . . . . .	6
2.2	Elementary Monoidal Category . . . . .	30
2.3	Strict Monoidal Category . . . . .	38
2.4	Opposite Monoidal Category . . . . .	38
2.5	Monoidal Language . . . . .	39
2.6	Coherence . . . . .	67
<b>3</b>	<b>Monoidal Functor</b>	<b>93</b>
3.1	Strict Monoidal Functor . . . . .	103
<b>4</b>	<b>The Free Monoidal Category</b>	<b>108</b>
4.1	Syntactic Construction . . . . .	108
4.2	Proof of Freeness . . . . .	143
4.3	Strict Subcategory . . . . .	154
<b>5</b>	<b>Cartesian Monoidal Category</b>	<b>175</b>

# Chapter 1

## Introduction

A *monoidal category* is a category  $C$  equipped with a binary “tensor product” functor  $\otimes : C \times C \rightarrow C$ , which is associative up to a given natural isomorphism, and an object  $\mathcal{I}$  that behaves up to isomorphism like a unit for  $\otimes$ . The associativity and unit isomorphisms are assumed to satisfy certain axioms known as *coherence conditions*. Monoidal categories were introduced by Bénabou [1] and MacLane [4]. MacLane showed that the axioms for a monoidal category imply that all diagrams in a large class are commutative. This result, known as MacLane’s Coherence Theorem, is the first important result in the theory of monoidal categories.

Monoidal categories are important partly because of their ubiquity. The category of sets and functions is monoidal; more generally any category with binary products and a terminal object becomes a monoidal category if we take the categorical product as  $\otimes$  and the terminal object as  $\mathcal{I}$ . The category of vector spaces over a field, with linear maps as morphisms, not only admits monoidal structure with respect to the categorical product, but also with respect to the usual tensor product of vector spaces. Monoidal categories serve as the starting point for enriched category theory in that they provide a setting in which ordinary categories, having “homs in the category of sets,” can be generalized to “categories having homs in a monoidal category  $\mathcal{V}$ ”. In addition, the theory of monoidal categories can be regarded as a stepping stone to the theory of bicategories, as monoidal categories are the same thing as one-object bicategories.

Building on the formalization of basic category theory set out in the author’s previous AFP article [6], the present article formalizes some basic aspects of the theory of monoidal categories. In Chapter 2, we give a definition of the notion of monoidal category and develop consequences of the axioms. We then give a proof of MacLane’s coherence theorem. The proof is syntactic: we define a language of terms built from arrows of a given category  $C$  using constructors that correspond to formal composition and tensor product as well as to the associativity and unit isomorphisms and their formal inverses, we then define a mapping that interprets terms of the language in an arbitrary monoidal category  $D$  via a valuation functor  $V : C \rightarrow D$ , and finally we syntactically characterize a class of equations between terms that hold in any such interpretation. Among these equations are all those that relate formally parallel “canonical” terms, where a term is

canonical if the only arrows of  $C$  that are used in its construction are identities. Thus, all formally parallel canonical terms have identical interpretations in any monoidal category, which is the content of MacLane’s coherence theorem.

In Chapter 3, we define the notion of a *monoidal functor* between monoidal categories. A monoidal functor from a monoidal category  $C$  to a monoidal category  $D$  is a functor  $F : C \rightarrow D$ , equipped with additional data that express that the monoidal structure is preserved by  $F$  up to natural isomorphism. A monoidal functor is *strict* if it preserves the monoidal structure “on the nose” (*i.e.* the natural isomorphism is an identity). We also define the notion of an *equivalence of monoidal categories*, which is a monoidal functor  $F : C \rightarrow D$  that is part of an ordinary equivalence of categories between  $C$  and  $D$ .

In Chapter 4, we use the language of terms defined in Chapter 2 to give a syntactic construction of the free monoidal category  $\mathcal{F}C$  generated by a category  $C$ . The arrows  $\mathcal{F}C$  are defined to be certain equivalence classes of terms, where composition and tensor product, as well as the associativity and unit isomorphisms, are determined by the syntactic operations. After proving that the construction does in fact yield a monoidal category, we establish its freeness: every functor from  $C$  to a monoidal category  $D$  extends uniquely to a strict monoidal functor from  $\mathcal{F}C$  to  $D$ . We then consider the subcategory  $\mathcal{F}_S C$  of  $\mathcal{F}C$  whose arrows are equivalence classes of terms that we call “diagonal.” Diagonal terms amount to lists of arrows of  $C$ , composition in  $\mathcal{F}_S C$  is given by elementwise composition of compatible lists of arrows, and tensor product in  $\mathcal{F}_S C$  is given by concatenation of lists. We show that the subcategory  $\mathcal{F}_S C$  is monoidally equivalent to the category  $\mathcal{F}C$  and in addition that  $\mathcal{F}_S C$  is the free strict monoidal category generated by  $C$ .

The formalizations of the notions of monoidal category and monoidal functor that we give here are not quite the traditional ones. The traditional definition of monoidal category assumes as given not only an “associator” natural isomorphism, which expresses the associativity of the tensor product, but also left and right “unitor” isomorphisms, which correspond to unit laws. However, as pointed out in [2], it is not necessary to take the unitors as given, because they are uniquely determined by the other structure and the condition that left and right tensoring with the unit object are endo-equivalences. This leads to a definition of monoidal category that requires fewer data to be given and fewer conditions to be verified in applications. As this is likely to be especially important in a formal setting, we adopt this more economical definition and go to the trouble to obtain the unitors as defined notions. A similar situation occurs with the definition of monoidal functor. The traditional definition requires two natural isomorphisms to be given: one that expresses the preservation of tensor product and another that expresses the preservation of the unit object. Once again, as indicated in [2], it is logically unnecessary to take the latter isomorphism as given, since there is a canonical definition of it in terms of the other structure. We adopt the more economical definition of monoidal functor and prove that the traditionally assumed structure can be derived from it.

Finally, the proof of the coherence theorem given here potentially has some novel aspects. A typical syntactic proof of this theorem, such as that described in [5], involves the identification, for each term constructed as a formal tensor product of the unit object  $\mathcal{I}$  and “primitive objects” (*i.e.* the elements of a given set of generators), of a “reduction”

isomorphism obtained by composing “basic reductions” in which occurrences of  $\mathcal{I}$  are eliminated using components of the left and right unitors and “parentheses are moved to one end” using components of the associator. The construction of these reductions is performed, as in [5], using an approach that can be thought of as the application of an iterative strategy for normalizing a term. My thoughts were initially along these lines, and I did succeed in producing a formal proof of the coherence theorem in this way. However, proving the termination of the reduction strategy was complicated by the necessity of using of a “rank function” on terms, and the lemmas required for the remainder of the proof had to be proved by induction on rank, which was messy. At some point, I realized that it ought to be possible to define reductions in a structurally recursive way, which would permit the lemmas in the rest of the proof to be proved by structural induction, rather than induction on rank. It took some time to find the right definitions, but in the end this approach worked out more simply, and is what is presented here.

### **Revision Notes**

The original version of this document dates from May, 2017. The current version of this document incorporates revisions made in mid-2020 after the release of Isabelle2020. Aside from various minor improvements, the main change was the addition of a new theory, concerning cartesian monoidal categories, which coordinates with material on cartesian categories that was simultaneously added to [6]. The new theory defines “cartesian monoidal category” as an extension of “monoidal category” obtained by adding additional functors, natural transformations, and coherence conditions. The main results proved are that the underlying category of a cartesian monoidal category is a cartesian category, and that every cartesian category extends to a cartesian monoidal category.

## Chapter 2

# Monoidal Category

In this theory, we define the notion “monoidal category,” and develop consequences of the definition. The main result is a proof of MacLane’s coherence theorem.

```
theory MonoidalCategory  
imports Category3.EquivalenceOfCategories  
begin
```

### 2.1 Monoidal Category

A typical textbook presentation defines a monoidal category to be a category  $C$  equipped with (among other things) a binary “tensor product” functor  $\otimes: C \times C \rightarrow C$  and an “associativity” natural isomorphism  $\alpha$ , whose components are isomorphisms  $\alpha(a, b, c): (a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c)$  for objects  $a, b$ , and  $c$  of  $C$ . This way of saying things avoids an explicit definition of the functors that are the domain and codomain of  $\alpha$  and, in particular, what category serves as the domain of these functors. The domain category is in fact the product category  $C \times C \times C$  and the domain and codomain of  $\alpha$  are the functors  $T o (T \times C): C \times C \times C \rightarrow C$  and  $T o (C \times T): C \times C \times C \rightarrow C$ . In a formal development, though, we can’t gloss over the fact that  $C \times C \times C$  has to mean either  $C \times (C \times C)$  or  $(C \times C) \times C$ , which are not formally identical, and that associativities are somehow involved in the definitions of the functors  $T o (T \times C)$  and  $T o (C \times T)$ . Here we define the *binary-endofunctor* locale to codify our choices about what  $C \times C \times C$ ,  $T o (T \times C)$ , and  $T o (C \times T)$  actually mean. In particular, we choose  $C \times C \times C$  to be  $C \times (C \times C)$  and define the functors  $T o (T \times C)$ , and  $T o (C \times T)$  accordingly.

```
locale binary-endofunctor =  
  C: category C +  
  CC: product-category C C +  
  CCC: product-category C CC.comp +  
  binary-functor C C C T  
for C :: 'a comp      (infixr · 55)  
and T :: 'a * 'a ⇒ 'a  
begin
```

**definition** *ToTC*  
**where** *ToTC*  $f \equiv$  if *CCC.arr*  $f$  then  $T (T (fst\ f, fst\ (snd\ f)), snd\ (snd\ f))$  else *C.null*

**lemma** *functor-ToTC*:  
**shows** *functor* *CCC.comp* *C* *ToTC*  
**using** *ToTC-def* **apply** *unfold-locales*  
**apply** *auto*[4]  
**proof** –  
**fix**  $f\ g$   
**assume**  $gf: CCC.seq\ g\ f$   
**show**  $ToTC\ (CCC.comp\ g\ f) = ToTC\ g \cdot ToTC\ f$   
**using**  $gf$  **unfolding** *CCC.seq-char* *CC.seq-char* *ToTC-def*  
**apply** *auto*  
**by** (*metis* *CC.comp-simp* *CC.seqI* *fst-conv* *preserves-comp* *preserves-seq* *snd-conv*)  
**qed**

**lemma** *ToTC-simp* [*simp*]:  
**assumes** *C.arr*  $f$  **and** *C.arr*  $g$  **and** *C.arr*  $h$   
**shows**  $ToTC\ (f, g, h) = T (T (f, g), h)$   
**using** *assms* *ToTC-def* *CCC.arr-char* **by** *simp*

**definition** *ToCT*  
**where** *ToCT*  $f \equiv$  if *CCC.arr*  $f$  then  $T (fst\ f, T (fst\ (snd\ f), snd\ (snd\ f)))$  else *C.null*

**lemma** *functor-ToCT*:  
**shows** *functor* *CCC.comp* *C* *ToCT*  
**using** *ToCT-def* **apply** *unfold-locales*  
**apply** *auto*[4]  
**proof** –  
**fix**  $f\ g$   
**assume**  $gf: CCC.seq\ g\ f$   
**show**  $ToCT\ (CCC.comp\ g\ f) = ToCT\ g \cdot ToCT\ f$   
**using**  $gf$  **unfolding** *CCC.seq-char* *CC.seq-char* *ToCT-def*  
**apply** *auto*  
**by** (*metis* *CC.comp-simp* *CC.seq-char* *as-nat-trans.preserves-comp-2* *fst-conv* *preserves-reflects-arr* *snd-conv*)  
**qed**

**lemma** *ToCT-simp* [*simp*]:  
**assumes** *C.arr*  $f$  **and** *C.arr*  $g$  **and** *C.arr*  $h$   
**shows**  $ToCT\ (f, g, h) = T (f, T (g, h))$   
**using** *assms* *ToCT-def* *CCC.arr-char* **by** *simp*

**end**

Our primary definition for “monoidal category” follows the somewhat non-traditional development in [2]. There a monoidal category is defined to be a category  $C$  equipped with a binary *tensor product* functor  $T: C \times C \rightarrow C$ , an *associativity isomorphism*,

which is a natural isomorphism  $\alpha: T o (T \times C) \rightarrow T o (C \times T)$ , a *unit object*  $\mathcal{I}$  of  $C$ , and an isomorphism  $\iota: T (\mathcal{I}, \mathcal{I}) \rightarrow \mathcal{I}$ , subject to two axioms: the *pentagon axiom*, which expresses the commutativity of certain pentagonal diagrams involving components of  $\alpha$ , and the left and right *unit axioms*, which state that the endofunctors  $T (\mathcal{I}, -)$  and  $T (-, \mathcal{I})$  are equivalences of categories. This definition is formalized in the *monoidal-category* locale.

In more traditional developments, the definition of monoidal category involves additional left and right *unitors* isomorphisms  $\lambda$  and  $\varrho$  and associated axioms involving their components. However, as is shown in [2] and formalized here, the unitors are uniquely determined by  $\alpha$  and their values  $\lambda(\mathcal{I})$  and  $\varrho(\mathcal{I})$  at  $\mathcal{I}$ , which coincide. Treating  $\lambda$  and  $\varrho$  as defined notions results in a more economical basic definition of monoidal category that requires less data to be given, and has a similar effect on the definition of “monoidal functor.” Moreover, in the context of the formalization of categories that we use here, the unit object  $\mathcal{I}$  also need not be given separately, as it can be obtained as the codomain of the isomorphism  $\iota$ .

```

locale monoidal-category =
  category C +
  CC: product-category C C +
  CCC: product-category C CC.comp +
  T: binary-endofunctor C T +
   $\alpha$ : natural-isomorphism CCC.comp C T.ToTC T.ToCT  $\alpha$  +
  L: equivalence-functor C C  $\lambda f$ . T (cod  $\iota$ , f) +
  R: equivalence-functor C C  $\lambda f$ . T (f, cod  $\iota$ )
for C :: 'a comp      (infixr · 55)
and T :: 'a * 'a  $\Rightarrow$  'a
and  $\alpha$  :: 'a * 'a * 'a  $\Rightarrow$  'a
and  $\iota$  :: 'a +
assumes  $\iota$ -in-hom':  $\langle \iota : T (\text{cod } \iota, \text{cod } \iota) \rightarrow \text{cod } \iota \rangle$ 
and  $\iota$ -is-iso: iso  $\iota$ 
and pentagon:  $\llbracket \text{ide } a; \text{ide } b; \text{ide } c; \text{ide } d \rrbracket \Longrightarrow$ 
      T (a,  $\alpha$  (b, c, d)) ·  $\alpha$  (a, T (b, c), d) · T ( $\alpha$  (a, b, c), d) =
       $\alpha$  (a, b, T (c, d)) ·  $\alpha$  (T (a, b), c, d)
begin

```

We now define helpful notation and abbreviations to improve readability. We did not define and use the notation  $\otimes$  for the tensor product in the definition of the locale because to define  $\otimes$  as a binary operator requires that it be in curried form, whereas for  $T$  to be a binary functor requires that it take a pair as its argument.

**definition** *unity* :: 'a ( $\mathcal{I}$ )  
**where** *unity*  $\equiv$  cod  $\iota$

**abbreviation** *L* :: 'a  $\Rightarrow$  'a  
**where** *L* f  $\equiv$  T ( $\mathcal{I}$ , f)

**abbreviation** *R* :: 'a  $\Rightarrow$  'a  
**where** *R* f  $\equiv$  T (f,  $\mathcal{I}$ )



**abbreviation** *tensor* (infixr  $\otimes$  53)  
**where**  $f \otimes g \equiv T(f, g)$

**abbreviation** *assoc* (a[-, -, -])  
**where**  $a[a, b, c] \equiv \alpha(a, b, c)$

In HOL we can just give the definitions of the left and right unitors “up front” without any preliminary work. Later we will have to show that these definitions have the right properties. The next two definitions define the values of the unitors when applied to identities; that is, their components as natural transformations.

**definition** *lunit* (l[-])  
**where**  $lunit\ a \equiv THE\ f.\ \langle f : \mathcal{I} \otimes a \rightarrow a \rangle \wedge \mathcal{I} \otimes f = (\iota \otimes a) \cdot inv\ a[\mathcal{I}, \mathcal{I}, a]$

**definition** *runit* (r[-])  
**where**  $runit\ a \equiv THE\ f.\ \langle f : a \otimes \mathcal{I} \rightarrow a \rangle \wedge f \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$

The next two definitions extend the unitors to all arrows, not just identities. Unfortunately, the traditional symbol  $\lambda$  for the left unitor is already reserved for a higher purpose, so we have to make do with a poor substitute.

**abbreviation**  $\wr$   
**where**  $\wr f \equiv if\ arr\ f\ then\ f \cdot \wr[dom\ f]\ else\ null$

**abbreviation**  $\rho$   
**where**  $\rho f \equiv if\ arr\ f\ then\ f \cdot \rho[dom\ f]\ else\ null$

We now embark upon a development of the consequences of the monoidal category axioms. One of our objectives is to be able to show that an interpretation of the *monoidal-category* locale induces an interpretation of a locale corresponding to a more traditional definition of monoidal category. Another is to obtain the facts we need to prove the coherence theorem.

**lemma**  *$\iota$ -in-hom* [intro]:  
**shows**  $\langle \iota : \mathcal{I} \otimes \mathcal{I} \rightarrow \mathcal{I} \rangle$   
**using** *unity-def  $\iota$ -in-hom'* **by** *force*

**lemma** *ide-unity* [simp]:  
**shows** *ide*  $\mathcal{I}$   
**using**  *$\iota$ -in-hom unity-def* **by** *auto*

**lemma** *tensor-in-hom* [simp]:  
**assumes**  $\langle f : a \rightarrow b \rangle$  **and**  $\langle g : c \rightarrow d \rangle$   
**shows**  $\langle f \otimes g : a \otimes c \rightarrow b \otimes d \rangle$   
**using** *assms T.preserves-hom CC.arr-char* **by** *simp*

**lemma** *arr-tensor* [simp]:  
**assumes** *arr*  $f$  **and** *arr*  $g$   
**shows** *arr*  $(f \otimes g)$   
**using** *assms* **by** *simp*

**lemma** *dom-tensor* [*simp*]:  
**assumes**  $\langle f : a \rightarrow b \rangle$  **and**  $\langle g : c \rightarrow d \rangle$   
**shows**  $\text{dom } (f \otimes g) = a \otimes c$   
**using** *assms* **by** *fastforce*

**lemma** *cod-tensor* [*simp*]:  
**assumes**  $\langle f : a \rightarrow b \rangle$  **and**  $\langle g : c \rightarrow d \rangle$   
**shows**  $\text{cod } (f \otimes g) = b \otimes d$   
**using** *assms* **by** *fastforce*

**lemma** *tensor-preserves-ide* [*simp*]:  
**assumes** *ide a* **and** *ide b*  
**shows** *ide*  $(a \otimes b)$   
**using** *assms* *T.preserves-ide* *CC.ide-char* **by** *simp*

**lemma** *tensor-preserves-iso* [*simp*]:  
**assumes** *iso f* **and** *iso g*  
**shows** *iso*  $(f \otimes g)$   
**using** *assms* **by** *simp*

**lemma** *inv-tensor* [*simp*]:  
**assumes** *iso f* **and** *iso g*  
**shows**  $\text{inv } (f \otimes g) = \text{inv } f \otimes \text{inv } g$   
**using** *assms* *T.preserves-inv* **by** *auto*

**lemma** *interchange*:  
**assumes** *seq h g* **and** *seq h' g'*  
**shows**  $(h \otimes h') \cdot (g \otimes g') = h \cdot g \otimes h' \cdot g'$   
**using** *assms* *T.preserves-comp* [*of*  $(h, h')$   $(g, g')$ ] **by** *simp*

**lemma**  $\alpha$ -*simp*:  
**assumes** *arr f* **and** *arr g* **and** *arr h*  
**shows**  $\alpha (f, g, h) = (f \otimes g \otimes h) \cdot \text{a}[\text{dom } f, \text{dom } g, \text{dom } h]$   
**using** *assms*  $\alpha$ .*is-natural-1* [*of*  $(f, g, h)$ ] **by** *simp*

**lemma** *assoc-in-hom* [*intro*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows**  $\langle \text{a}[a, b, c] : (a \otimes b) \otimes c \rightarrow a \otimes b \otimes c \rangle$   
**using** *assms* *CCC.in-homE* **by** *auto*

**lemma** *arr-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows** *arr*  $\text{a}[a, b, c]$   
**using** *assms* *assoc-in-hom* **by** *simp*

**lemma** *dom-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows**  $\text{dom } \text{a}[a, b, c] = (a \otimes b) \otimes c$   
**using** *assms* *assoc-in-hom* **by** *simp*

**lemma** *cod-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows**  $\text{cod } a[a, b, c] = a \otimes b \otimes c$   
**using** *assms assoc-in-hom* **by** *simp*

**lemma** *assoc-naturality*:  
**assumes** *arr f0* **and** *arr f1* **and** *arr f2*  
**shows**  $a[\text{cod } f0, \text{cod } f1, \text{cod } f2] \cdot ((f0 \otimes f1) \otimes f2) =$   
 $(f0 \otimes f1 \otimes f2) \cdot a[\text{dom } f0, \text{dom } f1, \text{dom } f2]$   
**using** *assms  $\alpha$ .naturality* **by** *auto*

**lemma** *iso-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows** *iso*  $a[a, b, c]$   
**using** *assms  $\alpha$ .preserves-iso* **by** *simp*

The next result uses the fact that the functor  $L$  is an equivalence (and hence faithful) to show the existence of a unique solution to the characteristic equation used in the definition of a component  $l[a]$  of the left unitor. It follows that  $l[a]$ , as given by our definition using definite description, satisfies this characteristic equation and is therefore uniquely determined by  $\otimes$ ,  $\alpha$ , and  $\iota$ .

**lemma** *lunit-char*:  
**assumes** *ide a*  
**shows**  $\langle l[a] : \mathcal{I} \otimes a \rightarrow a \rangle$  **and**  $\mathcal{I} \otimes l[a] = (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a]$   
**and**  $\exists! f. \langle f : \mathcal{I} \otimes a \rightarrow a \rangle \wedge \mathcal{I} \otimes f = (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a]$   
**proof** –  
**obtain**  $F \eta \varepsilon$  **where**  $L$ : *equivalence-of-categories*  $C \ C \ F$   $(\lambda f. \mathcal{I} \otimes f) \eta \varepsilon$   
**using** *L.induces-equivalence unity-def* **by** *auto*  
**interpret**  $L$ : *equivalence-of-categories*  $C \ C \ F$   $\langle \lambda f. \mathcal{I} \otimes f \rangle \eta \varepsilon$   
**using**  $L$  **by** *auto*  
**let**  $?P = \lambda f. \langle f : \mathcal{I} \otimes a \rightarrow a \rangle \wedge \mathcal{I} \otimes f = (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a]$   
**have**  $\langle (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a] : \mathcal{I} \otimes \mathcal{I} \otimes a \rightarrow \mathcal{I} \otimes a \rangle$   
**proof**  
**show**  $\langle \iota \otimes a : (\mathcal{I} \otimes \mathcal{I}) \otimes a \rightarrow \mathcal{I} \otimes a \rangle$   
**using** *assms ide-in-hom* **by** (*intro tensor-in-hom, auto*)  
**show**  $\langle \text{inv } a[\mathcal{I}, \mathcal{I}, a] : \mathcal{I} \otimes \mathcal{I} \otimes a \rightarrow (\mathcal{I} \otimes \mathcal{I}) \otimes a \rangle$   
**using** *assms* **by** *auto*  
**qed**  
**moreover** **have** *ide*  $(\mathcal{I} \otimes a)$  **using** *assms* **by** *simp*  
**ultimately** **have**  $\exists f. ?P f$   
**using** *assms L.is-full* **by** *blast*  
**moreover** **have**  $\bigwedge f f'. ?P f \implies ?P f' \implies f = f'$   
**proof** –  
**fix**  $f f'$   
**assume**  $f: ?P f$  **and**  $f': ?P f'$   
**have** *par*  $f f'$  **using**  $f f'$  **by** *force*  
**show**  $f = f'$  **using**  $f f'$  *L.is-faithful* [*of f f'*] **by** *force*  
**qed**

**ultimately show**  $\exists!f. ?P f$  **by** *blast*  
**hence**  $1: ?P \mathbb{1}[a]$   
**unfolding** *lunit-def* **using** *theI'* [of  $?P$ ] **by** *auto*  
**show**  $\llbracket \mathbb{1}[a] : \mathcal{I} \otimes a \rightarrow a \rrbracket$  **using**  $1$  **by** *fast*  
**show**  $\mathcal{I} \otimes \mathbb{1}[a] = (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a]$  **using**  $1$  **by** *fast*  
**qed**

**lemma** *l-ide-simp*:  
**assumes** *ide a*  
**shows**  $\mathbb{1} a = \mathbb{1}[a]$   
**using** *assms lunit-char comp-cod-arr ide-in-hom* **by** (*metis in-homE*)

**lemma** *lunit-in-hom [intro]*:  
**assumes** *ide a*  
**shows**  $\llbracket \mathbb{1}[a] : \mathcal{I} \otimes a \rightarrow a \rrbracket$   
**using** *assms lunit-char(1)* **by** *blast*

**lemma** *arr-lunit [simp]*:  
**assumes** *ide a*  
**shows** *arr*  $\mathbb{1}[a]$   
**using** *assms lunit-in-hom* **by** *auto*

**lemma** *dom-lunit [simp]*:  
**assumes** *ide a*  
**shows** *dom*  $\mathbb{1}[a] = \mathcal{I} \otimes a$   
**using** *assms lunit-in-hom* **by** *auto*

**lemma** *cod-lunit [simp]*:  
**assumes** *ide a*  
**shows** *cod*  $\mathbb{1}[a] = a$   
**using** *assms lunit-in-hom* **by** *auto*

As the right-hand side of the characteristic equation for  $\mathcal{I} \otimes \mathbb{1}[a]$  is an isomorphism, and the equivalence functor  $L$  reflects isomorphisms, it follows that  $\mathbb{1}[a]$  is an isomorphism.

**lemma** *iso-lunit [simp]*:  
**assumes** *ide a*  
**shows** *iso*  $\mathbb{1}[a]$   
**using** *assms lunit-char(2)  $\iota$ -is-iso ide-unity isos-compose iso-assoc iso-inv-iso  $\iota$ -in-hom L.reflects-iso arr-lunit arr-tensor ideD(1) ide-is-iso lunit-in-hom tensor-preserves-iso unity-def*  
**by** *metis*

To prove that an arrow  $f$  is equal to  $\mathbb{1}[a]$  we need only show that it is parallel to  $\mathbb{1}[a]$  and that  $\mathcal{I} \otimes f$  satisfies the same characteristic equation as  $\mathcal{I} \otimes \mathbb{1}[a]$  does.

**lemma** *lunit-eqI*:  
**assumes**  $\llbracket f : \mathcal{I} \otimes a \rightarrow a \rrbracket$  **and**  $\mathcal{I} \otimes f = (\iota \otimes a) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, a]$   
**shows**  $f = \mathbb{1}[a]$   
**proof** –  
**have** *ide a* **using** *assms(1)* **by** *auto*

**thus** *?thesis*  
**using** *assms lunit-char the1-equality* **by** *blast*  
**qed**

The next facts establish the corresponding results for the components of the right unitor.

**lemma** *runit-char*:  
**assumes** *ide a*  
**shows**  $\langle r[a] : a \otimes \mathcal{I} \rightarrow a \rangle$  **and**  $r[a] \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$   
**and**  $\exists ! f. \langle f : a \otimes \mathcal{I} \rightarrow a \rangle \wedge f \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$   
**proof** –  
**obtain**  $F \eta \varepsilon$  **where**  $R$ : *equivalence-of-categories C C F*  $(\lambda f. f \otimes \mathcal{I}) \eta \varepsilon$   
**using** *R.induces-equivalence ι-in-hom* **by** *auto*  
**interpret**  $R$ : *equivalence-of-categories C C F*  $\langle \lambda f. f \otimes \mathcal{I} \rangle \eta \varepsilon$   
**using**  $R$  **by** *auto*  
**let**  $?P = \lambda f. \langle f : a \otimes \mathcal{I} \rightarrow a \rangle \wedge f \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$   
**have**  $\langle (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}] : (a \otimes \mathcal{I}) \otimes \mathcal{I} \rightarrow a \otimes \mathcal{I} \rangle$   
**using** *assms* **by** *fastforce*  
**moreover** **have** *ide*  $(a \otimes \mathcal{I})$  **using** *assms* **by** *simp*  
**ultimately** **have**  $\exists f. ?P f$   
**using** *assms R.is-full*  $[of\ a\ a \otimes \mathcal{I}\ (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]]$  **by** *blast*  
**moreover** **have**  $\bigwedge f f'. ?P f \implies ?P f' \implies f = f'$   
**proof** –  
**fix**  $f f'$   
**assume**  $f: ?P f$  **and**  $f': ?P f'$   
**have** *par*  $f f'$  **using**  $f f'$  **by** *force*  
**show**  $f = f'$  **using**  $f f'$  *R.is-faithful*  $[of\ f f']$  **by** *force*  
**qed**  
**ultimately** **show**  $\exists ! f. ?P f$  **by** *blast*  
**hence**  $1: ?P\ r[a]$  **unfolding** *runit-def* **using** *theI'*  $[of\ ?P]$  **by** *fast*  
**show**  $\langle r[a] : a \otimes \mathcal{I} \rightarrow a \rangle$  **using**  $1$  **by** *fast*  
**show**  $r[a] \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$  **using**  $1$  **by** *fast*  
**qed**

**lemma** *ρ-ide-simp*:  
**assumes** *ide a*  
**shows**  $\rho\ a = r[a]$   
**using** *assms runit-char*  $[of\ a]$  *comp-cod-arr* **by** *auto*

**lemma** *runit-in-hom*  $[intro]$ :  
**assumes** *ide a*  
**shows**  $\langle r[a] : a \otimes \mathcal{I} \rightarrow a \rangle$   
**using** *assms runit-char(1)* **by** *blast*

**lemma** *arr-runit*  $[simp]$ :  
**assumes** *ide a*  
**shows** *arr*  $r[a]$   
**using** *assms runit-in-hom* **by** *blast*

**lemma** *dom-runit* [*simp*]:  
**assumes** *ide a*  
**shows**  $\text{dom } r[a] = a \otimes \mathcal{I}$   
**using** *assms runit-in-hom* **by** *blast*

**lemma** *cod-runit* [*simp*]:  
**assumes** *ide a*  
**shows**  $\text{cod } r[a] = a$   
**using** *assms runit-in-hom* **by** *blast*

**lemma** *runit-eqI*:  
**assumes**  $\langle f : a \otimes \mathcal{I} \rightarrow a \rangle$  **and**  $f \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$   
**shows**  $f = r[a]$   
**proof** –  
**have** *ide a* **using** *assms(1)* **by** *auto*  
**thus** *?thesis*  
**using** *assms runit-char the1-equality* **by** *blast*  
**qed**

**lemma** *iso-runit* [*simp*]:  
**assumes** *ide a*  
**shows** *iso r[a]*  
**using** *assms  $\iota$ -is-iso iso-inv-iso isos-compose ide-is-iso R.preserves-reflects-arr arrI ide-unity iso-assoc runit-char tensor-preserves-iso R.reflects-iso unity-def*  
**by** *metis*

We can now show that the components of the left and right unitors have the naturality properties required of a natural transformation.

**lemma** *lunit-naturality*:  
**assumes** *arr f*  
**shows**  $l[\text{cod } f] \cdot (\mathcal{I} \otimes f) = f \cdot l[\text{dom } f]$   
**proof** –  
**interpret**  $\alpha'$ : *inverse-transformation CCC.comp C T.ToTC T.ToCT  $\alpha$  ..*  
**have** *par*:  $l[\text{cod } f] \cdot (\mathcal{I} \otimes f) \cdot (f \cdot l[\text{dom } f])$   
**using** *assms* **by** *simp*  
**moreover** **have**  $\mathcal{I} \otimes l[\text{cod } f] \cdot (\mathcal{I} \otimes f) = \mathcal{I} \otimes f \cdot l[\text{dom } f]$   
**proof** –  
**have**  $\mathcal{I} \otimes l[\text{cod } f] \cdot (\mathcal{I} \otimes f) = ((\iota \otimes \text{cod } f) \cdot ((\mathcal{I} \otimes \mathcal{I}) \otimes f)) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, \text{dom } f]$   
**using** *assms interchange [of  $\mathcal{I} \mathcal{I} \mathcal{I} \otimes f$ ] lunit-char(2)  $\iota$ -in-hom unity-def  $\alpha'$ .naturality [of  $(\mathcal{I}, \mathcal{I}, f)$ ] comp-assoc*  
**by** *auto*  
**also** **have**  $\dots = ((\mathcal{I} \otimes f) \cdot (\iota \otimes \text{dom } f)) \cdot \text{inv } a[\mathcal{I}, \mathcal{I}, \text{dom } f]$   
**using** *assms interchange comp-arr-dom comp-cod-arr  $\iota$ -in-hom* **by** *auto*  
**also** **have**  $\dots = (\mathcal{I} \otimes f) \cdot (\mathcal{I} \otimes l[\text{dom } f])$   
**using** *assms  $\iota$ -in-hom lunit-char(2) comp-assoc* **by** *auto*  
**also** **have**  $\dots = \mathcal{I} \otimes f \cdot l[\text{dom } f]$   
**using** *assms interchange* **by** *simp*  
**finally** **show** *?thesis* **by** *blast*

```

qed
ultimately show l[cod f] · (I ⊗ f) = f · l[dom f]
  using L.is-faithful-unity-def by metis
qed

```

**lemma** *runit-naturality*:

**assumes** *arr f*

**shows**  $r[\text{cod } f] \cdot (f \otimes I) = f \cdot r[\text{dom } f]$

**proof** –

**have** *par* ( $r[\text{cod } f] \cdot (f \otimes I)$ ) ( $f \cdot r[\text{dom } f]$ )

**using** *assms* **by** *force*

**moreover** **have**  $r[\text{cod } f] \cdot (f \otimes I) \otimes I = f \cdot r[\text{dom } f] \otimes I$

**proof** –

**have**  $r[\text{cod } f] \cdot (f \otimes I) \otimes I = (\text{cod } f \otimes \iota) \cdot a[\text{cod } f, I, I] \cdot ((f \otimes I) \otimes I)$

**using** *assms* *interchange* [of  $I I I \otimes f$   $r[\text{cod } f]$ ] *runit-char*(2) *ι-in-hom* *unity-def*  
*comp-assoc*

**by** *auto*

**also** **have**  $\dots = (\text{cod } f \otimes \iota) \cdot (f \otimes I \otimes I) \cdot a[\text{dom } f, I, I]$

**using** *assms* *α.naturality* [of  $(f, I, I)$ ] **by** *auto*

**also** **have**  $\dots = ((\text{cod } f \otimes \iota) \cdot (f \otimes I \otimes I)) \cdot a[\text{dom } f, I, I]$

**using** *comp-assoc* **by** *simp*

**also** **have**  $\dots = ((f \otimes I) \cdot (\text{dom } f \otimes \iota)) \cdot a[\text{dom } f, I, I]$

**using** *assms* *ι-in-hom* *interchange* *comp-arr-dom* *comp-cod-arr* **by** *auto*

**also** **have**  $\dots = (f \otimes I) \cdot (r[\text{dom } f] \otimes I)$

**using** *assms* *runit-char* *comp-assoc* **by** *auto*

**also** **have**  $\dots = f \cdot r[\text{dom } f] \otimes I$

**using** *assms* *interchange* **by** *simp*

**finally** **show** *?thesis* **by** *blast*

**qed**

**ultimately** **show**  $r[\text{cod } f] \cdot (f \otimes I) = f \cdot r[\text{dom } f]$

**using** *R.is-faithful-unity-def* **by** *metis*

**qed**

**end**

The following locale is an extension of *monoidal-category* that has the unitors and their inverses, as well as the inverse to the associator, conveniently pre-interpreted.

**locale** *extended-monoidal-category* =

*monoidal-category* +

$\alpha'$ : *inverse-transformation*  $CCC.comp C T.ToTC T.ToCT \alpha$  +

$\mathfrak{l}$ : *natural-isomorphism*  $C C L map \mathfrak{l}$  +

$\mathfrak{l}'$ : *inverse-transformation*  $C C L map \mathfrak{l}$  +

$\varrho$ : *natural-isomorphism*  $C C R map \varrho$  +

$\varrho'$ : *inverse-transformation*  $C C R map \varrho$

Next we show that an interpretation of *monoidal-category* extends to an interpretation of *extended-monoidal-category* and we arrange for the former locale to inherit from the latter.

**context** *monoidal-category*

**begin**

**interpretation**  $\alpha'$ : *inverse-transformation*  $CCC.comp\ C\ T.ToTC\ T.ToCT\ \alpha\ ..$

**interpretation**  $\mathfrak{l}$ : *natural-transformation*  $C\ C\ L\ map\ \mathfrak{l}$

**proof** –

**interpret**  $\mathfrak{l}$ : *transformation-by-components*  $C\ C\ L\ map\ \langle\lambda a. \mathfrak{l}[a]\rangle$

**using** *lunit-in-hom lunit-naturality unity-def  $\iota$ -in-hom' L.is-extensional*

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

**have**  $\mathfrak{l}.map = \mathfrak{l}$

**using**  *$\mathfrak{l}.is-natural-1\ \mathfrak{l}.is-extensional$*  **by** *auto*

**thus** *natural-transformation*  $C\ C\ L\ map\ \mathfrak{l}$

**using**  *$\mathfrak{l}.natural-transformation-axioms$*  **by** *auto*

**qed**

**interpretation**  $\mathfrak{l}$ : *natural-isomorphism*  $C\ C\ L\ map\ \mathfrak{l}$

**apply** *unfold-locales*

**using** *iso-lunit  $\mathfrak{l}$ -ide-simp* **by** *simp*

**interpretation**  $\varrho$ : *natural-transformation*  $C\ C\ R\ map\ \varrho$

**proof** –

**interpret**  $\varrho$ : *transformation-by-components*  $C\ C\ R\ map\ \langle\lambda a. \varrho[a]\rangle$

**using** *runit-naturality unity-def  $\iota$ -in-hom' R.is-extensional*

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

**have**  $\varrho.map = \varrho$

**using**  *$\varrho.is-natural-1\ \varrho.is-extensional$*  **by** *auto*

**thus** *natural-transformation*  $C\ C\ R\ map\ \varrho$

**using**  *$\varrho.natural-transformation-axioms$*  **by** *auto*

**qed**

**interpretation**  $\varrho$ : *natural-isomorphism*  $C\ C\ R\ map\ \varrho$

**apply** *unfold-locales*

**using**  *$\varrho$ -ide-simp* **by** *simp*

**lemma** *induces-extended-monoidal-category:*

**shows** *extended-monoidal-category*  $C\ T\ \alpha\ \iota\ ..$

**end**

**sublocale** *monoidal-category*  $\subseteq$  *extended-monoidal-category*

**using** *induces-extended-monoidal-category* **by** *auto*

**context** *monoidal-category*

**begin**

**abbreviation**  $\alpha'$

**where**  $\alpha' \equiv \alpha'.map$

**lemma** *natural-isomorphism- $\alpha'$ :*



**shows** *natural-isomorphism*  $CCC.comp\ C\ T.ToCT\ T.ToTC\ \alpha' ..$

**abbreviation**  $assoc' (a^{-1}[-, -, -])$

**where**  $a^{-1}[a, b, c] \equiv inv\ a[a, b, c]$

**lemma**  $\alpha'$ -*ide-simp*:

**assumes** *ide a and ide b and ide c*

**shows**  $\alpha' (a, b, c) = a^{-1}[a, b, c]$

**using** *assms*  $\alpha'$ .*inverts-components inverse-unique* **by force**

**lemma**  $\alpha'$ -*simp*:

**assumes** *arr f and arr g and arr h*

**shows**  $\alpha' (f, g, h) = ((f \otimes g) \otimes h) \cdot a^{-1}[dom\ f, dom\ g, dom\ h]$

**using** *assms*  $T.ToTC$ -*simp*  $\alpha'$ .*is-natural-1*  $\alpha'$ -*ide-simp* **by force**

**lemma** *assoc-inv*:

**assumes** *ide a and ide b and ide c*

**shows** *inverse-arrows*  $a[a, b, c]\ a^{-1}[a, b, c]$

**using** *assms* *inv-is-inverse* **by simp**

**lemma** *assoc'-in-hom* [*intro*]:

**assumes** *ide a and ide b and ide c*

**shows**  $\langle\langle a^{-1}[a, b, c] : a \otimes b \otimes c \rightarrow (a \otimes b) \otimes c \rangle\rangle$

**using** *assms* **by auto**

**lemma** *arr-assoc'* [*simp*]:

**assumes** *ide a and ide b and ide c*

**shows** *arr*  $a^{-1}[a, b, c]$

**using** *assms* **by simp**

**lemma** *dom-assoc'* [*simp*]:

**assumes** *ide a and ide b and ide c*

**shows** *dom*  $a^{-1}[a, b, c] = a \otimes b \otimes c$

**using** *assms* **by simp**

**lemma** *cod-assoc'* [*simp*]:

**assumes** *ide a and ide b and ide c*

**shows** *cod*  $a^{-1}[a, b, c] = (a \otimes b) \otimes c$

**using** *assms* **by simp**

**lemma** *comp-assoc-assoc'* [*simp*]:

**assumes** *ide a and ide b and ide c*

**shows**  $a[a, b, c] \cdot a^{-1}[a, b, c] = a \otimes (b \otimes c)$

**and**  $a^{-1}[a, b, c] \cdot a[a, b, c] = (a \otimes b) \otimes c$

**using** *assms* *assoc-inv* *comp-arr-inv* *comp-inv-arr* **by auto**

**lemma** *assoc'-naturality*:

**assumes** *arr f0 and arr f1 and arr f2*

**shows**  $((f0 \otimes f1) \otimes f2) \cdot a^{-1}[dom\ f0, dom\ f1, dom\ f2] =$

$a^{-1}[\text{cod } f0, \text{cod } f1, \text{cod } f2] \cdot (f0 \otimes f1 \otimes f2)$   
**using** *assms*  $\alpha'.\text{natural}$  **by** *auto*

**abbreviation**  $l'$   
**where**  $l' \equiv l'.\text{map}$

**abbreviation**  $\text{lunit}'$   $(l^{-1}[-])$   
**where**  $l^{-1}[a] \equiv \text{inv } l[a]$

**lemma**  $l'\text{-ide-simp}$ :  
**assumes** *ide a*  
**shows**  $l'.\text{map } a = l^{-1}[a]$   
**using** *assms*  $l'.\text{inverts-components}$   $l\text{-ide-simp}$  *inverse-unique* **by** *force*

**lemma**  $\text{lunit}'\text{-inv}$ :  
**assumes** *ide a*  
**shows** *inverse-arrows*  $l[a]$   $l^{-1}[a]$   
**using** *assms* *inv-is-inverse* **by** *simp*

**lemma**  $\text{lunit}'\text{-in-hom}$  [*intro*]:  
**assumes** *ide a*  
**shows**  $\langle l^{-1}[a] : a \rightarrow \mathcal{I} \otimes a \rangle$   
**using** *assms* **by** *auto*

**lemma**  $\text{comp-lunit}'\text{-lunit}'$  [*simp*]:  
**assumes** *ide a*  
**shows**  $l[a] \cdot l^{-1}[a] = a$   
**and**  $l^{-1}[a] \cdot l[a] = \mathcal{I} \otimes a$   
**proof** –  
**show**  $l[a] \cdot l^{-1}[a] = a$   
**using** *assms* *comp-arr-inv*  $\text{lunit}'\text{-inv}$  **by** *fastforce*  
**show**  $l^{-1}[a] \cdot l[a] = \mathcal{I} \otimes a$   
**using** *assms* *comp-arr-inv*  $\text{lunit}'\text{-inv}$  **by** *fastforce*  
**qed**

**lemma**  $\text{lunit}'\text{-natural}$ :  
**assumes** *arr f*  
**shows**  $(\mathcal{I} \otimes f) \cdot l^{-1}[\text{dom } f] = l^{-1}[\text{cod } f] \cdot f$   
**using** *assms*  $l'.\text{natural}$   $l'\text{-ide-simp}$  **by** *simp*

**abbreviation**  $\varrho'$   
**where**  $\varrho' \equiv \varrho'.\text{map}$

**abbreviation**  $\text{runit}'$   $(r^{-1}[-])$   
**where**  $r^{-1}[a] \equiv \text{inv } r[a]$

**lemma**  $\varrho'\text{-ide-simp}$ :  
**assumes** *ide a*  
**shows**  $\varrho'.\text{map } a = r^{-1}[a]$

**using** *assms*  $\varrho'$ .*inverts-components*  $\varrho$ -*ide-simp* *inverse-unique* **by** *auto*

**lemma** *runit-inv*:

**assumes** *ide a*

**shows** *inverse-arrows*  $r[a]$   $r^{-1}[a]$

**using** *assms inv-is-inverse* **by** *simp*

**lemma** *runit'-in-hom* [*intro*]:

**assumes** *ide a*

**shows**  $\langle r^{-1}[a] : a \rightarrow a \otimes \mathcal{I} \rangle$

**using** *assms* **by** *auto*

**lemma** *comp-runit-runit'* [*simp*]:

**assumes** *ide a*

**shows**  $r[a] \cdot r^{-1}[a] = a$

**and**  $r^{-1}[a] \cdot r[a] = a \otimes \mathcal{I}$

**proof** –

**show**  $r[a] \cdot r^{-1}[a] = a$

**using** *assms runit-inv* **by** *fastforce*

**show**  $r^{-1}[a] \cdot r[a] = a \otimes \mathcal{I}$

**using** *assms runit-inv* **by** *fastforce*

**qed**

**lemma** *runit'-naturality*:

**assumes** *arr f*

**shows**  $(f \otimes \mathcal{I}) \cdot r^{-1}[\text{dom } f] = r^{-1}[\text{cod } f] \cdot f$

**using** *assms*  $\varrho'$ .*naturality*  $\varrho'$ -*ide-simp* **by** *simp*

**lemma** *lunit-commutes-with-L*:

**assumes** *ide a*

**shows**  $l[\mathcal{I} \otimes a] = \mathcal{I} \otimes l[a]$

**using** *assms lunit-naturality lunit-in-hom iso-lunit iso-is-section*

*section-is-mono monoE L.preserves-ide arrI cod-lunit dom-lunit seqI*

*unity-def*

**by** *metis*

**lemma** *runit-commutes-with-R*:

**assumes** *ide a*

**shows**  $r[a \otimes \mathcal{I}] = r[a] \otimes \mathcal{I}$

**using** *assms runit-naturality runit-in-hom iso-runit iso-is-section*

*section-is-mono monoE R.preserves-ide arrI cod-runit dom-runit seqI*

*unity-def*

**by** *metis*

The components of the left and right unitors are related via a “triangle” diagram that also involves the associator. The proof follows [2], Proposition 2.2.3.

**lemma** *triangle*:

**assumes** *ide a* **and** *ide b*

**shows**  $(a \otimes l[b]) \cdot a[a, \mathcal{I}, b] = r[a] \otimes b$

**proof** –

We show that the lower left triangle in the following diagram commutes.

$$\begin{array}{ccccc}
 ((a \otimes \mathcal{I}) \otimes \mathcal{I}) \otimes b & \xrightarrow{a[a, \mathcal{I}, \mathcal{I}] \otimes b} & (a \otimes \mathcal{I} \otimes \mathcal{I}) \otimes b & & \\
 \downarrow a[a \otimes \mathcal{I}, \mathcal{I}, b] & \searrow (r[a] \otimes \mathcal{I}) \otimes b & \swarrow (a \otimes \iota) \otimes b & & \downarrow a[a, \mathcal{I} \otimes \mathcal{I}, b] \\
 & (a \otimes \mathcal{I}) \otimes b & & & \\
 & \downarrow a[a, \mathcal{I}, b] & & & \\
 & a \otimes \mathcal{I} \otimes b & & & \\
 & \swarrow r[a] \otimes \mathcal{I} \otimes b & \nwarrow a \otimes \iota \otimes b & & \\
 (a \otimes \mathcal{I}) \otimes \mathcal{I} \otimes b & & a \otimes (\mathcal{I} \otimes \mathcal{I}) \otimes b & & \\
 \swarrow a[a, \mathcal{I}, \mathcal{I} \otimes b] & \nearrow a \otimes l[\mathcal{I} \otimes b] & \swarrow a \otimes a[\mathcal{I}, \mathcal{I}, b] & & \\
 & a \otimes \mathcal{I} \otimes \mathcal{I} \otimes b & & & 
 \end{array}$$

**have** \*:  $(a \otimes l[\mathcal{I} \otimes b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b] = r[a] \otimes \mathcal{I} \otimes b$

**proof** –

**have** 1:  $((a \otimes l[\mathcal{I} \otimes b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b]) \cdot a[a \otimes \mathcal{I}, \mathcal{I}, b]$   
 $= (r[a] \otimes \mathcal{I} \otimes b) \cdot a[a \otimes \mathcal{I}, \mathcal{I}, b]$

**proof** –

**have**  $((a \otimes l[\mathcal{I} \otimes b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b]) \cdot a[a \otimes \mathcal{I}, \mathcal{I}, b] =$   
 $((a \otimes l[\mathcal{I} \otimes b]) \cdot (a \otimes a[\mathcal{I}, \mathcal{I}, b])) \cdot a[a, \mathcal{I} \otimes \mathcal{I}, b] \cdot (a[a, \mathcal{I}, \mathcal{I}] \otimes b)$

**using** *assms pentagon comp-assoc* **by** *auto*

**also have** ...  $= (a \otimes ((\mathcal{I} \otimes l[b]) \cdot a[\mathcal{I}, \mathcal{I}, b])) \cdot a[a, \mathcal{I} \otimes \mathcal{I}, b] \cdot (a[a, \mathcal{I}, \mathcal{I}] \otimes b)$

**using** *assms interchange lunit-commutes-with-L* **by** *simp*

**also have** ...  $= ((a \otimes (\iota \otimes b)) \cdot a[a, \mathcal{I} \otimes \mathcal{I}, b]) \cdot (a[a, \mathcal{I}, \mathcal{I}] \otimes b)$

**using** *assms lunit-char ι-in-hom comp-arr-dom comp-assoc* **by** *auto*

**also have** ...  $= (a[a, \mathcal{I}, b] \cdot ((a \otimes \iota) \otimes b)) \cdot (a[a, \mathcal{I}, \mathcal{I}] \otimes b)$

**using** *assms ι-in-hom assoc-naturality [of a ι b]* **by** *fastforce*

**also have** ...  $= a[a, \mathcal{I}, b] \cdot ((r[a] \otimes \mathcal{I}) \otimes b)$

**using** *assms ι-in-hom interchange runit-char(2) comp-assoc* **by** *auto*

**also have** ...  $= (r[a] \otimes \mathcal{I} \otimes b) \cdot a[a \otimes \mathcal{I}, \mathcal{I}, b]$

**using** *assms assoc-naturality [of r[a] I b]* **by** *simp*

**finally show** *?thesis* **by** *blast*

**qed**

**show** *?thesis*

**proof** –

**have** *epi*  $a[a \otimes \mathcal{I}, \mathcal{I}, b]$

**using** *assms iso-assoc iso-is-retraction retraction-is-epi* **by** *simp*

**thus** *?thesis*

**using** 1 *assms epiE [of a[a ⊗ I, I, b] (a ⊗ l[I ⊗ b]) · a[a, I, I ⊗ b]]*

**by** *fastforce*

**qed**

**qed**

In [2] it merely states that the preceding result suffices “because any object of  $C$  is isomorphic to one of the form  $\mathcal{I} \otimes b$ .” However, it seems a little bit more involved than that to formally transport the equation (\*) along the isomorphism  $l[b]$  from  $\mathcal{I} \otimes b$  to  $b$ .

**have**  $(a \otimes l[b]) \cdot a[a, \mathcal{I}, b] = ((a \otimes l[b]) \cdot (a \otimes l[\mathcal{I} \otimes b]) \cdot (a \otimes \mathcal{I} \otimes l^{-1}[b])) \cdot (a \otimes \mathcal{I} \otimes l[b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b] \cdot ((a \otimes \mathcal{I}) \otimes l^{-1}[b])$

**proof** –

**have**  $a[a, \mathcal{I}, b] = (a \otimes \mathcal{I} \otimes l[b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b] \cdot ((a \otimes \mathcal{I}) \otimes l^{-1}[b])$

**proof** –

**have**  $(a \otimes \mathcal{I} \otimes l[b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b] \cdot ((a \otimes \mathcal{I}) \otimes l^{-1}[b]) = ((a \otimes \mathcal{I} \otimes l[b]) \cdot (a \otimes \mathcal{I} \otimes l^{-1}[b])) \cdot a[a, \mathcal{I}, b]$

**using** *assms assoc-naturality [of a  $\mathcal{I} l^{-1}[b]$  comp-*assoc* by simp*

**also have**  $\dots = a[a, \mathcal{I}, b]$

**using** *assms inv-is-inverse interchange comp-cod-arr by simp*

**finally show** *?thesis by auto*

**qed**

**moreover have**  $a \otimes l[b] = (a \otimes l[b]) \cdot (a \otimes l[\mathcal{I} \otimes b]) \cdot (a \otimes \mathcal{I} \otimes l^{-1}[b])$

**using** *assms lunit-commutes-with-L comp-arr-dom interchange by auto*

**ultimately show** *?thesis by argo*

**qed**

**also have**  $\dots = (a \otimes l[b]) \cdot (a \otimes l[\mathcal{I} \otimes b]) \cdot ((a \otimes \mathcal{I} \otimes l^{-1}[b]) \cdot (a \otimes \mathcal{I} \otimes l[b])) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b] \cdot ((a \otimes \mathcal{I}) \otimes l^{-1}[b])$

**using** *assms comp-*assoc* by auto*

**also have**  $\dots = (a \otimes l[b]) \cdot ((a \otimes l[\mathcal{I} \otimes b]) \cdot a[a, \mathcal{I}, \mathcal{I} \otimes b]) \cdot ((a \otimes \mathcal{I}) \otimes l^{-1}[b])$

**using** *assms interchange comp-cod-arr comp-*assoc* by auto*

**also have**  $\dots = r[a] \otimes b$

**using** *assms \* interchange runit-char(1) comp-arr-dom comp-cod-arr by auto*

**finally show** *?thesis by blast*

**qed**

**lemma** *lunit-tensor-gen:*

**assumes** *ide a and ide b and ide c*

**shows**  $(a \otimes l[b \otimes c]) \cdot (a \otimes a[\mathcal{I}, b, c]) = a \otimes l[b] \otimes c$

**proof** –

We show that the lower right triangle in the following diagram commutes.

$$\begin{array}{ccccc}
((a \otimes \mathcal{I}) \otimes b) \otimes c & \xrightarrow{a[a, \mathcal{I}, b] \otimes c} & (a \otimes \mathcal{I} \otimes b) \otimes c & & \\
\downarrow a[a \otimes \mathcal{I}, b, c] & \searrow r[a] \otimes b \otimes c & \swarrow (a \otimes l[b]) \otimes c & & \\
& (a \otimes b) \otimes c & & & \\
& \downarrow a[a, b, c] & & & \\
& a \otimes b \otimes c & & & \\
& \uparrow a \otimes l[b \otimes c] & & & \\
(a \otimes \mathcal{I}) \otimes b \otimes c & \xrightarrow{r[a] \otimes b \otimes c} & a \otimes b \otimes c & \xleftarrow{a \otimes l[b] \otimes c} & a \otimes (\mathcal{I} \otimes b) \otimes c \\
& \searrow a[a, \mathcal{I}, b \otimes c] & \uparrow a \otimes l[b \otimes c] & \swarrow a \otimes a[\mathcal{I}, b, c] & \\
& & a \otimes \mathcal{I} \otimes b \otimes c & & 
\end{array}$$

have  $((a \otimes l[b \otimes c]) \cdot (a \otimes a[\mathcal{I}, b, c])) \cdot (a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c)) =$   
 $((a \otimes l[b \otimes c]) \cdot a[a, \mathcal{I}, b \otimes c]) \cdot a[a \otimes \mathcal{I}, b, c]$

using *assms pentagon comp-assoc* by *simp*

also have ... =  $(r[a] \otimes (b \otimes c)) \cdot a[a \otimes \mathcal{I}, b, c]$

using *assms triangle* by *auto*

also have ... =  $a[a, b, c] \cdot ((r[a] \otimes b) \otimes c)$

using *assms assoc-naturality* [of  $r[a]$   $b$   $c$ ] by *auto*

also have ... =  $(a[a, b, c] \cdot ((a \otimes l[b]) \otimes c)) \cdot (a[a, \mathcal{I}, b] \otimes c)$

using *assms triangle interchange comp-assoc* by *auto*

also have ... =  $(a \otimes (l[b] \otimes c)) \cdot (a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c))$

using *assms assoc-naturality* [of  $a$   $l[b]$   $c$ ] *comp-assoc* by *auto*

finally have 1:  $((a \otimes l[b \otimes c]) \cdot (a \otimes a[\mathcal{I}, b, c])) \cdot a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c)$   
 $= (a \otimes (l[b] \otimes c)) \cdot a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c)$

by *blast*

The result follows by cancelling the isomorphism  $a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c)$

have 2: *iso*  $(a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c))$

using *assms isos-compose* by *simp*

moreover have

*seq*  $((a \otimes l[b \otimes c]) \cdot (a \otimes a[\mathcal{I}, b, c])) (a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c))$

using *assms* by *auto*

moreover have *seq*  $(a \otimes (l[b] \otimes c)) (a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c))$

using *assms* by *auto*

ultimately show *?thesis*

using 1 2 *assms iso-is-retraction retraction-is-epi*

*epiE* [of  $a[a, \mathcal{I} \otimes b, c] \cdot (a[a, \mathcal{I}, b] \otimes c)$

$(a \otimes l[b \otimes c]) \cdot (a \otimes a[\mathcal{I}, b, c])$   $a \otimes l[b] \otimes c$

by *auto*

qed

The following result is quoted without proof as Theorem 7 of [3] where it is attributed to MacLane [4]. It also appears as [5], Exercise 1, page 161. I did not succeed within a

few hours to construct a proof following MacLane's hint. The proof below is based on [2], Proposition 2.2.4.

**lemma** *lunit-tensor'*:

**assumes** *ide a and ide b*

**shows**  $l[a \otimes b] \cdot a[\mathcal{I}, a, b] = l[a] \otimes b$

**proof** –

**have**  $\mathcal{I} \otimes (l[a \otimes b] \cdot a[\mathcal{I}, a, b]) = \mathcal{I} \otimes (l[a] \otimes b)$

**using** *assms interchange [of  $\mathcal{I}$   $\mathcal{I}$ ] lunit-tensor-gen by simp*

**moreover have** *par*  $(l[a \otimes b] \cdot a[\mathcal{I}, a, b]) (l[a] \otimes b)$

**using** *assms by simp*

**ultimately show** *?thesis*

**using** *assms L.is-faithful [of  $l[a \otimes b] \cdot a[\mathcal{I}, a, b]$   $l[a] \otimes b$ ] unity-def by simp*

**qed**

**lemma** *lunit-tensor*:

**assumes** *ide a and ide b*

**shows**  $l[a \otimes b] = (l[a] \otimes b) \cdot a^{-1}[\mathcal{I}, a, b]$

**using** *assms lunit-tensor' invert-side-of-triangle by simp*

We next show the corresponding result for the right unitor.

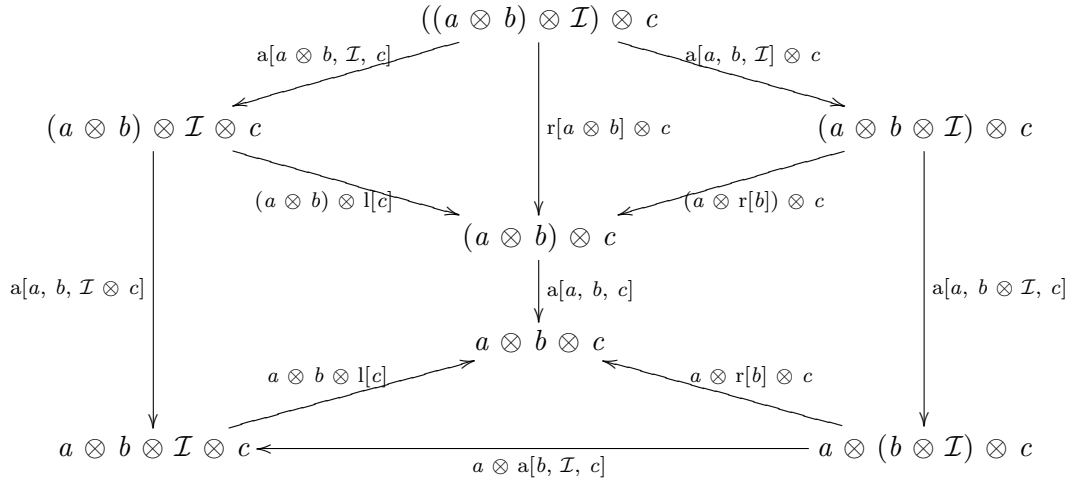
**lemma** *runit-tensor-gen*:

**assumes** *ide a and ide b and ide c*

**shows**  $r[a \otimes b] \otimes c = ((a \otimes r[b]) \otimes c) \cdot (a[a, b, \mathcal{I}] \otimes c)$

**proof** –

We show that the upper right triangle in the following diagram commutes.



**have**  $r[a \otimes b] \otimes c = ((a \otimes b) \otimes l[c]) \cdot a[a \otimes b, \mathcal{I}, c]$

**using** *assms triangle by simp*

**also have**  $\dots = (a^{-1}[a, b, c]) \cdot (a \otimes b \otimes l[c]) \cdot a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c]$

**using** *assms assoc-naturality [of  $a \otimes b \otimes l[c]$ ] comp-arr-dom comp-cod-arr*

*invert-side-of-triangle(1)*

**by force**

**also have** ... =  $a^{-1}[a, b, c] \cdot (a \otimes b \otimes 1[c]) \cdot a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c]$

**using comp-assoc by force**

**also have** ... =  $a^{-1}[a, b, c] \cdot ((a \otimes (r[b] \otimes c)) \cdot (a \otimes a^{-1}[b, \mathcal{I}, c])) \cdot a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c]$

**using assms triangle [of b c] interchange invert-side-of-triangle(2) by force**

**also have** ... =  $((a \otimes r[b]) \otimes c) \cdot a^{-1}[a, b \otimes \mathcal{I}, c] \cdot (a \otimes a^{-1}[b, \mathcal{I}, c]) \cdot a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c]$

**using assms assoc'-naturality [of a r[b] c] comp-assoc by force**

**also have** ... =  $((a \otimes r[b]) \otimes c) \cdot a^{-1}[a, b \otimes \mathcal{I}, c] \cdot (a \otimes a^{-1}[b, \mathcal{I}, c]) \cdot a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c]$

**using comp-assoc by simp**

**also have** ... =  $((a \otimes r[b]) \otimes c) \cdot (a[a, b, \mathcal{I}] \otimes c)$

**using assms pentagon invert-side-of-triangle(1)**

*invert-side-of-triangle(1)*

[of  $a[a, b, \mathcal{I} \otimes c] \cdot a[a \otimes b, \mathcal{I}, c] \ a \otimes a[b, \mathcal{I}, c]$   
 $a[a, b \otimes \mathcal{I}, c] \cdot (a[a, b, \mathcal{I}] \otimes c)$ ]

**by force**

**finally show ?thesis by blast**

**qed**

**lemma runit-tensor:**

**assumes ide a and ide b**

**shows**  $r[a \otimes b] = (a \otimes r[b]) \cdot a[a, b, \mathcal{I}]$

**proof** –

**have**  $((a \otimes r[b]) \cdot a[a, b, \mathcal{I}]) \otimes \mathcal{I} = r[a \otimes b] \otimes \mathcal{I}$

**using assms interchange [of  $\mathcal{I} \ \mathcal{I}$ ] runit-tensor-gen by simp**

**moreover have par**  $((a \otimes r[b]) \cdot a[a, b, \mathcal{I}]) \ r[a \otimes b]$

**using assms by simp**

**ultimately show ?thesis**

**using assms R.is-faithful [of  $(a \otimes r[b]) \cdot (a[a, b, \mathcal{I}]) \ r[a \otimes b]$ ] unity-def**

**by argo**

**qed**

**lemma runit-tensor':**

**assumes ide a and ide b**

**shows**  $r[a \otimes b] \cdot a^{-1}[a, b, \mathcal{I}] = a \otimes r[b]$

**using assms runit-tensor invert-side-of-triangle by force**

Sometimes inverted forms of the triangle and pentagon axioms are useful.

**lemma triangle':**

**assumes ide a and ide b**

**shows**  $(a \otimes 1[b]) = (r[a] \otimes b) \cdot a^{-1}[a, \mathcal{I}, b]$

**proof** –

**have**  $(r[a] \otimes b) \cdot a^{-1}[a, \mathcal{I}, b] = ((a \otimes 1[b]) \cdot a[a, \mathcal{I}, b]) \cdot a^{-1}[a, \mathcal{I}, b]$

**using assms triangle by auto**

**also have** ... =  $(a \otimes 1[b])$

**using assms comp-arr-dom comp-assoc by auto**



finally show *?thesis by auto*  
qed

lemma *pentagon'*:

assumes *ide a and ide b and ide c and ide d*

shows  $((a^{-1}[a, b, c] \otimes d) \cdot a^{-1}[a, b \otimes c, d]) \cdot (a \otimes a^{-1}[b, c, d])$   
 $= a^{-1}[a \otimes b, c, d] \cdot a^{-1}[a, b, c \otimes d]$

proof –

have  $((a^{-1}[a, b, c] \otimes d) \cdot a^{-1}[a, b \otimes c, d]) \cdot (a \otimes a^{-1}[b, c, d])$   
 $= \text{inv } ((a \otimes a[b, c, d]) \cdot (a[a, b \otimes c, d] \cdot (a[a, b, c] \otimes d)))$

using *assms isos-compose inv-comp by simp*

also have  $\dots = \text{inv } (a[a, b, c \otimes d] \cdot a[a \otimes b, c, d])$

using *assms pentagon by auto*

also have  $\dots = a^{-1}[a \otimes b, c, d] \cdot a^{-1}[a, b, c \otimes d]$

using *assms inv-comp by simp*

finally show *?thesis by auto*

qed

The following non-obvious fact is Corollary 2.2.5 from [2]. The statement that  $l[\mathcal{I}] = r[\mathcal{I}]$  is Theorem 6 from [3]. MacLane [5] does not show this, but assumes it as an axiom.

lemma *unitor-coincidence*:

shows  $l[\mathcal{I}] = \iota$  and  $r[\mathcal{I}] = \iota$

proof –

have  $l[\mathcal{I}] \otimes \mathcal{I} = (\mathcal{I} \otimes l[\mathcal{I}]) \cdot a[\mathcal{I}, \mathcal{I}, \mathcal{I}]$

using *lunit-tensor' [of  $\mathcal{I}$   $\mathcal{I}$ ] lunit-commutes-with-L [of  $\mathcal{I}$ ] by simp*

moreover have  $r[\mathcal{I}] \otimes \mathcal{I} = (\mathcal{I} \otimes r[\mathcal{I}]) \cdot a[\mathcal{I}, \mathcal{I}, \mathcal{I}]$

using *triangle [of  $\mathcal{I}$   $\mathcal{I}$ ] by simp*

moreover have  $\iota \otimes \mathcal{I} = (\mathcal{I} \otimes l[\mathcal{I}]) \cdot a[\mathcal{I}, \mathcal{I}, \mathcal{I}]$

using *lunit-char comp-arr-dom  $\iota$ -in-hom comp-assoc by auto*

ultimately have  $l[\mathcal{I}] \otimes \mathcal{I} = \iota \otimes \mathcal{I} \wedge r[\mathcal{I}] \otimes \mathcal{I} = \iota \otimes \mathcal{I}$

by *argo*

moreover have *par*  $l[\mathcal{I}] \iota \wedge$  *par*  $r[\mathcal{I}] \iota$

using  *$\iota$ -in-hom by force*

ultimately have *1*:  $l[\mathcal{I}] = \iota \wedge r[\mathcal{I}] = \iota$

using *R.is-faithful unity-def bymetis*

show  $l[\mathcal{I}] = \iota$  using *1 by auto*

show  $r[\mathcal{I}] = \iota$  using *1 by auto*

qed

lemma  *$\iota$ -triangle*:

shows  $\iota \otimes \mathcal{I} = (\mathcal{I} \otimes \iota) \cdot a[\mathcal{I}, \mathcal{I}, \mathcal{I}]$

and  $(\iota \otimes \mathcal{I}) \cdot a^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}] = \mathcal{I} \otimes \iota$

using *triangle [of  $\mathcal{I}$   $\mathcal{I}$ ] triangle' [of  $\mathcal{I}$   $\mathcal{I}$ ] unitor-coincidence by auto*

The only isomorphism that commutes with  $\iota$  is  $\mathcal{I}$ .

lemma *iso-commuting-with- $\iota$ -equals- $\mathcal{I}$* :

assumes  $\langle f : \mathcal{I} \rightarrow \mathcal{I} \rangle$  and *iso f* and  $f \cdot \iota = \iota \cdot (f \otimes f)$

shows  $f = \mathcal{I}$

proof –

```

have 1: f ⊗ f = f ⊗ I
proof -
  have f ⊗ f = (inv ι · ι) · (f ⊗ f)
    using assms ι-in-hom ι-is-iso inv-is-inverse comp-inv-arr comp-cod-arr [of f ⊗ f]
    by force
  also have ... = (inv ι · f) · ι
    using assms ι-is-iso inv-is-inverse comp-assoc by force
  also have ... = ((f ⊗ I) · inv ι) · ι
    using assms unitor-coincidence runit'-naturality [of f] by auto
  also have ... = (f ⊗ I) · (inv ι · ι)
    using comp-assoc by force
  also have ... = f ⊗ I
    using assms ι-in-hom ι-is-iso inv-is-inverse comp-inv-arr
      comp-arr-dom [of f ⊗ I I ⊗ I]
    by force
  finally show ?thesis by blast
qed
moreover have (f ⊗ f) · (inv f ⊗ I) = I ⊗ f ∧ (f ⊗ I) · (inv f ⊗ I) = I ⊗ I
  using assms interchange comp-arr-inv inv-is-inverse comp-arr-dom by auto
ultimately have I ⊗ f = I ⊗ I by metis
moreover have par f I
  using assms by auto
ultimately have f = I
  using L.is-faithful unity-def by metis
thus ?thesis using 1 by blast
qed

```

end

We now show that the unit  $\iota$  of a monoidal category is unique up to a unique isomorphism (Proposition 2.2.6 of [2]).

```

locale monoidal-category-with-alternate-unit =
  monoidal-category C T α ι +
  C1: monoidal-category C T α ι1
for C :: 'a comp (infixr · 55)
and T :: 'a * 'a ⇒ 'a
and α :: 'a * 'a * 'a ⇒ 'a
and ι :: 'a
and ι1 :: 'a
begin

```

```

  no-notation C1.tensor (infixr ⊗ 53)
  no-notation C1.unity (I)
  no-notation C1.lunit (l[-])
  no-notation C1.runit (r[-])
  no-notation C1.assoc (a[-, -, -])
  no-notation C1.assoc' (a-1[-, -, -])

```

```

  notation C1.tensor (infixr ⊗1 53)

```

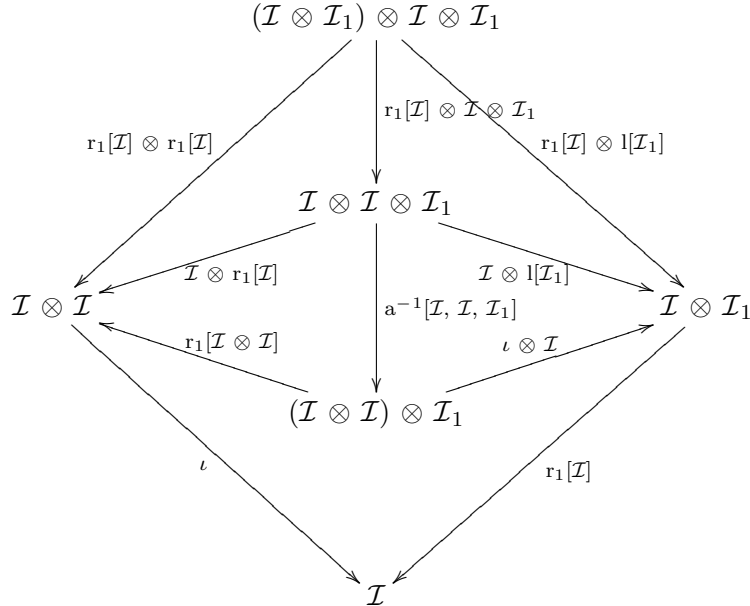
**notation**  $C_1.unity$   $(\mathcal{I}_1)$   
**notation**  $C_1.lunit$   $(l_1[-])$   
**notation**  $C_1.runit$   $(r_1[-])$   
**notation**  $C_1.assoc$   $(a_1[-, -, -])$   
**notation**  $C_1.assoc'$   $(a_1^{-1}[-, -, -])$

**definition**  $i$   
**where**  $i \equiv l[\mathcal{I}_1] \cdot inv\ r_1[\mathcal{I}]$

**lemma**  $iso-i$ :  
**shows**  $\langle i : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle$  **and**  $iso\ i$   
**proof** –  
**show**  $\langle i : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle$   
**using**  $C_1.iso-runit\ inv-in-hom\ i-def\ by\ auto$   
**show**  $iso\ i$   
**using**  $iso-lunit\ C_1.iso-runit\ isos-compose\ i-def\ by\ simp$   
**qed**

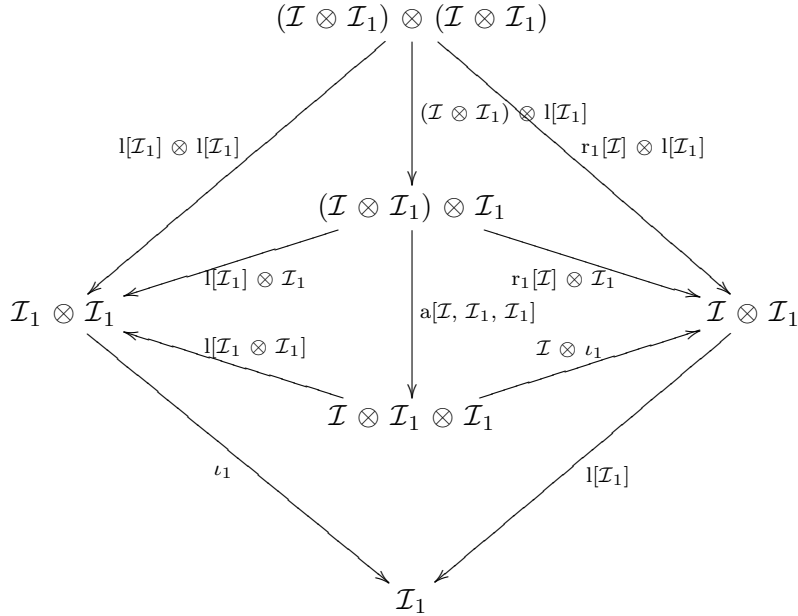
The following is Exercise 2.2.7 of [2].

**lemma**  $i-maps-\iota-to-\iota_1$ :  
**shows**  $i \cdot \iota = \iota_1 \cdot (i \otimes i)$   
**proof** –  
**have**  $1: inv\ r_1[\mathcal{I}] \cdot \iota = (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1]) \cdot (inv\ r_1[\mathcal{I}] \otimes inv\ r_1[\mathcal{I}])$   
**proof** –  
**have**  $\iota \cdot (r_1[\mathcal{I}] \otimes r_1[\mathcal{I}]) = r_1[\mathcal{I}] \cdot (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1])$   
**proof** –



**have**  $\iota \cdot (r_1[\mathcal{I}] \otimes r_1[\mathcal{I}]) = \iota \cdot (I \otimes r_1[\mathcal{I}]) \cdot (r_1[\mathcal{I}] \otimes I \otimes \mathcal{I}_1)$   
**using**  $interchange\ comp-cod-arr\ comp-arr-dom\ by\ simp$

**also have** ... =  $\iota \cdot (r_1[\mathcal{I} \otimes \mathcal{I}] \cdot a^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}_1]) \cdot (r_1[\mathcal{I}] \otimes \mathcal{I} \otimes \mathcal{I}_1)$   
**using**  $C_1.runit-tensor'$  **by** *auto*  
**also have** ... =  $(\iota \cdot r_1[\mathcal{I} \otimes \mathcal{I}]) \cdot a^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes \mathcal{I} \otimes \mathcal{I}_1)$   
**using** *comp-assoc* **by** *auto*  
**also have** ... =  $(r_1[\mathcal{I}] \cdot (\iota \otimes \mathcal{I}_1)) \cdot a^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes \mathcal{I} \otimes \mathcal{I}_1)$   
**using**  $C_1.runit-naturality$  [of  $\iota$ ]  *$\iota$ -in-hom* **by** *fastforce*  
**also have** ... =  $r_1[\mathcal{I}] \cdot ((\iota \otimes \mathcal{I}_1) \cdot a^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}_1]) \cdot (r_1[\mathcal{I}] \otimes \mathcal{I} \otimes \mathcal{I}_1)$   
**using** *comp-assoc* **by** *auto*  
**also have** ... =  $r_1[\mathcal{I}] \cdot (\mathcal{I} \otimes l[\mathcal{I}_1]) \cdot (r_1[\mathcal{I}] \otimes \mathcal{I} \otimes \mathcal{I}_1)$   
**using** *lunit-tensor lunit-commutes-with-L unitor-coincidence* **by** *simp*  
**also have** ... =  $r_1[\mathcal{I}] \cdot (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1])$   
**using** *interchange comp-arr-dom comp-cod-arr* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**moreover have** *seq*  $\iota (r_1[\mathcal{I}] \otimes r_1[\mathcal{I}]) \wedge$  *seq*  $r_1[\mathcal{I}] (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1])$   
**using**  *$\iota$ -in-hom* **by** *fastforce*  
**moreover have** *iso*  $r_1[\mathcal{I}] \wedge$  *iso*  $(r_1[\mathcal{I}] \otimes r_1[\mathcal{I}])$   
**using**  $C_1.iso-runit$  *tensor-preserves-iso* **by** *force*  
**ultimately show** *?thesis*  
**using** *invert-opposite-sides-of-square inv-tensor* **by** *metis*  
**qed**  
**have**  $l[\mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1]) = \iota_1 \cdot (l[\mathcal{I}_1] \otimes l[\mathcal{I}_1])$   
**proof** –



**have**  $l[\mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1]) = l[\mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes \mathcal{I}_1) \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using** *interchange comp-arr-dom comp-cod-arr* **by** *force*  
**also have** ... =  $l[\mathcal{I}_1] \cdot ((\mathcal{I} \otimes \iota_1) \cdot a[\mathcal{I}, \mathcal{I}_1, \mathcal{I}_1]) \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using**  $C_1.runit-tensor$   $C_1.unitor-coincidence$   $C_1.runit-commutes-with-R$  **by** *simp*

**also have** ... =  $(l[\mathcal{I}_1] \cdot (\mathcal{I} \otimes \iota_1)) \cdot a[\mathcal{I}, \mathcal{I}_1, \mathcal{I}_1] \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using** *comp-assoc* **by** *fastforce*  
**also have** ... =  $(\iota_1 \cdot l[\mathcal{I}_1 \otimes \mathcal{I}_1]) \cdot a[\mathcal{I}, \mathcal{I}_1, \mathcal{I}_1] \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using** *lunit-naturality* [of  $\iota_1$ ] *C<sub>1</sub>. $\iota$ -in-hom* *lunit-commutes-with-L* **by** *fastforce*  
**also have** ... =  $\iota_1 \cdot (l[\mathcal{I}_1 \otimes \mathcal{I}_1] \cdot a[\mathcal{I}, \mathcal{I}_1, \mathcal{I}_1]) \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using** *comp-assoc* **by** *force*  
**also have** ... =  $\iota_1 \cdot (l[\mathcal{I}_1] \otimes \mathcal{I}_1) \cdot ((\mathcal{I} \otimes \mathcal{I}_1) \otimes l[\mathcal{I}_1])$   
**using** *lunit-tensor'* **by** *auto*  
**also have** ... =  $\iota_1 \cdot (l[\mathcal{I}_1] \otimes l[\mathcal{I}_1])$   
**using** *interchange comp-arr-dom comp-cod-arr* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**show** *?thesis*  
**proof** –

$$\begin{array}{ccccc}
 \mathcal{I}_1 \otimes \mathcal{I}_1 & \xleftarrow{l[\mathcal{I}_1] \otimes l[\mathcal{I}_1]} & (\mathcal{I} \otimes \mathcal{I}_1) \otimes (\mathcal{I} \otimes \mathcal{I}_1) & \xrightarrow{r_1[\mathcal{I}] \otimes r_1[\mathcal{I}]} & \mathcal{I}_1 \otimes \mathcal{I}_1 \\
 \downarrow \iota_1 & & \downarrow r_1[\mathcal{I}] \otimes l[\mathcal{I}_1] & & \downarrow \iota \\
 \mathcal{I}_1 & \xleftarrow{l[\mathcal{I}_1]} & \mathcal{I} \otimes \mathcal{I}_1 & \xrightarrow{r_1[\mathcal{I}]} & \mathcal{I}
 \end{array}$$

**have**  $i \cdot \iota = l[\mathcal{I}_1] \cdot \text{inv } r_1[\mathcal{I}] \cdot \iota$   
**using** *i-def comp-assoc* **by** *auto*  
**also have** ... =  $(l[\mathcal{I}_1] \cdot (r_1[\mathcal{I}] \otimes l[\mathcal{I}_1])) \cdot (\text{inv } r_1[\mathcal{I}] \otimes \text{inv } r_1[\mathcal{I}])$   
**using** *1 comp-assoc* **by** *simp*  
**also have** ... =  $\iota_1 \cdot (l[\mathcal{I}_1] \otimes l[\mathcal{I}_1]) \cdot (\text{inv } r_1[\mathcal{I}] \otimes \text{inv } r_1[\mathcal{I}])$   
**using** *2 comp-assoc* **by** *fastforce*  
**also have** ... =  $\iota_1 \cdot (i \otimes i)$   
**using** *interchange i-def* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *inv-i-iso- $\iota$* :

**assumes**  $\langle f : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle$  **and** *iso*  $f$  **and**  $f \cdot \iota = \iota_1 \cdot (f \otimes f)$

**shows**  $\langle \text{inv } i \cdot f : \mathcal{I} \rightarrow \mathcal{I} \rangle$  **and** *iso*  $(\text{inv } i \cdot f)$

**and**  $(\text{inv } i \cdot f) \cdot \iota = \iota \cdot (\text{inv } i \cdot f \otimes \text{inv } i \cdot f)$

**proof** –

**show** *1*:  $\langle \text{inv } i \cdot f : \mathcal{I} \rightarrow \mathcal{I} \rangle$

**using** *assms iso-i inv-in-hom* **by** *blast*

**show** *iso*  $(\text{inv } i \cdot f)$

**using** *assms 1 iso-i inv-in-hom*

**by** *(intro isos-compose, auto)*

**show**  $(\text{inv } i \cdot f) \cdot \iota = \iota \cdot (\text{inv } i \cdot f \otimes \text{inv } i \cdot f)$

**proof** –

**have**  $(\text{inv } i \cdot f) \cdot \iota = (\text{inv } i \cdot \iota_1) \cdot (f \otimes f)$

**using** *assms iso-i comp-assoc* **by** *auto*

```

also have ... = ( $\iota \cdot (\text{inv } i \otimes \text{inv } i)$ )  $\cdot$  ( $f \otimes f$ )
  using assms iso-i invert-opposite-sides-of-square
           inv-tensor  $\iota$ -in-hom  $C_1$ . $\iota$ -in-hom tensor-in-hom tensor-preserves-iso
           inv-in-hom i-maps- $\iota$ -to- $\iota_1$  unity-def seqI'
  by metis
also have ... =  $\iota \cdot (\text{inv } i \cdot f \otimes \text{inv } i \cdot f)$ 
  using assms 1 iso-i interchange comp-assoc by fastforce
finally show ?thesis by blast
qed
qed

```

```

lemma unit-unique-upto-unique-iso:
shows  $\exists! f. \langle\langle f : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle\rangle \wedge \text{iso } f \wedge f \cdot \iota = \iota_1 \cdot (f \otimes f)$ 
proof
  show  $\langle\langle i : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle\rangle \wedge \text{iso } i \wedge i \cdot \iota = \iota_1 \cdot (i \otimes i)$ 
    using iso-i i-maps- $\iota$ -to- $\iota_1$  by auto
  show  $\bigwedge f. \langle\langle f : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle\rangle \wedge \text{iso } f \wedge f \cdot \iota = \iota_1 \cdot (f \otimes f) \implies f = i$ 
proof –
  fix  $f$ 
  assume  $f: \langle\langle f : \mathcal{I} \rightarrow \mathcal{I}_1 \rangle\rangle \wedge \text{iso } f \wedge f \cdot \iota = \iota_1 \cdot (f \otimes f)$ 
  have  $\text{inv } i \cdot f = \mathcal{I}$ 
    using f inv-i-iso- $\iota$  iso-commuting-with- $\iota$ -equals- $\mathcal{I}$  by blast
  hence ide ( $C (\text{inv } i) f$ )
    using iso-i by simp
  thus  $f = i$ 
    using section-retraction-of-iso(2) [of inv i f] inverse-arrow-unique inv-is-inverse
      iso-i
    by blast
qed
qed

```

end

## 2.2 Elementary Monoidal Category

Although the economy of data assumed by *monoidal-category* is useful for general results, to establish interpretations it is more convenient to work with a traditional definition of monoidal category. The following locale provides such a definition. It permits a monoidal category to be specified by giving the tensor product and the components of the associator and unitors, which are required only to satisfy elementary conditions that imply functoriality and naturality, without having to worry about extensionality or formal interpretations for the various functors and natural transformations.

```

locale elementary-monoidal-category =
  category C
for  $C :: 'a \text{ comp}$  (infixr  $\cdot$  55)
and tensor  $:: 'a \Rightarrow 'a \Rightarrow 'a$  (infixr  $\otimes$  53)
and unity  $:: 'a$  ( $\mathcal{I}$ )

```

```

and lunit :: 'a ⇒ 'a (l[-])
and runit :: 'a ⇒ 'a (r[-])
and assoc :: 'a ⇒ 'a ⇒ 'a ⇒ 'a (a[-, -, -]) +
assumes ide-unity [simp]: ide  $\mathcal{I}$ 
and iso-lunit: ide  $a \implies \text{iso } l[a]$ 
and iso-runit: ide  $a \implies \text{iso } r[a]$ 
and iso-assoc:  $\llbracket \text{ide } a; \text{ide } b; \text{ide } c \rrbracket \implies \text{iso } a[a, b, c]$ 
and tensor-in-hom [simp]:  $\llbracket \langle f : a \rightarrow b \rangle; \langle g : c \rightarrow d \rangle \rrbracket \implies \langle f \otimes g : a \otimes c \rightarrow b \otimes d \rangle$ 
and tensor-preserves-ide:  $\llbracket \text{ide } a; \text{ide } b \rrbracket \implies \text{ide } (a \otimes b)$ 
and interchange:  $\llbracket \text{seq } g \ f; \text{seq } g' \ f' \rrbracket \implies (g \otimes g') \cdot (f \otimes f') = g \cdot f \otimes g' \cdot f'$ 
and lunit-in-hom [simp]: ide  $a \implies \langle l[a] : \mathcal{I} \otimes a \rightarrow a \rangle$ 
and lunit-naturality: arr  $f \implies l[\text{cod } f] \cdot (\mathcal{I} \otimes f) = f \cdot l[\text{dom } f]$ 
and runit-in-hom [simp]: ide  $a \implies \langle r[a] : a \otimes \mathcal{I} \rightarrow a \rangle$ 
and runit-naturality: arr  $f \implies r[\text{cod } f] \cdot (f \otimes \mathcal{I}) = f \cdot r[\text{dom } f]$ 
and assoc-in-hom [simp]:
   $\llbracket \text{ide } a; \text{ide } b; \text{ide } c \rrbracket \implies \langle a[a, b, c] : (a \otimes b) \otimes c \rightarrow a \otimes b \otimes c \rangle$ 
and assoc-naturality:
   $\llbracket \text{arr } f0; \text{arr } f1; \text{arr } f2 \rrbracket \implies a[\text{cod } f0, \text{cod } f1, \text{cod } f2] \cdot ((f0 \otimes f1) \otimes f2)$ 
   $\quad = (f0 \otimes (f1 \otimes f2)) \cdot a[\text{dom } f0, \text{dom } f1, \text{dom } f2]$ 
and triangle:  $\llbracket \text{ide } a; \text{ide } b \rrbracket \implies (a \otimes l[b]) \cdot a[a, \mathcal{I}, b] = r[a] \otimes b$ 
and pentagon:  $\llbracket \text{ide } a; \text{ide } b; \text{ide } c; \text{ide } d \rrbracket \implies$ 
   $(a \otimes a[b, c, d]) \cdot a[a, b \otimes c, d] \cdot (a[a, b, c] \otimes d)$ 
   $\quad = a[a, b, c \otimes d] \cdot a[a \otimes b, c, d]$ 

```

An interpretation for the *monoidal-category* locale readily induces an interpretation for the *elementary-monoidal-category* locale.

```

context monoidal-category
begin

```

```

  lemma induces-elementary-monoidal-category:
  shows elementary-monoidal-category  $C$  tensor  $\mathcal{I}$  lunit runit assoc
  using  $\iota$ -in-hom iso-assoc tensor-preserves-ide assoc-in-hom tensor-in-hom
  assoc-naturality lunit-naturality runit-naturality lunit-in-hom runit-in-hom
  iso-lunit iso-runit interchange pentagon triangle
  by unfold-locales auto

```

```

end

```

```

context elementary-monoidal-category
begin

```

```

  interpretation CC: product-category  $C$   $C$  ..
  interpretation CCC: product-category  $C$   $CC$ .comp ..

```

```

  definition T :: 'a * 'a ⇒ 'a
  where  $T \ f \equiv \text{if } CC.\text{arr } f \text{ then } (fst \ f \otimes snd \ f) \text{ else null}$ 

```

```

  lemma T-simp [simp]:
  assumes arr f and arr g

```

**shows**  $T (f, g) = f \otimes g$   
**using** *assms T-def by simp*

**lemma** *arr-tensor [simp]*:  
**assumes** *arr f and arr g*  
**shows** *arr (f  $\otimes$  g)*  
**using** *assms tensor-in-hom by blast*

**lemma** *dom-tensor [simp]*:  
**assumes** *arr f and arr g*  
**shows** *dom (f  $\otimes$  g) = dom f  $\otimes$  dom g*  
**using** *assms tensor-in-hom by blast*

**lemma** *cod-tensor [simp]*:  
**assumes** *arr f and arr g*  
**shows** *cod (f  $\otimes$  g) = cod f  $\otimes$  cod g*  
**using** *assms tensor-in-hom by blast*

**interpretation** *T: binary-endofunctor C T*  
**using** *tensor-in-hom interchange T-def*  
**apply** *unfold-locales*  
**apply** *auto[4]*  
**by** *(elim CC.seqE, auto)*

**lemma** *binary-endofunctor-T*:  
**shows** *binary-endofunctor C T ..*

**interpretation** *ToTC: functor CCC.comp C T.ToTC*  
**using** *T.functor-ToTC by auto*

**interpretation** *ToCT: functor CCC.comp C T.ToCT*  
**using** *T.functor-ToCT by auto*

**definition**  $\alpha$

**where**  $\alpha f \equiv$  *if CCC.arr f then (fst f  $\otimes$  (fst (snd f)  $\otimes$  snd (snd f))) .*  
*a[dom (fst f), dom (fst (snd f)), dom (snd (snd f))]*  
*else null*

**lemma**  *$\alpha$ -ide-simp [simp]*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\alpha (a, b, c) = a[a, b, c]$   
**unfolding**  $\alpha$ -*def using assms assoc-in-hom comp-cod-arr*  
**by** *(metis CC.arrI CCC.arrI fst-conv ide-char in-homE snd-conv)*

**lemma** *arr-assoc [simp]*:  
**assumes** *ide a and ide b and ide c*  
**shows** *arr a[a, b, c]*  
**using** *assms assoc-in-hom by blast*



**lemma** *dom-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows**  $\text{dom } a[a, b, c] = (a \otimes b) \otimes c$   
**using** *assms assoc-in-hom* **by** *blast*

**lemma** *cod-assoc* [*simp*]:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows**  $\text{cod } a[a, b, c] = a \otimes b \otimes c$   
**using** *assms assoc-in-hom* **by** *blast*

**interpretation**  $\alpha$ : *natural-isomorphism CCC.comp C T.ToTC T.ToCT*  $\alpha$   
**proof** –

**interpret**  $\alpha$ : *transformation-by-components CCC.comp C T.ToTC T.ToCT*  $\alpha$   
**apply** *unfold-locales*  
**unfolding**  $\alpha$ -*def* *T.ToTC-def* *T.ToCT-def* *T-def*  
**using** *comp-arr-dom* *comp-cod-arr* *assoc-naturality*  
**by** *simp-all*

**interpret**  $\alpha$ : *natural-isomorphism CCC.comp C T.ToTC T.ToCT*  $\alpha$ .*map*  
**using** *iso-assoc*  $\alpha$ .*map-simp-ide* *assoc-in-hom* *tensor-preserves-ide*  $\alpha$ -*def*  
**by** (*unfold-locales*, *auto*)

**have**  $\alpha = \alpha$ .*map*

**using** *assoc-naturality*  $\alpha$ -*def* *comp-cod-arr* *T.ToTC-def* *T-def*  $\alpha$ .*map-def* **by** *auto*

**thus** *natural-isomorphism CCC.comp C T.ToTC T.ToCT*  $\alpha$

**using**  $\alpha$ .*natural-isomorphism-axioms* **by** *simp*

**qed**

**interpretation**  $\alpha'$ : *inverse-transformation CCC.comp C T.ToTC T.ToCT*  $\alpha$  ..

**interpretation** *L*: *functor C C*  $\langle \lambda f. T (\mathcal{I}, f) \rangle$   
**using** *T.fixing-ide-gives-functor-1* **by** *auto*

**interpretation** *R*: *functor C C*  $\langle \lambda f. T (f, \mathcal{I}) \rangle$   
**using** *T.fixing-ide-gives-functor-2* **by** *auto*

**interpretation**  $\mathfrak{l}$ : *natural-isomorphism C C*  $\langle \lambda f. T (\mathcal{I}, f) \rangle$  *map*  
 $\langle \lambda f. \text{if arr } f \text{ then } f \cdot \mathfrak{l}[\text{dom } f] \text{ else null} \rangle$

**proof** –

**interpret**  $\mathfrak{l}$ : *transformation-by-components C C*  $\langle \lambda f. T (\mathcal{I}, f) \rangle$  *map*  $\langle \lambda a. \mathfrak{l}[a] \rangle$   
**using** *lunit-naturality* **by** (*unfold-locales*, *auto*)

**interpret**  $\mathfrak{l}$ : *natural-isomorphism C C*  $\langle \lambda f. T (\mathcal{I}, f) \rangle$  *map*  $\mathfrak{l}$ .*map*  
**using** *iso-lunit* **by** (*unfold-locales*, *simp*)

**have**  $\mathfrak{l}$ .*map* =  $(\lambda f. \text{if arr } f \text{ then } f \cdot \mathfrak{l}[\text{dom } f] \text{ else null})$

**using**  $\mathfrak{l}$ .*map-def* *lunit-naturality* **by** *fastforce*

**thus** *natural-isomorphism C C*  $(\lambda f. T (\mathcal{I}, f))$  *map*  $(\lambda f. \text{if arr } f \text{ then } f \cdot \mathfrak{l}[\text{dom } f] \text{ else null})$

**using**  $\mathfrak{l}$ .*natural-isomorphism-axioms* **by** *force*

**qed**

**interpretation**  $\mathfrak{r}$ : *natural-isomorphism C C*  $\langle \lambda f. T (f, \mathcal{I}) \rangle$  *map*  
 $\langle \lambda f. \text{if arr } f \text{ then } f \cdot \mathfrak{r}[\text{dom } f] \text{ else null} \rangle$

**proof** –  
**interpret**  $\varrho$ : *transformation-by-components*  $C C \langle \lambda f. T (f, \mathcal{I}) \rangle \text{map} \langle \lambda a. r[a] \rangle$   
**using** *runit-naturality* **by** (*unfold-locales, auto*)  
**interpret**  $\varrho$ : *natural-isomorphism*  $C C \langle \lambda f. T (f, \mathcal{I}) \rangle \text{map} \varrho.\text{map}$   
**using** *iso-runit*  $\varrho.\text{map-simp-ide}$  **by** (*unfold-locales, simp*)  
**have** ( $\lambda f. \text{if arr } f \text{ then } f \cdot r[\text{dom } f] \text{ else null}$ ) =  $\varrho.\text{map}$   
**using**  $\varrho.\text{map-def}$  *runit-naturality* *T-simp* **by** *fastforce*  
**thus** *natural-isomorphism*  $C C \langle \lambda f. T (f, \mathcal{I}) \rangle \text{map} \langle \lambda f. \text{if arr } f \text{ then } f \cdot r[\text{dom } f] \text{ else null} \rangle$   
**using**  $\varrho.\text{natural-isomorphism-axioms}$  **by** *force*  
**qed**

The endofunctors  $\lambda f. T (\mathcal{I}, f)$  and  $\lambda f. T (f, \mathcal{I})$  are equivalence functors, due to the existence of the unitors.

**interpretation**  $L$ : *equivalence-functor*  $C C \langle \lambda f. T (\mathcal{I}, f) \rangle$   
**proof** –  
**interpret** *endofunctor*  $C \langle \lambda f. T (\mathcal{I}, f) \rangle \dots$   
**show** *equivalence-functor*  $C C \langle \lambda f. T (\mathcal{I}, f) \rangle$   
**using** *isomorphic-to-identity-is-equivalence*  $l.\text{natural-isomorphism-axioms}$  **by** *simp*  
**qed**

**interpretation**  $R$ : *equivalence-functor*  $C C \langle \lambda f. T (f, \mathcal{I}) \rangle$   
**proof** –  
**interpret** *endofunctor*  $C \langle \lambda f. T (f, \mathcal{I}) \rangle \dots$   
**show** *equivalence-functor*  $C C \langle \lambda f. T (f, \mathcal{I}) \rangle$   
**using** *isomorphic-to-identity-is-equivalence*  $\varrho.\text{natural-isomorphism-axioms}$  **by** *simp*  
**qed**

To complete an interpretation of the *monoidal-category* locale, we define  $\iota \equiv l[\mathcal{I}]$ . We could also have chosen  $\iota \equiv \varrho [\mathcal{I}]$  as the two are equal, though to prove that requires some work yet.

**definition**  $\iota$   
**where**  $\iota \equiv l[\mathcal{I}]$

**lemma**  *$\iota$ -in-hom*:  
**shows**  $\langle \iota : \mathcal{I} \otimes \mathcal{I} \rightarrow \mathcal{I} \rangle$   
**using** *lunit-in-hom*  $\iota\text{-def}$  **by** *simp*

**lemma** *induces-monoidal-category*:  
**shows** *monoidal-category*  $C T \alpha \iota$

**proof** –  
**have**  $1: \langle \iota : \mathcal{I} \otimes \mathcal{I} \rightarrow \mathcal{I} \rangle$   
**using** *lunit-in-hom*  $\iota\text{-def}$  **by** *simp*  
**interpret**  $L$ : *equivalence-functor*  $C C \langle \lambda f. T (\text{cod } \iota, f) \rangle$   
**proof** –  
**have** ( $\lambda f. T (\mathcal{I}, f)$ ) = ( $\lambda f. T (\text{cod } \iota, f)$ ) **using**  $1$  **by** *fastforce*  
**thus** *equivalence-functor*  $C C \langle \lambda f. T (\text{cod } \iota, f) \rangle$   
**using**  $L.\text{equivalence-functor-axioms}$  *T-def* **by** *simp*  
**qed**  
**interpret**  $R$ : *equivalence-functor*  $C C \langle \lambda f. T (f, \text{cod } \iota) \rangle$

```

proof –
  have  $(\lambda f. T (f, \mathcal{I})) = (\lambda f. T (f, \text{cod } \iota))$  using 1 by fastforce
  thus equivalence-functor  $C C (\lambda f. T (f, \text{cod } \iota))$ 
  using R.equivalence-functor-axioms  $T\text{-def}$  by simp
qed
show ?thesis
proof
  show  $\llbracket \iota : T (\text{cod } \iota, \text{cod } \iota) \rightarrow \text{cod } \iota \rrbracket$  using 1 by fastforce
  show iso  $\iota$  using iso-lunit  $\iota\text{-def}$  by simp
  show  $\bigwedge a b c d. \llbracket \text{ide } a; \text{ide } b; \text{ide } c; \text{ide } d \rrbracket \implies$ 
     $T (a, \alpha (b, c, d)) \cdot \alpha (a, T (b, c), d) \cdot T (\alpha (a, b, c), d)$ 
     $= \alpha (a, b, T (c, d)) \cdot \alpha (T (a, b), c, d)$ 
  using pentagon tensor-preserves-ide by simp
qed
qed

```

**interpretation**  $MC$ : *monoidal-category*  $C T \alpha \iota$   
**using** *induces-monoidal-category* **by** *auto*

We now show that the notions defined in the interpretation  $MC$  agree with their counterparts in the present locale. These facts are needed if we define an interpretation for the *elementary-monoidal-category* locale, use it to obtain the induced interpretation for *monoidal-category*, and then want to transfer facts obtained in the induced interpretation back to the original one.

**lemma** *I-agreement*:  
**shows**  $MC.\text{unity} = \mathcal{I}$   
**by** (*metis*  $MC.\text{unity-def}$   $\iota\text{-def}$  *ide-unity in-homE lunit-in-hom*)

**lemma** *L-agreement*:  
**shows**  $MC.L = (\lambda f. T (\mathcal{I}, f))$   
**using**  $\iota\text{-in-hom}$   $MC.\text{unity-def}$  **by** *auto*

**lemma** *R-agreement*:  
**shows**  $MC.R = (\lambda f. T (f, \mathcal{I}))$   
**using**  $\iota\text{-in-hom}$   $MC.\text{unity-def}$  **by** *auto*

We wish to show that the components of the unitors  $MC.l$  and  $MC.r$  defined in the induced interpretation  $MC$  agree with those given by the parameters *lunit* and *runit* to the present locale. To avoid a lengthy development that repeats work already done in the *monoidal-category* locale, we establish the agreement in a special case and then use the properties already shown for  $MC$  to prove the general case. In particular, we first show that  $l[Z] = MC.lunit MC.unity$  and  $r[Z] = MC.runit MC.unity$ , from which it follows by facts already proved for  $MC$  that both are equal to  $\iota$ . We then show that for an arbitrary identity  $a$  the arrows  $l[a]$  and  $r[a]$  satisfy the equations that uniquely characterize the components  $MC.lunit a$  and  $MC.runit a$ , respectively, and are therefore equal to those components.

**lemma** *unitor-coincidence*:  
**shows**  $l[Z] = \iota$  **and**  $r[Z] = \iota$

**proof** –  
**have**  $r[\mathcal{I}] = MC.runit\ MC.unity$   
**proof** (*intro*  $MC.runit-eqI$ )  
**show**  $1: \langle r[\mathcal{I}] : MC.tensor\ MC.unity\ MC.unity \rightarrow MC.unity \rangle$   
**using** *runit-in-hom* *ι-in-hom*  $MC.unity-def$  **by** *auto*  
**show**  $MC.tensor\ r[\mathcal{I}]\ MC.unity$   
 $= MC.tensor\ MC.unity\ \iota \cdot MC.assoc\ MC.unity\ MC.unity\ MC.unity$   
**using** *T-def* *ι-in-hom* *ι-def* *triangle*  $MC.unity-def$   
**by** (*metis*  $1\ MC.ide-unity\ T-simp\ \alpha-ide-simp\ ideD(1)\ in-homE$ )  
**qed**  
**moreover** **have**  $l[\mathcal{I}] = MC.lunit\ MC.unity$   
**proof** (*intro*  $MC.lunit-eqI$ )  
**show**  $\langle l[\mathcal{I}] : MC.tensor\ MC.unity\ MC.unity \rightarrow MC.unity \rangle$   
**using** *lunit-in-hom* [*of*  $\mathcal{I}$ ] *ι-in-hom*  $MC.unity-def$  **by** *auto*  
**show**  $MC.tensor\ MC.unity\ l[\mathcal{I}]$   
 $= MC.tensor\ \iota\ MC.unity \cdot MC.assoc'\ MC.unity\ MC.unity\ MC.unity$   
**using** *T-def* *lunit-in-hom* *ι-in-hom* *ι-def*  $MC.\iota-triangle$  **by** *argo*  
**qed**  
**moreover** **have**  $MC.lunit\ MC.unity = \iota \wedge MC.runit\ MC.unity = \iota$   
**using**  $MC.unitor-coincidence$  **by** *simp*  
**ultimately** **have**  $1: l[\mathcal{I}] = \iota \wedge r[\mathcal{I}] = \iota$  **by** *simp*  
**show**  $l[\mathcal{I}] = \iota$  **using**  $1$  **by** *simp*  
**show**  $r[\mathcal{I}] = \iota$  **using**  $1$  **by** *simp*  
**qed**

**lemma** *lunit-char*:  
**assumes** *ide a*  
**shows**  $\mathcal{I} \otimes l[a] = (\iota \otimes a) \cdot MC.assoc'\ MC.unity\ MC.unity\ a$   
**proof** –  
**have**  $(r[\mathcal{I}] \otimes a) \cdot MC.assoc'\ MC.unity\ MC.unity\ a$   
 $= ((\mathcal{I} \otimes l[a]) \cdot a[\mathcal{I}, \mathcal{I}, a]) \cdot MC.assoc'\ MC.unity\ MC.unity\ a$   
**using** *assms triangle* **by** *simp*  
**also** **have**  $\dots = \mathcal{I} \otimes l[a]$   
**using** *assms*  $MC.assoc-inv\ comp-arr-inv\ \mathcal{I}-agreement\ iso-assoc\ comp-arr-dom\ comp-assoc$   
**by** (*metis*  $MC.\iota-is-iso\ \alpha-ide-simp\ arr-tensor\ ideD(1)\ ide-unity\ invert-side-of-triangle(2)$   
*iso-is-arr triangle unitor-coincidence(2)*)  
**finally** **have**  $\mathcal{I} \otimes l[a] = (r[\mathcal{I}] \otimes a) \cdot MC.assoc'\ MC.unity\ MC.unity\ a$  **by** *argo*  
**thus**  $\mathcal{I} \otimes l[a] = (\iota \otimes a) \cdot MC.assoc'\ MC.unity\ MC.unity\ a$   
**using** *unitor-coincidence* **by** *auto*  
**qed**

**lemma** *runit-char*:  
**assumes** *ide a*  
**shows**  $r[a] \otimes \mathcal{I} = (a \otimes \iota) \cdot a[a, \mathcal{I}, \mathcal{I}]$   
**using** *assms triangle* *ι-def* **by** *simp*

**lemma** *l-agreement*:  
**shows**  $MC.l = (\lambda f. \text{if } arr\ f \text{ then } f \cdot l[dom\ f] \text{ else } null)$   
**proof**

```

fix f
have  $\neg \text{arr } f \implies \text{MC.l } f = \text{null}$  by simp
moreover have  $\text{arr } f \implies \text{MC.l } f = f \cdot \text{l}[dom\ f]$ 
proof –
  have  $\bigwedge a. \text{ide } a \implies \text{l}[a] = \text{MC.lunit } a$ 
    using I-agreement T-def lunit-char ι-in-hom iso-lunit
    apply (intro MC.lunit-eqI)
    apply auto
    by blast
  thus ?thesis
    by (metis ide-dom ext seqE)
qed
ultimately show  $\text{MC.l } f = (\text{if } \text{arr } f \text{ then } f \cdot \text{l}[dom\ f] \text{ else null})$  by simp
qed

```

```

lemma ρ-agreement:
shows  $\text{MC.ρ} = (\lambda f. \text{if } \text{arr } f \text{ then } f \cdot \text{r}[dom\ f] \text{ else null})$ 
proof
  fix f
  have  $\neg \text{arr } f \implies \text{MC.ρ } f = \text{null}$  by simp
  moreover have  $\text{arr } f \implies \text{MC.ρ } f = f \cdot \text{r}[dom\ f]$ 
  proof –
    have  $\bigwedge a. \text{ide } a \implies \text{r}[a] = \text{MC.runit } a$ 
      using I-agreement T-def runit-char ι-in-hom iso-runit
      apply (intro MC.runit-eqI)
      apply auto
      by blast
    thus ?thesis
      by (metis ide-dom local.ext seqE)
  qed
  ultimately show  $\text{MC.ρ } f = (\text{if } \text{arr } f \text{ then } f \cdot \text{r}[dom\ f] \text{ else null})$  by simp
qed

```

```

lemma lunit-agreement:
assumes ide a
shows  $\text{MC.lunit } a = \text{l}[a]$ 
  using assms comp-cod-arr l-agreement
  by (metis (no-types, lifting) MC.l-ide-simp ide-char in-homE lunit-in-hom)

```

```

lemma runit-agreement:
assumes ide a
shows  $\text{MC.runit } a = \text{r}[a]$ 
  using assms comp-cod-arr ρ-agreement
  by (metis (no-types, lifting) MC.ρ-ide-simp ide-char in-homE runit-in-hom)

```

**end**

## 2.3 Strict Monoidal Category

A monoidal category is *strict* if the components of the associator and unitors are all identities.

```

locale strict-monoidal-category =
  monoidal-category +
assumes strict-assoc: [ ide a0; ide a1; ide a2 ]  $\implies$  ide a[a0, a1, a2]
and strict-lunit: ide a  $\implies$  l[a] = a
and strict-runit: ide a  $\implies$  r[a] = a
begin

  lemma strict-unit:
  shows  $\iota = \mathcal{I}$ 
  using strict-lunit unitor-coincidence(1) by auto

  lemma tensor-assoc [simp]:
  assumes arr f0 and arr f1 and arr f2
  shows  $(f0 \otimes f1) \otimes f2 = f0 \otimes f1 \otimes f2$ 
  proof -
  have  $(f0 \otimes f1) \otimes f2 = a[\text{cod } f0, \text{cod } f1, \text{cod } f2] \cdot ((f0 \otimes f1) \otimes f2)$ 
  using assms assoc-in-hom [of cod f0 cod f1 cod f2] strict-assoc comp-cod-arr
  by auto
  also have  $\dots = (f0 \otimes f1 \otimes f2) \cdot a[\text{dom } f0, \text{dom } f1, \text{dom } f2]$ 
  using assms assoc-naturality by simp
  also have  $\dots = f0 \otimes f1 \otimes f2$ 
  using assms assoc-in-hom [of dom f0 dom f1 dom f2] strict-assoc comp-arr-dom
  by auto
  finally show ?thesis by simp
qed

end

```

## 2.4 Opposite Monoidal Category

The *opposite* of a monoidal category has the same underlying category, but the arguments to the tensor product are reversed and the associator is inverted and its arguments reversed.

```

locale opposite-monoidal-category =
  C: monoidal-category C TC  $\alpha_C$   $\iota$ 
for C :: 'a comp (infixr · 55)
and TC :: 'a * 'a  $\Rightarrow$  'a
and  $\alpha_C$  :: 'a * 'a * 'a  $\Rightarrow$  'a
and  $\iota$  :: 'a
begin

  abbreviation T
  where  $T f \equiv T_C (\text{snd } f, \text{fst } f)$ 

```

```

abbreviation  $\alpha$ 
where  $\alpha f \equiv C.\alpha' (snd (snd f), fst (snd f), fst f)$ 

end

sublocale opposite-monoidal-category  $\subseteq$  monoidal-category  $C T \alpha \iota$ 
proof –
  interpret  $T$ : binary-endofunctor  $C T$ 
    using  $C.T.is-extensional C.CC.seq-char C.interchange$  by (unfold-locales, auto)
  interpret  $ToTC$ : functor  $C.CCC.comp C T.ToTC$ 
    using  $T.functor-ToTC$  by auto
  interpret  $ToCT$ : functor  $C.CCC.comp C T.ToCT$ 
    using  $T.functor-ToCT$  by auto
  interpret  $\alpha$ : natural-transformation  $C.CCC.comp C T.ToTC T.ToCT \alpha$ 
    using  $C.\alpha'.is-extensional C.CCC.dom-char C.CCC.cod-char T.ToTC-def T.ToCT-def$ 
       $C.\alpha'.simp C.\alpha'.naturality$ 
    by (unfold-locales) auto
  interpret  $\alpha$ : natural-isomorphism  $C.CCC.comp C T.ToTC T.ToCT \alpha$ 
    using  $C.\alpha'.components-are-iso$  by (unfold-locales, simp)
  interpret  $L$ : equivalence-functor  $C C \langle \lambda f. T (C.cod \iota, f) \rangle$ 
    using  $C.R.equivalence-functor-axioms$  by simp
  interpret  $R$ : equivalence-functor  $C C \langle \lambda f. T (f, C.cod \iota) \rangle$ 
    using  $C.L.equivalence-functor-axioms$  by simp
  show monoidal-category  $C T \alpha \iota$ 
    using  $C.\iota-in-hom C.\iota-is-iso C.unity-def C.pentagon' C.comp-assoc$ 
    by (unfold-locales) auto
qed

context opposite-monoidal-category
begin

  lemma lunit-simp:
  assumes  $C.ide a$ 
  shows  $lunit a = C.runit a$ 
    using  $assms lunit-char C.iso-assoc$  by (intro C.runit-eqI, auto)

  lemma runit-simp:
  assumes  $C.ide a$ 
  shows  $runit a = C.lunit a$ 
    using  $assms runit-char C.iso-assoc$  by (intro C.lunit-eqI, auto)

end

```

## 2.5 Monoidal Language

In this section we assume that a category  $C$  is given, and we define a formal syntax of terms constructed from arrows of  $C$  using function symbols that correspond to unity,

composition, tensor, the associator and its formal inverse, and the left and right unitors and their formal inverses. We will use this syntax to state and prove the coherence theorem and then to construct the free monoidal category generated by  $C$ .

```

locale monoidal-language =
  C: category C
  for C :: 'a comp                (infixr · 55)
begin

  datatype (discs-sels) 't term =
    Prim 't                        (⟨-⟩)
  | Unity                          (I)
  | Tensor 't term 't term         (infixr ⊗ 53)
  | Comp '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[-, -, -])

```

**lemma** not-is-Tensor-Unity:

```

shows ¬ is-Tensor Unity
by simp

```

We define formal domain and codomain functions on terms.

```

primrec Dom :: 'a term ⇒ 'a term
where Dom ⟨f⟩ = ⟨C.dom f⟩
  | Dom I = I
  | Dom (t ⊗ u) = (Dom t ⊗ Dom u)
  | Dom (t · u) = Dom u
  | Dom l[t] = (I ⊗ Dom t)
  | Dom l-1[t] = Dom t
  | Dom r[t] = (Dom t ⊗ I)
  | 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 ⟨f⟩ = ⟨C.cod f⟩
  | Cod I = I
  | Cod (t ⊗ u) = (Cod t ⊗ Cod u)
  | Cod (t · u) = Cod t
  | Cod l[t] = Cod t
  | Cod l-1[t] = (I ⊗ Cod t)
  | Cod r[t] = Cod t
  | Cod r-1[t] = (Cod t ⊗ I)
  | 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 arrows of  $C$  in such a way that composition is applied only to formally composable pairs of terms.

**primrec**  $Arr :: 'a\ term \Rightarrow bool$   
**where**  $Arr \langle f \rangle = C.arr\ f$   
 $| Arr\ \mathcal{I} = True$   
 $| Arr\ (t \otimes u) = (Arr\ t \wedge Arr\ u)$   
 $| Arr\ (t \cdot u) = (Arr\ t \wedge Arr\ u \wedge Dom\ t = Cod\ u)$   
 $| Arr\ \mathbf{l}[t] = Arr\ t$   
 $| Arr\ \mathbf{l}^{-1}[t] = Arr\ t$   
 $| Arr\ \mathbf{r}[t] = Arr\ t$   
 $| Arr\ \mathbf{r}^{-1}[t] = Arr\ t$   
 $| Arr\ \mathbf{a}[t, u, v] = (Arr\ t \wedge Arr\ u \wedge Arr\ v)$   
 $| Arr\ \mathbf{a}^{-1}[t, u, v] = (Arr\ t \wedge Arr\ u \wedge Arr\ v)$

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

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

**abbreviation**  $Hom :: 'a\ term \Rightarrow 'a\ term \Rightarrow 'a\ term\ set$   
**where**  $Hom\ a\ b \equiv \{ t. Arr\ t \wedge Dom\ t = a \wedge Cod\ t = b \}$

A term is a “formal identity” if it is constructed from identity arrows of  $C$  and  $\mathcal{I}$  using only the  $\otimes$  operator.

**primrec**  $Ide :: 'a\ term \Rightarrow bool$   
**where**  $Ide \langle f \rangle = C.ide\ f$   
 $| Ide\ \mathcal{I} = True$   
 $| Ide\ (t \otimes u) = (Ide\ t \wedge Ide\ u)$   
 $| Ide\ (t \cdot u) = False$   
 $| Ide\ \mathbf{l}[t] = False$   
 $| Ide\ \mathbf{l}^{-1}[t] = False$   
 $| Ide\ \mathbf{r}[t] = False$   
 $| Ide\ \mathbf{r}^{-1}[t] = False$   
 $| Ide\ \mathbf{a}[t, u, v] = False$   
 $| Ide\ \mathbf{a}^{-1}[t, u, v] = False$

**lemma**  $Ide\text{-implies-}Arr$  [*simp*]:  
**shows**  $Ide\ t \Longrightarrow Arr\ t$   
**by** (*induct t*) *auto*

**lemma**  $Arr\text{-implies-}Ide\text{-}Dom$  [*elim*]:  
**shows**  $Arr\ t \Longrightarrow Ide\ (Dom\ t)$   
**by** (*induct t*) *auto*

**lemma**  $Arr\text{-implies-}Ide\text{-}Cod$  [*elim*]:  
**shows**  $Arr\ t \Longrightarrow Ide\ (Cod\ t)$   
**by** (*induct t*) *auto*

**lemma** *Ide-in-Hom* [*simp*]:  
**shows**  $Ide\ t \implies t \in Hom\ t\ t$   
**by** (*induct t*) *auto*

A formal arrow is “canonical” if the only arrows of  $C$  used in its construction are identities.

**primrec** *Can* :: 'a term  $\Rightarrow$  bool  
**where**  $Can\ \langle f \rangle = C.ide\ f$   
|  $Can\ \mathcal{I} = True$   
|  $Can\ (t \otimes u) = (Can\ t \wedge Can\ u)$   
|  $Can\ (t \cdot u) = (Can\ t \wedge Can\ u \wedge Dom\ t = Cod\ u)$   
|  $Can\ \mathbf{l}[t] = Can\ t$   
|  $Can\ \mathbf{l}^{-1}[t] = Can\ t$   
|  $Can\ \mathbf{r}[t] = Can\ t$   
|  $Can\ \mathbf{r}^{-1}[t] = Can\ t$   
|  $Can\ \mathbf{a}[t, u, v] = (Can\ t \wedge Can\ u \wedge Can\ v)$   
|  $Can\ \mathbf{a}^{-1}[t, u, v] = (Can\ t \wedge Can\ u \wedge Can\ v)$

**lemma** *Ide-implies-Can*:  
**shows**  $Ide\ t \implies Can\ t$   
**by** (*induct t*) *auto*

**lemma** *Can-implies-Arr*:  
**shows**  $Can\ t \implies Arr\ t$   
**by** (*induct t*) *auto*

We next define the formal inverse of a term. This is only sensible for formal arrows built using only isomorphisms of  $C$ ; in particular, for canonical formal arrows.

**primrec** *Inv* :: 'a term  $\Rightarrow$  'a term  
**where**  $Inv\ \langle f \rangle = \langle C.inv\ f \rangle$   
|  $Inv\ \mathcal{I} = \mathcal{I}$   
|  $Inv\ (t \otimes u) = (Inv\ t \otimes Inv\ u)$   
|  $Inv\ (t \cdot u) = (Inv\ u \cdot Inv\ t)$   
|  $Inv\ \mathbf{l}[t] = \mathbf{l}^{-1}[Inv\ t]$   
|  $Inv\ \mathbf{l}^{-1}[t] = \mathbf{l}[Inv\ t]$   
|  $Inv\ \mathbf{r}[t] = \mathbf{r}^{-1}[Inv\ t]$   
|  $Inv\ \mathbf{r}^{-1}[t] = \mathbf{r}[Inv\ t]$   
|  $Inv\ \mathbf{a}[t, u, v] = \mathbf{a}^{-1}[Inv\ t, Inv\ u, Inv\ v]$   
|  $Inv\ \mathbf{a}^{-1}[t, u, v] = \mathbf{a}[Inv\ t, Inv\ u, Inv\ v]$

**lemma** *Inv-preserves-Ide*:  
**shows**  $Ide\ t \implies Ide\ (Inv\ t)$   
**by** (*induct t*) *auto*

**lemma** *Inv-preserves-Can*:  
**assumes**  $Can\ t$   
**shows**  $Can\ (Inv\ t)$  **and**  $Dom\ (Inv\ t) = Cod\ t$  **and**  $Cod\ (Inv\ t) = Dom\ t$   
**proof** –  
**have**  $0$ :  $Can\ t \implies Can\ (Inv\ t) \wedge Dom\ (Inv\ t) = Cod\ t \wedge Cod\ (Inv\ t) = Dom\ t$

```

  by (induct t) auto
  show Can (Inv t) using assms 0 by blast
  show Dom (Inv t) = Cod t using assms 0 by blast
  show Cod (Inv t) = Dom t using assms 0 by blast
qed

```

```

lemma Inv-in-Hom [simp]:
  assumes Can t
  shows Inv t ∈ Hom (Cod t) (Dom t)
  using assms Inv-preserves-Can Can-implies-Arr by simp

```

```

lemma Inv-Ide [simp]:
  assumes Ide a
  shows Inv a = a
  using assms by (induct a) auto

```

```

lemma Inv-Inv [simp]:
  assumes Can t
  shows Inv (Inv t) = t
  using assms by (induct t) auto

```

We call a term “diagonal” if it is either  $\mathcal{I}$  or it is constructed from arrows of  $C$  using only the  $\otimes$  operator associated to the right. Essentially, such terms are lists of arrows of  $C$ , where  $\mathcal{I}$  represents the empty list and  $\otimes$  is used as the list constructor. We call them “diagonal” because terms can be regarded as defining “interconnection matrices” of arrows connecting “inputs” to “outputs”, and from this point of view diagonal terms correspond to diagonal matrices. The matrix point of view is suggestive for the extension of the results presented here to the symmetric monoidal and cartesian monoidal cases.

```

fun Diag :: 'a term ⇒ bool
where Diag  $\mathcal{I}$  = True
  | Diag ⟨f⟩ = C.arr f
  | Diag (⟨f⟩ ⊗ u) = (C.arr f ∧ Diag u ∧ u ≠  $\mathcal{I}$ )
  | Diag - = False

```

```

lemma Diag-TensorE:
  assumes Diag (Tensor t u)
  shows ⟨un-Prim t⟩ = t and C.arr (un-Prim t) and Diag t and Diag u and u ≠  $\mathcal{I}$ 
  proof -
    have 1: t = ⟨un-Prim t⟩ ∧ C.arr (un-Prim t) ∧ Diag t ∧ Diag u ∧ u ≠  $\mathcal{I}$ 
      using assms by (cases t; simp; cases u; simp)
    show ⟨un-Prim t⟩ = t using 1 by simp
    show C.arr (un-Prim t) using 1 by simp
    show Diag t using 1 by simp
    show Diag u using 1 by simp
    show u ≠  $\mathcal{I}$  using 1 by simp
  qed

```

```

lemma Diag-implies-Arr [elim]:
  shows Diag t ⇒ Arr t

```

**apply** (*induct t, simp-all*)  
**by** (*simp add: Diag-TensorE*)

**lemma** *Dom-preserves-Diag* [*elim*]:

**shows**  $Diag\ t \implies Diag\ (Dom\ t)$

**apply** (*induct t, simp-all*)

**proof** –

**fix**  $u\ v$

**assume**  $I2: Diag\ v \implies Diag\ (Dom\ v)$

**assume**  $uv: Diag\ (u \otimes v)$

**show**  $Diag\ (Dom\ u \otimes Dom\ v)$

**proof** –

**have**  $1: is-Prim\ (Dom\ u) \wedge C.arr\ (un-Prim\ (Dom\ u)) \wedge$   
 $Dom\ u = \langle C.dom\ (un-Prim\ u) \rangle$

**using**  $uv$  **by** (*cases u; simp; cases v, simp-all*)

**have**  $2: Diag\ v \wedge v \neq \mathcal{I} \wedge \neg is-Comp\ v \wedge \neg is-Lunit'\ v \wedge \neg is-Runit'\ v$

**using**  $uv$  **by** (*cases u; simp; cases v, simp-all*)

**have**  $Diag\ (Dom\ v) \wedge Dom\ v \neq \mathcal{I}$

**using**  $2\ I2$  **by** (*cases v, simp-all*)

**thus** *?thesis* **using**  $1$  **by** *force*

**qed**

**qed**

**lemma** *Cod-preserves-Diag* [*elim*]:

**shows**  $Diag\ t \implies Diag\ (Cod\ t)$

**apply** (*induct t, simp-all*)

**proof** –

**fix**  $u\ v$

**assume**  $I2: Diag\ v \implies Diag\ (Cod\ v)$

**assume**  $uv: Diag\ (u \otimes v)$

**show**  $Diag\ (Cod\ u \otimes Cod\ v)$

**proof** –

**have**  $1: is-Prim\ (Cod\ u) \wedge C.arr\ (un-Prim\ (Cod\ u)) \wedge Cod\ u = \langle C.cod\ (un-Prim\ u) \rangle$   
**using**  $uv$  **by** (*cases u; simp; cases v; simp*)

**have**  $2: Diag\ v \wedge v \neq \mathcal{I} \wedge \neg is-Comp\ v \wedge \neg is-Lunit'\ v \wedge \neg is-Runit'\ v$

**using**  $uv$  **by** (*cases u; simp; cases v; simp*)

**have**  $Diag\ (Cod\ v) \wedge Cod\ v \neq \mathcal{I}$

**using**  $I2\ 2$  **by** (*cases v, simp-all*)

**thus** *?thesis* **using**  $1$  **by** *force*

**qed**

**qed**

**lemma** *Inv-preserves-Diag*:

**assumes** *Can t* **and** *Diag t*

**shows**  $Diag\ (Inv\ t)$

**proof** –

**have**  $Can\ t \wedge Diag\ t \implies Diag\ (Inv\ t)$

**apply** (*induct t, simp-all*)

**by** (*metis (no-types, lifting) Can.simps(1) Inv.simps(1) Inv.simps(2) Diag.simps(3)*)

*Inv-Inv Diag-TensorE(1) C.inv-ide*  
**thus** *?thesis using assms by blast*  
**qed**

The following function defines the “dimension” of a term, which is the number of arrows of  $(\cdot)$  it contains. For diagonal terms, this is just the length of the term when regarded as a list of arrows of  $(\cdot)$ . Alternatively, if a term is regarded as defining an interconnection matrix, then the dimension is the number of inputs (or outputs).

**primrec** *dim* :: 'a term  $\Rightarrow$  nat  
**where** *dim*  $\langle f \rangle = 1$   
| *dim*  $\mathcal{I} = 0$   
| *dim*  $(t \otimes u) = (\text{dim } t + \text{dim } u)$   
| *dim*  $(t \cdot u) = \text{dim } t$   
| *dim*  $\mathbf{l}[t] = \text{dim } t$   
| *dim*  $\mathbf{l}^{-1}[t] = \text{dim } t$   
| *dim*  $\mathbf{r}[t] = \text{dim } t$   
| *dim*  $\mathbf{r}^{-1}[t] = \text{dim } t$   
| *dim*  $\mathbf{a}[t, u, v] = \text{dim } t + \text{dim } u + \text{dim } v$   
| *dim*  $\mathbf{a}^{-1}[t, u, v] = \text{dim } t + \text{dim } u + \text{dim } v$

The following function defines a tensor product for diagonal terms. If terms are regarded as lists, this is just list concatenation. If terms are regarded as matrices, this corresponds to constructing a block diagonal matrix.

**fun** *TensorDiag* (infixr  $[\otimes]$  53)  
**where**  $\mathcal{I} [\otimes] u = u$   
|  $t [\otimes] \mathcal{I} = t$   
|  $\langle f \rangle [\otimes] u = \langle f \rangle \otimes u$   
|  $(t \otimes u) [\otimes] v = t [\otimes] (u [\otimes] v)$   
|  $t [\otimes] u = \text{undefined}$

**lemma** *TensorDiag-Prim* [simp]:  
**assumes**  $t \neq \mathcal{I}$   
**shows**  $\langle f \rangle [\otimes] t = \langle f \rangle \otimes t$   
**using** *assms* **by** (*cases t, simp-all*)

**lemma** *TensorDiag-term-Unity* [simp]:  
**shows**  $t [\otimes] \mathcal{I} = t$   
**by** (*cases t = \mathcal{I}; cases t, simp-all*)

**lemma** *TensorDiag-Diag*:  
**assumes** *Diag*  $(t \otimes u)$   
**shows**  $t [\otimes] u = t \otimes u$   
**using** *assms TensorDiag-Prim* **by** (*cases t, simp-all*)

**lemma** *TensorDiag-preserves-Diag*:  
**assumes** *Diag*  $t$  **and** *Diag*  $u$   
**shows** *Diag*  $(t [\otimes] u)$   
**and** *Dom*  $(t [\otimes] u) = \text{Dom } t [\otimes] \text{Dom } u$   
**and** *Cod*  $(t [\otimes] u) = \text{Cod } t [\otimes] \text{Cod } u$

**proof** –  
**have**  $0$ :  $\bigwedge u. \llbracket \text{Diag } t; \text{Diag } u \rrbracket \implies$   
 $\text{Diag } (t \llbracket \otimes \rrbracket u) \wedge \text{Dom } (t \llbracket \otimes \rrbracket u) = \text{Dom } t \llbracket \otimes \rrbracket \text{Dom } u \wedge$   
 $\text{Cod } (t \llbracket \otimes \rrbracket u) = \text{Cod } t \llbracket \otimes \rrbracket \text{Cod } u$

**apply** (*induct t, simp-all*)

**proof** –  
**fix**  $f :: 'a$  **and**  $u :: 'a$  *term*  
**assume**  $f: C.\text{arr } f$   
**assume**  $u: \text{Diag } u$   
**show**  $\text{Diag } (\langle f \rangle \llbracket \otimes \rrbracket u) \wedge \text{Dom } (\langle f \rangle \llbracket \otimes \rrbracket u) = \langle C.\text{dom } f \rangle \llbracket \otimes \rrbracket \text{Dom } u \wedge$   
 $\text{Cod } (\langle f \rangle \llbracket \otimes \rrbracket u) = \langle C.\text{cod } f \rangle \llbracket \otimes \rrbracket \text{Cod } u$   
**using**  $u f$  **by** (*cases u, simp-all*)

**next**  
**fix**  $u v w$   
**assume**  $I1$ :  $\bigwedge u. \text{Diag } v \implies \text{Diag } u \implies \text{Diag } (v \llbracket \otimes \rrbracket u) \wedge$   
 $\text{Dom } (v \llbracket \otimes \rrbracket u) = \text{Dom } v \llbracket \otimes \rrbracket \text{Dom } u \wedge$   
 $\text{Cod } (v \llbracket \otimes \rrbracket u) = \text{Cod } v \llbracket \otimes \rrbracket \text{Cod } u$

**assume**  $I2$ :  $\bigwedge u. \text{Diag } w \implies \text{Diag } u \implies \text{Diag } (w \llbracket \otimes \rrbracket u) \wedge$   
 $\text{Dom } (w \llbracket \otimes \rrbracket u) = \text{Dom } w \llbracket \otimes \rrbracket \text{Dom } u \wedge$   
 $\text{Cod } (w \llbracket \otimes \rrbracket u) = \text{Cod } w \llbracket \otimes \rrbracket \text{Cod } u$

**assume**  $vw: \text{Diag } (v \otimes w)$   
**assume**  $u: \text{Diag } u$   
**show**  $\text{Diag } ((v \otimes w) \llbracket \otimes \rrbracket u) \wedge$   
 $\text{Dom } ((v \otimes w) \llbracket \otimes \rrbracket u) = (\text{Dom } v \otimes \text{Dom } w) \llbracket \otimes \rrbracket \text{Dom } u \wedge$   
 $\text{Cod } ((v \otimes w) \llbracket \otimes \rrbracket u) = (\text{Cod } v \otimes \text{Cod } w) \llbracket \otimes \rrbracket \text{Cod } u$

**proof** –  
**have**  $v: v = \langle \text{un-Prim } v \rangle \wedge \text{Diag } v$   
**using**  $vw$  *Diag-implies-Arr* *Diag-TensorE* [of  $v w$ ] **by** *force*  
**have**  $w: \text{Diag } w$   
**using**  $vw$  **by** (*simp add: Diag-TensorE*)  
**have**  $u = \mathcal{I} \implies ?thesis$  **by** (*simp add: vw*)  
**moreover** **have**  $u \neq \mathcal{I} \implies ?thesis$   
**using**  $u v w I1 I2$  *Dom-preserves-Diag* [of  $u$ ] *Cod-preserves-Diag* [of  $u$ ]  
**by** (*cases u, simp-all*)  
**ultimately show** *?thesis* **by** *blast*

**qed**  
**qed**  
**show**  $\text{Diag } (t \llbracket \otimes \rrbracket u)$  **using**  $assms\ 0$  **by** *blast*  
**show**  $\text{Dom } (t \llbracket \otimes \rrbracket u) = \text{Dom } t \llbracket \otimes \rrbracket \text{Dom } u$  **using**  $assms\ 0$  **by** *blast*  
**show**  $\text{Cod } (t \llbracket \otimes \rrbracket u) = \text{Cod } t \llbracket \otimes \rrbracket \text{Cod } u$  **using**  $assms\ 0$  **by** *blast*

**qed**

**lemma** *TensorDiag-in-Hom*:  
**assumes**  $\text{Diag } t$  **and**  $\text{Diag } u$   
**shows**  $t \llbracket \otimes \rrbracket u \in \text{Hom } (\text{Dom } t \llbracket \otimes \rrbracket \text{Dom } u) (\text{Cod } t \llbracket \otimes \rrbracket \text{Cod } u)$   
**using**  $assms$  *TensorDiag-preserves-Diag* *Diag-implies-Arr* **by** *simp*

**lemma** *Dom-TensorDiag*:  
**assumes**  $\text{Diag } t$  **and**  $\text{Diag } u$

**shows**  $Dom (t \lfloor \otimes \rfloor u) = Dom t \lfloor \otimes \rfloor Dom u$   
**using** *assms TensorDiag-preserves-Diag(2)* **by** *simp*

**lemma** *Cod-TensorDiag*:  
**assumes** *Diag t* **and** *Diag u*  
**shows**  $Cod (t \lfloor \otimes \rfloor u) = Cod t \lfloor \otimes \rfloor Cod u$   
**using** *assms TensorDiag-preserves-Diag(3)* **by** *simp*

**lemma** *not-is-Tensor-TensorDiagE*:  
**assumes**  $\neg is-Tensor (t \lfloor \otimes \rfloor u)$  **and** *Diag t* **and** *Diag u*  
**and**  $t \neq \mathcal{I}$  **and**  $u \neq \mathcal{I}$   
**shows** *False*

**proof** –  
**have**  $\llbracket \neg is-Tensor (t \lfloor \otimes \rfloor u); Diag t; Diag u; t \neq \mathcal{I}; u \neq \mathcal{I} \rrbracket \implies False$   
**apply** (*induct t, simp-all*)  
**proof** –  
**fix**  $v w$   
**assume**  $I2: \neg is-Tensor (w \lfloor \otimes \rfloor u) \implies Diag w \implies w \neq \mathcal{I} \implies False$   
**assume**  $1: \neg is-Tensor ((v \otimes w) \lfloor \otimes \rfloor u)$   
**assume**  $vw: Diag (v \otimes w)$   
**assume**  $u: Diag u$   
**assume**  $2: u \neq \mathcal{I}$   
**show** *False*  
**proof** –  
**have**  $v: v = \langle un-Prim v \rangle$   
**using**  $vw$  *Diag-TensorE* [*of v w*] **by** *force*  
**have**  $w: Diag w \wedge w \neq \mathcal{I}$   
**using**  $vw$  *Diag-TensorE* [*of v w*] **by** *simp*  
**have**  $(v \otimes w) \lfloor \otimes \rfloor u = v \otimes (w \lfloor \otimes \rfloor u)$   
**proof** –  
**have**  $(v \otimes w) \lfloor \otimes \rfloor u = v \lfloor \otimes \rfloor (w \lfloor \otimes \rfloor u)$   
**using**  $u$   $2$  **by** (*cases u, simp-all*)  
**also have**  $\dots = v \otimes (w \lfloor \otimes \rfloor u)$   
**using**  $u v w$   $I2$  *TensorDiag-Prim not-is-Tensor-Unity* **by** *metis*  
**finally show** *?thesis* **by** *simp*  
**qed**  
**thus** *?thesis* **using**  $1$  **by** *simp*  
**qed**  
**thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**lemma** *TensorDiag-assoc*:  
**assumes** *Diag t* **and** *Diag u* **and** *Diag v*  
**shows**  $(t \lfloor \otimes \rfloor u) \lfloor \otimes \rfloor v = t \lfloor \otimes \rfloor (u \lfloor \otimes \rfloor v)$   
**proof** –  
**have**  $\bigwedge n t u v. \llbracket dim t = n; Diag t; Diag u; Diag v \rrbracket \implies$   
 $(t \lfloor \otimes \rfloor u) \lfloor \otimes \rfloor v = t \lfloor \otimes \rfloor (u \lfloor \otimes \rfloor v)$   
**proof** –

```

fix n
show  $\wedge t u v. \llbracket \dim t = n; \text{Diag } t; \text{Diag } u; \text{Diag } v \rrbracket \implies$ 
       $(t \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v = t \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v)$ 
proof (induction n rule: nat-less-induct)
  fix n :: nat and t :: 'a term and u v
  assume I:  $\forall m < n. \forall t u v. \dim t = m \longrightarrow \text{Diag } t \longrightarrow \text{Diag } u \longrightarrow \text{Diag } v \longrightarrow$ 
       $(t \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v = t \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v)$ 
  assume dim:  $\dim t = n$ 
  assume t: Diag t
  assume u: Diag u
  assume v: Diag v
  show  $(t \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v = t \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v)$ 
  proof -
    have  $t = \mathcal{I} \implies ?thesis$  by simp
    moreover have  $u = \mathcal{I} \implies ?thesis$  by simp
    moreover have  $v = \mathcal{I} \implies ?thesis$  by simp
    moreover have  $t \neq \mathcal{I} \wedge u \neq \mathcal{I} \wedge v \neq \mathcal{I} \wedge \text{is-Prim } t \implies ?thesis$ 
      using v by (cases t, simp-all, cases u, simp-all; cases v, simp-all)
    moreover have  $t \neq \mathcal{I} \wedge u \neq \mathcal{I} \wedge v \neq \mathcal{I} \wedge \text{is-Tensor } t \implies ?thesis$ 
  proof (cases t; simp)
    fix w :: 'a term and x :: 'a term
    assume 1:  $u \neq \mathcal{I} \wedge v \neq \mathcal{I}$ 
    assume 2:  $t = (w \otimes x)$ 
    show  $((w \otimes x) \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v = (w \otimes x) \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v)$ 
    proof -
      have w:  $w = \langle \text{un-Prim } w \rangle$ 
      using t 1 2 Diag-TensorE [of w x] by auto
      have x: Diag x
      using t w 1 2 by (cases w, simp-all)
      have  $((w \otimes x) \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v = (w \llbracket \otimes \rrbracket (x \llbracket \otimes \rrbracket u)) \llbracket \otimes \rrbracket v$ 
      using u v w x 1 2 by (cases u, simp-all)
      also have ... =  $(w \otimes (x \llbracket \otimes \rrbracket u)) \llbracket \otimes \rrbracket v$ 
      using t w u 1 2 TensorDiag-Prim not-is-Tensor-TensorDiagE Diag-TensorE
        not-is-Tensor-Unity
      by metis
      also have ... =  $w \llbracket \otimes \rrbracket ((x \llbracket \otimes \rrbracket u) \llbracket \otimes \rrbracket v)$ 
      using u v w x 1 by (cases u, simp-all; cases v, simp-all)
      also have ... =  $w \llbracket \otimes \rrbracket (x \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v))$ 
    proof -
      have dim x < dim t
      using w 2 by (cases w, simp-all)
      thus ?thesis
      using u v x dim I by simp
    qed
    also have ... =  $(w \otimes x) \llbracket \otimes \rrbracket (u \llbracket \otimes \rrbracket v)$ 
  proof -
    have 3: is-Tensor (u  $\llbracket \otimes \rrbracket v$ )
    using u v 1 not-is-Tensor-TensorDiagE by auto
    obtain u' :: 'a term and v' where uv':  $u \llbracket \otimes \rrbracket v = u' \otimes v'$ 

```



using  $\mathcal{I}$  is-Tensor-def by blast  
 thus ?thesis by simp  
 qed  
 finally show ?thesis by simp  
 qed  
 qed  
 moreover have  $t = \mathcal{I} \vee \text{is-Prim } t \vee \text{is-Tensor } t$   
 using  $t$  by (cases  $t$ , simp-all)  
 ultimately show ?thesis by blast  
 qed  
 qed  
 qed  
 thus ?thesis using assms by blast  
 qed

**lemma** *TensorDiag-preserves-Ide*:

**assumes** *Ide*  $t$  and *Ide*  $u$  and *Diag*  $t$  and *Diag*  $u$

**shows** *Ide*  $(t \llbracket \otimes \rrbracket u)$

using *assms*

by (*metis* (*mono-tags*, *lifting*) *Arr-implies-Ide-Dom* *Ide-in-Hom* *Diag-implies-Arr*  
*TensorDiag-preserves-Diag(1)* *TensorDiag-preserves-Diag(2)* *mem-Collect-eq*)

**lemma** *TensorDiag-preserves-Can*:

**assumes** *Can*  $t$  and *Can*  $u$  and *Diag*  $t$  and *Diag*  $u$

**shows** *Can*  $(t \llbracket \otimes \rrbracket u)$

**proof** –

have  $\bigwedge u. \llbracket \text{Can } t \wedge \text{Diag } t; \text{Can } u \wedge \text{Diag } u \rrbracket \implies \text{Can } (t \llbracket \otimes \rrbracket u)$

**proof** (*induct*  $t$ ; *simp*)

show  $\bigwedge x u. C.\text{ide } x \wedge C.\text{arr } x \implies \text{Can } u \wedge \text{Diag } u \implies \text{Can } (\langle x \rangle \llbracket \otimes \rrbracket u)$

by (*metis* *Ide.simps(1)* *Ide.simps(2)* *Ide-implies-Can* *Diag.simps(2)* *TensorDiag-Prim*  
*TensorDiag-preserves-Ide* *Can.simps(3)*)

show  $\bigwedge t1 t2 u. (\bigwedge u. \text{Diag } t1 \implies \text{Can } u \wedge \text{Diag } u \implies \text{Can } (t1 \llbracket \otimes \rrbracket u)) \implies$   
 $(\bigwedge u. \text{Diag } t2 \implies \text{Can } u \wedge \text{Diag } u \implies \text{Can } (t2 \llbracket \otimes \rrbracket u)) \implies$   
 $\text{Can } t1 \wedge \text{Can } t2 \wedge \text{Diag } (t1 \otimes t2) \implies \text{Can } u \wedge \text{Diag } u \implies$   
 $\text{Can } ((t1 \otimes t2) \llbracket \otimes \rrbracket u)$

by (*metis* *Diag-TensorE(3)* *Diag-TensorE(4)* *TensorDiag-Diag* *TensorDiag-assoc*  
*TensorDiag-preserves-Diag(1)*)

qed

thus ?thesis using *assms* by blast

qed

**lemma** *Inv-TensorDiag*:

**assumes** *Can*  $t$  and *Can*  $u$  and *Diag*  $t$  and *Diag*  $u$

**shows** *Inv*  $(t \llbracket \otimes \rrbracket u) = \text{Inv } t \llbracket \otimes \rrbracket \text{Inv } u$

**proof** –

have  $\bigwedge u. \llbracket \text{Can } t \wedge \text{Diag } t; \text{Can } u \wedge \text{Diag } u \rrbracket \implies \text{Inv } (t \llbracket \otimes \rrbracket u) = \text{Inv } t \llbracket \otimes \rrbracket \text{Inv } u$

**proof** (*induct*  $t$ , *simp-all*)

fix  $f u$

assume  $f: C.\text{ide } f \wedge C.\text{arr } f$

```

assume  $u$ :  $Can\ u \wedge\ Diag\ u$ 
show  $Inv\ (\langle f \rangle\ [\otimes]\ u) = \langle f \rangle\ [\otimes]\ Inv\ u$ 
  using  $f\ u$  by ( $cases\ u$ ,  $simp$ -all)
next
fix  $t\ u\ v$ 
assume  $I1$ :  $\bigwedge v. \llbracket Diag\ t; Can\ v \wedge\ Diag\ v \rrbracket \implies Inv\ (t\ [\otimes]\ v) = Inv\ t\ [\otimes]\ Inv\ v$ 
assume  $I2$ :  $\bigwedge v. \llbracket Diag\ u; Can\ v \wedge\ Diag\ v \rrbracket \implies Inv\ (u\ [\otimes]\ v) = Inv\ u\ [\otimes]\ Inv\ v$ 
assume  $tu$ :  $Can\ t \wedge\ Can\ u \wedge\ Diag\ (t\ \otimes\ u)$ 
have  $t$ :  $Can\ t \wedge\ Diag\ t$ 
  using  $tu\ Diag$ -TensorE [of  $t\ u$ ] by force
have  $u$ :  $Can\ u \wedge\ Diag\ u$ 
  using  $t\ tu$  by ( $cases\ t$ ,  $simp$ -all)
assume  $v$ :  $Can\ v \wedge\ Diag\ v$ 
show  $Inv\ ((t\ \otimes\ u)\ [\otimes]\ v) = (Inv\ t\ \otimes\ Inv\ u)\ [\otimes]\ Inv\ v$ 
proof -
  have  $v = Unity \implies ?thesis$  by  $simp$ 
  moreover have  $v \neq Unity \implies ?thesis$ 
  proof -
    assume  $1$ :  $v \neq Unity$ 
    have  $Inv\ ((t\ \otimes\ u)\ [\otimes]\ v) = Inv\ (t\ [\otimes]\ (u\ [\otimes]\ v))$ 
      using  $1$  by ( $cases\ v$ ,  $simp$ -all)
    also have  $\dots = Inv\ t\ [\otimes]\ Inv\ (u\ [\otimes]\ v)$ 
      using  $t\ u\ v\ I1\ TensorDiag$ -preserves-Diag  $TensorDiag$ -preserves-Can
       $Inv$ -preserves-Diag  $Inv$ -preserves-Can
      by  $simp$ 
    also have  $\dots = Inv\ t\ [\otimes]\ (Inv\ u\ [\otimes]\ Inv\ v)$ 
      using  $t\ u\ v\ I2\ TensorDiag$ -preserves-Diag  $TensorDiag$ -preserves-Can
       $Inv$ -preserves-Diag  $Inv$ -preserves-Can
      by  $simp$ 
    also have  $\dots = (Inv\ t\ \otimes\ Inv\ u)\ [\otimes]\ Inv\ v$ 
      using  $v\ 1$  by ( $cases\ v$ ,  $simp$ -all)
    finally show  $?thesis$  by blast
  qed
  ultimately show  $?thesis$  by blast
qed
qed
thus  $?thesis$  using  $assms$  by blast
qed

```

The following function defines composition for compatible diagonal terms, by “pushing the composition down” to arrows of  $C$ .

```

fun  $CompDiag$  :: 'a term  $\Rightarrow$  'a term  $\Rightarrow$  'a term    (infixr  $[\cdot]$  55)
where  $\mathcal{I}\ [\cdot]\ u = u$ 
  |  $\langle f \rangle\ [\cdot]\ \langle g \rangle = \langle f \cdot g \rangle$ 
  |  $(u\ \otimes\ v)\ [\cdot]\ (w\ \otimes\ x) = (u\ [\cdot]\ w\ \otimes\ v\ [\cdot]\ x)$ 
  |  $t\ [\cdot]\ \mathcal{I} = t$ 
  |  $t\ [\cdot]\ - = undefined \cdot undefined$ 

```

Note that the last clause above is not relevant to diagonal terms. We have chosen a provably non-diagonal value in order to validate associativity.

**lemma** *CompDiag-preserves-Diag*:  
**assumes** *Diag t* **and** *Diag u* **and**  $Dom\ t = Cod\ u$   
**shows**  $Diag\ (t\ [\cdot]\ u)$   
**and**  $Dom\ (t\ [\cdot]\ u) = Dom\ u$   
**and**  $Cod\ (t\ [\cdot]\ u) = Cod\ t$   
**proof** –  
**have**  $0: \bigwedge u. \llbracket Diag\ t; Diag\ u; Dom\ t = Cod\ u \rrbracket \implies$   
 $Diag\ (t\ [\cdot]\ u) \wedge Dom\ (t\ [\cdot]\ u) = Dom\ u \wedge Cod\ (t\ [\cdot]\ u) = Cod\ t$   
**proof** (*induct t, simp-all add: Diag-TensorE*)  
**fix**  $f\ u$   
**assume**  $f: C.arr\ f$   
**assume**  $u: Diag\ u$   
**assume**  $1: \langle C.dom\ f \rangle = Cod\ u$   
**show**  $Diag\ (\langle f \rangle\ [\cdot]\ u) \wedge Dom\ (\langle f \rangle\ [\cdot]\ u) = Dom\ u \wedge Cod\ (\langle f \rangle\ [\cdot]\ u) = \langle C.cod\ f \rangle$   
**using**  $f\ u\ 1$  **by** (*cases u, simp-all*)  
**next**  
**fix**  $u\ v\ w$   
**assume**  $I2: \bigwedge u. \llbracket Diag\ u; Dom\ w = Cod\ u \rrbracket \implies$   
 $Diag\ (w\ [\cdot]\ u) \wedge Dom\ (w\ [\cdot]\ u) = Dom\ u \wedge Cod\ (w\ [\cdot]\ u) = Cod\ w$   
**assume**  $vw: Diag\ (v \otimes w)$   
**have**  $v: Diag\ v$   
**using**  $vw\ Diag-TensorE$  [*of v w*] **by force**  
**have**  $w: Diag\ w$   
**using**  $vw\ Diag-TensorE$  [*of v w*] **by force**  
**assume**  $u: Diag\ u$   
**assume**  $1: (Dom\ v \otimes Dom\ w) = Cod\ u$   
**show**  $Diag\ ((v \otimes w)\ [\cdot]\ u) \wedge Dom\ ((v \otimes w)\ [\cdot]\ u) = Dom\ u \wedge$   
 $Cod\ ((v \otimes w)\ [\cdot]\ u) = Cod\ v \otimes Cod\ w$   
**using**  $u\ v\ w\ 1$   
**proof** (*cases u, simp-all*)  
**fix**  $x\ y$   
**assume**  $2: u = Tensor\ x\ y$   
**have**  $4: is-Prim\ x \wedge x = \langle un-Prim\ x \rangle \wedge C.arr\ (un-Prim\ x) \wedge Diag\ y \wedge y \neq \mathcal{I}$   
**using**  $u\ 2$  **by** (*cases x, cases y, simp-all*)  
**have**  $5: is-Prim\ v \wedge v = \langle un-Prim\ v \rangle \wedge C.arr\ (un-Prim\ v) \wedge Diag\ w \wedge w \neq \mathcal{I}$   
**using**  $v\ w\ vw$  **by** (*cases v, simp-all*)  
**have**  $6: C.dom\ (un-Prim\ v) = C.cod\ (un-Prim\ x) \wedge Dom\ w = Cod\ y$   
**using**  $1\ 2\ 4\ 5$  **apply** (*cases u, simp-all*)  
**by** (*metis Cod.simps(1) Dom.simps(1) term.simps(1)*)  
**have**  $(v \otimes w)\ [\cdot]\ u = \langle un-Prim\ v \cdot un-Prim\ x \rangle \otimes w\ [\cdot]\ y$   
**using**  $2\ 4\ 5\ 6\ CompDiag.simps(2)$  [*of un-Prim v un-Prim x*] **by simp**  
**moreover** **have**  $Diag\ (\langle un-Prim\ v \cdot un-Prim\ x \rangle \otimes w\ [\cdot]\ y)$   
**proof** –  
**have**  $Diag\ (w\ [\cdot]\ y)$   
**using**  $I2\ 4\ 5\ 6$  **by simp**  
**thus** *?thesis*  
**using**  $4\ 5\ 6\ Diag.simps(3)$  [*of un-Prim v un-Prim x (w [\cdot] y)*]  
**by** (*cases w; cases y*) *auto*  
**qed**

**ultimately show**  $Diag (v [\cdot] x \otimes w [\cdot] y) \wedge$   
 $Dom (v [\cdot] x) = Dom x \wedge Dom (w [\cdot] y) = Dom y \wedge$   
 $Cod (v [\cdot] x) = Cod v \wedge Cod (w [\cdot] y) = Cod w$   
**using**  $4\ 5\ 6\ I2$   
**by**  $(metis (full-types) C.cod-comp C.dom-comp Cod.simps(1) CompDiag.simps(2)$   
 $Dom.simps(1) C.seqI)$   
**qed**  
**qed**  
**show**  $Diag (t [\cdot] u)$  **using**  $assms\ 0$  **by**  $blast$   
**show**  $Dom (t [\cdot] u) = Dom u$  **using**  $assms\ 0$  **by**  $blast$   
**show**  $Cod (t [\cdot] u) = Cod t$  **using**  $assms\ 0$  **by**  $blast$   
**qed**

**lemma**  $CompDiag-in-Hom$ :  
**assumes**  $Diag\ t$  **and**  $Diag\ u$  **and**  $Dom\ t = Cod\ u$   
**shows**  $t [\cdot] u \in Hom (Dom\ u) (Cod\ t)$   
**using**  $assms\ CompDiag-preserves-Diag\ Diag-implies-Arr$  **by**  $simp$

**lemma**  $Dom-CompDiag$ :  
**assumes**  $Diag\ t$  **and**  $Diag\ u$  **and**  $Dom\ t = Cod\ u$   
**shows**  $Dom (t [\cdot] u) = Dom u$   
**using**  $assms\ CompDiag-preserves-Diag(2)$  **by**  $simp$

**lemma**  $Cod-CompDiag$ :  
**assumes**  $Diag\ t$  **and**  $Diag\ u$  **and**  $Dom\ t = Cod\ u$   
**shows**  $Cod (t [\cdot] u) = Cod t$   
**using**  $assms\ CompDiag-preserves-Diag(3)$  **by**  $simp$

**lemma**  $CompDiag-Cod-Diag [simp]$ :  
**assumes**  $Diag\ t$   
**shows**  $Cod\ t [\cdot] t = t$   
**proof** –  
**have**  $Diag\ t \implies Cod\ t [\cdot] t = t$   
**using**  $C.comp-cod-arr$   
**apply**  $(induct\ t, auto)$   
**by**  $(auto\ simp\ add: Diag-TensorE)$   
**thus**  $?thesis$  **using**  $assms$  **by**  $blast$   
**qed**

**lemma**  $CompDiag-Diag-Dom [simp]$ :  
**assumes**  $Diag\ t$   
**shows**  $t [\cdot] Dom\ t = t$   
**proof** –  
**have**  $Diag\ t \implies t [\cdot] Dom\ t = t$   
**using**  $C.comp-arr-dom$   
**apply**  $(induct\ t, auto)$   
**by**  $(auto\ simp\ add: Diag-TensorE)$   
**thus**  $?thesis$  **using**  $assms$  **by**  $blast$   
**qed**

**lemma** *CompDiag-Ide-Diag* [*simp*]:  
**assumes** *Diag t* **and** *Ide a* **and** *Dom a = Cod t*  
**shows**  $a \llbracket \cdot \rrbracket t = t$   
**using** *assms Ide-in-Hom* **by** *simp*

**lemma** *CompDiag-Diag-Ide* [*simp*]:  
**assumes** *Diag t* **and** *Ide a* **and** *Dom t = Cod a*  
**shows**  $t \llbracket \cdot \rrbracket a = t$   
**using** *assms Ide-in-Hom* **by** *auto*

**lemma** *CompDiag-assoc*:  
**assumes** *Diag t* **and** *Diag u* **and** *Diag v*  
**and** *Dom t = Cod u* **and** *Dom u = Cod v*  
**shows**  $(t \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = t \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$   
**proof** –

**have**  $\bigwedge u v. \llbracket \text{Diag } t; \text{Diag } u; \text{Diag } v; \text{Dom } t = \text{Cod } u; \text{Dom } u = \text{Cod } v \rrbracket \implies$   
 $(t \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = t \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$

**proof** (*induct t, simp-all*)

**fix**  $f u v$

**assume**  $f: C.\text{arr } f$

**assume**  $u: \text{Diag } u$

**assume**  $v: \text{Diag } v$

**assume**  $1: \langle C.\text{dom } f \rangle = \text{Cod } u$

**assume**  $2: \text{Dom } u = \text{Cod } v$

**show**  $(\langle f \rangle \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = \langle f \rangle \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$

**using** *C.comp-assoc* **by** (*cases u, simp-all; cases v, simp-all*)

**next**

**fix**  $u v w x$

**assume**  $I1: \bigwedge u v. \llbracket \text{Diag } w; \text{Diag } u; \text{Diag } v; \text{Dom } w = \text{Cod } u; \text{Dom } u = \text{Cod } v \rrbracket \implies$   
 $(w \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = w \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$

**assume**  $I2: \bigwedge u v. \llbracket \text{Diag } x; \text{Diag } u; \text{Diag } v; \text{Dom } x = \text{Cod } u; \text{Dom } u = \text{Cod } v \rrbracket \implies$   
 $(x \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = x \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$

**assume**  $wx: \text{Diag } (w \otimes x)$

**assume**  $u: \text{Diag } u$

**assume**  $v: \text{Diag } v$

**assume**  $1: (\text{Dom } w \otimes \text{Dom } x) = \text{Cod } u$

**assume**  $2: \text{Dom } u = \text{Cod } v$

**show**  $((w \otimes x) \llbracket \cdot \rrbracket u) \llbracket \cdot \rrbracket v = (w \otimes x) \llbracket \cdot \rrbracket (u \llbracket \cdot \rrbracket v)$

**proof** –

**have**  $w: \text{Diag } w$

**using**  $wx$  *Diag-TensorE* **by** *blast*

**have**  $x: \text{Diag } x$

**using**  $wx$  *Diag-TensorE* **by** *blast*

**have** *is-Tensor*  $u$

**using**  $u$   $1$  **by** (*cases u*) *simp-all*

**thus** *?thesis*

**using**  $u v$  **apply** (*cases u, simp-all, cases v, simp-all*)

**proof** –

```

fix u1 u2 v1 v2
assume 3: u = (u1  $\otimes$  u2)
assume 4: v = (v1  $\otimes$  v2)
show (w [·] u1) [·] v1 = w [·] u1 [·] v1  $\wedge$ 
      (x [·] u2) [·] v2 = x [·] u2 [·] v2
proof –
  have Diag u1  $\wedge$  Diag u2
    using u 3 Diag-TensorE by blast
  moreover have Diag v1  $\wedge$  Diag v2
    using v 4 Diag-TensorE by blast
  ultimately show ?thesis using w x I1 I2 1 2 3 4 by simp
qed
qed
qed
qed
thus ?thesis using assms by blast
qed

```

**lemma** *CompDiag-preserves-Ide*:  
**assumes** *Ide t* **and** *Ide u* **and** *Diag t* **and** *Diag u* **and**  $Dom\ t = Cod\ u$   
**shows** *Ide* (t [·] u)  
**proof** –  
**have**  $\bigwedge u. \llbracket Ide\ t; Ide\ u; Diag\ t; Diag\ u; Dom\ t = Cod\ u \rrbracket \implies Ide\ (CompDiag\ t\ u)$   
**by** (*induct t*; *simp*)  
**thus** ?thesis **using** assms **by** blast  
**qed**

**lemma** *CompDiag-preserves-Can*:  
**assumes** *Can t* **and** *Can u* **and** *Diag t* **and** *Diag u* **and**  $Dom\ t = Cod\ u$   
**shows** *Can* (t [·] u)  
**proof** –  
**have**  $\bigwedge u. \llbracket Can\ t \wedge Diag\ t; Can\ u \wedge Diag\ u; Dom\ t = Cod\ u \rrbracket \implies Can\ (t\ [·]\ u)$   
**proof** (*induct t*, *simp-all*)  
**fix** t u v  
**assume** I1:  $\bigwedge v. \llbracket Diag\ t; Can\ v \wedge Diag\ v; Dom\ t = Cod\ v \rrbracket \implies Can\ (t\ [·]\ v)$   
**assume** I2:  $\bigwedge v. \llbracket Diag\ u; Can\ v \wedge Diag\ v; Dom\ u = Cod\ v \rrbracket \implies Can\ (u\ [·]\ v)$   
**assume** tu:  $Can\ t \wedge Can\ u \wedge Diag\ (t \otimes u)$   
**have** t:  $Can\ t \wedge Diag\ t$   
**using** tu *Diag-TensorE* **by** blast  
**have** u:  $Can\ u \wedge Diag\ u$   
**using** tu *Diag-TensorE* **by** blast  
**assume** v:  $Can\ v \wedge Diag\ v$   
**assume** 1:  $(Dom\ t \otimes Dom\ u) = Cod\ v$   
**show** *Can* ((t  $\otimes$  u) [·] v)  
**proof** –  
**have** 2:  $(Dom\ t \otimes Dom\ u) = Cod\ v$  **using** 1 **by** *simp*  
**show** ?thesis  
**using** v 2  
**proof** (*cases v*; *simp*)

```

fix w x
assume wx: v = (w ⊗ x)
have Can w ∧ Diag w using v wx Diag-TensorE [of w x] by auto
moreover have Can x ∧ Diag x using v wx Diag-TensorE [of w x] by auto
moreover have Dom t = Cod w using 2 wx by simp
moreover have ux: Dom u = Cod x using 2 wx by simp
ultimately show Can (t [·] w) ∧ Can (u [·] x)
using t u I1 I2 by simp
qed
qed
qed
thus ?thesis using assms by blast
qed

```

**lemma** *Inv-CompDiag*:

**assumes** Can t **and** Can u **and** Diag t **and** Diag u **and** Dom t = Cod u

**shows** Inv (t [·] u) = Inv u [·] Inv t

**proof** –

**have**  $\bigwedge u. \llbracket \text{Can } t \wedge \text{Diag } t; \text{Can } u \wedge \text{Diag } u; \text{Dom } t = \text{Cod } u \rrbracket \implies$   
 $\text{Inv } (t \text{ [}\cdot\text{]} u) = \text{Inv } u \text{ [}\cdot\text{]} \text{Inv } t$

**proof** (*induct t, simp-all*)

**show**  $\bigwedge x u. \llbracket C.\text{ide } x \wedge C.\text{arr } x; \text{Can } u \wedge \text{Diag } u; \langle x \rangle = \text{Cod } u \rrbracket \implies$   
 $\text{Inv } u = \text{Inv } u \text{ [}\cdot\text{]} \text{Inv } (\text{Cod } u)$

**by** (*metis CompDiag-Diag-Dom Inv-Ide Inv-preserves-Can(2) Inv-preserves-Diag*  
*Ide.simps(1)*)

**show**  $\bigwedge u. \text{Can } u \wedge \text{Diag } u \implies \mathcal{I} = \text{Cod } u \implies \text{Inv } u = \text{Inv } u \text{ [}\cdot\text{]} \mathcal{I}$

**by** (*simp add: Inv-preserves-Can(2) Inv-preserves-Diag*)

**fix** t u v

**assume** tu: Can t ∧ Can u ∧ Diag (t ⊗ u)

**have** t: Can t ∧ Diag t

**using** tu *Diag-TensorE* **by** blast

**have** u: Can u ∧ Diag u

**using** tu *Diag-TensorE* **by** blast

**assume** I1:  $\bigwedge v. \llbracket \text{Diag } t; \text{Can } v \wedge \text{Diag } v; \text{Dom } t = \text{Cod } v \rrbracket \implies$

$\text{Inv } (t \text{ [}\cdot\text{]} v) = \text{Inv } v \text{ [}\cdot\text{]} \text{Inv } t$

**assume** I2:  $\bigwedge v. \llbracket \text{Diag } u; \text{Can } v \wedge \text{Diag } v; \text{Dom } u = \text{Cod } v \rrbracket \implies$

$\text{Inv } (u \text{ [}\cdot\text{]} v) = \text{Inv } v \text{ [}\cdot\text{]} \text{Inv } u$

**assume** v: Can v ∧ Diag v

**assume** 1: (Dom t ⊗ Dom u) = Cod v

**show** Inv ((t ⊗ u) [·] v) = Inv v [·] (Inv t ⊗ Inv u)

**using** v 1

**proof** (*cases v, simp-all*)

**fix** w x

**assume** wx: v = (w ⊗ x)

**have** Can w ∧ Diag w **using** v wx *Diag-TensorE* [of w x] **by** auto

**moreover** **have** Can x ∧ Diag x **using** v wx *Diag-TensorE* [of w x] **by** auto

**moreover** **have** Dom t = Cod w **using** wx 1 **by** simp

**moreover** **have** Dom u = Cod x **using** wx 1 **by** simp

**ultimately** **show** Inv (t [·] w) = Inv w [·] Inv t ∧

$Inv (u [\cdot] x) = Inv x [\cdot] Inv u$

**using**  $t u I1 I2$  **by** *simp*

**qed**

**qed**

**thus** *?thesis* **using** *assms* **by** *blast*

**qed**

**lemma** *Can-and-Diag-implies-Ide*:

**assumes** *Can t* **and** *Diag t*

**shows** *Ide t*

**proof** –

**have**  $\llbracket Can\ t; Diag\ t \rrbracket \implies Ide\ t$

**apply** (*induct t, simp-all*)

**by** (*simp add: Diag-TensorE*)

**thus** *?thesis* **using** *assms* **by** *blast*

**qed**

**lemma** *CompDiag-Can-Inv [simp]*:

**assumes** *Can t* **and** *Diag t*

**shows**  $t [\cdot] Inv\ t = Cod\ t$

**using** *assms Can-and-Diag-implies-Ide Ide-in-Hom* **by** *simp*

**lemma** *CompDiag-Inv-Can [simp]*:

**assumes** *Can t* **and** *Diag t*

**shows**  $Inv\ t [\cdot] t = Dom\ t$

**using** *assms Can-and-Diag-implies-Ide Ide-in-Hom* **by** *simp*

The next fact is a syntactic version of the interchange law, for diagonal terms.

**lemma** *CompDiag-TensorDiag*:

**assumes** *Diag t* **and** *Diag u* **and** *Diag v* **and** *Diag w*

**and** *Seq t v* **and** *Seq u w*

**shows**  $(t [\otimes] u) [\cdot] (v [\otimes] w) = (t [\cdot] v) [\otimes] (u [\cdot] w)$

**proof** –

**have**  $\bigwedge u\ v\ w. \llbracket Diag\ t; Diag\ u; Diag\ v; Diag\ w; Seq\ t\ v; Seq\ u\ w \rrbracket \implies$

$(t [\otimes] u) [\cdot] (v [\otimes] w) = (t [\cdot] v) [\otimes] (u [\cdot] w)$

**proof** (*induct t, simp-all*)

**fix**  $u\ v\ w$

**assume**  $u: Diag\ u$

**assume**  $v: Diag\ v$

**assume**  $w: Diag\ w$

**assume**  $uw: Seq\ u\ w$

**show**  $Arr\ v \wedge \mathcal{I} = Cod\ v \implies u [\cdot] (v [\otimes] w) = v [\otimes] (u [\cdot] w)$

**using**  $u\ v\ w\ uw$  **by** (*cases v*) *simp-all*

**show**  $\bigwedge f. \llbracket C.arr\ f; Arr\ v \wedge \langle C.dom\ f \rangle = Cod\ v \rrbracket \implies$

$\langle f \rangle [\otimes] u [\cdot] (v [\otimes] w) = \langle f \rangle [\cdot] v [\otimes] (u [\cdot] w)$

**proof** –

**fix**  $f$

**assume**  $f: C.arr\ f$

**assume**  $1: Arr\ v \wedge \langle C.dom\ f \rangle = Cod\ v$



**show**  $(\langle f \rangle [\otimes] u) [\cdot] (v [\otimes] w) = (\langle f \rangle [\cdot] v) [\otimes] (u [\cdot] w)$   
**proof** –  
**have**  $2: v = \langle \text{un-Prim } v \rangle \wedge C.\text{arr } (\text{un-Prim } v)$  **using**  $v$  **1** **by**  $(\text{cases } v)$  *simp-all*  
**have**  $u = \mathcal{I} \implies ?thesis$   
**using**  $v$   $w$   $uw$   $1$   $2$  *Cod.simps(3)* *CompDiag-Cod-Diag* *Dom.simps(2)*  
*TensorDiag-Prim* *TensorDiag-term-Unity* *TensorDiag-preserves-Diag(3)*  
**by**  $(\text{cases } w)$  *simp-all*  
**moreover** **have**  $u \neq \mathcal{I} \implies ?thesis$   
**proof** –  
**assume**  $3: u \neq \mathcal{I}$   
**hence**  $4: w \neq \mathcal{I}$  **using**  $u$   $w$   $uw$  **by**  $(\text{cases } u, \text{simp-all}; \text{cases } w, \text{simp-all})$   
**have**  $(\langle f \rangle [\otimes] u) [\cdot] (v [\otimes] w) = (\langle f \rangle \otimes u) [\cdot] (v \otimes w)$   
**proof** –  
**have**  $\langle f \rangle [\otimes] u = \langle f \rangle \otimes u$   
**using**  $u$   $f$   $3$  *TensorDiag-Diag* **by**  $(\text{cases } u)$  *simp-all*  
**moreover** **have**  $v [\otimes] w = v \otimes w$   
**using**  $w$   $2$   $4$  *TensorDiag-Diag* **by**  $(\text{cases } v, \text{simp-all}; \text{cases } w, \text{simp-all})$   
**ultimately** **show**  $?thesis$  **by** *simp*  
**qed**  
**also** **have**  $5: \dots = (\langle f \rangle [\cdot] v) \otimes (u [\cdot] w)$  **by** *simp*  
**also** **have**  $\dots = (\langle f \rangle [\cdot] v) [\otimes] (u [\cdot] w)$   
**using**  $f$   $u$   $w$   $uw$   $1$   $2$   $3$   $4$   $5$   
*TensorDiag-Diag* *TensorDiag-Prim* *TensorDiag-preserves-Diag(1)*  
*CompDiag-preserves-Diag(1)*  
**by**  $(\text{metis } \text{Cod.simps(3)} \text{ Dom.simps(1)} \text{ Dom.simps(3)} \text{ Diag.simps(2)})$   
**finally** **show**  $?thesis$  **by** *blast*  
**qed**  
**ultimately** **show**  $?thesis$  **by** *blast*  
**qed**  
**qed**  
**fix**  $t1$   $t2$   
**assume**  $I2: \bigwedge u v w. \llbracket \text{Diag } t2; \text{Diag } u; \text{Diag } v; \text{Diag } w; \text{Arr } v \wedge \text{Dom } t2 = \text{Cod } v; \text{Seq } u w \rrbracket \implies$   
 $(t2 [\otimes] u) [\cdot] (v [\otimes] w) = (t2 [\cdot] v) [\otimes] (u [\cdot] w)$   
**assume**  $t12: \text{Diag } (t1 \otimes t2)$   
**have**  $t1: t1 = \langle \text{un-Prim } t1 \rangle \wedge C.\text{arr } (\text{un-Prim } t1) \wedge \text{Diag } t1$   
**using**  $t12$  **by**  $(\text{cases } t1)$  *simp-all*  
**have**  $t2: \text{Diag } t2 \wedge t2 \neq \mathcal{I}$   
**using**  $t12$  **by**  $(\text{cases } t1)$  *simp-all*  
**assume**  $1: \text{Arr } t1 \wedge \text{Arr } t2 \wedge \text{Arr } v \wedge \text{Dom } t1 \otimes \text{Dom } t2 = \text{Cod } v$   
**show**  $((t1 \otimes t2) [\otimes] u) [\cdot] (v [\otimes] w) = ((t1 \otimes t2) [\cdot] v) [\otimes] (u [\cdot] w)$   
**proof** –  
**have**  $u = \mathcal{I} \implies ?thesis$   
**using**  $w$   $uw$  *TensorDiag-term-Unity* *CompDiag-Cod-Diag* **by**  $(\text{cases } w)$  *simp-all*  
**moreover** **have**  $u \neq \mathcal{I} \implies ?thesis$   
**proof** –  
**assume**  $u': u \neq \mathcal{I}$   
**hence**  $w': w \neq \mathcal{I}$  **using**  $u$   $w$   $uw$  **by**  $(\text{cases } u; \text{simp}; \text{cases } w; \text{simp})$   
**show**  $?thesis$

```

using 1 v
proof (cases v, simp-all)
fix v1 v2
assume v12: v = Tensor v1 v2
have v1: v1 = ⟨un-Prim v1⟩ ∧ C.arr (un-Prim v1) ∧ Diag v1
  using v v12 by (cases v1) simp-all
have v2: Diag v2 ∧ v2 ≠  $\mathcal{I}$ 
  using v v12 by (cases v1) simp-all
have 2: v = (⟨un-Prim v1⟩ ⊗ v2)
  using v1 v12 by simp
show ((t1 ⊗ t2) [⊗] u) [·] ((v1 ⊗ v2) [⊗] w)
  = ((t1 [·] v1) ⊗ (t2 [·] v2)) [⊗] (u [·] w)
proof -
have 3: (t1 ⊗ t2) [⊗] u = t1 [⊗] (t2 [⊗] u)
  using u u' by (cases u) simp-all
have 4: v [⊗] w = v1 [⊗] (v2 [⊗] w)
  using v w v1 v2 2 TensorDiag-assoc TensorDiag-Diag by metis
have ((t1 ⊗ t2) [⊗] u) [·] ((v1 ⊗ v2) [⊗] w)
  = (t1 [⊗] (t2 [⊗] u)) [·] (v1 [⊗] (v2 [⊗] w))
  using 3 4 v12 by simp
also have ... = (t1 [·] v1) [⊗] ((t2 [⊗] u) [·] (v2 [⊗] w))
proof -
have is-Tensor (t2 [⊗] u)
  using t2 u u' not-is-Tensor-TensorDiagE by auto
moreover have is-Tensor (v2 [⊗] w)
  using v2 v12 w w' not-is-Tensor-TensorDiagE by auto
ultimately show ?thesis
  using u u' v w t1 v1 t12 v12 TensorDiag-Prim not-is-Tensor-Unity
  by (metis (no-types, lifting) CompDiag.simps(2) CompDiag.simps(3)
    is-Tensor-def)
qed
also have ... = (t1 [·] v1) [⊗] (t2 [·] v2) [⊗] (u [·] w)
  using u w uw t2 v2 1 2 Diag-implies-Arr I2 by fastforce
also have ... = ((t1 [·] v1) ⊗ (t2 [·] v2)) [⊗] (u [·] w)
proof -
have u [·] w ≠ Unity
proof -
have Arr v1 ∧ ⟨C.dom (un-Prim t1)⟩ = Cod v1
  using t1 v1 1 2 by (cases t1, auto)
thus ?thesis
  using t1 t2 v1 v2 u w uw u' CompDiag-preserves-Diag
    TensorDiag-preserves-Diag TensorDiag-Prim
  by (metis (mono-tags, lifting) Cod.simps(2) Cod.simps(3)
    TensorDiag.simps(2) term.distinct(3))
qed
hence ((t1 [·] v1) ⊗ (t2 [·] v2)) [⊗] (u [·] w)
  = (t1 [·] v1) [⊗] ((t2 [·] v2) [⊗] (u [·] w))
  by (cases u [·] w) simp-all
thus ?thesis by argo

```

```

      qed
    finally show ?thesis by blast
  qed
  qed
  qed
  ultimately show ?thesis by blast
  qed
  qed
  thus ?thesis using assms by blast
  qed

```

The following function reduces an arrow to diagonal form. The precise relationship between a term and its diagonalization is developed below.

```

fun Diagonalize :: 'a term  $\Rightarrow$  'a term  ([_])
where [⟨f⟩] = ⟨f⟩
  | [I] = I
  | [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** *Diag-Diagonalize*:

**assumes** *Arr t*

**shows** *Diag [t]* **and** *Dom [t] = [Dom t]* **and** *Cod [t] = [Cod t]*

**proof** –

**have** *0*: *Arr t*  $\implies$  *Diag [t] ∧ Dom [t] = [Dom t] ∧ Cod [t] = [Cod t]*

**using** *TensorDiag-preserves-Diag CompDiag-preserves-Diag TensorDiag-assoc*

**apply** (*induct t*)

**apply** *auto*

**apply** (*metis (full-types)*)

**by** (*metis (full-types)*)

**show** *Diag [t]* **using** *assms 0* **by** *blast*

**show** *Dom [t] = [Dom t]* **using** *assms 0* **by** *blast*

**show** *Cod [t] = [Cod t]* **using** *assms 0* **by** *blast*

**qed**

**lemma** *Diagonalize-in-Hom*:

**assumes** *Arr t*

**shows**  $[t] \in \text{Hom } [Dom\ t] [Cod\ t]$

**using** *assms Diag-Diagonalize Diag-implies-Arr* **by** *blast*

**lemma** *Diagonalize-Dom*:

**assumes** *Arr t*

**shows**  $[Dom\ t] = Dom\ [t]$

**using** *assms Diagonalize-in-Hom* **by** *simp*

**lemma** *Diagonalize-Cod*:  
**assumes** *Arr t*  
**shows**  $\lfloor \text{Cod } t \rfloor = \text{Cod } \lfloor t \rfloor$   
**using** *assms Diagonalize-in-Hom* **by** *simp*

**lemma** *Diagonalize-preserves-Ide*:  
**assumes** *Ide a*  
**shows** *Ide*  $\lfloor a \rfloor$   
**proof** –  
  **have** *Ide a*  $\implies$  *Ide*  $\lfloor a \rfloor$   
  **using** *Ide-implies-Arr TensorDiag-preserves-Ide Diag-Diagonalize*  
  **by** (*induct a*) *simp-all*  
  **thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

The diagonalizations of canonical arrows are identities.

**lemma** *Ide-Diagonalize-Can*:  
**assumes** *Can t*  
**shows** *Ide*  $\lfloor t \rfloor$   
**proof** –  
  **have** *Can t*  $\implies$  *Ide*  $\lfloor t \rfloor$   
  **using** *Can-implies-Arr TensorDiag-preserves-Ide Diag-Diagonalize CompDiag-preserves-Ide*  
  *TensorDiag-preserves-Diag*  
  **by** (*induct t*) *auto*  
  **thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**lemma** *Diagonalize-preserves-Can*:  
**assumes** *Can t*  
**shows** *Can*  $\lfloor t \rfloor$   
**using** *assms Ide-Diagonalize-Can Ide-implies-Can* **by** *auto*

**lemma** *Diagonalize-Diag [simp]*:  
**assumes** *Diag t*  
**shows**  $\lfloor t \rfloor = t$   
**proof** –  
  **have** *Diag t*  $\implies$   $\lfloor t \rfloor = t$   
  **apply** (*induct t, simp-all*)  
  **using** *TensorDiag-Prim Diag-TensorE* **by** *metis*  
  **thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**lemma** *Diagonalize-Diagonalize [simp]*:  
**assumes** *Arr t*  
**shows**  $\lfloor \lfloor t \rfloor \rfloor = \lfloor t \rfloor$   
**using** *assms Diag-Diagonalize Diagonalize-Diag* **by** *blast*

**lemma** *Diagonalize-Tensor*:

**assumes**  $Arr\ t$  **and**  $Arr\ u$   
**shows**  $\lfloor t \otimes u \rfloor = \lfloor \lfloor t \rfloor \otimes \lfloor u \rfloor \rfloor$   
**using** *assms Diagonalize-Diagonalize* **by** *simp*

**lemma** *Diagonalize-Tensor-Unity-Arr* [*simp*]:  
**assumes**  $Arr\ u$   
**shows**  $\lfloor \mathcal{I} \otimes u \rfloor = \lfloor u \rfloor$   
**using** *assms* **by** *simp*

**lemma** *Diagonalize-Tensor-Arr-Unity* [*simp*]:  
**assumes**  $Arr\ t$   
**shows**  $\lfloor t \otimes \mathcal{I} \rfloor = \lfloor t \rfloor$   
**using** *assms* **by** *simp*

**lemma** *Diagonalize-Tensor-Prim-Arr* [*simp*]:  
**assumes**  $arr\ f$  **and**  $Arr\ u$  **and**  $\lfloor u \rfloor \neq Unity$   
**shows**  $\lfloor \langle f \rangle \otimes u \rfloor = \langle f \rangle \otimes \lfloor u \rfloor$   
**using** *assms* **by** *simp*

**lemma** *Diagonalize-Tensor-Tensor*:  
**assumes**  $Arr\ t$  **and**  $Arr\ u$  **and**  $Arr\ v$   
**shows**  $\lfloor (t \otimes u) \otimes v \rfloor = \lfloor \lfloor t \rfloor \otimes (\lfloor u \rfloor \otimes \lfloor v \rfloor) \rfloor$   
**using** *assms Diag-Diagonalize Diagonalize-Diagonalize* **by** (*simp add: TensorDiag-assoc*)

**lemma** *Diagonalize-Comp-Cod-Arr*:  
**assumes**  $Arr\ t$   
**shows**  $\lfloor Cod\ t \cdot t \rfloor = \lfloor t \rfloor$   
**proof** –  
**have**  $Arr\ t \implies \lfloor Cod\ t \cdot t \rfloor = \lfloor t \rfloor$   
**using** *C.comp-cod-arr*  
**apply** (*induct t, simp-all*)  
**using** *CompDiag-TensorDiag Arr-implies-Ide-Cod Ide-in-Hom Diag-Diagonalize Diagonalize-in-Hom*  
**apply** *simp*  
**using** *CompDiag-preserves-Diag CompDiag-Cod-Diag Diag-Diagonalize*  
**apply** *metis*  
**using** *CompDiag-TensorDiag Arr-implies-Ide-Cod Ide-in-Hom TensorDiag-in-Hom TensorDiag-preserves-Diag Diag-Diagonalize Diagonalize-in-Hom TensorDiag-assoc*  
**by** *simp-all*  
**thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**lemma** *Diagonalize-Comp-Arr-Dom*:  
**assumes**  $Arr\ t$   
**shows**  $\lfloor t \cdot Dom\ t \rfloor = \lfloor t \rfloor$   
**proof** –  
**have**  $Arr\ t \implies \lfloor t \cdot Dom\ t \rfloor = \lfloor t \rfloor$   
**by** (*metis CompDiag-Diag-Dom Diag-Diagonalize(1-2) Diagonalize.simps(4)*)  
**thus** *?thesis* **using** *assms* **by** *blast*

qed

**lemma** *Diagonalize-Inv*:

**assumes** *Can t*

**shows**  $[Inv\ t] = Inv\ [t]$

**proof** –

**have**  $Can\ t \implies [Inv\ t] = Inv\ [t]$

**proof** (*induct t, simp-all*)

**fix**  $u\ v$

**assume**  $I1: [Inv\ u] = Inv\ [u]$

**assume**  $I2: [Inv\ v] = Inv\ [v]$

**show**  $Can\ u \wedge Can\ v \implies Inv\ [u] [\otimes] Inv\ [v] = Inv\ ([u] [\otimes] [v])$

**using** *Inv-TensorDiag Diag-Diagonalize Can-implies-Arr Diagonalize-preserves-Can I1 I2*

**by** *simp*

**show**  $Can\ u \wedge Can\ v \wedge Dom\ u = Cod\ v \implies Inv\ [v] [\cdot] Inv\ [u] = Inv\ ([u] [\cdot] [v])$

**using** *Inv-CompDiag Diag-Diagonalize Can-implies-Arr Diagonalize-in-Hom Diagonalize-preserves-Can I1 I2*

**by** *simp*

**fix**  $w$

**assume**  $I3: [Inv\ w] = Inv\ [w]$

**assume**  $uvw: Can\ u \wedge Can\ v \wedge Can\ w$

**show**  $Inv\ [u] [\otimes] (Inv\ [v] [\otimes] Inv\ [w]) = Inv\ (([u] [\otimes] [v]) [\otimes] [w])$

**using**  $uvw\ I1\ I2\ I3$

*Inv-TensorDiag Diag-Diagonalize Can-implies-Arr Diagonalize-preserves-Can TensorDiag-preserves-Diag TensorDiag-preserves-Can TensorDiag-assoc*

**by** *simp*

**show**  $(Inv\ [u] [\otimes] Inv\ [v]) [\otimes] Inv\ [w] = Inv\ ([u] [\otimes] ([v] [\otimes] [w]))$

**by** (*simp add: Can-implies-Arr Ide-Diagonalize-Can TensorDiag-assoc*

*TensorDiag-preserves-Diag(1) TensorDiag-preserves-Ide Diag-Diagonalize(1) uvw*)

qed

**thus** *?thesis using assms by blast*

qed

Our next objective is to begin making the connection, to be completed in a subsequent section, between arrows and their diagonalizations. To summarize, an arrow  $t$  and its diagonalization  $[t]$  are opposite sides of a square whose other sides are certain canonical terms  $Dom\ t \downarrow \in Hom\ (Dom\ t)\ [Dom\ t]$  and  $Cod\ t \downarrow \in Hom\ (Cod\ t)\ [Cod\ t]$ , where  $Dom\ t \downarrow$  and  $Cod\ t \downarrow$  are defined by the function *red* below. The coherence theorem amounts to the statement that every such square commutes when the formal terms involved are evaluated in the evident way in any monoidal category.

Function *red* defined below takes an identity term  $a$  to a canonical arrow  $a \downarrow \in Hom\ a\ [a]$ . The auxiliary function *red2* takes a pair  $(a, b)$  of diagonal identity terms and produces a canonical arrow  $a \Downarrow b \in Hom\ (a \otimes b)\ [a \otimes b]$ . The canonical arrow  $a \downarrow$  amounts to a “parallel innermost reduction” from  $a$  to  $[a]$ , where the reduction steps are canonical arrows that involve the unitors and associator only in their uninverted forms. In general, a parallel innermost reduction from  $a$  will not be unique: at some points there is a choice available between left and right unitors and at other points there are

choices between unitors and associators. These choices are inessential, and the ordering of the clauses in the function definitions below resolves them in an arbitrary way. What is more important is having chosen an innermost reduction, which is what allows us to write these definitions in structurally recursive form.

The essence of coherence is that the axioms for a monoidal category allow us to prove that any reduction from  $a$  to  $\llbracket a \rrbracket$  is equivalent (under evaluation of terms) to a parallel innermost reduction. The problematic cases are terms of the form  $((a \otimes b) \otimes c) \otimes d$ , which present a choice between an inner and outer reduction that lead to terms with different structures. It is of course the pentagon axiom that ensures the confluence (under evaluation) of the two resulting paths.

Although simple in appearance, the structurally recursive definitions below were difficult to get right even after I started to understand what I was doing. I wish I could have just written them down straightaway. If so, then I could have avoided laboriously constructing and then throwing away thousands of lines of proof text that used a non-structural, “operational” approach to defining a reduction from  $a$  to  $\llbracket a \rrbracket$ .

```

fun red2                                (infixr  $\Downarrow$  53)
where  $\mathcal{I} \Downarrow a = \mathbf{l}[a]$ 
      |  $\langle f \rangle \Downarrow \mathcal{I} = \mathbf{r}[\langle f \rangle]$ 
      |  $\langle f \rangle \Downarrow a = \langle f \rangle \otimes a$ 
      |  $(a \otimes b) \Downarrow \mathcal{I} = \mathbf{r}[a \otimes b]$ 
      |  $(a \otimes b) \Downarrow c = (a \Downarrow \llbracket b \otimes c \rrbracket) \cdot (a \otimes (b \Downarrow c)) \cdot \mathbf{a}[a, b, c]$ 
      |  $a \Downarrow b = \text{undefined}$ 

```

```

fun red                                  ( $\Downarrow$  [56] 56)
where  $\mathcal{I} \Downarrow = \mathcal{I}$ 
      |  $\langle f \rangle \Downarrow = \langle f \rangle$ 
      |  $(a \otimes b) \Downarrow = (\text{if } \text{Diag } (a \otimes b) \text{ then } a \otimes b \text{ else } (\llbracket a \rrbracket \Downarrow \llbracket b \rrbracket) \cdot (a \Downarrow \otimes b \Downarrow))$ 
      |  $a \Downarrow = \text{undefined}$ 

```

```

lemma red-Diag [simp]:
assumes Diag a
shows  $a \Downarrow = a$ 
using assms by (cases a) simp-all

```

```

lemma red2-Diag:
assumes Diag (a  $\otimes$  b)
shows  $a \Downarrow b = a \otimes b$ 
proof –
  have a:  $a = \langle \text{un-Prim } a \rangle$ 
    using assms Diag-TensorE by metis
  have b: Diag b  $\wedge$   $b \neq \mathcal{I}$ 
    using assms Diag-TensorE by metis
  show ?thesis using a b
  apply (cases b)
  apply simp-all
  apply (metis red2.simps(3))
  by (metis red2.simps(4))

```

qed

**lemma** *Can-red2*:

**assumes** *Ide a* **and** *Diag a* **and** *Ide b* **and** *Diag b*

**shows**  $\text{Can } (a \Downarrow b)$

**and**  $a \Downarrow b \in \text{Hom } (a \otimes b) \llbracket a \otimes b \rrbracket$

**proof** –

**have**  $0: \bigwedge b. \llbracket \text{Ide } a \wedge \text{Diag } a; \text{Ide } b \wedge \text{Diag } b \rrbracket \implies$   
 $\text{Can } (a \Downarrow b) \wedge a \Downarrow b \in \text{Hom } (a \otimes b) \llbracket a \otimes b \rrbracket$

**proof** (*induct a, simp-all*)

**fix** *b*

**show**  $\text{Ide } b \wedge \text{Diag } b \implies \text{Can } b \wedge \text{Dom } b = b \wedge \text{Cod } b = b$

**using** *Ide-implies-Can Ide-in-Hom Diagonalize-Diag* **by** *auto*

**fix** *f*

**show**  $\llbracket C.\text{ide } f \wedge C.\text{arr } f; \text{Ide } b \wedge \text{Diag } b \rrbracket \implies$

$\text{Can } (\langle f \rangle \Downarrow b) \wedge \text{Arr } (\langle f \rangle \Downarrow b) \wedge \text{Dom } (\langle f \rangle \Downarrow b) = \langle f \rangle \otimes b \wedge$   
 $\text{Cod } (\langle f \rangle \Downarrow b) = \langle f \rangle \llbracket \otimes \rrbracket b$

**using** *Ide-implies-Can Ide-in-Hom* **by** (*cases b; auto*)

**next**

**fix** *a b c*

**assume** *ab*:  $\text{Ide } a \wedge \text{Ide } b \wedge \text{Diag } (\text{Tensor } a \ b)$

**assume** *c*:  $\text{Ide } c \wedge \text{Diag } c$

**assume** *I1*:  $\bigwedge c. \llbracket \text{Diag } a; \text{Ide } c \wedge \text{Diag } c \rrbracket \implies$

$\text{Can } (a \Downarrow c) \wedge \text{Arr } (a \Downarrow c) \wedge \text{Dom } (a \Downarrow c) = a \otimes c \wedge$   
 $\text{Cod } (a \Downarrow c) = a \llbracket \otimes \rrbracket c$

**assume** *I2*:  $\bigwedge c. \llbracket \text{Diag } b; \text{Ide } c \wedge \text{Diag } c \rrbracket \implies$

$\text{Can } (b \Downarrow c) \wedge \text{Arr } (b \Downarrow c) \wedge \text{Dom } (b \Downarrow c) = b \otimes c \wedge$   
 $\text{Cod } (b \Downarrow c) = b \llbracket \otimes \rrbracket c$

**show**  $\text{Can } ((a \otimes b) \Downarrow c) \wedge \text{Arr } ((a \otimes b) \Downarrow c) \wedge$

$\text{Dom } ((a \otimes b) \Downarrow c) = (a \otimes b) \otimes c \wedge$

$\text{Cod } ((a \otimes b) \Downarrow c) = (\llbracket a \rrbracket \llbracket \otimes \rrbracket \llbracket b \rrbracket) \llbracket \otimes \rrbracket c$

**proof** –

**have** *a*:  $\text{Diag } a \wedge \text{Ide } a$

**using** *ab Diag-TensorE* **by** *blast*

**have** *b*:  $\text{Diag } b \wedge \text{Ide } b$

**using** *ab Diag-TensorE* **by** *blast*

**have** *c* =  $\mathcal{I} \implies ?thesis$

**proof** –

**assume** *1*:  $c = \mathcal{I}$

**have** *2*:  $(a \otimes b) \Downarrow c = \mathbf{r}[a \otimes b]$

**using** *1* **by** *simp*

**have** *3*:  $\text{Can } (a \Downarrow b) \wedge \text{Arr } (a \Downarrow b) \wedge \text{Dom } (a \Downarrow b) = a \otimes b \wedge \text{Cod } (a \Downarrow b) = a \otimes b$

**using** *a b ab 1 2 I1 Diagonalize-Diag Diagonalize.simps(3)* **by** *metis*

**hence** *4*:  $\text{Seq } (a \Downarrow b) \mathbf{r}[a \otimes b]$

**using** *ab*

**by** (*metis (mono-tags, lifting) Arr.simps(7) Cod.simps(3) Cod.simps(7)*)

*Diag-implies-Arr Ide-in-Hom mem-Collect-eq*)

**have**  $\text{Can } ((a \otimes b) \Downarrow c)$

**using** *1 2 3 4 ab* **by** (*simp add: Ide-implies-Can*)



**moreover have**  $Dom ((a \otimes b) \Downarrow c) = (a \otimes b) \otimes c$   
**using** *1 2 3 4 a b ab I1 Ide-in-Hom TensorDiag-preserves-Diag* **by** *simp*  
**moreover have**  $Cod ((a \otimes b) \Downarrow c) = \lfloor (a \otimes b) \otimes c \rfloor$   
**using** *1 2 3 4 ab* **using** *Diagonalize-Diag* **by** *fastforce*  
**ultimately show** *?thesis* **using** *Can-implies-Arr* **by** (*simp add: 1 ab*)  
**qed**  
**moreover have**  $c \neq \mathcal{I} \implies ?thesis$   
**proof** –  
**assume**  $1: c \neq \mathcal{I}$   
**have**  $2: (a \otimes b) \Downarrow c = (a \Downarrow \lfloor b \otimes c \rfloor) \cdot (a \otimes b \Downarrow c) \cdot \mathbf{a}[a, b, c]$   
**using** *1 a b ab c* **by** (*cases c; simp*)  
**have**  $3: Can (a \Downarrow \lfloor b \otimes c \rfloor) \wedge Dom (a \Downarrow \lfloor b \otimes c \rfloor) = a \otimes \lfloor b \otimes c \rfloor \wedge$   
 $Cod (a \Downarrow \lfloor b \otimes c \rfloor) = \lfloor a \otimes (b \otimes c) \rfloor$   
**proof** –  
**have**  $Can (a \Downarrow \lfloor b \otimes c \rfloor) \wedge Dom (a \Downarrow \lfloor b \otimes c \rfloor) = a \otimes \lfloor b \otimes c \rfloor \wedge$   
 $Cod (a \Downarrow \lfloor b \otimes c \rfloor) = \lfloor a \otimes \lfloor b \otimes c \rfloor \rfloor$   
**using** *a c ab 1 2 I1 Diag-implies-Arr Diag-Diagonalize(1)*  
*Diagonalize-preserves-Ide TensorDiag-preserves-Ide*  
*TensorDiag-preserves-Diag(1)*  
**by** *auto*  
**moreover have**  $\lfloor a \otimes \lfloor b \otimes c \rfloor \rfloor = \lfloor a \otimes (b \otimes c) \rfloor$   
**using** *ab c Diagonalize-Tensor Diagonalize-Diagonalize Diag-implies-Arr*  
**by** (*metis Arr.simps(3) Diagonalize.simps(3)*)  
**ultimately show** *?thesis* **by** *metis*  
**qed**  
**have**  $4: Can (b \Downarrow c) \wedge Dom (b \Downarrow c) = b \otimes c \wedge Cod (b \Downarrow c) = \lfloor b \otimes c \rfloor$   
**using** *b c ab 1 2 I2* **by** *simp*  
**hence**  $Can (a \otimes (b \Downarrow c)) \wedge Dom (a \otimes (b \Downarrow c)) = a \otimes (b \otimes c) \wedge$   
 $Cod (a \otimes (b \Downarrow c)) = a \otimes \lfloor b \otimes c \rfloor$   
**using** *ab Ide-implies-Can Ide-in-Hom* **by** *force*  
**moreover have**  $\lfloor a \otimes \lfloor b \otimes c \rfloor \rfloor = \lfloor a \otimes b \rfloor \lfloor \otimes \rfloor \lfloor c \rfloor$   
**proof** –  
**have**  $\lfloor a \otimes \lfloor b \otimes c \rfloor \rfloor = a \lfloor \otimes \rfloor (b \lfloor \otimes \rfloor c)$   
**using** *a b c 4*  
**by** (*metis Arr-implies-Ide-Dom Can-implies-Arr Ide-implies-Arr*  
*Diag-Diagonalize(1) Diagonalize.simps(3) Diagonalize-Diag*)  
**also have**  $\dots = (a \lfloor \otimes \rfloor b) \lfloor \otimes \rfloor c$   
**using** *a b ab c TensorDiag-assoc* **by** *metis*  
**also have**  $\dots = \lfloor a \otimes b \rfloor \lfloor \otimes \rfloor \lfloor c \rfloor$   
**using** *a b c* **by** (*metis Diagonalize.simps(3) Diagonalize-Diag*)  
**finally show** *?thesis* **by** *blast*  
**qed**  
**moreover have**  $Can \mathbf{a}[a, b, c] \wedge Dom \mathbf{a}[a, b, c] = (a \otimes b) \otimes c \wedge$   
 $Cod \mathbf{a}[a, b, c] = a \otimes (b \otimes c)$   
**using** *ab c Ide-implies-Can Ide-in-Hom* **by** *auto*  
**ultimately show** *?thesis*  
**using** *c 2 3 4 Diagonalize-Diagonalize Ide-implies-Can*  
*Diagonalize-Diag Arr-implies-Ide-Dom Can-implies-Arr*  
**by** (*metis Can.simps(4) Cod.simps(4) Dom.simps(4) Diagonalize.simps(3)*)

**qed**  
**ultimately show ?thesis by blast**  
**qed**  
**qed**  
**show**  $\text{Can } (a \Downarrow b)$  **using**  $\text{assms } 0$  **by blast**  
**show**  $a \Downarrow b \in \text{Hom } (a \otimes b) [a \otimes b]$  **using**  $0 \text{ assms by blast}$   
**qed**

**lemma** *red2-in-Hom*:  
**assumes**  $\text{Ide } a$  **and**  $\text{Diag } a$  **and**  $\text{Ide } b$  **and**  $\text{Diag } b$   
**shows**  $a \Downarrow b \in \text{Hom } (a \otimes b) [a \otimes b]$   
**using**  $\text{assms } \text{Can-red2 } \text{Can-implies-Arr}$  **by simp**

**lemma** *Can-red*:  
**assumes**  $\text{Ide } a$   
**shows**  $\text{Can } (a \Downarrow)$  **and**  $a \Downarrow \in \text{Hom } a [a]$   
**proof** –  
**have**  $0: \text{Ide } a \implies \text{Can } (a \Downarrow) \wedge a \Downarrow \in \text{Hom } a [a]$   
**proof** (*induct a, simp-all*)  
**fix**  $b \ c$   
**assume**  $b: \text{Can } (b \Downarrow) \wedge \text{Arr } (b \Downarrow) \wedge \text{Dom } (b \Downarrow) = b \wedge \text{Cod } (b \Downarrow) = [b]$   
**assume**  $c: \text{Can } (c \Downarrow) \wedge \text{Arr } (c \Downarrow) \wedge \text{Dom } (c \Downarrow) = c \wedge \text{Cod } (c \Downarrow) = [c]$   
**assume**  $bc: \text{Ide } b \wedge \text{Ide } c$   
**show**  $(\text{Diag } (b \otimes c)) \longrightarrow$   
 $\text{Can } b \wedge \text{Can } c \wedge \text{Dom } b = b \wedge \text{Dom } c = c \wedge \text{Cod } b \otimes \text{Cod } c = [b] [ \otimes ] [c] \wedge$   
 $(\neg \text{Diag } (b \otimes c)) \longrightarrow$   
 $\text{Can } ([b] \Downarrow [c]) \wedge \text{Dom } ([b] \Downarrow [c]) = [b] \otimes [c] \wedge \text{Arr } ([b] \Downarrow [c]) \wedge$   
 $\text{Dom } ([b] \Downarrow [c]) = [b] \otimes [c] \wedge \text{Cod } ([b] \Downarrow [c]) = [b] [ \otimes ] [c]$   
**proof**  
**show**  $\text{Diag } (b \otimes c) \longrightarrow$   
 $\text{Can } b \wedge \text{Can } c \wedge \text{Dom } b = b \wedge \text{Dom } c = c \wedge \text{Cod } b \otimes \text{Cod } c = [b] [ \otimes ] [c]$   
**using**  $bc \ \text{Diag-TensorE } \text{Ide-implies-Can } \text{Inv-preserves-Can}(2)$   
 $\text{CompDiag-Ide-Diag } \text{Inv-Ide } \text{Diagonalize.simps}(3) \ \text{Diagonalize-Diag}$   
**by** (*metis CompDiag-Inv-Can*)  
**show**  $\neg \text{Diag } (b \otimes c) \longrightarrow$   
 $\text{Can } ([b] \Downarrow [c]) \wedge \text{Dom } ([b] \Downarrow [c]) = [b] \otimes [c] \wedge \text{Arr } ([b] \Downarrow [c]) \wedge$   
 $\text{Dom } ([b] \Downarrow [c]) = [b] \otimes [c] \wedge \text{Cod } ([b] \Downarrow [c]) = [b] [ \otimes ] [c]$   
**using**  $b \ c \ bc \ \text{Ide-in-Hom } \text{Ide-implies-Can } \text{Can-red2 } \text{Diag-Diagonalize}$   
 $\text{Diagonalize-preserves-Ide } \text{TensorDiag-preserves-Diag } \text{TensorDiag-preserves-Ide}$   
**by force**  
**qed**  
**qed**  
**show**  $\text{Can } (a \Downarrow)$  **using**  $\text{assms } 0$  **by blast**  
**show**  $a \Downarrow \in \text{Hom } a [a]$  **using**  $\text{assms } 0$  **by blast**  
**qed**

**lemma** *red-in-Hom*:  
**assumes**  $\text{Ide } a$   
**shows**  $a \Downarrow \in \text{Hom } a [a]$

**using** *assms Can-red Can-implies-Arr* **by** *simp*

**lemma** *Diagonalize-red* [*simp*]:

**assumes** *Ide a*

**shows**  $\lfloor a \Downarrow \rfloor = \lfloor a \rfloor$

**using** *assms Can-red Ide-Diagonalize-Can Diagonalize-in-Hom Ide-in-Hom* **by** *fastforce*

**lemma** *Diagonalize-red2* [*simp*]:

**assumes** *Ide a* **and** *Ide b* **and** *Diag a* **and** *Diag b*

**shows**  $\lfloor a \Downarrow b \rfloor = \lfloor a \otimes b \rfloor$

**using** *assms Can-red2 Ide-Diagonalize-Can Diagonalize-in-Hom [of a \Downarrow b]*  
*red2-in-Hom Ide-in-Hom*

**by** *simp*

**end**

## 2.6 Coherence

If  $D$  is a monoidal category, then a functor  $V: C \rightarrow D$  extends in an evident way to an evaluation map that interprets each formal arrow of the monoidal language of  $C$  as an arrow of  $D$ .

**locale** *evaluation-map* =

*monoidal-language C* +

*monoidal-category D T  $\alpha$   $\iota$*  +

*V: functor C D V*

**for**  $C :: 'c \text{ comp}$  (**infixr**  $\cdot_C$  55)

**and**  $D :: 'd \text{ comp}$  (**infixr**  $\cdot$  55)

**and**  $T :: 'd * 'd \Rightarrow 'd$

**and**  $\alpha :: 'd * 'd * 'd \Rightarrow 'd$

**and**  $\iota :: 'd$

**and**  $V :: 'c \Rightarrow 'd$

**begin**

**no-notation** *C.in-hom* ( $\ll - : - \rightarrow - \gg$ )

**notation** *unity* ( $\mathcal{I}$ )

**notation** *runit* ( $\mathbf{r}[-]$ )

**notation** *lunit* ( $\mathbf{l}[-]$ )

**notation** *assoc'* ( $\mathbf{a}^{-1}[-, -, -]$ )

**notation** *runit'* ( $\mathbf{r}^{-1}[-]$ )

**notation** *lunit'* ( $\mathbf{l}^{-1}[-]$ )

**primrec** *eval* ::  $'c \text{ term} \Rightarrow 'd$  ( $\{\!\{-\}\!\}$ )

**where**  $\{\!\{f\}\!\} = V f$

|  $\{\!\{\mathcal{I}\}\!\} = \mathcal{I}$

|  $\{\!\{t \otimes u\}\!\} = \{\!\{t\}\!\} \otimes \{\!\{u\}\!\}$

|  $\{\!\{t \cdot u\}\!\} = \{\!\{t\}\!\} \cdot \{\!\{u\}\!\}$

|  $\{\!\{\mathbf{l}[t]\}\!\} = \iota \{\!\{t\}\!\}$

```

|  $\{\mathbf{1}^{-1}[t]\} = \mathbf{1}' \{t\}$ 
|  $\{\mathbf{r}[t]\} = \varrho \{t\}$ 
|  $\{\mathbf{r}^{-1}[t]\} = \varrho' \{t\}$ 
|  $\{\mathbf{a}[t, u, v]\} = \alpha (\{t\}, \{u\}, \{v\})$ 
|  $\{\mathbf{a}^{-1}[t, u, v]\} = \alpha' (\{t\}, \{u\}, \{v\})$ 

```

Identity terms evaluate to identities of  $D$  and evaluation preserves domain and codomain.

**lemma** *ide-eval-Ide* [*simp*]:

**shows**  $Ide\ t \implies ide\ \{t\}$

**by** (*induct t, auto*)

**lemma** *eval-in-hom*:

**shows**  $Arr\ t \implies \langle \{t\} : \{Dom\ t\} \rightarrow \{Cod\ t\} \rangle$

**apply** (*induct t*)

**apply** *auto*[4]

**apply** *fastforce*

**proof** –

**fix**  $t\ u\ v$

**assume**  $I$ :  $Arr\ t \implies \langle \{t\} : \{Dom\ t\} \rightarrow \{Cod\ t\} \rangle$

**show**  $Arr\ \mathbf{1}^{-1}[t] \implies \langle \{\mathbf{1}^{-1}[t]\} : \{Dom\ \mathbf{1}^{-1}[t]\} \rightarrow \{Cod\ \mathbf{1}^{-1}[t]\} \rangle$

**using**  $I\ arr\text{-}dom\text{-}iff\text{-}arr$  [*of*  $\{t\}$ ] **by** *force*

**show**  $Arr\ \mathbf{r}[t] \implies \langle \{\mathbf{r}[t]\} : \{Dom\ \mathbf{r}[t]\} \rightarrow \{Cod\ \mathbf{r}[t]\} \rangle$

**using**  $I\ arr\text{-}cod\text{-}iff\text{-}arr$  [*of*  $\{t\}$ ] **by** *force*

**show**  $Arr\ \mathbf{r}^{-1}[t] \implies \langle \{\mathbf{r}^{-1}[t]\} : \{Dom\ \mathbf{r}^{-1}[t]\} \rightarrow \{Cod\ \mathbf{r}^{-1}[t]\} \rangle$

**using**  $I\ arr\text{-}dom\text{-}iff\text{-}arr$  [*of*  $\{t\}$ ] **by** *force*

**assume**  $I1$ :  $Arr\ t \implies \langle \{t\} : \{Dom\ t\} \rightarrow \{Cod\ t\} \rangle$

**assume**  $I2$ :  $Arr\ u \implies \langle \{u\} : \{Dom\ u\} \rightarrow \{Cod\ u\} \rangle$

**assume**  $I3$ :  $Arr\ v \implies \langle \{v\} : \{Dom\ v\} \rightarrow \{Cod\ v\} \rangle$

**show**  $Arr\ \mathbf{a}[t, u, v] \implies \langle \{\mathbf{a}[t, u, v]\} : \{Dom\ \mathbf{a}[t, u, v]\} \rightarrow \{Cod\ \mathbf{a}[t, u, v]\} \rangle$

**proof** –

**assume**  $1$ :  $Arr\ \mathbf{a}[t, u, v]$

**have**  $t$ :  $\langle \{t\} : dom\ \{t\} \rightarrow cod\ \{t\} \rangle$  **using**  $1\ I1$  **by** *auto*

**have**  $u$ :  $\langle \{u\} : dom\ \{u\} \rightarrow cod\ \{u\} \rangle$  **using**  $1\ I2$  **by** *auto*

**have**  $v$ :  $\langle \{v\} : dom\ \{v\} \rightarrow cod\ \{v\} \rangle$  **using**  $1\ I3$  **by** *auto*

**have**  $\{\mathbf{a}[t, u, v]\} = (\{t\} \otimes \{u\} \otimes \{v\}) \cdot a[dom\ \{t\}, dom\ \{u\}, dom\ \{v\}]$

**using**  $t\ u\ v\ \alpha\text{-}simp$  [*of*  $\{t\}\ \{u\}\ \{v\}$ ] **by** *auto*

**moreover** **have**  $\langle (\{t\} \otimes \{u\} \otimes \{v\}) \cdot a[dom\ \{t\}, dom\ \{u\}, dom\ \{v\}] :$

$(dom\ \{t\} \otimes dom\ \{u\}) \otimes dom\ \{v\} \rightarrow cod\ \{t\} \otimes cod\ \{u\} \otimes cod\ \{v\} \rangle$

**using**  $t\ u\ v$  **by** (*elim in-homE, auto*)

**moreover** **have**  $\{Dom\ t\} = dom\ \{t\} \wedge \{Dom\ u\} = dom\ \{u\} \wedge \{Dom\ v\} = dom\ \{v\} \wedge$

$\{Cod\ t\} = cod\ \{t\} \wedge \{Cod\ u\} = cod\ \{u\} \wedge \{Cod\ v\} = cod\ \{v\}$

**using**  $1\ I1\ I2\ I3$  **by** *auto*

**ultimately** **show**  $\langle \{\mathbf{a}[t, u, v]\} : \{Dom\ \mathbf{a}[t, u, v]\} \rightarrow \{Cod\ \mathbf{a}[t, u, v]\} \rangle$

**by** *simp*

**qed**

**show**  $Arr\ \mathbf{a}^{-1}[t, u, v] \implies \langle \{\mathbf{a}^{-1}[t, u, v]\} : \{Dom\ \mathbf{a}^{-1}[t, u, v]\} \rightarrow \{Cod\ \mathbf{a}^{-1}[t, u, v]\} \rangle$

**proof** –

**assume**  $1$ :  $Arr\ \mathbf{a}^{-1}[t, u, v]$

**have**  $t$ :  $\langle\langle t \rangle\rangle : \text{dom } \langle t \rangle \rightarrow \text{cod } \langle t \rangle$  **using** 1 I1 **by** *auto*  
**have**  $u$ :  $\langle\langle u \rangle\rangle : \text{dom } \langle u \rangle \rightarrow \text{cod } \langle u \rangle$  **using** 1 I2 **by** *auto*  
**have**  $v$ :  $\langle\langle v \rangle\rangle : \text{dom } \langle v \rangle \rightarrow \text{cod } \langle v \rangle$  **using** 1 I3 **by** *auto*  
**have**  $\langle\langle a^{-1}[t, u, v] \rangle\rangle = ((\langle t \rangle \otimes \langle u \rangle) \otimes \langle v \rangle) \cdot a^{-1}[\text{dom } \langle t \rangle, \text{dom } \langle u \rangle, \text{dom } \langle v \rangle]$   
**using** 1 I1 I2 I3  $\alpha'$ -*simp* [of  $\langle t \rangle$   $\langle u \rangle$   $\langle v \rangle$ ] **by** *auto*  
**moreover** **have**  $\langle\langle (\langle t \rangle \otimes \langle u \rangle) \otimes \langle v \rangle \rangle\rangle \cdot a^{-1}[\text{dom } \langle t \rangle, \text{dom } \langle u \rangle, \text{dom } \langle v \rangle] :$   
 $\text{dom } \langle t \rangle \otimes \text{dom } \langle u \rangle \otimes \text{dom } \langle v \rangle \rightarrow (\text{cod } \langle t \rangle \otimes \text{cod } \langle u \rangle) \otimes \text{cod } \langle v \rangle$   
**using**  $t\ u\ v$  *assoc'-in-hom* [of  $\text{dom } \langle t \rangle$   $\text{dom } \langle u \rangle$   $\text{dom } \langle v \rangle$ ]  
**by** (*elim in-homE*, *auto*)  
**moreover** **have**  $\langle\langle \text{Dom } t \rangle\rangle = \text{dom } \langle t \rangle \wedge \langle\langle \text{Dom } u \rangle\rangle = \text{dom } \langle u \rangle \wedge \langle\langle \text{Dom } v \rangle\rangle = \text{dom } \langle v \rangle \wedge$   
 $\langle\langle \text{Cod } t \rangle\rangle = \text{cod } \langle t \rangle \wedge \langle\langle \text{Cod } u \rangle\rangle = \text{cod } \langle u \rangle \wedge \langle\langle \text{Cod } v \rangle\rangle = \text{cod } \langle v \rangle$   
**using** 1 I1 I2 I3 **by** *auto*  
**ultimately** **show**  $\langle\langle a^{-1}[t, u, v] \rangle\rangle : \langle\langle \text{Dom } a^{-1}[t, u, v] \rangle\rangle \rightarrow \langle\langle \text{Cod } a^{-1}[t, u, v] \rangle\rangle$   
**by** *simp*  
**qed**  
**qed**

**lemma** *arr-eval* [*simp*]:  
**assumes**  $\text{Arr } f$   
**shows**  $\text{arr } \langle f \rangle$   
**using** *assms eval-in-hom* **by** *auto*

**lemma** *dom-eval* [*simp*]:  
**assumes**  $\text{Arr } f$   
**shows**  $\text{dom } \langle f \rangle = \langle\langle \text{Dom } f \rangle\rangle$   
**using** *assms eval-in-hom* **by** *auto*

**lemma** *cod-eval* [*simp*]:  
**assumes**  $\text{Arr } f$   
**shows**  $\text{cod } \langle f \rangle = \langle\langle \text{Cod } f \rangle\rangle$   
**using** *assms eval-in-hom* **by** *auto*

**lemma** *eval-Prim* [*simp*]:  
**assumes**  $C.\text{arr } f$   
**shows**  $\langle\langle f \rangle\rangle = V f$   
**by** *simp*

**lemma** *eval-Tensor* [*simp*]:  
**assumes**  $\text{Arr } t$  **and**  $\text{Arr } u$   
**shows**  $\langle t \otimes u \rangle = \langle t \rangle \otimes \langle u \rangle$   
**using** *assms eval-in-hom* **by** *auto*

**lemma** *eval-Comp* [*simp*]:  
**assumes**  $\text{Arr } t$  **and**  $\text{Arr } u$  **and**  $\text{Dom } t = \text{Cod } u$   
**shows**  $\langle t \cdot u \rangle = \langle t \rangle \cdot \langle u \rangle$   
**using** *assms* **by** *simp*

**lemma** *eval-Lunit* [*simp*]:  
**assumes**  $\text{Arr } t$

**shows**  $\{\mathbf{l}[t]\} = \mathbf{l}[\{\text{Cod } t\}] \cdot (\mathcal{I} \otimes \{t\})$   
**using** *assms lunit-naturality [of {t}] by simp*

**lemma** *eval-Lunit' [simp]*:  
**assumes** *Arr t*  
**shows**  $\{\mathbf{l}^{-1}[t]\} = \mathbf{l}^{-1}[\{\text{Cod } t\}] \cdot \{t\}$   
**using** *assms lunit'-naturality [of {t}] l'.map-simp [of {t}] l-ide-simp*  
*Arr-implies-Ide-Cod*  
**by** *simp*

**lemma** *eval-Runit [simp]*:  
**assumes** *Arr t*  
**shows**  $\{\mathbf{r}[t]\} = \mathbf{r}[\{\text{Cod } t\}] \cdot (\{t\} \otimes \mathcal{I})$   
**using** *assms runit-naturality [of {t}] by simp*

**lemma** *eval-Runit' [simp]*:  
**assumes** *Arr t*  
**shows**  $\{\mathbf{r}^{-1}[t]\} = \mathbf{r}^{-1}[\{\text{Cod } t\}] \cdot \{t\}$   
**using** *assms runit'-naturality [of {t}] r'.map-simp [of {t}] r-ide-simp*  
*Arr-implies-Ide-Cod*  
**by** *simp*

**lemma** *eval-Assoc [simp]*:  
**assumes** *Arr t and Arr u and Arr v*  
**shows**  $\{\mathbf{a}[t, u, v]\} = \mathbf{a}[\text{cod } \{t\}, \text{cod } \{u\}, \text{cod } \{v\}] \cdot ((\{t\} \otimes \{u\}) \otimes \{v\})$   
**using** *assms a.is-natural-2 [of ({t}, {u}, {v})] by auto*

**lemma** *eval-Assoc' [simp]*:  
**assumes** *Arr t and Arr u and Arr v*  
**shows**  $\{\mathbf{a}^{-1}[t, u, v]\} = \mathbf{a}^{-1}[\text{cod } \{t\}, \text{cod } \{u\}, \text{cod } \{v\}] \cdot (\{t\} \otimes \{u\} \otimes \{v\})$   
**using** *assms a'-simp [of {t} {u} {v}] assoc'-naturality [of {t} {u} {v}]*  
**by** *simp*

The following are conveniences for the case of identity arguments to avoid having to get rid of the extra identities that are introduced by the general formulas above.

**lemma** *eval-Lunit-Ide [simp]*:  
**assumes** *Ide a*  
**shows**  $\{\mathbf{l}[a]\} = \mathbf{l}[\{a\}]$   
**using** *assms comp-cod-arr by simp*

**lemma** *eval-Lunit'-Ide [simp]*:  
**assumes** *Ide a*  
**shows**  $\{\mathbf{l}^{-1}[a]\} = \mathbf{l}^{-1}[\{a\}]$   
**using** *assms comp-cod-arr by simp*

**lemma** *eval-Runit-Ide [simp]*:  
**assumes** *Ide a*  
**shows**  $\{\mathbf{r}[a]\} = \mathbf{r}[\{a\}]$   
**using** *assms comp-cod-arr by simp*

**lemma** *eval-Runit'-Ide* [*simp*]:  
**assumes** *Ide a*  
**shows**  $\{\mathbf{r}^{-1}[a]\} = \mathbf{r}^{-1}\{\{a\}\}$   
**using** *assms comp-cod-arr* **by** *simp*

**lemma** *eval-Assoc-Ide* [*simp*]:  
**assumes** *Ide a* **and** *Ide b* **and** *Ide c*  
**shows**  $\{\mathbf{a}[a, b, c]\} = \mathbf{a}\{\{a\}, \{b\}, \{c\}\}$   
**using** *assms* **by** *simp*

**lemma** *eval-Assoc'-Ide* [*simp*]:  
**assumes** *Ide a* **and** *Ide b* **and** *Ide c*  
**shows**  $\{\mathbf{a}^{-1}[a, b, c]\} = \mathbf{a}^{-1}\{\{a\}, \{b\}, \{c\}\}$   
**using** *assms  $\alpha'$ -ide-simp* **by** *simp*

Canonical arrows evaluate to isomorphisms in  $D$ , and formal inverses evaluate to inverses in  $D$ .

**lemma** *iso-eval-Can*:  
**shows** *Can t*  $\implies$  *iso*  $\{t\}$   
**using** *Can-implies-Arr* *V'.preserves-iso* *Q'.preserves-iso* *alpha.preserves-iso* *alpha'.preserves-iso*  
*Arr-implies-Ide-Dom*  
**by** (*induct t*) *auto*

**lemma** *eval-Inv-Can*:  
**shows** *Can t*  $\implies$   $\{\mathbf{Inv} t\} = \mathbf{inv} \{t\}$   
**apply** (*induct t*)  
**using** *iso-eval-Can inv-comp* *Can-implies-Arr*  
**apply** *auto*[4]

**proof** –  
**fix**  $t$   
**assume**  $I$ : *Can t*  $\implies$   $\{\mathbf{Inv} t\} = \mathbf{inv} \{t\}$   
**show** *Can*  $\mathbf{l}[t]$   $\implies$   $\{\mathbf{Inv} \mathbf{l}[t]\} = \mathbf{inv} \{\mathbf{l}[t]\}$   
**using**  $I$  *V'.is-natural-2* [*of inv*  $\{t\}$ ] *iso-eval-Can l-ide-simp iso-is-arr*  
*comp-cod-arr inv-comp*  
**by** *simp*  
**show** *Can*  $\mathbf{r}[t]$   $\implies$   $\{\mathbf{Inv} \mathbf{r}[t]\} = \mathbf{inv} \{\mathbf{r}[t]\}$   
**using**  $I$  *Q'.is-natural-2* [*of inv*  $\{t\}$ ] *iso-eval-Can Q-ide-simp iso-is-arr*  
*comp-cod-arr inv-comp*  
**by** *simp*  
**show** *Can*  $\mathbf{l}^{-1}[t]$   $\implies$   $\{\mathbf{Inv} \mathbf{l}^{-1}[t]\} = \mathbf{inv} \{\mathbf{l}^{-1}[t]\}$

**proof** –  
**assume**  $t$ : *Can*  $\mathbf{l}^{-1}[t]$   
**hence**  $1$ : *iso*  $\{t\}$  **using** *iso-eval-Can* **by** *simp*  
**have**  $\mathbf{inv} \{\mathbf{l}^{-1}[t]\} = \mathbf{inv} (V' \{t\})$   
**using**  $t$  **by** *simp*  
**also have**  $\dots = \mathbf{inv} (\mathbf{l}^{-1}[\mathbf{cod} \{t\}] \cdot \{t\})$   
**using**  $1$  *V'.is-natural-2* [*of*  $\{t\}$ ] *V'-ide-simp iso-is-arr* **by** *auto*  
**also have**  $\dots = \{\mathbf{Inv} \mathbf{l}^{-1}[t]\}$

using  $t I 1$  *iso-is-arr inv-comp* by *auto*  
 finally show *?thesis* by *simp*  
 qed  
 show  $\text{Can } \mathbf{r}^{-1}[t] \implies \{\!\{ \text{Inv } \mathbf{r}^{-1}[t] \}\!\} = \text{inv } \{\mathbf{r}^{-1}[t]\}$   
 proof –  
 assume  $t$ :  $\text{Can } \mathbf{r}^{-1}[t]$   
 hence  $1$ : *iso*  $\{\!\{t\}\!\}$  using *iso-eval-Can* by *simp*  
 have  $\text{inv } \{\mathbf{r}^{-1}[t]\} = \text{inv } (\varrho' \{\!\{t\}\!\})$   
 using  $t$  by *simp*  
 also have  $\dots = \text{inv } (\mathbf{r}^{-1}[\text{cod } \{\!\{t\}\!\}] \cdot \{\!\{t\}\!\})$   
 using  $1$   $\varrho'$ -*is-natural-2* [of  $\{\!\{t\}\!\}$ ]  $\varrho'$ -*ide-simp iso-is-arr* by *auto*  
 also have  $\dots = \{\!\{ \text{Inv } \mathbf{r}^{-1}[t] \}\!\}$   
 using  $t I 1$  *iso-is-arr inv-comp* by *auto*  
 finally show *?thesis* by *simp*  
 qed  
 next  
 fix  $t u v$   
 assume  $I1$ :  $\text{Can } t \implies \{\!\{ \text{Inv } t \}\!\} = \text{inv } \{\!\{t\}\!\}$   
 assume  $I2$ :  $\text{Can } u \implies \{\!\{ \text{Inv } u \}\!\} = \text{inv } \{\!\{u\}\!\}$   
 assume  $I3$ :  $\text{Can } v \implies \{\!\{ \text{Inv } v \}\!\} = \text{inv } \{\!\{v\}\!\}$   
 show  $\text{Can } \mathbf{a}[t, u, v] \implies \{\!\{ \text{Inv } \mathbf{a}[t, u, v] \}\!\} = \text{inv } \{\mathbf{a}[t, u, v]\}$   
 proof –  
 assume  $tuv$ :  $\text{Can } \mathbf{a}[t, u, v]$   
 have  $t$ : *iso*  $\{\!\{t\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $u$ : *iso*  $\{\!\{u\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $v$ : *iso*  $\{\!\{v\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $\{\!\{ \text{Inv } \mathbf{a}[t, u, v] \}\!\} = \alpha' (\text{inv } \{\!\{t\}\!\}, \text{inv } \{\!\{u\}\!\}, \text{inv } \{\!\{v\}\!\})$   
 using  $tuv I1 I2 I3$  by *simp*  
 also have  $\dots = \text{inv } (\mathbf{a}[\text{cod } \{\!\{t\}\!\}, \text{cod } \{\!\{u\}\!\}, \text{cod } \{\!\{v\}\!\}] \cdot ((\{\!\{t\}\!\} \otimes \{\!\{u\}\!\}) \otimes \{\!\{v\}\!\})$   
 using  $t u v$   $\alpha'$ -*simp iso-is-arr inv-comp* by *auto*  
 also have  $\dots = \text{inv } ((\{\!\{t\}\!\} \otimes \{\!\{u\}\!\} \otimes \{\!\{v\}\!\}) \cdot \mathbf{a}[\text{dom } \{\!\{t\}\!\}, \text{dom } \{\!\{u\}\!\}, \text{dom } \{\!\{v\}\!\}])$   
 using  $t u v$  *iso-is-arr assoc-naturality* by *simp*  
 also have  $\dots = \text{inv } \{\mathbf{a}[t, u, v]\}$   
 using  $t u v$  *iso-is-arr*  $\alpha$ -*simp* [of  $\{\!\{t\}\!\}$   $\{\!\{u\}\!\}$   $\{\!\{v\}\!\}$ ] by *simp*  
 finally show *?thesis* by *simp*  
 qed  
 show  $\text{Can } \mathbf{a}^{-1}[t, u, v] \implies \{\!\{ \text{Inv } \mathbf{a}^{-1}[t, u, v] \}\!\} = \text{inv } \{\mathbf{a}^{-1}[t, u, v]\}$   
 proof –  
 assume  $tuv$ :  $\text{Can } \mathbf{a}^{-1}[t, u, v]$   
 have  $t$ : *iso*  $\{\!\{t\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $u$ : *iso*  $\{\!\{u\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $v$ : *iso*  $\{\!\{v\}\!\}$  using  $tuv$  *iso-eval-Can* by *auto*  
 have  $\{\!\{ \text{Inv } \mathbf{a}^{-1}[t, u, v] \}\!\} = \alpha (\text{inv } \{\!\{t\}\!\}, \text{inv } \{\!\{u\}\!\}, \text{inv } \{\!\{v\}\!\})$   
 using  $tuv I1 I2 I3$  by *simp*  
 also have  $\dots = (\text{inv } \{\!\{t\}\!\} \otimes \text{inv } \{\!\{u\}\!\} \otimes \text{inv } \{\!\{v\}\!\}) \cdot \mathbf{a}[\text{cod } \{\!\{t\}\!\}, \text{cod } \{\!\{u\}\!\}, \text{cod } \{\!\{v\}\!\}]$   
 using  $t u v$  *iso-is-arr*  $\alpha$ -*simp* [of  $\text{inv } \{\!\{t\}\!\}$   $\text{inv } \{\!\{u\}\!\}$   $\text{inv } \{\!\{v\}\!\}$ ] by *simp*  
 also have  $\dots = \text{inv } (\mathbf{a}^{-1}[\text{cod } \{\!\{t\}\!\}, \text{cod } \{\!\{u\}\!\}, \text{cod } \{\!\{v\}\!\}] \cdot (\{\!\{t\}\!\} \otimes \{\!\{u\}\!\} \otimes \{\!\{v\}\!\}))$   
 using  $t u v$  *iso-is-arr inv-comp* by *auto*  
 also have  $\dots = \text{inv } (((\{\!\{t\}\!\} \otimes \{\!\{u\}\!\}) \otimes \{\!\{v\}\!\}) \cdot \mathbf{a}^{-1}[\text{dom } \{\!\{t\}\!\}, \text{dom } \{\!\{u\}\!\}, \text{dom } \{\!\{v\}\!\}])$



```

    using t u v iso-is-arr assoc'-naturality by simp
  also have ... = inv {a-1[t, u, v]}
    using t u v iso-is-arr α'-simp by auto
  finally show ?thesis by blast
qed
qed

```

The operation  $[\cdot]$  evaluates to composition in  $D$ .

**lemma** *eval-CompDiag*:

**assumes** *Diag t* and *Diag u* and *Seq t u*

**shows**  $\{t [\cdot] u\} = \{t\} \cdot \{u\}$

**proof** –

**have**  $\bigwedge u. \llbracket \text{Diag } t; \text{Diag } u; \text{Seq } t \ u \rrbracket \implies \{t [\cdot] u\} = \{t\} \cdot \{u\}$

**using** *eval-in-hom comp-cod-arr*

**proof** (*induct t, simp-all*)

**fix**  $u \ f$

**assume**  $u: \text{Diag } u$

**assume**  $f: C.\text{arr } f$

**assume**  $1: \text{Arr } u \wedge \langle C.\text{dom } f \rangle = \text{Cod } u$

**show**  $\{\langle f \rangle [\cdot] u\} = V f \cdot \{u\}$

**using**  $f \ u \ 1 \ \text{as-nat-trans.preserves-comp-2}$  **by** (*cases u; simp*)

**next**

**fix**  $u \ v \ w$

**assume**  $I1: \bigwedge u. \llbracket \text{Diag } v; \text{Diag } u; \text{Arr } u \wedge \text{Dom } v = \text{Cod } u \rrbracket \implies \{v [\cdot] u\} = \{v\} \cdot \{u\}$

**assume**  $I2: \bigwedge u. \llbracket \text{Diag } w; \text{Diag } u; \text{Arr } u \wedge \text{Dom } w = \text{Cod } u \rrbracket \implies \{w [\cdot] u\} = \{w\} \cdot \{u\}$

**assume**  $vw: \text{Diag } (\text{Tensor } v \ w)$

**have**  $v: \text{Diag } v \wedge v = \text{Prim } (\text{un-Prim } v)$

**using**  $vw$  **by** (*simp add: Diag-TensorE*)

**have**  $w: \text{Diag } w$

**using**  $vw$  **by** (*simp add: Diag-TensorE*)

**assume**  $u: \text{Diag } u$

**assume**  $1: \text{Arr } v \wedge \text{Arr } w \wedge \text{Arr } u \wedge \text{Dom } v \otimes \text{Dom } w = \text{Cod } u$

**show**  $\{(v \otimes w) [\cdot] u\} = (\{v\} \otimes \{w\}) \cdot \{u\}$

**using**  $u \ 1 \ \text{eval-in-hom CompDiag-in-Hom}$

**proof** (*cases u, simp-all*)

**fix**  $x \ y$

**assume**  $3: u = x \otimes y$

**assume**  $4: \text{Arr } v \wedge \text{Arr } w \wedge \text{Dom } v = \text{Cod } x \wedge \text{Dom } w = \text{Cod } y$

**have**  $x: \text{Diag } x$

**using**  $u \ 1 \ 3 \ \text{Diag-TensorE [of } x \ y]$  **by** *simp*

**have**  $y: \text{Diag } y$

**using**  $u \ x \ 1 \ 3 \ \text{Diag-TensorE [of } x \ y]$  **by** *simp*

**show**  $\{v [\cdot] x\} \otimes \{w [\cdot] y\} = (\{v\} \otimes \{w\}) \cdot (\{x\} \otimes \{y\})$

**using**  $v \ w \ x \ y \ 4 \ I1 \ I2 \ \text{CompDiag-in-Hom eval-in-hom Diag-implies-Arr interchange}$

**by** *auto*

**qed**

**qed**

**thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

For identity terms  $a$  and  $b$ , the reduction  $(a \otimes b)\downarrow$  factors (under evaluation in  $D$ ) into the parallel reduction  $a\downarrow \otimes b\downarrow$ , followed by a reduction of its codomain  $\lfloor a \rfloor \downarrow \lfloor b \rfloor$ .

**lemma** *eval-red-Tensor*:  
**assumes** *Ide a* **and** *Ide b*  
**shows**  $\{(a \otimes b)\downarrow\} = \{\lfloor a \rfloor \downarrow \lfloor b \rfloor\} \cdot (\{a\downarrow\} \otimes \{b\downarrow\})$   
**proof** –  
**have** *Diag (a ⊗ b) ⇒ ?thesis*  
**using** *assms Can-red2 Ide-implies-Arr red-Diag*  
*Diagonalize-Diag red2-Diag Can-implies-Arr iso-eval-Can iso-is-arr*  
**apply** *simp*  
**using** *Diag-TensorE eval-Tensor Diagonalize-Diag Diag-implies-Arr red-Diag*  
*tensor-preserves-ide ide-eval-Ide dom-eval comp-arr-dom*  
**by** *metis*  
**moreover** **have**  $\neg \text{Diag } (a \otimes b) \implies ?thesis$   
**using** *assms Can-red2* **by** (*simp add: Can-red(1) iso-eval-Can*)  
**ultimately show** *?thesis* **by** *blast*  
**qed**

**lemma** *eval-red2-Diag-Unity*:  
**assumes** *Ide a* **and** *Diag a*  
**shows**  $\{a \downarrow \mathcal{I}\} = r[\{a\}]$   
**using** *assms tensor-preserves-ide ρ-ide-simp unitor-coincidence ι-in-hom comp-cod-arr*  
**by** (*cases a, auto*)

Define a formal arrow  $t$  to be “coherent” if the square formed by  $t$ ,  $\lfloor t \rfloor$  and the reductions  $Dom\ t\downarrow$  and  $Cod\ t\downarrow$  commutes under evaluation in  $D$ . We will show that all formal arrows are coherent. Since the diagonalizations of canonical arrows are identities, a corollary is that parallel canonical arrows have equal evaluations.

**abbreviation** *coherent*  
**where** *coherent t*  $\equiv \{Cod\ t\downarrow\} \cdot \{t\} = \{\lfloor t \rfloor\} \cdot \{Dom\ t\downarrow\}$

Diagonal arrows are coherent, since for such arrows  $t$  the reductions  $Dom\ t\downarrow$  and  $Cod\ t\downarrow$  are identities.

**lemma** *Diag-implies-coherent*:  
**assumes** *Diag t*  
**shows** *coherent t*  
**using** *assms Diag-implies-Arr Arr-implies-Ide-Dom Arr-implies-Ide-Cod*  
*Dom-preserves-Diag Cod-preserves-Diag Diagonalize-Diag red-Diag*  
*comp-arr-dom comp-cod-arr*  
**by** *simp*

The evaluation of a coherent arrow  $t$  has a canonical factorization in  $D$  into the evaluations of a reduction  $Dom\ t\downarrow$ , diagonalization  $\lfloor t \rfloor$ , and inverse reduction  $Inv\ (Cod\ t\downarrow)$ . This will later allow us to use the term  $Inv\ (Cod\ t\downarrow) \cdot \lfloor t \rfloor \cdot Dom\ t\downarrow$  as a normal form for  $t$ .

**lemma** *canonical-factorization*:  
**assumes**  $Arr\ t$   
**shows**  $coherent\ t \longleftrightarrow \{\!\{t\}\!\} = inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$   
**proof**  
  **assume**  $1: coherent\ t$   
  **have**  $inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\} = inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{t\}\!\}$   
  **using**  $1$  **by** *simp*  
  **also have**  $\dots = (inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{Cod\ t\downarrow\}\!\}) \cdot \{\!\{t\}\!\}$   
  **using** *comp-assoc* **by** *simp*  
  **also have**  $\dots = \{\!\{t\}\!\}$   
  **using** *assms 1 red-in-Hom inv-in-hom Arr-implies-Ide-Cod Can-red iso-eval-Can comp-cod-arr Ide-in-Hom inv-is-inverse*  
  **by** (*simp add: comp-inv-arr*)  
  **finally show**  $\{\!\{t\}\!\} = inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$  **by** *simp*  
  **next**  
  **assume**  $1: \{\!\{t\}\!\} = inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$   
  **hence**  $\{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{t\}\!\} = \{\!\{Cod\ t\downarrow\}\!\} \cdot inv\ \{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$  **by** *simp*  
  **also have**  $\dots = (\{\!\{Cod\ t\downarrow\}\!\} \cdot inv\ \{\!\{Cod\ t\downarrow\}\!\}) \cdot \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$   
  **using** *comp-assoc* **by** *simp*  
  **also have**  $\dots = \{\!\{[t]\}\!\} \cdot \{\!\{Dom\ t\downarrow\}\!\}$   
  **using** *assms 1 red-in-Hom Arr-implies-Ide-Cod Can-red iso-eval-Can inv-is-inverse Diagonalize-in-Hom comp-arr-inv comp-cod-arr Arr-implies-Ide-Dom Diagonalize-in-Hom*  
  **by** *auto*  
  **finally show**  $coherent\ t$  **by** *blast*  
**qed**

A canonical arrow is coherent if and only if its formal inverse is.

**lemma** *Can-implies-coherent-iff-coherent-Inv*:  
**assumes**  $Can\ t$   
**shows**  $coherent\ t \longleftrightarrow coherent\ (Inv\ t)$   
**proof**  
  **have**  $1: \bigwedge t. Can\ t \implies coherent\ t \implies coherent\ (Inv\ t)$   
  **proof** –  
  **fix**  $t$   
  **assume**  $Can\ t$   
  **hence**  $t: Can\ t \wedge Arr\ t \wedge Ide\ (Dom\ t) \wedge Ide\ (Cod\ t) \wedge arr\ \{\!\{t\}\!\} \wedge iso\ \{\!\{t\}\!\} \wedge inverse\ arrows\ \{\!\{t\}\!\} (inv\ \{\!\{t\}\!\}) \wedge Can\ [t] \wedge Arr\ [t] \wedge arr\ \{\!\{[t]\}\!\} \wedge iso\ \{\!\{[t]\}\!\} \wedge [t] \in Hom\ [Dom\ t] [Cod\ t] \wedge inverse\ arrows\ \{\!\{[t]\}\!\} (inv\ \{\!\{[t]\}\!\}) \wedge Inv\ t \in Hom\ (Cod\ t) (Dom\ t)$   
  **using** *assms Can-implies-Arr Arr-implies-Ide-Dom Arr-implies-Ide-Cod iso-eval-Can inv-is-inverse Diagonalize-in-Hom Diagonalize-preserves-Can Inv-in-Hom*  
  **by** *simp*  
  **assume** *coh: coherent t*  
  **have**  $\{\!\{Cod\ (Inv\ t)\downarrow\}\!\} \cdot \{\!\{Inv\ t\}\!\} = (inv\ \{\!\{[t]\}\!\} \cdot \{\!\{[t]\}\!\}) \cdot \{\!\{Cod\ (Inv\ t)\downarrow\}\!\} \cdot \{\!\{Inv\ t\}\!\}$   
  **using**  $t$  *red-in-Hom comp-cod-arr comp-inv-arr*  
  **by** (*simp add: canonical-factorization coh Diagonalize-preserves-Can <Can t> inv-is-inverse*)  
  **also have**  $\dots = inv\ \{\!\{[t]\}\!\} \cdot (\{\!\{Cod\ t\downarrow\}\!\} \cdot \{\!\{t\}\!\}) \cdot inv\ \{\!\{t\}\!\}$

```

    using t eval-Inv-Can coh comp-assoc by auto
  also have ... =  $\{\llbracket \text{Inv } t \rrbracket\} \cdot \{\llbracket \text{Dom } (\text{Inv } t) \downarrow \rrbracket\}$ 
  using t Diagonalize-Inv eval-Inv-Can comp-arr-inv red-in-Hom comp-arr-dom comp-assoc
    by auto
  finally show coherent (Inv t) by blast
qed
show coherent t  $\implies$  coherent (Inv t) using assms 1 by simp
show coherent (Inv t)  $\implies$  coherent t
proof -
  assume coherent (Inv t)
  hence coherent (Inv (Inv t))
    using assms 1 Inv-preserves-Can by blast
  thus ?thesis using assms by simp
qed
qed

```

Some special cases of coherence are readily dispatched.

```

lemma coherent-Unity:
shows coherent  $\mathcal{I}$ 
  by simp

```

```

lemma coherent-Prim:
assumes Arr  $\langle f \rangle$ 
shows coherent  $\langle f \rangle$ 
  using assms by simp

```

```

lemma coherent-Lunit-Ide:
assumes Ide a
shows coherent  $\mathbb{I}[a]$ 
proof -
  have a: Ide a  $\wedge$  Arr a  $\wedge$  Dom a = a  $\wedge$  Cod a = a  $\wedge$ 
    ide  $\{\llbracket a \rrbracket\} \wedge$  ide  $\{\llbracket \mathbb{I}[a] \rrbracket\} \wedge \{\llbracket a \downarrow \rrbracket\} \in \text{hom } \{\llbracket a \rrbracket\} \{\llbracket \mathbb{I}[a] \rrbracket\}$ 
    using assms Ide-implies-Arr Ide-in-Hom Diagonalize-preserves-Ide red-in-Hom by auto
  thus ?thesis
    using a lunit-naturality [of  $\{\llbracket a \downarrow \rrbracket\}$ ] comp-cod-arr by auto
qed

```

```

lemma coherent-Runit-Ide:
assumes Ide a
shows coherent  $\mathbf{r}[a]$ 
proof -
  have a: Ide a  $\wedge$  Arr a  $\wedge$  Dom a = a  $\wedge$  Cod a = a  $\wedge$ 
    ide  $\{\llbracket a \rrbracket\} \wedge$  ide  $\{\llbracket \mathbb{I}[a] \rrbracket\} \wedge \{\llbracket a \downarrow \rrbracket\} \in \text{hom } \{\llbracket a \rrbracket\} \{\llbracket \mathbb{I}[a] \rrbracket\}$ 
    using assms Ide-implies-Arr Ide-in-Hom Diagonalize-preserves-Ide red-in-Hom
    by auto
  have  $\{\llbracket \text{Cod } \mathbf{r}[a] \downarrow \rrbracket\} \cdot \{\llbracket \mathbf{r}[a] \rrbracket\} = \{\llbracket a \downarrow \rrbracket\} \cdot \mathbf{r}\{\llbracket a \rrbracket\}$ 
    using a runit-in-hom comp-cod-arr by simp
  also have ... =  $\mathbf{r}\{\llbracket \mathbb{I}[a] \rrbracket\} \cdot (\{\llbracket a \downarrow \rrbracket\} \otimes \mathcal{I})$ 
    using a eval-Runit runit-naturality [of  $\{\llbracket \text{red } a \rrbracket\}$ ] by auto

```

**also have**  $\dots = \{\llbracket r[a] \rrbracket\} \cdot \{\text{Dom } r[a] \downarrow\}$   
**proof** –  
**have**  $\neg \text{Diag } (a \otimes \mathcal{I})$  **by** (*cases a; simp*)  
**thus** *?thesis*  
**using** *a comp-cod-arr red2-in-Hom eval-red2-Diag-Unity Diag-Diagonalize*  
*Diagonalize-preserves-Ide*  
**by** *auto*  
**qed**  
**finally show** *?thesis* **by** *blast*  
**qed**

**lemma** *coherent-Lunit'-Ide*:  
**assumes** *Ide a*  
**shows** *coherent l<sup>-1</sup>[a]*  
**using** *assms Ide-implies-Can coherent-Lunit-Ide*  
*Can-implies-coherent-iff-coherent-Inv [of Lunit a]* **by** *simp*

**lemma** *coherent-Runit'-Ide*:  
**assumes** *Ide a*  
**shows** *coherent r<sup>-1</sup>[a]*  
**using** *assms Ide-implies-Can coherent-Runit-Ide*  
*Can-implies-coherent-iff-coherent-Inv [of Runit a]* **by** *simp*

To go further, we need the next result, which is in some sense the crux of coherence: For diagonal identities  $a$ ,  $b$ , and  $c$ , the reduction  $((a \llbracket \otimes \rrbracket b) \downarrow c) \cdot ((a \downarrow b) \otimes c)$  from  $(a \otimes b) \otimes c$  that first reduces the subterm  $a \otimes b$  and then reduces the result, is equivalent under evaluation in  $D$  to the reduction that first applies the associator  $\mathbf{a}[a, b, c]$  and then applies the reduction  $(a \downarrow b \llbracket \otimes \rrbracket c) \cdot (a \otimes b \downarrow c)$  from  $a \otimes b \otimes c$ . The triangle and pentagon axioms are used in the proof.

**lemma** *coherence-key-fact*:  
**assumes** *Ide a ∧ Diag a* **and** *Ide b ∧ Diag b* **and** *Ide c ∧ Diag c*  
**shows**  $\{(a \llbracket \otimes \rrbracket b) \downarrow c\} \cdot \{(a \downarrow b) \otimes \{c\}\}$   
 $= \{(a \downarrow (b \llbracket \otimes \rrbracket c))\} \cdot (\{a\} \otimes \{b \downarrow c\}) \cdot \mathbf{a}[\{a\}, \{b\}, \{c\}]$   
**proof** –  
**have**  $b = \mathcal{I} \implies ?thesis$   
**using** *assms not-is-Tensor-TensorDiagE eval-red2-Diag-Unity triangle*  
*comp-cod-arr comp-assoc*  
**by** *simp*

The triangle is used!

**moreover have**  $c = \mathcal{I} \implies ?thesis$   
**using** *assms TensorDiag-preserves-Diag TensorDiag-preserves-Ide*  
*not-is-Tensor-TensorDiagE eval-red2-Diag-Unity*  
*red2-in-Hom runit-tensor runit-naturality [of {a ↓ b}] comp-assoc*  
**by** *simp*  
**moreover have**  $\llbracket b \neq \mathcal{I}; c \neq \mathcal{I} \rrbracket \implies ?thesis$   
**proof** –  
**assume**  $b': b \neq \mathcal{I}$   
**hence**  $b: \text{Ide } b \wedge \text{Diag } b \wedge \text{Arr } b \wedge b \neq \mathcal{I} \wedge$

$ide \{b\} \wedge arr \{b\} \wedge [b] = b \wedge b \downarrow = b \wedge Dom b = b \wedge Cod b = b$   
**using** *assms Diagonalize-preserves-Ide Ide-in-Hom* **by** *simp*  
**assume**  $c': c \neq \mathcal{I}$   
**hence**  $c: Ide c \wedge Diag c \wedge Arr c \wedge c \neq \mathcal{I} \wedge$   
 $ide \{c\} \wedge arr \{c\} \wedge [c] = c \wedge c \downarrow = c \wedge Dom c = c \wedge Cod c = c$   
**using** *assms Diagonalize-preserves-Ide Ide-in-Hom* **by** *simp*  
**have**  $\bigwedge a. Ide a \wedge Diag a \implies$   
 $\{ (a \lfloor \otimes \rfloor b) \downarrow c \} \cdot (\{a \downarrow b\} \otimes \{c\})$   
 $= (\{a \downarrow (b \lfloor \otimes \rfloor c)\} \cdot (\{a\} \otimes \{b \downarrow c\})) \cdot a[\{a\}, \{b\}, \{c\}]$   
**proof** –  
**fix**  $a :: 'c$  term  
**show**  $Ide a \wedge Diag a \implies$   
 $\{ (a \lfloor \otimes \rfloor b) \downarrow c \} \cdot (\{a \downarrow b\} \otimes \{c\})$   
 $= (\{a \downarrow (b \lfloor \otimes \rfloor c)\} \cdot (\{a\} \otimes \{b \downarrow c\})) \cdot a[\{a\}, \{b\}, \{c\}]$   
**apply** (*induct a*)  
**using**  $b\ c$  *TensorDiag-in-Hom* **apply** *simp-all*  
**proof** –  
**show**  $\{b \downarrow c\} \cdot (\{b\} \cdot 1[\{b\}] \otimes \{c\})$   
 $= ((\{b \lfloor \otimes \rfloor c\} \cdot 1[\{b \lfloor \otimes \rfloor c\}]) \cdot (\mathcal{I} \otimes \{b \downarrow c\})) \cdot a[\mathcal{I}, \{b\}, \{c\}]$   
**proof** –  
**have**  $\{b \lfloor \otimes \rfloor c\} \cdot (1[\{b \lfloor \otimes \rfloor c\}] \cdot (\mathcal{I} \otimes \{b \downarrow c\})) \cdot a[\mathcal{I}, \{b\}, \{c\}] =$   
 $\{b \lfloor \otimes \rfloor c\} \cdot (\{b \downarrow c\} \cdot 1[\{b\}] \otimes \{c\}) \cdot a[\mathcal{I}, \{b\}, \{c\}]$   
**using**  $b\ c$  *red2-in-Hom lunit-naturality* [*of*  $\{b \downarrow c\}$ ] **by** *simp*  
**thus** *?thesis*  
**using**  $b\ c$  *red2-in-Hom lunit-tensor comp-arr-dom comp-cod-arr comp-assoc* **by** *simp*  
**qed**  
**show**  $\bigwedge f. C.ide f \wedge C.arr f \implies$   
 $\{ (\langle f \rangle \otimes b) \downarrow c \} \cdot (\{ \langle f \rangle \downarrow b \} \otimes \{c\})$   
 $= (\{ \langle f \rangle \downarrow (b \lfloor \otimes \rfloor c) \} \cdot (V f \otimes \{b \downarrow c\})) \cdot a[V f, \{b\}, \{c\}]$   
**proof** –  
**fix**  $f$   
**assume**  $f: C.ide f \wedge C.arr f$   
**show**  $\{ (\langle f \rangle \otimes b) \downarrow c \} \cdot (\{ \langle f \rangle \downarrow b \} \otimes \{c\})$   
 $= (\{ \langle f \rangle \downarrow (b \lfloor \otimes \rfloor c) \} \cdot (V f \otimes \{b \downarrow c\})) \cdot a[V f, \{b\}, \{c\}]$   
**proof** –  
**have**  $\{ (\langle f \rangle \otimes b) \downarrow c \} \cdot (\{ \langle f \rangle \downarrow b \} \otimes \{c\})$   
 $= ((V f \otimes \{b \lfloor \otimes \rfloor c\}) \cdot (V f \otimes \{b \downarrow c\})) \cdot a[V f, \{b\}, \{c\}] \cdot$   
 $((V f \otimes \{b\}) \otimes \{c\})$   
**proof** –  
**have**  $\{ \langle f \rangle \downarrow b \} = V f \otimes \{b\}$   
**using** *assms f b c red2-Diag* **by** *simp*  
**moreover** **have**  $\{ \langle f \rangle \downarrow (b \lfloor \otimes \rfloor c) \} = V f \otimes \{b \lfloor \otimes \rfloor c\}$   
**proof** –  
**have** *is-Tensor*  $(b \lfloor \otimes \rfloor c)$   
**using** *assms b c not-is-Tensor-TensorDiagE* **by** *blast*  
**thus** *?thesis*  
**using** *assms f b c red2-Diag TensorDiag-preserves-Diag(1)*  
**by** (*cases b \lfloor \otimes \rfloor c; simp*)  
**qed**

**ultimately show** *?thesis*  
**using** *assms b c* **by** (*cases c, simp-all*)  
**qed**  
**also have** ... =  $((V f \otimes \{b \lfloor \otimes \rfloor c\}) \cdot (V f \otimes \{b \Downarrow c\})) \cdot a[V f, \{b\}, \{c\}]$   
**using** *b c f TensorDiag-in-Hom red2-in-Hom comp-arr-dom comp-cod-arr*  
**by** *simp*  
**also have** ... =  $(\{ \langle f \rangle \Downarrow (b \lfloor \otimes \rfloor c) \} \cdot (V f \otimes \{b \Downarrow c\})) \cdot a[V f, \{b\}, \{c\}]$   
**using** *b c f Ide-implies-Arr TensorDiag-preserves-Ide not-is-Tensor-TensorDiagE*  
**by** (*cases b \lfloor \otimes \rfloor c, simp-all; blast*)  
**finally show** *?thesis* **by** *blast*  
**qed**  
**qed**  
**fix** *d e*  
**assume** *I: Diag e*  $\implies \{ (e \lfloor \otimes \rfloor b) \Downarrow c \} \cdot (\{ e \Downarrow b \} \otimes \{ c \})$   
 $= (\{ e \Downarrow b \lfloor \otimes \rfloor c \} \cdot (\{ e \} \otimes \{ b \Downarrow c \})) \cdot a[\{ e \}, \{ b \}, \{ c \}]$   
**assume** *de: Ide d*  $\wedge$  *Ide e*  $\wedge$  *Diag (d*  $\otimes$  *e)*  
**show**  $\{ ((d \otimes e) \lfloor \otimes \rfloor b) \Downarrow c \} \cdot (\{ (d \otimes e) \Downarrow b \} \otimes \{ c \})$   
 $= (\{ (d \otimes e) \Downarrow (b \lfloor \otimes \rfloor c) \} \cdot (\{ d \} \otimes \{ e \} \otimes \{ b \Downarrow c \})) \cdot a[\{ d \} \otimes \{ e \}, \{ b \}, \{ c \}]$   
**proof** –  
**let** *?f = un-Prim d*  
**have** *is-Prim d*  
**using** *de* **by** (*cases d, simp-all*)  
**hence**  $d = \langle ?f \rangle \wedge C.ide \ ?f$   
**using** *de* **by** (*cases d, simp-all*)  
**hence** *d: Ide d*  $\wedge$  *Arr d*  $\wedge$  *Dom d = d*  $\wedge$  *Cod d = d*  $\wedge$  *Diag d*  $\wedge$   
 $d = \langle ?f \rangle \wedge C.ide \ ?f \wedge ide \ \{ d \} \wedge arr \ \{ d \}$   
**using** *de ide-eval-Ide Ide-implies-Arr Diag-Diagonalize(1) Ide-in-Hom*  
*Diag-TensorE [of d e]*  
**by** *simp*  
**have** *Diag e*  $\wedge$   $e \neq \mathcal{I}$   
**using** *de Diag-TensorE* **by** *metis*  
**hence** *e: Ide e*  $\wedge$  *Arr e*  $\wedge$  *Dom e = e*  $\wedge$  *Cod e = e*  $\wedge$  *Diag e*  $\wedge$   
 $e \neq \mathcal{I} \wedge ide \ \{ e \} \wedge arr \ \{ e \}$   
**using** *de Ide-in-Hom* **by** *simp*  
**have** *1: is-Tensor (e*  $\lfloor \otimes \rfloor$  *b)*  $\wedge$  *is-Tensor (b*  $\lfloor \otimes \rfloor$  *c)*  $\wedge$  *is-Tensor (e*  $\lfloor \otimes \rfloor$  *(b*  $\lfloor \otimes \rfloor$  *c))*  
**using** *b c e de not-is-Tensor-TensorDiagE TensorDiag-preserves-Diag*  
*not-is-Tensor-TensorDiagE [of e b \lfloor \otimes \rfloor c]*  
**by** *auto*  
**have**  $\{ ((d \otimes e) \lfloor \otimes \rfloor b) \Downarrow c \} \cdot (\{ (d \otimes e) \Downarrow b \} \otimes \{ c \})$   
 $= ((\{ d \} \otimes \{ (e \lfloor \otimes \rfloor b) \lfloor \otimes \rfloor c \}) \cdot (\{ d \} \otimes \{ (e \lfloor \otimes \rfloor b) \Downarrow c \})) \cdot$   
 $a[\{ d \}, \{ e \lfloor \otimes \rfloor b \}, \{ c \}] \cdot$   
 $((\{ d \} \otimes \{ e \lfloor \otimes \rfloor b \}) \cdot (\{ d \} \otimes \{ e \Downarrow b \})) \cdot a[\{ d \}, \{ e \}, \{ b \}] \otimes \{ c \}$   
**proof** –  
**have**  $\{ ((d \otimes e) \lfloor \otimes \rfloor b) \Downarrow c \}$   
 $= (\{ d \} \otimes \{ (e \lfloor \otimes \rfloor b) \lfloor \otimes \rfloor c \}) \cdot (\{ d \} \otimes \{ (e \lfloor \otimes \rfloor b) \Downarrow c \}) \cdot$   
 $a[\{ d \}, \{ e \lfloor \otimes \rfloor b \}, \{ c \}]$   
**proof** –  
**have**  $((d \otimes e) \lfloor \otimes \rfloor b) \Downarrow c = (d \otimes (e \lfloor \otimes \rfloor b)) \Downarrow c$   
**using** *b c d e de 1 TensorDiag-Diag TensorDiag-preserves-Diag TensorDiag-assoc*

*TensorDiag-Prim not-is-Tensor-Unity*

by *metis*

also have ... =  $(d \Downarrow (e \lfloor \otimes \rfloor b) \lfloor \otimes \rfloor c) \cdot (d \otimes ((e \lfloor \otimes \rfloor b) \Downarrow c)) \cdot \mathbf{a}[d, e \lfloor \otimes \rfloor b, c]$

using *c d 1* by (*cases c*) *simp-all*

also have ... =  $(d \otimes ((e \lfloor \otimes \rfloor b) \lfloor \otimes \rfloor c)) \cdot (d \otimes ((e \lfloor \otimes \rfloor b) \Downarrow c)) \cdot \mathbf{a}[d, e \lfloor \otimes \rfloor b, c]$

by (*metis 1 Diagonalize-Diag TensorDiag-assoc TensorDiag-preserves-Diag(1) b c d e is-Tensor-def red2.simps(4)*)

finally show *?thesis*

using *b c d e TensorDiag-in-Hom red2-in-Hom TensorDiag-preserves-Diag TensorDiag-preserves-Ide*

by *simp*

qed

moreover have  $\{\{d \otimes e\} \Downarrow b\} = (\{d\} \otimes \{e \lfloor \otimes \rfloor b\}) \cdot (\{d\} \otimes \{e \Downarrow b\}) \cdot \mathbf{a}[\{d\}, \{e\}, \{b\}]$

proof –

have  $(d \otimes e) \Downarrow b = (d \Downarrow (e \lfloor \otimes \rfloor b)) \cdot (d \otimes (e \Downarrow b)) \cdot \mathbf{a}[d, e, b]$

using *b c d e de 1 TensorDiag-Prim Diagonalize-Diag*

by (*cases b*) *simp-all*

also have ... =  $(d \otimes (e \lfloor \otimes \rfloor b)) \cdot (d \otimes (e \Downarrow b)) \cdot \mathbf{a}[d, e, b]$

using *b d e 1 TensorDiag-preserves-Diag red2-Diag*

by (*metis Diag.simps(3) de term.disc(12)*)

finally have  $(d \otimes e) \Downarrow b = (d \otimes (e \lfloor \otimes \rfloor b)) \cdot (d \otimes (e \Downarrow b)) \cdot \mathbf{a}[d, e, b]$

by *simp*

thus *?thesis* using *b d e eval-in-hom TensorDiag-in-Hom red2-in-Hom* by *simp*

qed

ultimately show *?thesis* by *argo*

qed

also have ... =  $(\{d\} \otimes \{(e \lfloor \otimes \rfloor b) \Downarrow c\}) \cdot \mathbf{a}[\{d\}, \{e \lfloor \otimes \rfloor b\}, \{c\}] \cdot ((\{d\} \otimes \{e \Downarrow b\}) \otimes \{c\}) \cdot (\mathbf{a}[\{d\}, \{e\}, \{b\}] \otimes \{c\})$

using *b c d e red2-in-Hom TensorDiag-preserves-Ide*

*TensorDiag-preserves-Diag interchange comp-cod-arr comp-assoc*

by *simp*

also have ... =  $(\{d\} \otimes \{(e \lfloor \otimes \rfloor b) \Downarrow c\}) \cdot (\{d\} \otimes (\{e \Downarrow b\} \otimes \{c\})) \cdot \mathbf{a}[\{d\}, \{e\} \otimes \{b\}, \{c\}] \cdot (\mathbf{a}[\{d\}, \{e\}, \{b\}] \otimes \{c\})$

using *b c d e TensorDiag-in-Hom red2-in-Hom TensorDiag-preserves-Ide*

*TensorDiag-preserves-Diag assoc-naturality [of {d} {e ↓ b} {c}]*

*comp-permute [of a[{d}], {e ↓ b}, {c}] ({d} ⊗ {e ↓ b}) ⊗ {c}*

$\{d\} \otimes (\{e \Downarrow b\} \otimes \{c\}) \mathbf{a}[\{d\}, \{e\} \otimes \{b\}, \{c\}]$

by *simp*

also have ... =  $(\{d\} \otimes \{(e \lfloor \otimes \rfloor b) \Downarrow c\}) \cdot (\{e \Downarrow b\} \otimes \{c\}) \cdot \mathbf{a}[\{d\}, \{e\} \otimes \{b\}, \{c\}] \cdot (\mathbf{a}[\{d\}, \{e\}, \{b\}] \otimes \{c\})$

using *b c d e TensorDiag-in-Hom red2-in-Hom TensorDiag-preserves-Ide*

*TensorDiag-preserves-Diag interchange*

*comp-reduce [of {d} ⊗ {(e ↓ b) ↓ c}*

$\{d\} \otimes (\{e \Downarrow b\} \otimes \{c\})$

$\{d\} \otimes \{(e \lfloor \otimes \rfloor b) \Downarrow c\} \cdot (\{e \Downarrow b\} \otimes \{c\})$

$\mathbf{a}[\{d\}, \{e\} \otimes \{b\}, \{c\}] \cdot (\mathbf{a}[\{d\}, \{e\}, \{b\}] \otimes \{c\})$



**by simp**  
**also have** ... = ((( $\{d\} \otimes \{e \downarrow (b \lfloor \otimes \rfloor c)\}$ ) · ( $\{d\} \otimes \{e\} \otimes \{b \downarrow c\}$ )) ·  
( $\{d\} \otimes a[\{e\}, \{b\}, \{c\}]$ )) ·  
 $a[\{d\}, \{e\} \otimes \{b\}, \{c\}] \cdot (a[\{d\}, \{e\}, \{b\}] \otimes \{c\})$   
**using**  $b \ c \ d \ e \ I \ TensorDiag\text{-in-Hom} \ red2\text{-in-Hom} \ TensorDiag\text{-preserves-Ide}$   
*TensorDiag-preserves-Diag interchange*  
**by simp**  
**also have** ... = (( $\{d\} \otimes \{e \downarrow (b \lfloor \otimes \rfloor c)\}$ ) · ( $\{d\} \otimes (\{e\} \otimes \{b \downarrow c\})$ )) ·  
 $a[\{d\}, \{e\}, \{b\} \otimes \{c\}] \cdot a[\{d\} \otimes \{e\}, \{b\}, \{c\}]$   
**using**  $b \ c \ d \ e \ comp\text{-assoc} \ red2\text{-in-Hom} \ TensorDiag\text{-in-Hom} \ ide\text{-eval-Ide}$   
*TensorDiag-preserves-Diag tensor-preserves-ide TensorDiag-preserves-Ide*  
*pentagon*  
**by simp**

The pentagon is used!

**also have** ... = ((( $\{d\} \otimes \{e \lfloor \otimes \rfloor b \lfloor \otimes \rfloor c\}$ ) · ( $\{d\} \otimes \{e \downarrow b \lfloor \otimes \rfloor c\}$ ) ·  
 $a[\{d\}, \{e\}, \{b \lfloor \otimes \rfloor c\}] \cdot ((\{d\} \otimes \{e\}) \otimes \{b \downarrow c\})$ ) ·  
 $a[\{d\} \otimes \{e\}, \{b\}, \{c\}]$   
**using**  $b \ c \ d \ e \ red2\text{-in-Hom} \ TensorDiag\text{-preserves-Ide} \ TensorDiag\text{-preserves-Diag}$   
*assoc-naturality [of  $\{d\} \ \{e\} \ \{b \downarrow c\}] \ comp\text{-cod-arr} \ comp\text{-assoc}$*   
**by simp**  
**also have** ... = ( $\{d \otimes e\} \downarrow (b \lfloor \otimes \rfloor c)$ ) · (( $\{d\} \otimes \{e\}$ ) · ( $\{b \downarrow c\}$ )) ·  
 $a[\{d\} \otimes \{e\}, \{b\}, \{c\}]$   
**proof** –  
**have**  $\{d \otimes e\} \downarrow (b \lfloor \otimes \rfloor c)$   
= ( $\{d\} \otimes \{e \lfloor \otimes \rfloor (b \lfloor \otimes \rfloor c)\}$ ) · ( $\{d\} \otimes \{e \downarrow (b \lfloor \otimes \rfloor c)\}$ ) ·  
 $a[\{d\}, \{e\}, \{b \lfloor \otimes \rfloor c\}]$   
**proof** –  
**have**  $(d \otimes e) \downarrow (b \lfloor \otimes \rfloor c)$   
=  $(d \downarrow (e \lfloor \otimes \rfloor [b \lfloor \otimes \rfloor c])) \cdot (d \otimes (e \downarrow (b \lfloor \otimes \rfloor c))) \cdot a[d, e, b \lfloor \otimes \rfloor c]$   
**using**  $b \ c \ e \ not\text{-is-Tensor-TensorDiagE}$   
**by** (*cases  $b \lfloor \otimes \rfloor c$* ) *auto*  
**also have** ... =  $(d \downarrow (e \lfloor \otimes \rfloor (b \lfloor \otimes \rfloor c))) \cdot (d \otimes (e \downarrow (b \lfloor \otimes \rfloor c))) \cdot$   
 $a[d, e, b \lfloor \otimes \rfloor c]$   
**using**  $b \ c \ d \ e \ 1 \ TensorDiag\text{-preserves-Diag} \ Diagonalize\text{-Diag}$  **by simp**  
**also have** ... =  $(d \otimes (e \lfloor \otimes \rfloor (b \lfloor \otimes \rfloor c))) \cdot (d \otimes (e \downarrow (b \lfloor \otimes \rfloor c))) \cdot$   
 $a[d, e, b \lfloor \otimes \rfloor c]$   
**using**  $b \ c \ d \ e \ 1 \ TensorDiag\text{-preserves-Diag}(1) \ red2\text{-Diag}$   
**by** (*metis Diag.simps(3) de not-is-Tensor-Unity*)  
**finally have**  $(d \otimes e) \downarrow (b \lfloor \otimes \rfloor c)$   
=  $(d \otimes (e \lfloor \otimes \rfloor (b \lfloor \otimes \rfloor c))) \cdot (d \otimes (e \downarrow (b \lfloor \otimes \rfloor c))) \cdot$   
 $a[d, e, b \lfloor \otimes \rfloor c]$   
**by blast**  
**thus** *?thesis*  
**using**  $b \ c \ d \ e \ red2\text{-in-Hom} \ TensorDiag\text{-in-Hom} \ TensorDiag\text{-preserves-Diag}$   
*TensorDiag-preserves-Ide*  
**by simp**  
**qed**  
**thus** *?thesis* **using**  $d \ e \ b \ c$  **by simp**

qed  
 finally show *?thesis* by *simp*  
 qed  
 qed  
 thus *?thesis* using *assms(1)* by *blast*  
 qed  
 ultimately show *?thesis* by *blast*  
 qed

lemma *coherent-Assoc-Ide*:

assumes *Ide a* and *Ide b* and *Ide c*

shows *coherent a[a, b, c]*

proof –

have *a*:  $Ide\ a \wedge Arr\ a \wedge Dom\ a = a \wedge Cod\ a = a \wedge$   
 $ide\ \{a\} \wedge ide\ \{\llbracket a \rrbracket\} \wedge \llbracket \{a\} \rrbracket : \{a\} \rightarrow \{\llbracket a \rrbracket\}$

using *assms Ide-implies-Arr Ide-in-Hom Diagonalize-preserves-Ide red-in-Hom* by *auto*

have *b*:  $Ide\ b \wedge Arr\ b \wedge Dom\ b = b \wedge Cod\ b = b \wedge$

$ide\ \{b\} \wedge ide\ \{\llbracket b \rrbracket\} \wedge \llbracket \{b\} \rrbracket : \{b\} \rightarrow \{\llbracket b \rrbracket\}$

using *assms Ide-implies-Arr Ide-in-Hom Diagonalize-preserves-Ide red-in-Hom* by *auto*

have *c*:  $Ide\ c \wedge Arr\ c \wedge Dom\ c = c \wedge Cod\ c = c \wedge$

$ide\ \{c\} \wedge ide\ \{\llbracket c \rrbracket\} \wedge \llbracket \{c\} \rrbracket : \{c\} \rightarrow \{\llbracket c \rrbracket\}$

using *assms Ide-implies-Arr Ide-in-Hom Diagonalize-preserves-Ide red-in-Hom* by *auto*

have  $\{\llbracket Cod\ a[a, b, c] \rrbracket\} \cdot \{a[a, b, c]\}$

$= (\{\llbracket a \rrbracket \rrbracket \downarrow (\llbracket b \rrbracket \llbracket \otimes \rrbracket \llbracket c \rrbracket)\} \cdot (\{\llbracket a \rrbracket\} \otimes (\{\llbracket b \rrbracket \rrbracket \downarrow \llbracket c \rrbracket\})) \cdot$   
 $(\{\llbracket a \rrbracket\} \otimes \{\llbracket b \rrbracket\} \otimes \{\llbracket c \rrbracket\})) \cdot a[\{\llbracket a \rrbracket\}, \{\llbracket b \rrbracket\}, \{\llbracket c \rrbracket\}]$

using *a b c red-in-Hom red2-in-Hom Diagonalize-in-Hom Diag-Diagonalize*

*Diagonalize-preserves-Ide interchange Ide-in-Hom eval-red-Tensor*

*comp-cod-arr [of \{\llbracket a \rrbracket\}]*

by *simp*

also have ... =  $((\{\llbracket a \rrbracket \rrbracket \downarrow (\llbracket b \rrbracket \llbracket \otimes \rrbracket \llbracket c \rrbracket)\} \cdot (\{\llbracket a \rrbracket\} \otimes \{\llbracket b \rrbracket \rrbracket \downarrow \llbracket c \rrbracket\})) \cdot$

$a[\{\llbracket a \rrbracket\}, \{\llbracket b \rrbracket\}, \{\llbracket c \rrbracket\}]) \cdot ((\{\llbracket a \rrbracket\} \otimes \{\llbracket b \rrbracket\}) \otimes \{\llbracket c \rrbracket\})$

using *a b c red-in-Hom Diag-Diagonalize TensorDiag-preserves-Diag*

*assoc-naturality [of \{\llbracket a \rrbracket\} \{\llbracket b \rrbracket\} \{\llbracket c \rrbracket\}] comp-assoc*

by *simp*

also have ... =  $(\{\llbracket (\llbracket a \rrbracket \llbracket \otimes \rrbracket \llbracket b \rrbracket) \rrbracket \rrbracket \downarrow \llbracket c \rrbracket\} \cdot (\{\llbracket a \rrbracket \rrbracket \downarrow \llbracket b \rrbracket\} \otimes \{\llbracket \llbracket c \rrbracket \rrbracket\})) \cdot$

$((\{\llbracket a \rrbracket\} \otimes \{\llbracket b \rrbracket\}) \otimes \{\llbracket c \rrbracket\})$

using *a b c Diag-Diagonalize Diagonalize-preserves-Ide coherence-key-fact* by *simp*

also have ... =  $\{\llbracket a[a, b, c] \rrbracket\} \cdot \{\llbracket Dom\ a[a, b, c] \rrbracket\}$

using *a b c red-in-Hom red2-in-Hom TensorDiag-preserves-Diag*

*Diagonalize-preserves-Ide TensorDiag-preserves-Ide Diag-Diagonalize interchange*

*eval-red-Tensor TensorDiag-assoc comp-cod-arr [of \{\llbracket c \rrbracket\}]*

*comp-cod-arr [of \{\llbracket (\llbracket a \rrbracket \llbracket \otimes \rrbracket \llbracket b \rrbracket) \rrbracket \rrbracket \downarrow \llbracket c \rrbracket\} \cdot (\{\llbracket a \rrbracket \rrbracket \downarrow \llbracket b \rrbracket\} \cdot (\{\llbracket a \rrbracket\} \otimes \{\llbracket b \rrbracket\}) \otimes \{\llbracket c \rrbracket\})]*

*comp-assoc*

by *simp*

finally show *?thesis* by *blast*

qed

lemma *coherent-Assoc'-Ide*:

**assumes** *Ide a and Ide b and Ide c*  
**shows** *coherent a<sup>-1</sup>[a, b, c]*  
**proof** –  
**have** *Can a[a, b, c]* **using** *assms Ide-implies-Can* **by** *simp*  
**moreover have** *a<sup>-1</sup>[a, b, c] = Inv a[a, b, c]*  
**using** *assms Inv-Ide* **by** *simp*  
**ultimately show** *?thesis*  
**using** *assms Ide-implies-Can coherent-Assoc-Ide Inv-Ide*  
*Can-implies-coherent-iff-coherent-Inv*  
**by** *metis*  
**qed**

The next lemma implies coherence for the special case of a term that is the tensor of two diagonal arrows.

**lemma** *eval-red2-naturality:*  
**assumes** *Diag t and Diag u*  
**shows**  $\{\{Cod\ t \Downarrow Cod\ u\} \cdot (\{\{t\} \otimes \{u\}\}) = \{\{t\} [\otimes] u\} \cdot \{\{Dom\ t \Downarrow Dom\ u\}\}$   
**proof** –  
**have**  $*$ :  $\bigwedge t\ u. Diag\ (t \otimes u) \implies arr\ \{\{t\}\} \wedge arr\ \{\{u\}\}$   
**using** *Diag-implies-Arr* **by** *force*  
**have**  $t = \mathcal{I} \implies ?thesis$   
**using** *assms Diag-implies-Arr lunit-naturality [of \{u\}]*  
*Arr-implies-Ide-Dom Arr-implies-Ide-Cod comp-cod-arr*  
**by** *simp*  
**moreover have**  $t \neq \mathcal{I} \wedge u = \mathcal{I} \implies ?thesis$   
**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod*  
*Diag-implies-Arr Dom-preserves-Diag Cod-preserves-Diag*  
*eval-red2-Diag-Unity runit-naturality [of \{t\}]*  
**by** *simp*  
**moreover have**  $t \neq \mathcal{I} \wedge u \neq \mathcal{I} \implies ?thesis$   
**using**  $*$  *Arr-implies-Ide-Dom Arr-implies-Ide-Cod*  
*Diag-implies-Arr Dom-preserves-Diag Cod-preserves-Diag*  
**apply** (*induct t, simp-all*)  
**proof** –  
**fix**  $f$   
**assume**  $f: C.arr\ f$   
**assume**  $u \neq \mathcal{I}$   
**hence**  $u: u \neq \mathcal{I} \wedge$   
 $Diag\ u \wedge Diag\ (Dom\ u) \wedge Diag\ (Cod\ u) \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge$   
 $arr\ \{\{u\}\} \wedge arr\ \{\{Dom\ u\}\} \wedge arr\ \{\{Cod\ u\}\} \wedge ide\ \{\{Dom\ u\}\} \wedge ide\ \{\{Cod\ u\}\}$   
**using** *assms(2) Diag-implies-Arr Dom-preserves-Diag Cod-preserves-Diag*  
*Arr-implies-Ide-Dom Arr-implies-Ide-Cod*  
**by** *simp*  
**hence**  $1: Dom\ u \neq \mathcal{I} \wedge Cod\ u \neq \mathcal{I}$  **using**  $u$  **by** (*cases u, simp-all*)  
**show**  $\{\{C.cod\ f\} \Downarrow Cod\ u\} \cdot (Vf \otimes \{u\}) = (Vf \otimes \{u\}) \cdot \{\{C.dom\ f\} \Downarrow Dom\ u\}$   
**using**  $f\ u\ 1$  *Diag-implies-Arr red2-Diag comp-arr-dom comp-cod-arr* **by** *simp*  
**next**  
**fix**  $v\ w$   
**assume**  $I2: \llbracket w \neq Unity; Diag\ w \rrbracket \implies$

$\{\!\{ \text{Cod } w \Downarrow \text{Cod } u \}\!\} \cdot (\{\!\{ w \}\!\} \otimes \{\!\{ u \}\!\}) = \{\!\{ w \llbracket \otimes \rrbracket u \}\!\} \cdot \{\!\{ \text{Dom } w \Downarrow \text{Dom } u \}\!\}$

**assume**  $u \neq \mathcal{I}$

**hence**  $u: u \neq \mathcal{I} \wedge \text{Arr } u \wedge \text{Arr } (\text{Dom } u) \wedge \text{Arr } (\text{Cod } u) \wedge$   
 $\text{Diag } u \wedge \text{Diag } (\text{Dom } u) \wedge \text{Diag } (\text{Cod } u) \wedge \text{Ide } (\text{Dom } u) \wedge \text{Ide } (\text{Cod } u) \wedge$   
 $\text{arr } \{\!\{ u \}\!\} \wedge \text{arr } \{\!\{ \text{Dom } u \}\!\} \wedge \text{arr } \{\!\{ \text{Cod } u \}\!\} \wedge \text{ide } \{\!\{ \text{Dom } u \}\!\} \wedge \text{ide } \{\!\{ \text{Cod } u \}\!\}$

**using** *assms(2) Diag-implies-Arr Dom-preserves-Diag Cod-preserves-Diag*  
*Arr-implies-Ide-Dom Arr-implies-Ide-Cod*

**by** *simp*

**assume**  $vw: \text{Diag } (v \otimes w)$

**let**  $?f = \text{un-Prim } v$

**have**  $v = \langle ?f \rangle \wedge C.\text{arr } ?f$

**using**  $vw$  **by** (*metis Diag-TensorE(1) Diag-TensorE(2)*)

**hence**  $\text{Arr } v \wedge v = \langle \text{un-Prim } v \rangle \wedge C.\text{arr } ?f \wedge \text{Diag } v$  **by** (*cases v; simp*)

**hence**  $v: v = \langle ?f \rangle \wedge C.\text{arr } ?f \wedge \text{Arr } v \wedge \text{Ide } (\text{Dom } v) \wedge \text{Ide } (\text{Cod } v) \wedge \text{Diag } v \wedge$   
 $\text{Diag } (\text{Dom } v) \wedge \text{arr } \{\!\{ v \}\!\} \wedge \text{arr } \{\!\{ \text{Dom } v \}\!\} \wedge \text{arr } \{\!\{ \text{Cod } v \}\!\} \wedge$   
 $\text{ide } \{\!\{ \text{Dom } v \}\!\} \wedge \text{ide } \{\!\{ \text{Cod } v \}\!\}$

**by** (*cases v, simp-all*)

**have**  $\text{Diag } w \wedge w \neq \mathcal{I}$

**using**  $vw$  **by** (*metis Diag.simps(3)*)

**hence**  $w: w \neq \mathcal{I} \wedge \text{Arr } w \wedge \text{Arr } (\text{Dom } w) \wedge \text{Arr } (\text{Cod } w) \wedge$   
 $\text{Diag } w \wedge \text{Diag } (\text{Dom } w) \wedge \text{Diag } (\text{Cod } w) \wedge$   
 $\text{Ide } (\text{Dom } w) \wedge \text{Ide } (\text{Cod } w) \wedge$   
 $\text{arr } \{\!\{ w \}\!\} \wedge \text{arr } \{\!\{ \text{Dom } w \}\!\} \wedge \text{arr } \{\!\{ \text{Cod } w \}\!\} \wedge \text{ide } \{\!\{ \text{Dom } w \}\!\} \wedge \text{ide } \{\!\{ \text{Cod } w \}\!\}$

**using**  $vw * \text{Diag-implies-Arr Dom-preserves-Diag Cod-preserves-Diag Arr-implies-Ide-Dom}$   
 $\text{Arr-implies-Ide-Cod ide-eval-Ide Ide-implies-Arr Ide-in-Hom}$

**by** *simp*

**show**  $\{\!\{ (\text{Cod } v \otimes \text{Cod } w) \Downarrow \text{Cod } u \}\!\} \cdot (\{\!\{ v \}\!\} \otimes \{\!\{ w \}\!\}) \otimes \{\!\{ u \}\!\}$   
 $= \{\!\{ (v \otimes w) \llbracket \otimes \rrbracket u \}\!\} \cdot \{\!\{ (\text{Dom } v \otimes \text{Dom } w) \Downarrow \text{Dom } u \}\!\}$

**proof** –

**have**  $u': \text{Dom } u \neq \mathcal{I} \wedge \text{Cod } u \neq \mathcal{I}$  **using**  $u$  **by** (*cases u simp-all*)

**have**  $w': \text{Dom } w \neq \mathcal{I} \wedge \text{Cod } w \neq \mathcal{I}$  **using**  $w$  **by** (*cases w simp-all*)

**have**  $D: \text{Diag } (\text{Dom } v \otimes (\text{Dom } w \llbracket \otimes \rrbracket \text{Dom } u))$

**proof** –

**have**  $\text{Dom } w \llbracket \otimes \rrbracket \text{Dom } u \neq \mathcal{I}$

**using**  $u$   $u'$   $w$   $w'$  *not-is-Tensor-TensorDiagE* **by** *blast*

**moreover** **have**  $\text{Diag } (\text{Dom } w \llbracket \otimes \rrbracket \text{Dom } u)$

**using**  $u$   $w$  *TensorDiag-preserves-Diag* **by** *simp*

**moreover** **have**  $\text{Dom } v = \langle C.\text{dom } ?f \rangle$

**using**  $v$  **by** (*cases v, simp-all*)

**ultimately show** *?thesis*

**using**  $u$   $v$   $w$  *TensorDiag-preserves-Diag* **by** *auto*

**qed**

**have**  $C: \text{Diag } (\text{Cod } v \otimes (\text{Cod } w \llbracket \otimes \rrbracket \text{Cod } u))$

**proof** –

**have**  $\text{Cod } w \llbracket \otimes \rrbracket \text{Cod } u \neq \mathcal{I}$

**using**  $u$   $u'$   $w$   $w'$  *not-is-Tensor-TensorDiagE* **by** *blast*

**moreover** **have**  $\text{Diag } (\text{Cod } w \llbracket \otimes \rrbracket \text{Cod } u)$

**using**  $u$   $w$  *TensorDiag-preserves-Diag* **by** *simp*

**moreover** **have**  $\text{Cod } v = \langle C.\text{cod } ?f \rangle$

**using**  $v$  **by** (*cases v, simp-all*)  
**ultimately show** *?thesis*  
**using**  $u v w$  **by** (*cases Cod w  $\lfloor \otimes \rfloor$  Cod u*) *simp-all*  
**qed**  
**have**  $\{ (Cod\ v \otimes Cod\ w) \Downarrow Cod\ u \} \cdot ((\{v\} \otimes \{w\}) \otimes \{u\})$   
 $= (\{Cod\ v \Downarrow (Cod\ w \lfloor \otimes \rfloor Cod\ u)\} \cdot (\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}) \cdot$   
 $a[\{Cod\ v\}, \{Cod\ w\}, \{Cod\ u\}]) \cdot ((\{v\} \otimes \{w\}) \otimes \{u\})$   
**proof** –  
**have**  $(Cod\ v \otimes Cod\ w) \Downarrow Cod\ u$   
 $= (Cod\ v \Downarrow (Cod\ w \lfloor \otimes \rfloor \lfloor Cod\ u \rfloor)) \cdot (Cod\ v \otimes Cod\ w \Downarrow Cod\ u) \cdot$   
 $a[Cod\ v, Cod\ w, Cod\ u]$   
**using**  $u v w$  **by** (*cases u, simp-all*)  
**hence**  $\{ (Cod\ v \otimes Cod\ w) \Downarrow Cod\ u \}$   
 $= \{Cod\ v \Downarrow (Cod\ w \lfloor \otimes \rfloor Cod\ u)\} \cdot (\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}) \cdot$   
 $a[\{Cod\ v\}, \{Cod\ w\}, \{Cod\ u\}]$   
**using**  $u v w$  **by** *simp*  
**thus** *?thesis* **by** *argo*  
**qed**  
**also have**  $\dots = ((\{Cod\ v\} \otimes \{Cod\ w \lfloor \otimes \rfloor Cod\ u\}) \cdot (\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}) \cdot$   
 $a[\{Cod\ v\}, \{Cod\ w\}, \{Cod\ u\}]) \cdot ((\{v\} \otimes \{w\}) \otimes \{u\})$   
**using**  $u v w$  *C red2-Diag* **by** *simp*  
**also have**  $\dots = ((\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}) \cdot a[\{Cod\ v\}, \{Cod\ w\}, \{Cod\ u\}]) \cdot$   
 $((\{v\} \otimes \{w\}) \otimes \{u\})$   
**proof** –  
**have**  $(\{Cod\ v\} \otimes \{Cod\ w \lfloor \otimes \rfloor Cod\ u\}) \cdot (\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\})$   
 $= \{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}$   
**using**  $u v w$  *comp-cod-arr red2-in-Hom* **by** *simp*  
**moreover have**  
 $seq(\{Cod\ v\} \otimes \{Cod\ w \lfloor \otimes \rfloor Cod\ u\}) (\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\})$   
**using**  $u v w$  *red2-in-Hom TensorDiag-in-Hom Ide-in-Hom* **by** *simp*  
**moreover have**  $seq(\{Cod\ v\} \otimes \{Cod\ w \Downarrow Cod\ u\}) a[\{Cod\ v\}, \{Cod\ w\}, \{Cod\ u\}]$   
**using**  $u v w$  *red2-in-Hom* **by** *simp*  
**ultimately show** *?thesis*  
**using**  $u v w$  *comp-reduce* **by** *presburger*  
**qed**  
**also have**  
 $\dots = (\{v\} \otimes \{w \lfloor \otimes \rfloor u\}) \cdot \{Dom\ w \Downarrow Dom\ u\} \cdot a[\{Dom\ v\}, \{Dom\ w\}, \{Dom\ u\}]$   
**using**  $u v w$  *I2 red2-in-Hom TensorDiag-in-Hom interchange comp-reduce*  
*assoc-naturality* [of  $\{v\} \{w\} \{u\}$ ] *comp-cod-arr comp-assoc*  
**by** *simp*  
**also have**  $\dots = (\{v\} \otimes \{w \lfloor \otimes \rfloor u\}) \cdot (\{Dom\ v\} \otimes \{Dom\ w \Downarrow Dom\ u\}) \cdot$   
 $a[\{Dom\ v\}, \{Dom\ w\}, \{Dom\ u\}]$   
**using**  $u v w$  *red2-in-Hom TensorDiag-in-Hom interchange comp-reduce comp-arr-dom*  
**by** *simp*  
**also have**  $\dots = \{v \lfloor \otimes \rfloor w \lfloor \otimes \rfloor u\} \cdot (\{Dom\ v\} \otimes \{Dom\ w \Downarrow Dom\ u\}) \cdot$   
 $a[\{Dom\ v\}, \{Dom\ w\}, \{Dom\ u\}]$   
**using**  $u u' v w$  *not-is-Tensor-TensorDiagE TensorDiag-Prim* [of  $w \lfloor \otimes \rfloor u$  *?f*]  
**by** *force*  
**also have**  $\dots = \{v \lfloor \otimes \rfloor w \lfloor \otimes \rfloor u\} \cdot \{Dom\ v \lfloor \otimes \rfloor Dom\ w \lfloor \otimes \rfloor Dom\ u\} \cdot$

$(\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$

**proof** –

**have**

$$\{\!\{v \lfloor \otimes \rfloor w \lfloor \otimes \rfloor u\}\!\} \cdot (\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot$$

$$a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}] =$$

$$(\{\!\{v \lfloor \otimes \rfloor w \lfloor \otimes \rfloor u\}\!\} \cdot \{\!\{Dom\ v \lfloor \otimes \rfloor Dom\ w \lfloor \otimes \rfloor Dom\ u\}\!\}) \cdot$$

$$(\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$$

**using** *u v w comp-arr-dom TensorDiag-in-Hom TensorDiag-preserves-Diag* **by** *simp*

**also have** ... =  $\{\!\{v \lfloor \otimes \rfloor w \lfloor \otimes \rfloor u\}\!\} \cdot \{\!\{Dom\ v \lfloor \otimes \rfloor Dom\ w \lfloor \otimes \rfloor Dom\ u\}\!\} \cdot$

$$(\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$$

**using** *comp-assoc* **by** *simp*

**finally show** *?thesis* **by** *blast*

**qed**

**also have** ... =  $\{\!\{(v \lfloor \otimes \rfloor w) \lfloor \otimes \rfloor u\}\!\} \cdot \{\!\{(Dom\ v \otimes Dom\ w) \Downarrow Dom\ u\}\!\}$

**proof** –

**have**

$$\{\!\{(Dom\ v \otimes Dom\ w) \Downarrow Dom\ u\}\!\}$$

$$= \{\!\{Dom\ v \Downarrow (Dom\ w \lfloor \otimes \rfloor Dom\ u)\}\!\} \cdot (\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot$$

$$a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$$

**proof** –

**have**  $(Dom\ v \otimes Dom\ w) \Downarrow Dom\ u$

$$= (Dom\ v \Downarrow (Dom\ w \lfloor \otimes \rfloor \lfloor Dom\ u \rfloor)) \cdot (Dom\ v \otimes (Dom\ w \Downarrow Dom\ u)) \cdot$$

$$a[Dom\ v, Dom\ w, Dom\ u]$$

**using** *u u' v w red2-in-Hom TensorDiag-in-Hom tensor-in-hom Ide-in-Hom*

**by** *(cases u) auto*

**thus** *?thesis*

**using** *u v w red2-in-Hom* **by** *simp*

**qed**

**also have**

$$\dots = \{\!\{Dom\ v \lfloor \otimes \rfloor Dom\ w \lfloor \otimes \rfloor Dom\ u\}\!\} \cdot (\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot$$

$$a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$$

**using** *D TensorDiag-Diag red2-Diag* **by** *simp*

**finally have**

$$\{\!\{(Dom\ v \otimes Dom\ w) \Downarrow Dom\ u\}\!\}$$

$$= \{\!\{Dom\ v \lfloor \otimes \rfloor Dom\ w \lfloor \otimes \rfloor Dom\ u\}\!\} \cdot (\{\!\{Dom\ v\}\!\} \otimes \{\!\{Dom\ w \Downarrow Dom\ u\}\!\}) \cdot$$

$$a[\{\!\{Dom\ v\}\!\}, \{\!\{Dom\ w\}\!\}, \{\!\{Dom\ u\}\!\}]$$

**by** *blast*

**thus** *?thesis*

**using** *assms v w TensorDiag-assoc* **by** *auto*

**qed**

**finally show** *?thesis*

**using** *vw TensorDiag-Diag* **by** *simp*

**qed**

**qed**

**ultimately show** *?thesis* **by** *blast*

**qed**

**lemma** *Tensor-preserves-coherent:*

**assumes** *Arr t and Arr u and coherent t and coherent u*

shows *coherent* ( $t \otimes u$ )

**proof** –

**have**  $t$ :  $Arr\ t \wedge Ide\ (Dom\ t) \wedge Ide\ (Cod\ t) \wedge Ide\ [Dom\ t] \wedge Ide\ [Cod\ t] \wedge$   
 $arr\ \{\!|t|\!\} \wedge arr\ \{\!|Dom\ t|\!\} \wedge ide\ \{\!|Dom\ t|\!\} \wedge arr\ \{\!|Cod\ t|\!\} \wedge ide\ \{\!|Cod\ t|\!\}$   
**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod Diagonalize-preserves-Ide*  
**by** *auto*

**have**  $u$ :  $Arr\ u \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge Ide\ [Dom\ u] \wedge Ide\ [Cod\ u] \wedge$   
 $arr\ \{\!|u|\!\} \wedge arr\ \{\!|Dom\ u|\!\} \wedge ide\ \{\!|Dom\ u|\!\} \wedge arr\ \{\!|Cod\ u|\!\} \wedge ide\ \{\!|Cod\ u|\!\}$   
**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod Diagonalize-preserves-Ide*  
**by** *auto*

**have**  $\{\!|Cod\ (t \otimes u)\!\} \cdot (\{\!|t|\!\} \otimes \{\!|u|\!\})$   
 $= (\{\!|[Cod\ t]\!\} \Downarrow [Cod\ u]\!\} \cdot (\{\!|Cod\ t\!\} \otimes \{\!|Cod\ u\!\})) \cdot (\{\!|t|\!\} \otimes \{\!|u|\!\})$   
**using** *t u eval-red-Tensor by simp*

**also have**  $\dots = \{\!|[Cod\ t]\!\} \Downarrow [Cod\ u]\!\} \cdot (\{\!|Cod\ t\!\} \otimes \{\!|Cod\ u\!\}) \cdot (\{\!|t|\!\} \otimes \{\!|u|\!\})$   
**using** *comp-assoc by simp*

**also have**  $\dots = \{\!|[Cod\ t]\!\} \Downarrow [Cod\ u]\!\} \cdot (\{\!|[t]\!\} \otimes \{\!|[u]\!\}) \cdot (\{\!|Dom\ t\!\} \otimes \{\!|Dom\ u\!\})$   
**using** *assms t u Diagonalize-in-Hom red-in-Hom interchange by simp*

**also have**  $\dots = (\{\!|[Cod\ t]\!\} \Downarrow [Cod\ u]\!\} \cdot (\{\!|[t]\!\} \otimes \{\!|[u]\!\})) \cdot (\{\!|Dom\ t\!\} \otimes \{\!|Dom\ u\!\})$   
**using** *comp-assoc by simp*

**also have**  $\dots = (\{\!|[t]\!\} [ \otimes ] [u]\!\} \cdot \{\!|[Dom\ t]\!\} \Downarrow [Dom\ u]\!\}) \cdot (\{\!|Dom\ t\!\} \otimes \{\!|Dom\ u\!\})$   
**using** *assms t u Diag-Diagonalize Diagonalize-in-Hom*  
*eval-red2-naturality [of Diagonalize t Diagonalize u]*  
**by** *simp*

**also have**  $\dots = \{\!|[t]\!\} [ \otimes ] [u]\!\} \cdot \{\!|[Dom\ t]\!\} \Downarrow [Dom\ u]\!\} \cdot (\{\!|Dom\ t\!\} \otimes \{\!|Dom\ u\!\})$   
**using** *comp-assoc by simp*

**also have**  $\dots = \{\!|[t]\!\} [ \otimes ] [u]\!\} \cdot \{\!|(Dom\ t \otimes Dom\ u)\!\}$   
**using** *t u eval-red-Tensor by simp*

**finally have**  $\{\!|Cod\ (t \otimes u)\!\} \cdot (\{\!|t|\!\} \otimes \{\!|u|\!\}) = \{\!|[t]\!\} [ \otimes ] [u]\!\} \cdot \{\!|(Dom\ t \otimes Dom\ u)\!\}$   
**by** *blast*

**thus** *?thesis using t u by simp*

**qed**

**lemma** *Comp-preserves-coherent*:

**assumes**  $Arr\ t$  **and**  $Arr\ u$  **and**  $Dom\ t = Cod\ u$

**and** *coherent t and coherent u*

**shows** *coherent* ( $t \cdot u$ )

**proof** –

**have**  $t$ :  $Arr\ t \wedge Ide\ (Dom\ t) \wedge Ide\ (Cod\ t) \wedge Ide\ [Dom\ t] \wedge Ide\ [Cod\ t] \wedge$   
 $arr\ \{\!|t|\!\} \wedge arr\ \{\!|Dom\ t|\!\} \wedge ide\ \{\!|Dom\ t|\!\} \wedge arr\ \{\!|Cod\ t|\!\} \wedge ide\ \{\!|Cod\ t|\!\}$   
**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod Diagonalize-preserves-Ide*  
**by** *auto*

**have**  $u$ :  $Arr\ u \wedge Ide\ (Dom\ u) \wedge Ide\ (Cod\ u) \wedge Ide\ [Dom\ u] \wedge Ide\ [Cod\ u] \wedge$   
 $arr\ \{\!|u|\!\} \wedge arr\ \{\!|Dom\ u|\!\} \wedge ide\ \{\!|Dom\ u|\!\} \wedge arr\ \{\!|Cod\ u|\!\} \wedge ide\ \{\!|Cod\ u|\!\}$   
**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod Diagonalize-preserves-Ide*  
**by** *auto*

**have**  $\{\!|Cod\ (t \cdot u)\!\} \cdot \{\!|t \cdot u|\!\} = \{\!|Cod\ t\!\} \cdot \{\!|t|\!\} \cdot \{\!|u|\!\}$   
**using** *t u by simp*

**also have**  $\dots = (\{\!|Cod\ t\!\} \cdot \{\!|t|\!\}) \cdot \{\!|u|\!\}$

**proof** –

```

have seq  $\{Cod \downarrow\} \{t\}$ 
  using assms t red-in-Hom by (intro seqI, auto)
moreover have seq  $\{t\} \{u\}$ 
  using assms t u by auto
ultimately show ?thesis using comp-assoc by auto
qed
also have ... =  $\{[t \cdot u]\} \cdot \{Dom (t \cdot u)\downarrow\}$ 
  using t u assms red-in-Hom Diag-Diagonalize comp-assoc
  by (simp add: Diag-implies-Arr eval-CompDiag)
finally show coherent (t · u) by blast
qed

```

The main result: “Every formal arrow is coherent.”

**theorem** *coherence*:

**assumes** *Arr t*

**shows** *coherent t*

**proof** –

have *Arr t*  $\implies$  *coherent t*

**proof** (*induct t*)

fix *u v*

show  $\llbracket Arr\ u \implies\ coherent\ u;\ Arr\ v \implies\ coherent\ v \rrbracket \implies Arr\ (u \otimes v)$   
 $\implies coherent\ (u \otimes v)$

using *Tensor-preserves-coherent* by *simp*

show  $\llbracket Arr\ u \implies\ coherent\ u;\ Arr\ v \implies\ coherent\ v \rrbracket \implies Arr\ (u \cdot v)$   
 $\implies coherent\ (u \cdot v)$

using *Comp-preserves-coherent* by *simp*

next

show *coherent  $\mathcal{I}$*  by *simp*

fix *f*

show *Arr  $\langle f \rangle \implies coherent \langle f \rangle$*  by *simp*

next

fix *t*

assume *I*: *Arr t*  $\implies$  *coherent t*

show *Lunit*: *Arr  $\mathbf{l}[t] \implies coherent \mathbf{l}[t]$*

using *I Arr-implies-Ide-Dom coherent-Lunit-Ide Ide-in-Hom Ide-implies-Arr*  
*Comp-preserves-coherent [of t  $\mathbf{l}[Dom\ t]$  Diagonalize-Comp-Arr-Dom  $\mathbf{l}$ -ide-simp*  
 by *auto*

show *Runit*: *Arr  $\mathbf{r}[t] \implies coherent \mathbf{r}[t]$*

using *I Arr-implies-Ide-Dom coherent-Runit-Ide Ide-in-Hom Ide-implies-Arr*  
*Comp-preserves-coherent [of t  $\mathbf{r}[Dom\ t]$  Diagonalize-Comp-Arr-Dom  $\mathbf{r}$ -ide-simp*  
 by *auto*

show *Arr  $\mathbf{l}^{-1}[t] \implies coherent \mathbf{l}^{-1}[t]$*

**proof** –

assume *Arr  $\mathbf{l}^{-1}[t]$*

hence *t*: *Arr t* by *simp*

have *coherent ( $\mathbf{l}^{-1}[Cod\ t] \cdot t$ )*

using *t I Arr-implies-Ide-Cod coherent-Lunit'-Ide Ide-in-Hom*  
*Comp-preserves-coherent [of  $\mathbf{l}^{-1}[Cod\ t]$  t]*



```

    by fastforce
  thus ?thesis
    using t Arr-implies-Ide-Cod Ide-implies-Arr Ide-in-Hom Diagonalize-Comp-Cod-Arr
      eval-in-hom  $\mathcal{V}$ .is-natural-2 [of  $\{t\}$ ]
    by force
qed
show Arr  $\mathbf{r}^{-1}[t] \implies$  coherent  $\mathbf{r}^{-1}[t]$ 
proof -
  assume Arr  $\mathbf{r}^{-1}[t]$ 
  hence t: Arr t by simp
  have coherent ( $\mathbf{r}^{-1}[\text{Cod } t] \cdot t$ )
    using t I Arr-implies-Ide-Cod coherent-Runit'-Ide Ide-in-Hom
      Comp-preserves-coherent [of  $\mathbf{r}^{-1}[\text{Cod } t]$  t]
  by fastforce
  thus ?thesis
    using t Arr-implies-Ide-Cod Ide-implies-Arr Ide-in-Hom Diagonalize-Comp-Cod-Arr
      eval-in-hom  $\mathcal{Q}$ '.is-natural-2 [of  $\{t\}$ ]
    by force
qed
next
fix t u v
assume I1: Arr t  $\implies$  coherent t
assume I2: Arr u  $\implies$  coherent u
assume I3: Arr v  $\implies$  coherent v
show Arr  $\mathbf{a}[t, u, v] \implies$  coherent  $\mathbf{a}[t, u, v]$ 
proof -
  assume tuv: Arr  $\mathbf{a}[t, u, v]$ 
  have t: Arr t using tuv by simp
  have u: Arr u using tuv by simp
  have v: Arr v using tuv by simp
  have coherent ((t  $\otimes$  u  $\otimes$  v)  $\cdot$   $\mathbf{a}[\text{Dom } t, \text{Dom } u, \text{Dom } v]$ )
  proof -
    have Arr (t  $\otimes$  u  $\otimes$  v)  $\wedge$  coherent (t  $\otimes$  u  $\otimes$  v)
  proof
    have 1: Arr t  $\wedge$  coherent t using t I1 by simp
    have 2: Arr (u  $\otimes$  v)  $\wedge$  coherent (u  $\otimes$  v)
      using u v I2 I3 Tensor-preserves-coherent by force
    show Arr (t  $\otimes$  u  $\otimes$  v) using 1 2 by simp
    show coherent (t  $\otimes$  u  $\otimes$  v)
      using 1 2 Tensor-preserves-coherent by blast
  qed
  moreover have Arr  $\mathbf{a}[\text{Dom } t, \text{Dom } u, \text{Dom } v]$ 
    using t u v Arr-implies-Ide-Dom by simp
  moreover have coherent  $\mathbf{a}[\text{Dom } t, \text{Dom } u, \text{Dom } v]$ 
    using t u v Arr-implies-Ide-Dom coherent-Assoc-Ide by blast
  moreover have Dom (t  $\otimes$  u  $\otimes$  v) = Cod  $\mathbf{a}[\text{Dom } t, \text{Dom } u, \text{Dom } v]$ 
    using t u v Arr-implies-Ide-Dom Ide-in-Hom by simp
  ultimately show ?thesis
    using t u v Arr-implies-Ide-Dom Ide-implies-Arr

```

$Comp\text{-preserves-coherent}$  [of  $t \otimes u \otimes v$   $\mathbf{a}[Dom\ t, Dom\ u, Dom\ v]$ ]  
 by *blast*  
**qed**  
**moreover have**  $Par\ \mathbf{a}[t, u, v]\ ((t \otimes u \otimes v) \cdot \mathbf{a}[Dom\ t, Dom\ u, Dom\ v])$   
 using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-implies-Arr}\ Ide\text{-in-Hom}$  by *simp*  
**moreover have**  $\lfloor \mathbf{a}[t, u, v] \rfloor = \lfloor (t \otimes u \otimes v) \cdot \mathbf{a}[Dom\ t, Dom\ u, Dom\ v] \rfloor$   
**proof** –  
 have  $(\lfloor t \rfloor\ \lfloor \otimes \rfloor\ \lfloor u \rfloor)\ \lfloor \otimes \rfloor\ \lfloor v \rfloor$   
 $= (\lfloor t \rfloor\ \lfloor \otimes \rfloor\ \lfloor u \rfloor\ \lfloor \otimes \rfloor\ \lfloor v \rfloor)\ \lfloor \cdot \rfloor\ ((\lfloor Dom\ t \rfloor\ \lfloor \otimes \rfloor\ \lfloor Dom\ u \rfloor)\ \lfloor \otimes \rfloor\ \lfloor Dom\ v \rfloor)$   
**proof** –  
 have  $1: Diag\ \lfloor t \rfloor \wedge Diag\ \lfloor u \rfloor \wedge Diag\ \lfloor v \rfloor \wedge$   
 $Dom\ \lfloor t \rfloor = \lfloor Dom\ t \rfloor \wedge Dom\ \lfloor u \rfloor = \lfloor Dom\ u \rfloor \wedge Dom\ \lfloor v \rfloor = \lfloor Dom\ v \rfloor$   
 using  $t\ u\ v\ Diag\text{-Diagonalize}$  by *blast*  
**moreover have**  $Diag\ (\lfloor t \rfloor\ \lfloor \otimes \rfloor\ \lfloor u \rfloor)$   
 using  $1\ TensorDiag\text{-preserves-Diag}(1)$  by *blast*  
**moreover have**  $\wedge t. Arr\ t \implies \lfloor t \rfloor\ \lfloor \cdot \rfloor\ \lfloor Dom\ t \rfloor = \lfloor t \rfloor$   
 using  $t\ Diagonalize\text{-Comp-Arr-Dom}$  by *simp*  
**moreover have**  $Dom\ \lfloor \mathbf{a}[t, u, v] \rfloor = \lfloor Dom\ \mathbf{a}[t, u, v] \rfloor$   
 using  $Diag\text{-Diagonalize}\ tuv$  by *blast*  
**ultimately show** *?thesis*  
 using  $t\ u\ v\ tuv\ 1\ TensorDiag\text{-assoc}\ TensorDiag\text{-preserves-Diag}(2)$   
 by *(metis (no-types) Diagonalize.simps(9))*  
**qed**  
**thus** *?thesis*  
 using  $t\ u\ v\ Diagonalize\text{-Comp-Arr-Dom}\ CompDiag\text{-TensorDiag}\ Diag\text{-Diagonalize}$   
 by *simp*  
**qed**  
**moreover have**  $\{\mathbf{a}[t, u, v]\} = \{(t \otimes u \otimes v) \cdot \mathbf{a}[Dom\ t, Dom\ u, Dom\ v]\}$   
 using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-implies-Arr}\ \alpha\text{-simp}$  [of  $\{t\}\ \{u\}\ \{v\}$ ]  
 by *simp*  
**ultimately show** *coherent*  $\mathbf{a}[t, u, v]$  by *argo*  
**qed**  
**show**  $Arr\ \mathbf{a}^{-1}[t, u, v] \implies coherent\ \mathbf{a}^{-1}[t, u, v]$   
**proof** –  
 assume  $tuv: Arr\ \mathbf{a}^{-1}[t, u, v]$   
 have  $t: Arr\ t$  using  $tuv$  by *simp*  
 have  $u: Arr\ u$  using  $tuv$  by *simp*  
 have  $v: Arr\ v$  using  $tuv$  by *simp*  
 have  $coherent\ (((t \otimes u) \otimes v) \cdot \mathbf{a}^{-1}[Dom\ t, Dom\ u, Dom\ v])$   
**proof** –  
 have  $Arr\ ((t \otimes u) \otimes v) \wedge coherent\ ((t \otimes u) \otimes v)$   
**proof**  
 have  $1: Arr\ v \wedge coherent\ v$  using  $v\ I3$  by *simp*  
 have  $2: Arr\ (t \otimes u) \wedge coherent\ (t \otimes u)$   
 using  $t\ u\ I1\ I2\ Tensor\text{-preserves-coherent}$  by *force*  
**show**  $Arr\ ((t \otimes u) \otimes v)$  using  $1\ 2$  by *simp*  
**show**  $coherent\ ((t \otimes u) \otimes v)$   
 using  $1\ 2\ Tensor\text{-preserves-coherent}$  by *blast*  
**qed**

```

moreover have  $Arr\ a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}$  by  $simp$ 
moreover have  $coherent\ a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ coherent\text{-Assoc'-Ide}$  by  $blast$ 
moreover have  $Dom\ ((t\ \otimes\ u)\ \otimes\ v) = Cod\ a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-in-Hom}$  by  $simp$ 
ultimately show  $?thesis$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-implies-Arr}$ 
     $Comp\text{-preserves-coherent}\ [of\ ((t\ \otimes\ u)\ \otimes\ v)\ a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]]$ 
  by  $metis$ 
qed
moreover have  $Par\ a^{-1}[t,\ u,\ v]\ (((t\ \otimes\ u)\ \otimes\ v) \cdot a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v])$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-implies-Arr}\ Ide\text{-in-Hom}$  by  $simp$ 
moreover have  $[a^{-1}[t,\ u,\ v]] = [((t\ \otimes\ u)\ \otimes\ v) \cdot a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]]$ 
  using  $t\ u\ v\ Diagonalize\text{-Comp-Arr-Dom}\ CompDiag\text{-TensorDiag}\ Diag\text{-Diagonalize}$ 
     $TensorDiag\text{-assoc}\ TensorDiag\text{-preserves-Diag}\ TensorDiag\text{-in-Hom}$ 
     $CompDiag\text{-Diag-Dom}\ [of\ ([t]\ [u])\ [\otimes]\ [v]]$ 
  by  $simp$ 
moreover have  $\{a^{-1}[t,\ u,\ v]\} = \{((t\ \otimes\ u)\ \otimes\ v) \cdot a^{-1}[Dom\ t,\ Dom\ u,\ Dom\ v]\}$ 
  using  $t\ u\ v\ Arr\text{-implies-Ide-Dom}\ Ide\text{-implies-Arr}\ eval\text{-in-hom}\ comp\text{-cod-arr}$ 
     $\alpha'\text{-is-natural-1}\ \alpha'\text{-simp}$ 
  by  $simp$ 
ultimately show  $coherent\ a^{-1}[t,\ u,\ v]$  by  $argo$ 
qed
qed
thus  $?thesis$  using  $assms$  by  $blast$ 
qed

```

MacLane [5] says: “A coherence theorem asserts ‘Every diagram commutes,’” but that is somewhat misleading. A coherence theorem provides some kind of hopefully useful way of distinguishing diagrams that definitely commute from diagrams that might not. The next result expresses coherence for monoidal categories in this way. As the hypotheses can be verified algorithmically (using the functions *Dom*, *Cod*, *Arr*, and *Diagonalize*) if we are given an oracle for equality of arrows in *C*, the result provides a decision procedure, relative to *C*, for the word problem for the free monoidal category generated by *C*.

```

corollary  $eval\text{-eqI}$ :
assumes  $Par\ t\ u$  and  $[t] = [u]$ 
shows  $\{t\} = \{u\}$ 
  using  $assms\ coherence\ canonical\text{-factorization}$  by  $simp$ 

```

Our final corollary expresses coherence in a more “MacLane-like” fashion: parallel canonical arrows are equivalent under evaluation.

```

corollary  $maclane\text{-coherence}$ :
assumes  $Par\ t\ u$  and  $Can\ t$  and  $Can\ u$ 
shows  $\{t\} = \{u\}$ 
proof ( $intro\ eval\text{-eqI}$ )
  show  $Par\ t\ u$  by  $fact$ 
  show  $[t] = [u]$ 
proof –

```

```
    have  $Ide\ [t] \wedge Ide\ [u] \wedge Par\ [t]\ [u]$ 
      using assms eval-eqI Ide-Diagonalize-Can Diagonalize-in-Hom by simp
    thus ?thesis using Ide-in-Hom by auto
  qed
qed

end

end
```

## Chapter 3

# Monoidal Functor

```
theory MonoidalFunctor
imports MonoidalCategory
begin
```

A monoidal functor is a functor  $F$  between monoidal categories  $C$  and  $D$  that preserves the monoidal structure up to isomorphism. The traditional definition assumes a monoidal functor to be equipped with two natural isomorphisms, a natural isomorphism  $\varphi$  that expresses the preservation of tensor product and a natural isomorphism  $\psi$  that expresses the preservation of the unit object. These natural isomorphisms are subject to coherence conditions; the condition for  $\varphi$  involving the associator and the conditions for  $\psi$  involving the unitors. However, as pointed out in [2] (Section 2.4), it is not necessary to take the natural isomorphism  $\psi$  as given, since the mere assumption that  $F \mathcal{I}_C$  is isomorphic to  $\mathcal{I}_D$  is sufficient for there to be a canonical definition of  $\psi$  from which the coherence conditions can be derived. This leads to a more economical definition of monoidal functor, which is the one we adopt here.

```
locale monoidal-functor =
  C: monoidal-category C T_C  $\alpha_C$   $\iota_C$  +
  D: monoidal-category D T_D  $\alpha_D$   $\iota_D$  +
  functor C D F +
  CC: product-category C C +
  DD: product-category D D +
  FF: product-functor C C D D F F +
  FoT_C: composite-functor C.CC.comp C D T_C F +
  T_D o FF: composite-functor C.CC.comp D.CC.comp D FF.map T_D +
   $\varphi$ : natural-isomorphism C.CC.comp D T_D o FF.map FoT_C.map  $\varphi$ 
for C :: 'c comp (infixr  $\cdot_C$  55)
and T_C :: 'c * 'c  $\Rightarrow$  'c
and  $\alpha_C$  :: 'c * 'c * 'c  $\Rightarrow$  'c
and  $\iota_C$  :: 'c
and D :: 'd comp (infixr  $\cdot_D$  55)
and T_D :: 'd * 'd  $\Rightarrow$  'd
and  $\alpha_D$  :: 'd * 'd * 'd  $\Rightarrow$  'd
and  $\iota_D$  :: 'd
```

**and**  $F :: 'c \Rightarrow 'd$   
**and**  $\varphi :: 'c * 'c \Rightarrow 'd +$   
**assumes** *preserves-unity*:  $D.isomorphic\ D.unity\ (F\ C.unity)$   
**and** *assoc-coherence*:  
 $\llbracket C.ide\ a;\ C.ide\ b;\ C.ide\ c \rrbracket \Longrightarrow$   
 $F\ (\alpha_C\ (a,\ b,\ c)) \cdot_D\ \varphi\ (T_C\ (a,\ b),\ c) \cdot_D\ T_D\ (\varphi\ (a,\ b),\ F\ c)$   
 $=\ \varphi\ (a,\ T_C\ (b,\ c)) \cdot_D\ T_D\ (F\ a,\ \varphi\ (b,\ c)) \cdot_D\ \alpha_D\ (F\ a,\ F\ b,\ F\ c)$   
**begin**

<b>notation</b> $C.tensor$	<b>(infixr</b> $\otimes_C$ 53)
<b>and</b> $C.unity$	$(\mathcal{I}_C)$
<b>and</b> $C.lunit$	$(l_C[-])$
<b>and</b> $C.runit$	$(r_C[-])$
<b>and</b> $C.assoc$	$(a_C[-,\ -, -])$
<b>and</b> $D.tensor$	<b>(infixr</b> $\otimes_D$ 53)
<b>and</b> $D.unity$	$(\mathcal{I}_D)$
<b>and</b> $D.lunit$	$(l_D[-])$
<b>and</b> $D.runit$	$(r_D[-])$
<b>and</b> $D.assoc$	$(a_D[-,\ -, -])$

**lemma**  $\varphi$ -in-hom:  
**assumes**  $C.ide\ a$  **and**  $C.ide\ b$   
**shows**  $\llbracket \varphi\ (a,\ b) : F\ a \otimes_D F\ b \rightarrow_D F\ (a \otimes_C b) \rrbracket$   
**using** *assms by auto*

We wish to exhibit a canonical definition of an isomorphism  $\psi \in D.hom\ \mathcal{I}_D\ (F\ \mathcal{I}_C)$  that satisfies certain coherence conditions that involve the left and right unitors. In [2], the isomorphism  $\psi$  is defined by the equation  $l_D[F\ \mathcal{I}_C] = F\ l_C[\mathcal{I}_C] \cdot_D\ \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C) \cdot_D\ (\psi \otimes_D F\ \mathcal{I}_C)$ , which suffices for the definition because the functor  $- \otimes_D F\ \mathcal{I}_C$  is fully faithful. It is then asserted (Proposition 2.4.3) that the coherence condition  $l_D[F\ a] = F\ l_C[a] \cdot_D\ \varphi\ (\mathcal{I}_C,\ a) \cdot_D\ (\psi \otimes_D F\ a)$  is satisfied for any object  $a$  of  $C$ , as well as the corresponding condition for the right unitor. However, the proof is left as an exercise (Exercise 2.4.4). The organization of the presentation suggests that that one should derive the general coherence condition from the special case  $l_D[F\ \mathcal{I}_C] = F\ l_C[\mathcal{I}_C] \cdot_D\ \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C) \cdot_D\ (\psi \otimes_D F\ \mathcal{I}_C)$  used as the definition of  $\psi$ . However, I did not see how to do it that way, so I used a different approach. The isomorphism  $\iota_D' \equiv F\ \iota_C \cdot_D\ \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C)$  serves as an alternative unit for the monoidal category  $D$ . There is consequently a unique isomorphism that maps  $\iota_D$  to  $\iota_D'$ . We define  $\psi$  to be this isomorphism and then use the definition to establish the desired coherence conditions.

**abbreviation**  $\iota_1$   
**where**  $\iota_1 \equiv F\ \iota_C \cdot_D\ \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C)$

**lemma**  $\iota_1$ -in-hom:  
**shows**  $\llbracket \iota_1 : F\ \mathcal{I}_C \otimes_D F\ \mathcal{I}_C \rightarrow_D F\ \mathcal{I}_C \rrbracket$   
**using**  $C.l$ -in-hom **by** (*intro D.in-homI, auto*)

**lemma**  $\iota_1$ -is-iso:  
**shows**  $D.iso\ \iota_1$

using *C.i-is-iso C.i-in-hom  $\varphi$ -in-hom D.isos-compose* by *auto*

**interpretation** *D: monoidal-category-with-alternate-unit*  $D T_D \alpha_D \iota_D \iota_1$

**proof** –

have  $1: \exists \psi. \langle \psi : F \mathcal{I}_C \rightarrow_D \mathcal{I}_D \rangle \wedge D.iso \psi$

**proof** –

obtain  $\psi'$  where  $\psi': \langle \psi' : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.iso \psi'$

using *preserves-unity* by *auto*

have  $\langle D.inv \psi' : F \mathcal{I}_C \rightarrow_D \mathcal{I}_D \rangle \wedge D.iso (D.inv \psi')$

using  $\psi'$  by *simp*

thus *?thesis* by *auto*

qed

obtain  $\psi$  where  $\psi: \langle \psi : F \mathcal{I}_C \rightarrow_D \mathcal{I}_D \rangle \wedge D.iso \psi$

using  $1$  by *blast*

**interpret** *L: equivalence-functor*  $D D \langle \lambda f. (D.cod \iota_1) \otimes_D f \rangle$

**proof** –

**interpret** *L: functor*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle$

using *D.T.fixing-ide-gives-functor-1* by *simp*

**interpret** *L: endofunctor*  $D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle ..$

**interpret**  $\psi x$ : *natural-transformation*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle \langle \lambda f. \mathcal{I}_D \otimes_D f \rangle$   
 $\langle \lambda f. \psi \otimes_D f \rangle$

using  $\psi$  *D.T.fixing-arr-gives-natural-transformation-1* [of  $\psi$ ] by *auto*

**interpret**  $\psi x$ : *natural-isomorphism*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle \langle \lambda f. \mathcal{I}_D \otimes_D f \rangle \langle \lambda f. \psi \otimes_D f \rangle$

apply *unfold-locales* using  $\psi$  *D.tensor-preserves-iso* by *simp*

**interpret**  $\iota_D o \psi x$ : *vertical-composite*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle \langle \lambda f. \mathcal{I}_D \otimes_D f \rangle D.map$   
 $\langle \lambda f. \psi \otimes_D f \rangle D.l ..$

**interpret**  $\iota_D o \psi x$ : *natural-isomorphism*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle D.map \iota_D o \psi x.map$

using  $\psi x.natural-isomorphism-axioms$  *D.l.natural-isomorphism-axioms*  
*natural-isomorphisms-compose* by *blast*

**interpret** *L: equivalence-functor*  $D D \langle \lambda f. (F \mathcal{I}_C) \otimes_D f \rangle$

using *L.isomorphic-to-identity-is-equivalence*  $\iota_D o \psi x.natural-isomorphism-axioms$   
by *simp*

**show** *equivalence-functor*  $D D \langle \lambda f. (D.cod \iota_1) \otimes_D f \rangle$

using *L.equivalence-functor-axioms* *C.i-in-hom* by *auto*

qed

**interpret** *R: equivalence-functor*  $D D \langle \lambda f. T_D (f, D.cod \iota_1) \rangle$

**proof** –

**interpret** *R: functor*  $D D \langle \lambda f. T_D (f, F \mathcal{I}_C) \rangle$

using *D.T.fixing-ide-gives-functor-2* by *simp*

**interpret** *R: endofunctor*  $D \langle \lambda f. T_D (f, F \mathcal{I}_C) \rangle ..$

**interpret**  $x\psi$ : *natural-transformation*  $D D \langle \lambda f. f \otimes_D (F \mathcal{I}_C) \rangle \langle \lambda f. f \otimes_D \mathcal{I}_D \rangle$   
 $\langle \lambda f. f \otimes_D \psi \rangle$

using  $\psi$  *D.T.fixing-arr-gives-natural-transformation-2* [of  $\psi$ ] by *auto*

**interpret**  $x\psi$ : *natural-isomorphism*  $D D \langle \lambda f. f \otimes_D (F \mathcal{I}_C) \rangle \langle \lambda f. f \otimes_D \mathcal{I}_D \rangle \langle \lambda f. f \otimes_D \psi \rangle$

using  $\psi$  *D.tensor-preserves-iso* by (*unfold-locales*, *simp*)

**interpret**  $\rho_D o x\psi$ : *vertical-composite*  $D D \langle \lambda f. f \otimes_D (F \mathcal{I}_C) \rangle \langle \lambda f. f \otimes_D \mathcal{I}_D \rangle D.map$   
 $\langle \lambda f. f \otimes_D \psi \rangle D.\rho ..$

**interpret**  $\rho_D o x\psi$ : *natural-isomorphism*  $D D \langle \lambda f. f \otimes_D (F \mathcal{I}_C) \rangle D.map \rho_D o x\psi.map$

using  $x\psi.natural-isomorphism-axioms$  *D. $\rho$ .natural-isomorphism-axioms*

*natural-isomorphisms-compose* **by** *blast*

**interpret**  $R$ : *equivalence-functor*  $D D \langle \lambda f. f \otimes_D (F \mathcal{I}_C) \rangle$   
**using**  $R$ .*isomorphic-to-identity-is-equivalence*  $\varrho_D$  *or*  $\psi$ .*natural-isomorphism-axioms*  
**by** *simp*

**show** *equivalence-functor*  $D D (\lambda f. f \otimes_D (D.\text{cod } \iota_1))$   
**using**  $R$ .*equivalence-functor-axioms*  $C$ . *$\iota$ -in-hom* **by** *auto*

**qed**

**show** *monoidal-category-with-alternate-unit*  $D T_D \alpha_D \iota_D \iota_1$   
**using**  $D$ .*pentagon*  $C$ . *$\iota$ -is-iso*  $C$ . *$\iota$ -in-hom preserves-hom*  $\iota_1$ -*is-iso*  $\iota_1$ -*in-hom*  
**by** (*unfold-locales*, *auto*)

**qed**

**no-notation**  $D$ .*tensor* (**infixr**  $\otimes_D$  53)  
**notation**  $D$ . $C_1$ .*tensor* (**infixr**  $\otimes_D$  53)  
**no-notation**  $D$ .*assoc* ( $\text{a}_D[-, -, -]$ )  
**notation**  $D$ . $C_1$ .*assoc* ( $\text{a}_D[-, -, -]$ )  
**no-notation**  $D$ .*assoc'* ( $\text{a}_D^{-1}[-, -, -]$ )  
**notation**  $D$ . $C_1$ .*assoc'* ( $\text{a}_D^{-1}[-, -, -]$ )  
**notation**  $D$ . $C_1$ .*unity* ( $\mathcal{I}_1$ )  
**notation**  $D$ . $C_1$ .*lunit* ( $\text{l}_1[-]$ )  
**notation**  $D$ . $C_1$ .*runit* ( $\text{r}_1[-]$ )

**lemma**  $\mathcal{I}_1$ -*char* [*simp*]:  
**shows**  $\mathcal{I}_1 = F \mathcal{I}_C$   
**using**  $D$ . $C_1$ .*unity-def*  $\iota_1$ -*in-hom* **by** *auto*

**definition**  $\psi$   
**where**  $\psi \equiv \text{THE } \psi. \langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.\text{iso } \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$

**lemma**  $\psi$ -*char*:  
**shows**  $\langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle$  **and**  $D.\text{iso } \psi$  **and**  $\psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$   
**and**  $\exists ! \psi. \langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.\text{iso } \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$   
**proof** –  
**show**  $\exists ! \psi. \langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.\text{iso } \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$   
**using**  $D$ .*unit-unique-upto-unique-iso*  $\iota_1$ -*in-hom*  $D$ . $C_1$ . *$\iota$ -in-hom*  
**by** (*elim D.in-homE*, *auto*)  
**hence** 1:  $\langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.\text{iso } \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$   
**unfolding**  $\psi$ -*def*  
**using** *theI'* [*of*  $\lambda \psi. \langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.\text{iso } \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$ ]  
**by** *fast*  
**show**  $\langle \psi : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle$  **using** 1 **by** *simp*  
**show**  $D.\text{iso } \psi$  **using** 1 **by** *simp*  
**show**  $\psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$  **using** 1 **by** *simp*

**qed**

**lemma**  $\psi$ -*eqI*:  
**assumes**  $\langle f : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle$  **and**  $D.\text{iso } f$  **and**  $f \cdot_D \iota_D = \iota_1 \cdot_D (f \otimes_D f)$   
**shows**  $f = \psi$   
**using** *assms*  $\psi$ -*def*  $\psi$ -*char*



the1-equality [of  $\lambda f. \llbracket f: \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rrbracket \wedge D.iso\ f \wedge f \cdot_D \iota_D = \iota_1 \cdot_D (f \otimes_D f) f$ ]  
by simp

**lemma** *lunit-coherence1*:

**assumes** *C.ide a*

**shows**  $l_1[F a] \cdot_D (\psi \otimes_D F a) = l_D[F a]$

**proof** –

**have** *D.par* ( $l_1[F a] \cdot_D (\psi \otimes_D F a)$ )  $l_D[F a]$

**using** *assms* *D.C1.lunit-in-hom* *D.tensor-in-hom* *D.lunit-in-hom* *ψ-char(1)* *C.ι-in-hom*

**by** *auto*

The upper left triangle in the following diagram commutes.

$$\begin{array}{ccc}
 \mathcal{I} \otimes Fa & \xrightarrow{\psi \otimes Fa} & F\mathcal{I} \otimes Fa \\
 \uparrow \mathcal{I} \otimes l_1[Fa] & & \uparrow F\mathcal{I} \otimes l_1[Fa] \\
 \mathcal{I} \otimes (\mathcal{I} \otimes Fa) & \xrightarrow{\psi \otimes (\mathcal{I} \otimes Fa)} & F\mathcal{I} \otimes (\mathcal{I} \otimes Fa) \\
 \uparrow \mathcal{I} \otimes (\psi \otimes Fa) & & \uparrow \mathcal{I} \otimes (\psi \otimes Fa) \\
 \mathcal{I} \otimes (\mathcal{I} \otimes Fa) & \xrightarrow{\psi \otimes (\psi \otimes Fa)} & F\mathcal{I} \otimes (F\mathcal{I} \otimes Fa) \\
 \uparrow \mathcal{I} \otimes [Fa] & & \uparrow F\mathcal{I} \otimes [Fa] \\
 (\mathcal{I} \otimes \mathcal{I}) \otimes Fa & \xrightarrow{(\psi \otimes \psi) \otimes Fa} & (F\mathcal{I} \otimes F\mathcal{I}) \otimes Fa \\
 \uparrow \iota \otimes Fa & & \uparrow \iota_1 \otimes Fa \\
 (\mathcal{I} \otimes \mathcal{I}) \otimes Fa & & (F\mathcal{I} \otimes F\mathcal{I}) \otimes Fa \\
 \uparrow a[\mathcal{I}, \mathcal{I}, Fa] & & \uparrow a[F\mathcal{I}, F\mathcal{I}, Fa]
 \end{array}$$

**moreover have**  $(\mathcal{I}_D \otimes_D l_1[F a]) \cdot_D (\mathcal{I}_D \otimes_D \psi \otimes_D F a) = \mathcal{I}_D \otimes_D l_D[F a]$

**proof** –

**have**  $(\mathcal{I}_D \otimes_D l_1[F a]) \cdot_D (\mathcal{I}_D \otimes_D \psi \otimes_D F a)$

$= (\mathcal{I}_D \otimes_D l_1[F a]) \cdot_D (D.inv\ \psi \otimes_D F \mathcal{I}_C \otimes_D F a) \cdot_D (\psi \otimes_D \psi \otimes_D F a)$

**using** *assms* *ψ-char(1-2)* *D.interchange* [of *D.inv ψ*] *D.comp-cod-arr*

*D.inv-is-inverse* *D.comp-inv-arr*

**by** (*elim D.in-homE*, *simp*)

**also have**  $\dots = (D.inv\ \psi \otimes_D F a) \cdot_D (F \mathcal{I}_C \otimes_D l_1[F a]) \cdot_D (\psi \otimes_D \psi \otimes_D F a)$

**proof** –

**have**  $(\mathcal{I}_D \otimes_D l_1[F a]) \cdot_D (D.inv\ \psi \otimes_D F \mathcal{I}_C \otimes_D F a) =$

$(D.inv\ \psi \otimes_D F a) \cdot_D (F \mathcal{I}_C \otimes_D l_1[F a])$

**using** *assms* *ψ-char(1-2)* *D.interchange* [of  $\mathcal{I}_D$ ] *D.interchange* [of *D.inv ψ*]

*D.comp-arr-dom* *D.comp-cod-arr*

**by** (*elim D.in-homE*, *auto*)

**thus** *?thesis*

**using** *assms* *ψ-char(1-2)* *D.inv-in-hom*

*D.comp-permute* [of  $\mathcal{I}_D \otimes_D l_1[F a]$ ] *D.inv ψ*  $\otimes_D F \mathcal{I}_C \otimes_D F a$

*D.inv ψ*  $\otimes_D F a$   $F \mathcal{I}_C \otimes_D l_1[F a]$

by (*elim D.in-homE*, *auto*)  
**qed**  
**also have** ... =  $(D.inv \psi \otimes_D F a) \cdot_D (\iota_1 \otimes_D F a) \cdot_D D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a] \cdot_D$   
 $(\psi \otimes_D \psi \otimes_D F a)$   
 using *assms*  $\psi$ -char(1-2) *D.C1.lunit-char*(2) *D.comp-assoc* **by** *auto*  
**also have** ... =  $((D.inv \psi \otimes_D F a) \cdot_D (\iota_1 \otimes_D F a) \cdot_D ((\psi \otimes_D \psi) \otimes_D F a)) \cdot_D$   
 $D.inv \mathfrak{a}_D[\mathcal{I}_D, \mathcal{I}_D, F a]$   
 using *assms*  $\psi$ -char(1-2) *D.assoc'-naturality* [of  $\psi \psi F a$ ] *D.comp-assoc* **by** *auto*  
**also have** ... =  $(\iota_D \otimes_D F a) \cdot_D D.inv \mathfrak{a}_D[\mathcal{I}_D, \mathcal{I}_D, F a]$   
**proof** –  
 have  $(D.inv \psi \otimes_D F a) \cdot_D (\iota_1 \otimes_D F a) \cdot_D ((\psi \otimes_D \psi) \otimes_D F a) = \iota_D \otimes_D F a$   
**proof** –  
 have  $(D.inv \psi \otimes_D F a) \cdot_D (\iota_1 \otimes_D F a) \cdot_D ((\psi \otimes_D \psi) \otimes_D F a) =$   
 $D.inv \psi \cdot_D \psi \cdot_D \iota_D \otimes_D F a$   
 using *assms*  $\psi$ -char(1-3) *D.l-in-hom*  $\iota_1$ -in-hom *D.interchange*  
 by (*elim D.in-homE*, *auto*)  
**also have** ... =  $\iota_D \otimes_D F a$   
 using *assms*  $\psi$ -char(1-2) *D.inv-is-inverse* *D.comp-inv-arr* *D.comp-cod-arr*  
 $D.comp-reduce$  *D.l-in-hom*  
 by (*elim D.in-homE*, *auto*)  
**finally show** *?thesis* **by** *blast*  
**qed**  
**thus** *?thesis* **by** *simp*  
**qed**  
**also have** ... =  $\mathcal{I}_D \otimes_D 1_D[F a]$   
 using *assms* *D.lunit-char* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**ultimately show** *?thesis*  
 using *D.L.is-faithful* [of  $1_1[F a] \cdot_D (\psi \otimes_D F a) 1_D[F a]$ ] *D.l-in-hom* **by** *force*  
**qed**

**lemma** *lunit-coherence2*:  
**assumes** *C.ide a*  
**shows**  $F 1_C[a] \cdot_D \varphi(\mathcal{I}_C, a) = 1_1[F a]$   
**proof** –

We show that the lower left triangle in the following diagram commutes.

$$\begin{array}{ccccc}
(F\mathcal{I} \otimes F\mathcal{I}) \otimes Fa & \xrightarrow{\phi(\mathcal{I}, \mathcal{I}) \otimes Fa} & F(\mathcal{I} \otimes \mathcal{I}) \otimes Fa & & \\
\downarrow a[F\mathcal{I}, F\mathcal{I}, Fa] & \searrow \iota_1 \otimes Fa & \swarrow F\iota \otimes Fa & & \downarrow \phi(\mathcal{I} \otimes \mathcal{I}, a) \\
F\mathcal{I} \otimes (F\mathcal{I} \otimes Fa) & \xrightarrow{F\mathcal{I} \otimes l_1[Fa]} & F\mathcal{I} \otimes Fa & \xrightarrow{\phi(\mathcal{I}, a)} & F(\mathcal{I} \otimes a) & \xleftarrow{F(\iota \otimes a)} & F((\mathcal{I} \otimes \mathcal{I}) \otimes a) \\
\downarrow F\mathcal{I} \otimes \phi(\mathcal{I}, a) & \swarrow F\mathcal{I} \otimes F l[a] & & & \swarrow F(\mathcal{I} \otimes l[a]) & & \downarrow Fa[\mathcal{I}, \mathcal{I}, a] \\
F\mathcal{I} \otimes F(\mathcal{I} \otimes a) & \xrightarrow{\phi(\mathcal{I}, \mathcal{I} \otimes a)} & F(\mathcal{I} \otimes (\mathcal{I} \otimes a)) & & & & 
\end{array}$$

have  $(F \mathcal{I}_C \otimes_D F l_C[a]) \cdot_D (F \mathcal{I}_C \otimes_D \varphi(\mathcal{I}_C, a)) = F \mathcal{I}_C \otimes_D l_1[F a]$

**proof** –

$$\begin{aligned}
& \text{have } (F \mathcal{I}_C \otimes_D F l_C[a]) \cdot_D (F \mathcal{I}_C \otimes_D \varphi(\mathcal{I}_C, a)) \\
& = (F \mathcal{I}_C \otimes_D F l_C[a]) \cdot_D D.\text{inv}(\varphi(\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a] \cdot_D \\
& \quad \varphi(\mathcal{I}_C \otimes_C \mathcal{I}_C, a) \cdot_D (\varphi(\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a) \cdot_D D.\text{inv} \text{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]
\end{aligned}$$

**proof** –

$$\begin{aligned}
& \text{have } D.\text{inv}(\varphi(\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a] \cdot_D \varphi(\mathcal{I}_C \otimes_C \mathcal{I}_C, a) \cdot_D \\
& \quad (\varphi(\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a) \\
& = (F \mathcal{I}_C \otimes_D \varphi(\mathcal{I}_C, a)) \cdot_D \text{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]
\end{aligned}$$

**using** *assms*  $\varphi$ -in-hom *assoc-coherence* *D.invert-side-of-triangle(1)* **by** *simp*

**hence**  $F \mathcal{I}_C \otimes_D \varphi(\mathcal{I}_C, a)$

$$\begin{aligned}
& = (D.\text{inv}(\varphi(\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a] \cdot_D \varphi(\mathcal{I}_C \otimes_C \mathcal{I}_C, a) \cdot_D \\
& \quad (\varphi(\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a)) \cdot_D D.\text{inv} \text{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]
\end{aligned}$$

**using** *assms*  $\varphi$ -in-hom *D.invert-side-of-triangle(2)* **by** *simp*

**thus** *?thesis*

**using** *D.comp-assoc* **by** *simp*

**qed**

$$\begin{aligned}
& \text{also have } \dots = (F \mathcal{I}_C \otimes_D F l_C[a]) \cdot_D D.\text{inv}(\varphi(\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D \\
& \quad (D.\text{inv}(F(\mathcal{I}_C \otimes_C l_C[a])) \cdot_D F(\iota_C \otimes_C a)) \cdot_D \\
& \quad \varphi(\mathcal{I}_C \otimes_C \mathcal{I}_C, a) \cdot_D (\varphi(\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a) \cdot_D \\
& \quad D.\text{inv} \text{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]
\end{aligned}$$

**proof** –

$$\text{have } 1: F(\mathcal{I}_C \otimes_C l_C[a]) = F(\iota_C \otimes_C a) \cdot_D D.\text{inv}(F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a])$$

**using** *assms* *C.lunit-char(1-2)* *C.l-in-hom preserves-inv* **by** *auto*

$$\text{hence } F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a] = D.\text{inv}(F(\mathcal{I}_C \otimes_C l_C[a])) \cdot_D F(\iota_C \otimes_C a)$$

**proof** –

$$\begin{aligned}
& \text{have } F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a] \cdot_D D.\text{inv}(F(\iota_C \otimes_C a)) \\
& = D.\text{inv}(F(\iota_C \otimes_C a) \cdot_D D.\text{inv}(F \text{ac}[\mathcal{I}_C, \mathcal{I}_C, a]))
\end{aligned}$$

**using** *assms 1 preserves-iso C.ide-is-iso C.ι-is-iso C.ide-unity C.iso-assoc*  
*C.iso-lunit C.tensor-preserved-iso D.inv-comp D.inv-inv*  
*D.iso-inv-iso D.iso-is-arr*  
**by** *metis*  
**thus** *?thesis*  
**using** *assms 1 preserves-iso C.ide-is-iso C.ι-is-iso C.ide-unity C.iso-assoc*  
*C.iso-lunit C.tensor-preserved-iso D.inv-comp D.inv-inv*  
*D.iso-inv-iso D.iso-is-arr D.invert-side-of-triangle(2)*  
**by** *metis*  
**qed**  
**thus** *?thesis by argo*  
**qed**  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D$   
 $D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D (F (\iota_C \otimes_C a) \cdot_D \varphi (\mathcal{I}_C \otimes_C \mathcal{I}_C, a)) \cdot_D$   
 $(\varphi (\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a) \cdot_D D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]$   
**using** *D.comp-assoc by auto*  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D$   
 $D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D (\varphi (\mathcal{I}_C, a) \cdot_D (F \iota_C \otimes_D F a)) \cdot_D$   
 $(\varphi (\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a) \cdot_D D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]$   
**using** *assms φ.naturality [of (ι<sub>C</sub>, a)] C.ι-in-hom by auto*  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D$   
 $D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a) \cdot_D$   
 $((F \iota_C \otimes_D F a) \cdot_D (\varphi (\mathcal{I}_C, \mathcal{I}_C) \otimes_D F a)) \cdot_D$   
 $D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]$   
**using** *D.comp-assoc by auto*  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D$   
 $D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a) \cdot_D (\iota_1 \otimes_D F a) \cdot_D$   
 $D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]$   
**using** *assms D.interchange C.ι-in-hom by auto*  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D$   
 $D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a) \cdot_D$   
 $((F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a]) \cdot_D \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]) \cdot_D$   
 $D.inv \mathfrak{a}_D[F \mathcal{I}_C, F \mathcal{I}_C, F a]$   
**proof** –  
**have**  $(\iota_1 \otimes_D F a) \cdot_D \mathfrak{a}_D^{-1}[F \mathcal{I}_C, F \mathcal{I}_C, F a] = F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a]$   
**using** *assms D.C<sub>1</sub>.lunit-char [of F a] by auto*  
**thus** *?thesis*  
**using** *assms D.inv-is-inverse ι<sub>1</sub>-in-hom φ-in-hom D.invert-side-of-triangle(2)*  
**by** *simp*  
**qed**  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D$   
 $(D.inv (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D D.inv (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a)) \cdot_D$   
 $(F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a])$   
**using** *assms D.comp-arr-dom [of F I<sub>C</sub> ⊗<sub>D</sub> 1<sub>1</sub>[F a]] D.comp-assoc by auto*  
**also have** ... =  $(F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.inv (F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D (F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a])$   
**proof** –  
**have**  $D.inv (F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a])$   
 $= D.inv (D.inv (\varphi (\mathcal{I}_C, a)) \cdot_D F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a]) \cdot_D \varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a))$   
**using** *assms φ.naturality [of (I<sub>C</sub>, 1<sub>C</sub>[a])] D.invert-side-of-triangle(1) by simp*

**also have** ... =  $D.\text{inv } (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D D.\text{inv } (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a)$   
**using** *assms*  $D.\text{inv-comp } D.\text{inv-is-inverse } D.\text{isos-compose } D.\text{comp-assoc}$   
**by** *simp*  
**finally have**  $D.\text{inv } (F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a])$   
=  $D.\text{inv } (\varphi (\mathcal{I}_C, \mathcal{I}_C \otimes_C a)) \cdot_D D.\text{inv } (F (\mathcal{I}_C \otimes_C \mathbb{1}_C[a])) \cdot_D \varphi (\mathcal{I}_C, a)$   
**by** *blast*  
**thus** *?thesis* **by** *argo*  
**qed**  
**also have** ... =  $((F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a]) \cdot_D D.\text{inv } (F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a])) \cdot_D (F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a])$   
**using** *assms*  $D.\text{tensor-preserves-iso } D.\text{comp-assoc}$  **by** *simp*  
**also have** ... =  $F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a]$   
**using** *assms*  $D.\text{tensor-preserves-iso } D.\text{comp-arr-inv } D.\text{inv-is-inverse } D.\text{comp-cod-arr}$   
 $D.\text{interchange}$   
**by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**hence**  $F \mathcal{I}_C \otimes_D F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a) = F \mathcal{I}_C \otimes_D \mathbb{1}_1[F a]$   
**using** *assms*  $\varphi\text{-in-hom } D.\text{interchange}$  **by** *simp*  
**moreover have**  $D.\text{par } (F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a)) \mathbb{1}_1[F a]$   
**using** *assms*  $\varphi\text{-in-hom}$  **by** *simp*  
**ultimately show** *?thesis*  
**using**  $D.C_1.L.\text{is-faithful } [of F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a) \mathbb{1}_1[F a]] D.C_1.\text{unity-def}$  **by** *simp*  
**qed**

Combining the two previous lemmas yields the coherence result we seek. This is the condition that is traditionally taken as part of the definition of monoidal functor.

**lemma** *lunit-coherence*:

**assumes**  $C.\text{ide } a$

**shows**  $\mathbb{1}_D[F a] = F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a) \cdot_D (\psi \otimes_D F a)$

**proof** –

**have**  $\mathbb{1}_D[F a] \cdot_D D.\text{inv } (\psi \otimes_D F a) = \mathbb{1}_1[F a]$

**using** *assms*  $\text{lunit-coherence1 } \psi\text{-char}(2)$

$D.\text{invert-side-of-triangle}(2) [of \mathbb{1}_D[F a] \mathbb{1}_1[F a] \psi \otimes_D F a]$

**by** *auto*

**also have** ... =  $F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a)$

**using** *assms*  $\text{lunit-coherence2}$  **by** *simp*

**finally have**  $\mathbb{1}_D[F a] \cdot_D D.\text{inv } (\psi \otimes_D F a) = F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a)$

**by** *blast*

**hence**  $\mathbb{1}_D[F a] = (F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a)) \cdot_D (\psi \otimes_D F a)$

**using** *assms*  $\psi\text{-char}(2) \varphi\text{-in-hom}$

$D.\text{invert-side-of-triangle}(2) [of F \mathbb{1}_C[a] \cdot_D \varphi (\mathcal{I}_C, a) \mathbb{1}_D[F a] D.\text{inv } (\psi \otimes_D F a)]$

**by** *simp*

**thus** *?thesis*

**using** *assms*  $\psi\text{-char}(1) D.\text{comp-assoc}$  **by** *auto*

**qed**

We now want to obtain the corresponding result for the right unitor. To avoid a repetition of what would amount to essentially the same tedious diagram chases that were carried out above, we instead show here that  $F$  becomes a monoidal functor from

the opposite of  $C$  to the opposite of  $D$ , with  $\lambda f. \varphi (snd f, fst f)$  as the structure map. The fact that in the opposite monoidal categories the left and right unitors are exchanged then permits us to obtain the result for the right unitor from the result already proved for the left unitor.

**interpretation**  $C'$ : *opposite-monoidal-category*  $C T_C \alpha_C \iota_C ..$   
**interpretation**  $D'$ : *opposite-monoidal-category*  $D T_D \alpha_D \iota_D ..$   
**interpretation**  $T_D' oFF$ : *composite-functor*  $C.CC.comp D.CC.comp D.FF.map D'.T ..$   
**interpretation**  $FoT_{C'}$ : *composite-functor*  $C.CC.comp C D C'.T F ..$   
**interpretation**  $\varphi'$ : *natural-transformation*  $C.CC.comp D T_D' oFF.map FoT_{C'}.map$   
 $\langle \lambda f. \varphi (snd f, fst f) \rangle$   
**using**  $\varphi.is-natural-1$   $\varphi.is-natural-2$   $\varphi.is-extensional$  **by** (*unfold-locales, auto*)  
**interpretation**  $\varphi'$ : *natural-isomorphism*  $C.CC.comp D T_D' oFF.map FoT_{C'}.map$   
 $\langle \lambda f. \varphi (snd f, fst f) \rangle$   
**by** (*unfold-locales, simp*)  
**interpretation**  $F'$ : *monoidal-functor*  $C C'.T C'.\alpha \iota_C D D'.T D'.\alpha \iota_D F \langle \lambda f. \varphi (snd f, fst f) \rangle$   
**using** *preserves-unity* **apply** (*unfold-locales; simp*)  
**proof** –  
**fix**  $a b c$   
**assume**  $a: C.ide a$  **and**  $b: C.ide b$  **and**  $c: C.ide c$   
**have**  $(\varphi (c \otimes_C b, a) \cdot_D (\varphi (c, b) \otimes_D F a)) \cdot_D a_D^{-1}[F c, F b, F a] =$   
 $F (C.assoc' c b a) \cdot_D \varphi (c, b \otimes_C a) \cdot_D (F c \otimes_D \varphi (b, a))$   
**proof** –  
**have**  $D.seq (F a_C[c, b, a]) (\varphi (c \otimes_C b, a) \cdot_D (\varphi (c, b) \otimes_D F a))$   
**using**  $a b c$   $\varphi$ -*in-hom* **by** *simp*  
**moreover have**  $D.seq (\varphi (c, b \otimes_C a) \cdot_D (F c \otimes_D \varphi (b, a))) a_D[F c, F b, F a]$   
**using**  $a b c$   $\varphi$ -*in-hom* **by** *simp*  
**moreover have**  
 $F a_C[c, b, a] \cdot_D \varphi (c \otimes_C b, a) \cdot_D (\varphi (c, b) \otimes_D F a) =$   
 $(\varphi (c, b \otimes_C a) \cdot_D (F c \otimes_D \varphi (b, a))) \cdot_D a_D[F c, F b, F a]$   
**using**  $a b c$  *assoc-coherence*  $D.comp-assoc$  **by** *simp*  
**moreover have**  $D.iso (F a_C[c, b, a])$   
**using**  $a b c$  **by** *simp*  
**moreover have**  $D.iso a_D[F c, F b, F a]$   
**using**  $a b c$  **by** *simp*  
**moreover have**  $D.inv (F a_C[c, b, a]) = F (C.assoc' c b a)$   
**using**  $a b c$  *preserves-inv* **by** *simp*  
**ultimately show** *?thesis*  
**using**  $D.invert-opposite-sides-of-square$  **by** *simp*  
**qed**  
**thus**  $F (C.assoc' c b a) \cdot_D \varphi (c, b \otimes_C a) \cdot_D (F c \otimes_D \varphi (b, a)) =$   
 $\varphi (c \otimes_C b, a) \cdot_D (\varphi (c, b) \otimes_D F a) \cdot_D a_D^{-1}[F c, F b, F a]$   
**using**  $D.comp-assoc$  **by** *simp*  
**qed**  
**lemma** *induces-monoidal-functor-between-opposites:*  
**shows** *monoidal-functor*  $C C'.T C'.\alpha \iota_C D D'.T D'.\alpha \iota_D F (\lambda f. \varphi (snd f, fst f))$   
**..**

```

lemma runit-coherence:
assumes C.ide a
shows  $r_D[F a] = F r_C[a] \cdot_D \varphi(a, \mathcal{I}_C) \cdot_D (F a \otimes_D \psi)$ 
proof –
  have  $C'.lunit a = r_C[a]$ 
    using assms C'.lunit-simp by simp
  moreover have  $D'.lunit (F a) = r_D[F a]$ 
    using assms D'.lunit-simp by simp
  moreover have  $F'.\psi = \psi$ 
proof (intro  $\psi$ -eqI)
    show  $\langle F'.\psi : D'.unity \rightarrow_D F C'.unity \rangle$  using F'. $\psi$ -char(1) by simp
    show D.iso F'. $\psi$  using F'. $\psi$ -char(2) by simp
    show  $F'.\psi \cdot_D \iota_D = \iota_1 \cdot_D (F'.\psi \otimes_D F'.\psi)$  using F'. $\psi$ -char(3) by simp
  qed
  moreover have  $D'.lunit (F a) = F (C'.lunit a) \cdot_D \varphi(a, C'.unity) \cdot_D (F a \otimes_D F'.\psi)$ 
    using assms F'.lunit-coherence by simp
  ultimately show ?thesis by simp
qed

end

```

### 3.1 Strict Monoidal Functor

A strict monoidal functor preserves the monoidal structure “on the nose”.

```

locale strict-monoidal-functor =
  C: monoidal-category C T_C  $\alpha_C$   $\iota_C$  +
  D: monoidal-category D T_D  $\alpha_D$   $\iota_D$  +
  functor C D F
for  $C :: 'c \text{ comp}$  (infixr  $\cdot_C$  55)
and  $T_C :: 'c * 'c \Rightarrow 'c$ 
and  $\alpha_C :: 'c * 'c * 'c \Rightarrow 'c$ 
and  $\iota_C :: 'c$ 
and  $D :: 'd \text{ comp}$  (infixr  $\cdot_D$  55)
and  $T_D :: 'd * 'd \Rightarrow 'd$ 
and  $\alpha_D :: 'd * 'd * 'd \Rightarrow 'd$ 
and  $\iota_D :: 'd$ 
and  $F :: 'c \Rightarrow 'd +$ 
assumes strictly-preserves- $\iota$ : F  $\iota_C = \iota_D$ 
and strictly-preserves-T:  $\llbracket C.arr f; C.arr g \rrbracket \Longrightarrow F (T_C (f, g)) = T_D (F f, F g)$ 
and strictly-preserves- $\alpha$ -ide:  $\llbracket C.ide a; C.ide b; C.ide c \rrbracket \Longrightarrow$ 
 $F (\alpha_C (a, b, c)) = \alpha_D (F a, F b, F c)$ 
begin

  notation C.tensor (infixr  $\otimes_C$  53)
and C.unity ( $\mathcal{I}_C$ )
and C.lunit ( $l_C[-]$ )
and C.runit ( $r_C[-]$ )
and C.assoc ( $a_C[-, -, -]$ )

```

**and**  $D.tensor$  (**infixr**  $\otimes_D$  53)  
**and**  $D.unity$  ( $\mathcal{I}_D$ )  
**and**  $D.lunit$  ( $l_D[-]$ )  
**and**  $D.runit$  ( $r_D[-]$ )  
**and**  $D.assoc$  ( $a_D[-, -, -]$ )

**lemma** *strictly-preserves-tensor*:  
**assumes**  $C.arr\ f$  **and**  $C.arr\ g$   
**shows**  $F(f \otimes_C g) = Ff \otimes_D Fg$   
**using** *assms strictly-preserves-T by blast*

**lemma** *strictly-preserves- $\alpha$* :  
**assumes**  $C.arr\ f$  **and**  $C.arr\ g$  **and**  $C.arr\ h$   
**shows**  $F(\alpha_C(f, g, h)) = \alpha_D(Ff, Fg, Fh)$   
**proof** –  
**have**  $F(\alpha_C(f, g, h)) = F((f \otimes_C g \otimes_C h) \cdot_C \alpha_C(C.dom\ f, C.dom\ g, C.dom\ h))$   
**using** *assms C. $\alpha$ .is-natural-1 [of (f, g, h)] C.T.ToCT-simp by force*  
**also have**  $\dots = (Ff \otimes_D Fg \otimes_D Fh) \cdot_D \alpha_D(D.dom(Ff), D.dom(Fg), D.dom(Fh))$   
**using** *assms strictly-preserves- $\alpha$ -ide strictly-preserves-tensor by simp*  
**also have**  $\dots = \alpha_D(Ff, Fg, Fh)$   
**using** *assms D. $\alpha$ .is-natural-1 [of (Ff, Fg, Fh)] by simp*  
**finally show** *?thesis by blast*  
**qed**

**lemma** *strictly-preserves-unity*:  
**shows**  $F\ \mathcal{I}_C = \mathcal{I}_D$   
**using** *C. $\iota$ -in-hom strictly-preserves- $\iota$  C.unity-def D.unity-def by auto*

**lemma** *strictly-preserves-assoc*:  
**assumes**  $C.arr\ a$  **and**  $C.arr\ b$  **and**  $C.arr\ c$   
**shows**  $F\ a_C[a, b, c] = a_D[Fa, Fb, Fc]$   
**using** *assms strictly-preserves- $\alpha$  by simp*

**lemma** *strictly-preserves-lunit*:  
**assumes**  $C.ide\ a$   
**shows**  $F\ l_C[a] = l_D[Fa]$   
**proof** –  
**let**  $?P = \lambda f. f \in C.hom(\mathcal{I}_C \otimes_C a)\ a \wedge \mathcal{I}_C \otimes_C f = (\iota_C \otimes_C a) \cdot_C C.assoc'\ \mathcal{I}_C\ \mathcal{I}_C\ a$   
**let**  $?Q = \lambda f. f \in D.hom(\mathcal{I}_D \otimes_D Fa)\ (Fa) \wedge$   
 $\mathcal{I}_D \otimes_D f = (\iota_D \otimes_D Fa) \cdot_D D.assoc'\ \mathcal{I}_D\ \mathcal{I}_D\ (Fa)$   
**have** 1:  $?P\ l_C[a]$  **using** *C.lunit-char by simp*  
**hence**  $?Q\ (F\ l_C[a])$   
**proof** –  
**have**  $F\ l_C[a] \in D.hom(\mathcal{I}_D \otimes_D Fa)\ (Fa)$   
**using** *assms 1 strictly-preserves-unity strictly-preserves-tensor by auto*  
**moreover have**  
 $F((\iota_C \otimes_C a) \cdot_C C.assoc'\ \mathcal{I}_C\ \mathcal{I}_C\ a) = (\iota_D \otimes_D Fa) \cdot_D D.assoc'\ \mathcal{I}_D\ \mathcal{I}_D\ (Fa)$   
**using** *assms 1 strictly-preserves- $\iota$  strictly-preserves- $\alpha$  strictly-preserves-unity*  
*strictly-preserves-tensor preserves-inv C. $\iota$ -in-hom*



by *auto*  
**moreover have**  $\mathcal{I}_D \otimes_D F \mathcal{I}_C[a] = F (\mathcal{I}_C \otimes_C \mathcal{I}_C[a])$   
 using *assms strictly-preserves-unity strictly-preserves-tensor* by *simp*  
**ultimately show** *?thesis*  
 using *assms C.lunit-char(2)* by *simp*  
**qed**  
**thus** *?thesis* using *assms D.lunit-eqI* by *simp*  
**qed**

**lemma** *strictly-preserves-runit:*

**assumes** *C.ide a*

**shows**  $F \text{r}_C[a] = \text{r}_D[F a]$

**proof** –

**let**  $?P = \lambda f. f \in C.\text{hom} (a \otimes_C \mathcal{I}_C) a \wedge f \otimes_C \mathcal{I}_C = (a \otimes_C \iota_C) \cdot_C C.\text{assoc } a \mathcal{I}_C \mathcal{I}_C$

**let**  $?Q = \lambda f. f \in D.\text{hom} (F a \otimes_D \mathcal{I}_D) (F a) \wedge$

$f \otimes_D \mathcal{I}_D = (F a \otimes_D \iota_D) \cdot_D D.\text{assoc} (F a) \mathcal{I}_D \mathcal{I}_D$

**have** 1:  $?P \text{r}_C[a]$  using *assms C.runit-char* by *simp*

**hence**  $?Q (F \text{r}_C[a])$

**proof** –

**have**  $F \text{r}_C[a] \in D.\text{hom} (F a \otimes_D \mathcal{I}_D) (F a)$

using *assms 1 strictly-preserves-unity strictly-preserves-tensor* by *auto*

**moreover have**  $F ((a \otimes_C \iota_C) \cdot_C C.\text{assoc } a \mathcal{I}_C \mathcal{I}_C)$

$= (F a \otimes_D \iota_D) \cdot_D D.\text{assoc} (F a) \mathcal{I}_D \mathcal{I}_D$

using *assms 1 strictly-preserves-ι strictly-preserves-assoc strictly-preserves-unity strictly-preserves-tensor preserves-inv C.ι-in-hom*

by *auto*

**moreover have**  $F \text{r}_C[a] \otimes_D \mathcal{I}_D = F (\text{r}_C[a] \otimes_C \mathcal{I}_C)$

using *assms strictly-preserves-unity strictly-preserves-tensor* by *simp*

**ultimately show** *?thesis*

using *assms C.runit-char(2)* by *simp*

**qed**

**thus** *?thesis* using *assms D.runit-eqI* by *simp*

**qed**

The following are used to simplify the expression of the sublocale relationship between *strict-monoidal-functor* and *monoidal-functor*, as the definition of the latter mentions the structure map  $\varphi$ . For a strict monoidal functor, this is an identity transformation.

**interpretation** *FF: product-functor C C D D F F ..*

**interpretation** *FoTC: composite-functor C.CC.comp C D TC F ..*

**interpretation** *TDoFF: composite-functor C.CC.comp D.CC.comp D FF.map TD ..*

**lemma** *structure-is-trivial:*

**shows**  $T_D \circ FF.\text{map} = FoTC.\text{map}$

**proof**

**fix**  $x$

**have**  $C.CC.\text{arr } x \implies T_D \circ FF.\text{map } x = FoTC.\text{map } x$

**proof** –

**assume**  $x: C.CC.\text{arr } x$

**have**  $T_D \circ FF.\text{map } x = F (\text{fst } x) \otimes_D F (\text{snd } x)$

```

    using x by simp
  also have ... = FoTC.map x
    using x strictly-preserves-tensor [of fst x snd x] by simp
  finally show T_D o FF.map x = FoTC.map x by simp
qed
moreover have  $\neg C.CC.arr x \implies T_D o FF.map x = FoTC.map x$ 
  using T_D o FF.is-extensioal FoTC.is-extensioal by simp
ultimately show T_D o FF.map x = FoTC.map x by blast
qed

```

abbreviation  $\varphi$  where  $\varphi \equiv T_D o FF.map$

lemma *structure-is-natural-isomorphism*:

```

shows natural-isomorphism C.CC.comp D T_D o FF.map FoTC.map  $\varphi$ 
  using T_D o FF.as-nat-iso.natural-isomorphism-axioms structure-is-trivial by force

```

end

A strict monoidal functor is a monoidal functor.

sublocale *strict-monoidal-functor*  $\subseteq$  *monoidal-functor* C T\_C  $\alpha_C$   $\iota_C$  D T\_D  $\alpha_D$   $\iota_D$  F  $\varphi$

proof –

```

interpret FF: product-functor C C D D F F ..
interpret FoTC: composite-functor C.CC.comp C D T_C F ..
interpret T_D o FF: composite-functor C.CC.comp D.CC.comp D FF.map T_D ..
interpret  $\varphi$ : natural-isomorphism C.CC.comp D T_D o FF.map FoTC.map  $\varphi$ 
  using structure-is-natural-isomorphism by simp
show monoidal-functor C T_C  $\alpha_C$   $\iota_C$  D T_D  $\alpha_D$   $\iota_D$  F  $\varphi$ 
proof
  show D.isomorphic  $\mathcal{I}_D$  (F  $\mathcal{I}_C$ )
  proof (unfold D.isomorphic-def)
    have  $\langle \mathcal{I}_D : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.iso \mathcal{I}_D$ 
      using strictly-preserves-unity by auto
    thus  $\exists f. \langle f : \mathcal{I}_D \rightarrow_D F \mathcal{I}_C \rangle \wedge D.iso f$  by blast
  qed
  fix a b c
  assume a: C.ide a
  assume b: C.ide b
  assume c: C.ide c
  show F a_C[a, b, c]  $\cdot_D \varphi (a \otimes_C b, c) \cdot_D (\varphi (a, b) \otimes_D F c) =$ 
     $\varphi (a, b \otimes_C c) \cdot_D (F a \otimes_D \varphi (b, c)) \cdot_D a_D[F a, F b, F c]$ 
    using a b c strictly-preserves-tensor strictly-preserves-assoc
      D.comp-arr-dom D.comp-cod-arr
    by simp
  qed
qed

```

lemma *strict-monoidal-functors-compose*:

```

assumes strict-monoidal-functor B T_B  $\alpha_B$   $\iota_B$  C T_C  $\alpha_C$   $\iota_C$  F
and strict-monoidal-functor C T_C  $\alpha_C$   $\iota_C$  D T_D  $\alpha_D$   $\iota_D$  G

```

```

shows strict-monoidal-functor B TB αB ιB D TD αD ιD (G o F)
proof –
  interpret F: strict-monoidal-functor B TB αB ιB C TC αC ιC F
    using assms(1) by auto
  interpret G: strict-monoidal-functor C TC αC ιC D TD αD ιD G
    using assms(2) by auto
  interpret GoF: composite-functor B C D F G ..
  show ?thesis
    using F.strictly-preserves-T F.strictly-preserves-ι F.strictly-preserves-α
      G.strictly-preserves-T G.strictly-preserves-ι G.strictly-preserves-α
    by (unfold-locales, simp-all)
qed

```

An equivalence of monoidal categories is a monoidal functor whose underlying ordinary functor is also part of an ordinary equivalence of categories.

```

locale equivalence-of-monoidal-categories =
  C: monoidal-category C TC αC ιC +
  D: monoidal-category D TD αD ιD +
  equivalence-of-categories C D F G η ε +
  monoidal-functor D TD αD ιD C TC αC ιC F φ
for C :: 'c comp (infixr ·C 55)
and TC :: 'c * 'c ⇒ 'c
and αC :: 'c * 'c * 'c ⇒ 'c
and ιC :: 'c
and D :: 'd comp (infixr ·D 55)
and TD :: 'd * 'd ⇒ 'd
and αD :: 'd * 'd * 'd ⇒ 'd
and ιD :: 'd
and F :: 'd ⇒ 'c
and φ :: 'd * 'd ⇒ 'c
and ι :: 'c
and G :: 'c ⇒ 'd
and η :: 'd ⇒ 'd
and ε :: 'c ⇒ 'c

```

**end**

## Chapter 4

# The Free Monoidal Category

```
theory FreeMonoidalCategory
imports Category3.Subcategory MonoidalFunctor
begin
```

In this theory, we use the monoidal language of a category  $C$  defined in *Monoidal-Category.MonoidalCategory* to give a construction of the free monoidal category  $\mathcal{F}C$  generated by  $C$ . The arrows of  $\mathcal{F}C$  are the equivalence classes of formal arrows obtained by declaring two formal arrows to be equivalent if they are parallel and have the same diagonalization. Composition, tensor, and the components of the associator and unitors are all defined in terms of the corresponding syntactic constructs. After defining  $\mathcal{F}C$  and showing that it does indeed have the structure of a monoidal category, we prove the freeness: every functor from  $C$  to a monoidal category  $D$  extends uniquely to a strict monoidal functor from  $\mathcal{F}C$  to  $D$ .

We then consider the full subcategory  $\mathcal{F}_S C$  of  $\mathcal{F}C$  whose objects are the equivalence classes of diagonal identity terms (*i.e.* equivalence classes of lists of identity arrows of  $C$ ), and we show that this category is monoidally equivalent to  $\mathcal{F}C$ . In addition, we show that  $\mathcal{F}_S C$  is the free strict monoidal category, as any functor from  $C$  to a strict monoidal category  $D$  extends uniquely to a strict monoidal functor from  $\mathcal{F}_S C$  to  $D$ .

### 4.1 Syntactic Construction

```
locale free-monoidal-category =
  monoidal-language C
for C :: 'c comp
begin

  no-notation C.in-hom («- : - → -»)
  notation C.in-hom («- : - →C -»)
```

Two terms of the monoidal language of  $C$  are defined to be equivalent if they are parallel formal arrows with the same diagonalization.

```
abbreviation equiv
```

**where**  $\text{equiv } t \ u \equiv \text{Par } t \ u \wedge \lfloor t \rfloor = \lfloor u \rfloor$

Arrows of  $\mathcal{FC}$  will be the equivalence classes of formal arrows determined by the relation  $\text{equiv}$ . We define here the property of being an equivalence class of the relation  $\text{equiv}$ . Later we show that this property coincides with that of being an arrow of the category that we will construct.

**type-synonym**  $'a \ \text{arr} = 'a \ \text{term set}$

**definition**  $\text{ARR}$  **where**  $\text{ARR } f \equiv f \neq \{\} \wedge (\forall t. t \in f \longrightarrow f = \text{Collect } (\text{equiv } t))$

**lemma**  $\text{not-ARR-empty}$ :

**shows**  $\neg \text{ARR } \{\}$

**using**  $\text{ARR-def}$  **by**  $\text{simp}$

**lemma**  $\text{ARR-eqI}$ :

**assumes**  $\text{ARR } f$  **and**  $\text{ARR } g$  **and**  $f \cap g \neq \{\}$

**shows**  $f = g$

**using**  $\text{assms } \text{ARR-def}$  **by**  $\text{fastforce}$

We will need to choose a representative of each equivalence class as a normal form. The requirements we have of these representatives are: (1) the normal form of an arrow  $t$  is equivalent to  $t$ ; (2) equivalent arrows have identical normal forms; (3) a normal form is a canonical term if and only if its diagonalization is an identity. It follows from these properties and coherence that a term and its normal form have the same evaluation in any monoidal category. We choose here as a normal form for an arrow  $t$  the particular term  $\text{Inv } (\text{Cod } t \downarrow) \cdot \lfloor t \rfloor \cdot \text{Dom } t \downarrow$ . However, the only specific properties of this definition we actually use are the three we have just stated.

**definition**  $\text{norm } (\|-\|)$

**where**  $\|t\| = \text{Inv } (\text{Cod } t \downarrow) \cdot \lfloor t \rfloor \cdot \text{Dom } t \downarrow$

If  $t$  is a formal arrow, then  $t$  is equivalent to its normal form.

**lemma**  $\text{equiv-norm-Arr}$ :

**assumes**  $\text{Arr } t$

**shows**  $\text{equiv } \|t\| \ t$

**proof** –

**have**  $\text{Par } t \ (\text{Inv } (\text{Cod } t \downarrow) \cdot \lfloor t \rfloor \cdot \text{Dom } t \downarrow)$

**using**  $\text{assms } \text{Diagonalize-in-Hom } \text{red-in-Hom } \text{Inv-in-Hom } \text{Arr-implies-Ide-Dom}$   
 $\text{Arr-implies-Ide-Cod } \text{Ide-implies-Arr } \text{Can-red}$

**by**  $\text{auto}$

**moreover have**  $\lfloor (\text{Inv } (\text{Cod } t \downarrow) \cdot \lfloor t \rfloor \cdot \text{Dom } t \downarrow) \rfloor = \lfloor t \rfloor$

**using**  $\text{assms } \text{Arr-implies-Ide-Dom } \text{Arr-implies-Ide-Cod } \text{Diagonalize-preserves-Ide}$   
 $\text{Diagonalize-in-Hom } \text{Diagonalize-Inv } [\text{of } \text{Cod } t \downarrow] \text{Diag-Diagonalize}$   
 $\text{CompDiag-Diag-Dom } [\text{of } \lfloor t \rfloor] \text{CompDiag-Cod-Diag } [\text{of } \lfloor t \rfloor]$

**by**  $(\text{simp } \text{add: } \text{Diagonalize-red } [\text{of } \text{Cod } t] \text{Can-red}(1))$

**ultimately show**  $?thesis$  **using**  $\text{norm-def}$  **by**  $\text{simp}$

**qed**

Equivalent arrows have identical normal forms.

**lemma**  $\text{norm-respects-equiv}$ :

**assumes** *equiv t u*  
**shows**  $\|t\| = \|u\|$   
**using** *assms norm-def by simp*

The normal form of an arrow is canonical if and only if its diagonalization is an identity term.

**lemma** *Can-norm-iff-Ide-Diagonalize*:  
**assumes** *Arr t*  
**shows**  $\text{Can } \|t\| \longleftrightarrow \text{Ide } \lfloor t \rfloor$   
**using** *assms norm-def Can-implies-Arr Arr-implies-Ide-Dom Arr-implies-Ide-Cod Can-red Inv-preserves-Can Diagonalize-preserves-Can red-in-Hom Diagonalize-in-Hom Ide-Diagonalize-Can*  
**by** *fastforce*

We now establish various additional properties of normal forms that are consequences of the three already proved. The definition *norm-def* is not used subsequently.

**lemma** *norm-preserves-Can*:  
**assumes** *Can t*  
**shows**  $\text{Can } \|t\|$   
**using** *assms Can-implies-Arr Can-norm-iff-Ide-Diagonalize Ide-Diagonalize-Can by simp*

**lemma** *Par-Arr-norm*:  
**assumes** *Arr t*  
**shows**  $\text{Par } \|t\| \ t$   
**using** *assms equiv-norm-Arr by auto*

**lemma** *Diagonalize-norm [simp]*:  
**assumes** *Arr t*  
**shows**  $\lfloor \|t\| \rfloor = \lfloor t \rfloor$   
**using** *assms equiv-norm-Arr by auto*

**lemma** *unique-norm*:  
**assumes** *ARR f*  
**shows**  $\exists! t. \forall u. u \in f \longrightarrow \|u\| = t$   
**proof**  
**have**  $1: (\text{SOME } t. t \in f) \in f$   
**using** *assms ARR-def someI-ex [of  $\lambda t. t \in f$ ] by auto*  
**show**  $\bigwedge t. \forall u. u \in f \longrightarrow \|u\| = t \implies t = \|\text{SOME } t. t \in f\|$   
**using** *assms ARR-def 1 by auto*  
**show**  $\forall u. u \in f \longrightarrow \|u\| = \|\text{SOME } t. t \in f\|$   
**using** *assms ARR-def 1 norm-respects-equiv by blast*  
**qed**

**lemma** *Dom-norm*:  
**assumes** *Arr t*  
**shows**  $\text{Dom } \|t\| = \text{Dom } t$   
**using** *assms Par-Arr-norm by metis*

**lemma** *Cod-norm*:

**assumes**  $Arr\ t$   
**shows**  $Cod\ ||t|| = Cod\ t$   
**using** *assms Par-Arr-norm* **by** *metis*

**lemma** *norm-in-Hom*:  
**assumes**  $Arr\ t$   
**shows**  $||t|| \in Hom\ (Dom\ t)\ (Cod\ t)$   
**using** *assms Par-Arr-norm [of t]* **by** *simp*

As all the elements of an equivalence class have the same normal form, we can use the normal form of an arbitrarily chosen element as a canonical representative.

**definition** *rep* **where**  $rep\ f \equiv ||SOME\ t.\ t \in f||$

**lemma** *rep-in-ARR*:  
**assumes**  $ARR\ f$   
**shows**  $rep\ f \in f$   
**using** *assms ARR-def someI-ex [of  $\lambda t.\ t \in f$ ]* *equiv-norm-Arr rep-def ARR-def*  
**by** *fastforce*

**lemma** *Arr-rep-ARR*:  
**assumes**  $ARR\ f$   
**shows**  $Arr\ (rep\ f)$   
**using** *assms ARR-def rep-in-ARR* **by** *auto*

We next define a function *mkarr* that maps formal arrows to their equivalence classes. For terms that are not formal arrows, the function yields the empty set.

**definition** *mkarr* **where**  $mkarr\ t = Collect\ (equiv\ t)$

**lemma** *mkarr-extensionality*:  
**assumes**  $\neg Arr\ t$   
**shows**  $mkarr\ t = \{\}$   
**using** *assms mkarr-def* **by** *simp*

**lemma** *ARR-mkarr*:  
**assumes**  $Arr\ t$   
**shows**  $ARR\ (mkarr\ t)$   
**using** *assms ARR-def mkarr-def* **by** *auto*

**lemma** *mkarr-memb-ARR*:  
**assumes**  $ARR\ f$  **and**  $t \in f$   
**shows**  $mkarr\ t = f$   
**using** *assms ARR-def mkarr-def* **by** *simp*

**lemma** *mkarr-rep-ARR [simp]*:  
**assumes**  $ARR\ f$   
**shows**  $mkarr\ (rep\ f) = f$   
**using** *assms rep-in-ARR mkarr-memb-ARR* **by** *auto*

**lemma** *Arr-in-mkarr*:

**assumes**  $Arr\ t$   
**shows**  $t \in mkarr\ t$   
**using** *assms mkarr-def* **by** *simp*

Two terms are related by *equiv* iff they are both formal arrows and have identical normal forms.

**lemma** *equiv-iff-eq-norm*:  
**shows**  $equiv\ t\ u \longleftrightarrow Arr\ t \wedge Arr\ u \wedge \|t\| = \|u\|$   
**proof**  
**show**  $equiv\ t\ u \implies Arr\ t \wedge Arr\ u \wedge \|t\| = \|u\|$   
**using** *mkarr-def Arr-in-mkarr ARR-mkarr unique-norm* **by** *blast*  
**show**  $Arr\ t \wedge Arr\ u \wedge \|t\| = \|u\| \implies equiv\ t\ u$   
**using** *Par-Arr-norm Diagonalize-norm* **by** *metis*  
**qed**

**lemma** *norm-norm* [*simp*]:  
**assumes**  $Arr\ t$   
**shows**  $\|\|t\|\| = \|t\|$   
**proof** –  
**have**  $t \in mkarr\ t$   
**using** *assms Arr-in-mkarr* **by** *blast*  
**moreover have**  $\|t\| \in mkarr\ t$   
**using** *assms equiv-norm-Arr mkarr-def* **by** *simp*  
**ultimately show** *?thesis* **using** *assms ARR-mkarr unique-norm* **by** *auto*  
**qed**

**lemma** *norm-in-ARR*:  
**assumes**  $ARR\ f$  **and**  $t \in f$   
**shows**  $\|t\| \in f$   
**using** *assms ARR-def equiv-iff-eq-norm norm-norm Par-Arr-norm* **by** *fastforce*

**lemma** *norm-rep-ARR* [*simp*]:  
**assumes**  $ARR\ f$   
**shows**  $\|rep\ f\| = rep\ f$   
**using** *assms ARR-def someI-ex* [of  $\lambda t. t \in f$ ] *rep-def norm-norm* **by** *fastforce*

**lemma** *norm-memb-eq-rep-ARR*:  
**assumes**  $ARR\ f$  **and**  $t \in f$   
**shows**  $norm\ t = rep\ f$   
**using** *assms ARR-def someI-ex* [of  $\lambda t. t \in f$ ] *unique-norm rep-def* **by** *metis*

**lemma** *rep-mkarr*:  
**assumes**  $Arr\ f$   
**shows**  $rep\ (mkarr\ f) = \|f\|$   
**using** *assms ARR-mkarr Arr-in-mkarr norm-memb-eq-rep-ARR* **by** *fastforce*

To prove that two terms determine the same equivalence class, it suffices to show that they are parallel formal arrows with identical diagonalizations.

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



**assumes**  $Par\ f\ g$  **and**  $\lfloor f \rfloor = \lfloor g \rfloor$   
**shows**  $mkarr\ f = mkarr\ g$   
**using** *assms* **by** (*metis* *ARR-mkarr-equiv-iff-eq-norm* *rep-mkarr* *mkarr-rep-ARR*)

We use canonical representatives to lift the formal domain and codomain functions from terms to equivalence classes.

**abbreviation**  $DOM$  **where**  $DOM\ f \equiv Dom\ (rep\ f)$   
**abbreviation**  $COD$  **where**  $COD\ f \equiv Cod\ (rep\ f)$

**lemma** *DOM-mkarr*:  
**assumes**  $Arr\ t$   
**shows**  $DOM\ (mkarr\ t) = Dom\ t$   
**using** *assms* *rep-mkarr* **by** (*metis* *Par-Arr-norm*)

**lemma** *COD-mkarr*:  
**assumes**  $Arr\ t$   
**shows**  $COD\ (mkarr\ t) = Cod\ t$   
**using** *assms* *rep-mkarr* **by** (*metis* *Par-Arr-norm*)

A composition operation can now be defined on equivalence classes using the syntactic constructor *Comp*.

**definition** *comp* (**infixr**  $\cdot$  55)  
**where**  $comp\ f\ g \equiv (if\ ARR\ f \wedge ARR\ g \wedge DOM\ f = COD\ g$   
 $then\ mkarr\ ((rep\ f) \cdot (rep\ g))\ else\ \{\})$

We commence the task of showing that the composition *comp* so defined determines a category.

**interpretation** *partial-magma* *comp*  
**apply** *unfold-locales*  
**using** *comp-def* *not-ARR-empty* **by** *metis*

**notation** *in-hom* ( $\llcorner : - \rightarrow - \lrcorner$ )

The empty set serves as the null for the composition.

**lemma** *null-char*:  
**shows**  $null = \{\}$   
**proof** –  
**let**  $?P = \lambda n. \forall f. f \cdot n = n \wedge n \cdot f = n$   
**have**  $?P\ \{\}$  **using** *comp-def* *not-ARR-empty* **by** *simp*  
**moreover** **have**  $\exists! n. ?P\ n$  **using** *ex-un-null* **by** *metis*  
**ultimately** **show** *?thesis* **using** *null-def* *theI-unique* [*of*  $?P\ \{\}$ ]  
**by** (*metis* *comp-null(2)*)  
**qed**

**lemma** *ARR-comp*:  
**assumes**  $ARR\ f$  **and**  $ARR\ g$  **and**  $DOM\ f = COD\ g$   
**shows**  $ARR\ (f \cdot g)$   
**using** *assms* *comp-def* *Arr-rep-ARR* *ARR-mkarr(1)* **by** *simp*

**lemma** *DOM-comp* [*simp*]:  
**assumes** *ARR f and ARR g and DOM f = COD g*  
**shows**  $DOM (f \cdot g) = DOM g$   
**using** *assms comp-def ARR-comp Arr-rep-ARR DOM-mkarr by simp*

**lemma** *COD-comp* [*simp*]:  
**assumes** *ARR f and ARR g and DOM f = COD g*  
**shows**  $COD (f \cdot g) = COD f$   
**using** *assms comp-def ARR-comp Arr-rep-ARR COD-mkarr by simp*

**lemma** *comp-assoc*:  
**assumes**  $g \cdot f \neq null$  **and**  $h \cdot g \neq null$   
**shows**  $h \cdot (g \cdot f) = (h \cdot g) \cdot f$   
**proof** –  
**have** 1:  $ARR f \wedge ARR g \wedge ARR h \wedge DOM h = COD g \wedge DOM g = COD f$   
**using** *assms comp-def not-ARR-empty null-char by metis*  
**hence** 2:  $Arr (rep f) \wedge Arr (rep g) \wedge Arr (rep h) \wedge$   
 $Dom (rep h) = Cod (rep g) \wedge Dom (rep g) = Cod (rep f)$   
**using** *Arr-rep-ARR by simp*  
**have** 3:  $h \cdot g \cdot f = mkarr (rep h \cdot rep (mkarr (rep g \cdot rep f)))$   
**using** 1 2 3 *comp-def ARR-comp COD-comp by simp*  
**also have** ... =  $mkarr (rep h \cdot rep g \cdot rep f)$   
**proof** –  
**have** *equiv*  $(rep h \cdot rep (mkarr (rep g \cdot rep f))) (rep h \cdot rep g \cdot rep f)$   
**proof** –  
**have** *Par*  $(rep h \cdot rep (mkarr (rep g \cdot rep f))) (rep h \cdot rep g \cdot rep f)$   
**using** 1 2 3 *DOM-mkarr ARR-comp COD-comp mkarr-extensionality not-ARR-empty*  
**by** (*metis Arr.simps(4) Cod.simps(4) Dom.simps(4) snd-map-prod*)  
  
**moreover have**  $\lfloor rep h \cdot rep (mkarr (rep g \cdot rep f)) \rfloor = \lfloor rep h \cdot rep g \cdot rep f \rfloor$   
**using** 1 2 *Arr-rep-ARR rep-mkarr rep-in-ARR assms(1) ARR-comp mkarr-extensionality*  
 $comp-def equiv-iff-eq-norm norm-memb-eq-rep-ARR null-char$   
**by** *auto*  
**ultimately show** *?thesis using equiv-iff-eq-norm by blast*  
**qed**  
**thus** *?thesis*  
**using** *mkarr-def by force*  
**qed**  
**also have** ... =  $mkarr ((rep h \cdot rep g) \cdot rep f)$   
**proof** –  
**have** *Par*  $(rep h \cdot rep g \cdot rep f) ((rep h \cdot rep g) \cdot rep f)$   
**using** 2 *by simp*  
**moreover have**  $\lfloor rep h \cdot rep g \cdot rep f \rfloor = \lfloor (rep h \cdot rep g) \cdot rep f \rfloor$   
**using** 2 *Diag-Diagonalize by (simp add: CompDiag-assoc)*  
**ultimately show** *?thesis*  
**using** *equiv-iff-eq-norm by (simp add: mkarr-def)*  
**qed**  
**also have** ... =  $mkarr (rep (mkarr (rep h \cdot rep g)) \cdot rep f)$   
**proof** –

```

have equiv (rep (mkarr (rep h · rep g)) · rep f) ((rep h · rep g) · rep f)
proof –
  have Par (rep (mkarr (rep h · rep g)) · rep f) ((rep h · rep g) · rep f)
    using 1 2 Arr-rep-ARR DOM-comp ARR-comp COD-comp comp-def by auto
  moreover have [rep (mkarr (rep h · rep g)) · rep f] = [(rep h · rep g) · rep f]
  using assms(2) 1 2 ARR-comp Arr-rep-ARR mkarr-extensionality rep-mkarr rep-in-ARR
    equiv-iff-eq-norm norm-memb-eq-rep-ARR comp-def null-char
  by simp
  ultimately show ?thesis using equiv-iff-eq-norm by blast
qed
thus ?thesis
  using mkarr-def by auto
qed
also have ... = (h · g) · f
  using 1 comp-def ARR-comp DOM-comp by simp
finally show ?thesis by blast
qed

```

```

lemma Comp-in-comp-ARR:
assumes ARR f and ARR g and DOM f = COD g
and t ∈ f and u ∈ g
shows t · u ∈ f · g
proof –
  have equiv (t · u) (rep f · rep g)
proof –
  have 1: Par (t · u) (rep f · rep g)
    using assms ARR-def Arr-rep-ARR COD-mkarr DOM-mkarr mkarr-memb-ARR
      mkarr-extensionality
    by (metis (no-types, lifting) Arr.simps(4) Cod.simps(4) Dom.simps(4) snd-map-prod)

  moreover have [t · u] = [rep f · rep g]
    using assms 1 rep-in-ARR equiv-iff-eq-norm norm-memb-eq-rep-ARR
    by (metis (no-types, lifting) Arr.simps(4) Diagonalize.simps(4))
  ultimately show ?thesis by simp
qed
thus ?thesis
  using assms comp-def mkarr-def by simp
qed

```

Ultimately, we will show that that the identities of the category are those equivalence classes, all of whose members diagonalize to formal identity arrows, having the further property that their canonical representative is a formal endo-arrow.

```

definition IDE where IDE f ≡ ARR f ∧ (∀ t. t ∈ f → Ide [t]) ∧ DOM f = COD f

```

```

lemma IDE-implies-ARR:
assumes IDE f
shows ARR f
  using assms IDE-def ARR-def by auto

```

**lemma** *IDE-mkarr-Ide*:  
**assumes** *Ide a*  
**shows** *IDE (mkarr a)*  
**proof** –  
    **have** *DOM (mkarr a) = COD (mkarr a)*  
    **using** *assms mkarr-def equiv-iff-eq-norm Par-Arr-norm COD-mkarr DOM-mkarr Ide-in-Hom*  
    **by** (*metis Ide-implies-Can Inv-Ide Ide-implies-Arr Inv-preserves-Can(2)*)  
    **moreover have** *ARR (mkarr a)  $\wedge$  ( $\forall t. t \in \text{mkarr } a \longrightarrow \text{Ide } [t]$ )*  
    **proof** –  
        **have** *ARR (mkarr a)* **using** *assms ARR-mkarr Ide-implies-Arr* **by** *simp*  
        **moreover have**  $\forall t. t \in \text{mkarr } a \longrightarrow \text{Ide } [t]$   
        **using** *assms mkarr-def Diagonalize-preserves-Ide* **by** *fastforce*  
        **ultimately show** *?thesis* **by** *blast*  
    **qed**  
    **ultimately show** *?thesis* **using** *IDE-def* **by** *blast*  
**qed**

**lemma** *IDE-implies-ide*:  
**assumes** *IDE a*  
**shows** *ide a*  
**proof** (*unfold ide-def*)  
    **have** *a  $\cdot$  a  $\neq$  null*  
    **proof** –  
        **have** *rep a  $\cdot$  rep a  $\in$  a  $\cdot$  a*  
        **using** *assms IDE-def comp-def Arr-rep-ARR Arr-in-mkarr* **by** *simp*  
    **thus** *?thesis*  
    **using** *null-char* **by** *auto*  
**qed**  
**moreover have**  $\bigwedge f. (f \cdot a \neq \text{null} \longrightarrow f \cdot a = f) \wedge (a \cdot f \neq \text{null} \longrightarrow a \cdot f = f)$   
**proof**  
    **fix** *f :: 'c arr*  
    **show** *a  $\cdot$  f  $\neq$  null  $\longrightarrow$  a  $\cdot$  f = f*  
    **proof**  
        **assume** *f: a  $\cdot$  f  $\neq$  null*  
        **hence** *ARR f*  
        **using** *comp-def null-char* **by** *auto*  
        **have** *rep a  $\cdot$  rep f  $\in$  a  $\cdot$  f*  
        **using** *assms f Comp-in-comp-ARR comp-def rep-in-ARR null-char* **by** *metis*  
        **moreover have** *rep a  $\cdot$  rep f  $\in$  f*  
        **proof** –  
            **have** *rep f  $\in$  f*  
            **using**  $\langle \text{ARR } f \rangle$  *rep-in-ARR* **by** *auto*  
            **moreover have** *equiv (rep a  $\cdot$  rep f) (rep f)*  
            **proof** –  
                **have** *1: Par (rep a  $\cdot$  rep f) (rep f)*  
                **using** *assms f comp-def mkarr-extensionality Arr-rep-ARR IDE-def null-char*  
                **by** (*metis Cod.simps(4) Dom.simps(4)*)  
                **moreover have**  $[rep a \cdot rep f] = [rep f]$   
                **using** *assms f 1 comp-def IDE-def CompDiag-Ide-Diag Diag-Diagonalize(1)*  
        **qed**

```

      Diag-Diagonalize(2) Diag-Diagonalize(3) rep-in-ARR
    by auto
    ultimately show ?thesis by auto
  qed
  ultimately show ?thesis
    using  $\langle ARR\ f \rangle$  ARR-def by auto
  qed
  ultimately show  $a \cdot f = f$ 
    using mkarr-memb-ARR comp-def by auto
  qed
  show  $f \cdot a \neq null \longrightarrow f \cdot a = f$ 
  proof
    assume  $f: f \cdot a \neq null$ 
    hence ARR f
      using comp-def null-char by auto
    have  $rep\ f \cdot rep\ a \in f \cdot a$ 
      using assms f Comp-in-comp-ARR comp-def rep-in-ARR null-char by metis
    moreover have  $rep\ f \cdot rep\ a \in f$ 
    proof -
      have  $rep\ f \in f$ 
        using  $\langle ARR\ f \rangle$  rep-in-ARR by auto
      moreover have equiv  $(rep\ f \cdot rep\ a)$   $(rep\ f)$ 
    proof -
      have  $1: Par\ (rep\ f \cdot rep\ a)\ (rep\ f)$ 
        using assms f comp-def mkarr-extensionality Arr-rep-ARR IDE-def null-char
        by  $(metis\ Cod.simps(4)\ Dom.simps(4))$ 
      moreover have  $\lfloor rep\ f \cdot rep\ a \rfloor = \lfloor rep\ f \rfloor$ 
        using assms f 1 comp-def IDE-def CompDiag-Diag-Ide
          Diag-Diagonalize(1) Diag-Diagonalize(2) Diag-Diagonalize(3)
          rep-in-ARR
        by force
      ultimately show ?thesis by auto
    qed
  qed
  ultimately show ?thesis
    using  $\langle ARR\ f \rangle$  ARR-def by auto
  qed
  ultimately show  $f \cdot a = f$ 
    using mkarr-memb-ARR comp-def by auto
  qed
  qed
  ultimately show  $a \cdot a \neq null \wedge$ 
     $(\forall f. (f \cdot a \neq null \longrightarrow f \cdot a = f) \wedge (a \cdot f \neq null \longrightarrow a \cdot f = f))$ 
    by blast
  qed

lemma ARR-iff-has-domain:
shows  $ARR\ f \longleftrightarrow domains\ f \neq \{\}$ 
proof
  assume  $f: domains\ f \neq \{\}$ 

```

**show**  $ARR\ f$  **using**  $f$  *domains-def comp-def null-char* **by** *auto*  
**next**  
**assume**  $f: ARR\ f$   
**have**  $Ide\ (DOM\ f)$   
**using**  $f$  *ARR-def* **by** (*simp add: Arr-implies-Ide-Dom Arr-rep-ARR*)  
**hence**  $IDE\ (mkarr\ (DOM\ f))$  **using** *IDE-mkarr-Ide* **by** *metis*  
**hence**  $ide\ (mkarr\ (DOM\ f))$  **using** *IDE-implies-ide* **by** *simp*  
**moreover** **have**  $f \cdot mkarr\ (DOM\ f) = f$   
**proof** –  
**have**  $1: rep\ f \cdot DOM\ f \in f \cdot mkarr\ (DOM\ f)$   
**using**  $f$  *Comp-in-comp-ARR*  
**using** *IDE-implies-ARR Ide-in-Hom rep-in-ARR*  $\langle IDE\ (mkarr\ (DOM\ f)) \rangle$   
 $\langle Ide\ (DOM\ f) \rangle$  *Arr-in-mkarr COD-mkarr*  
**by** *fastforce*  
**moreover** **have**  $rep\ f \cdot DOM\ f \in f$   
**proof** –  
**have**  $2: rep\ f \in f$  **using**  $f$  *rep-in-ARR* **by** *simp*  
**moreover** **have** *equiv*  $(rep\ f \cdot DOM\ f)\ (rep\ f)$   
**by** (*metis 1 Arr.simps(4) Arr-rep-ARR COD-mkarr Cod.simps(4)*  
*Diagonalize-Comp-Arr-Dom Dom.simps(4) IDE-def Ide-implies-Arr*  
 $\langle IDE\ (mkarr\ (DOM\ f)) \rangle$   $\langle Ide\ (DOM\ f) \rangle$  *all-not-in-conv DOM-mkarr comp-def*)  
**ultimately** **show** *?thesis*  
**using**  $f$  *ARR-eqI 1*  $\langle ide\ (mkarr\ (DOM\ f)) \rangle$  *null-char ide-def* **by** *auto*  
**qed**  
**ultimately** **show** *?thesis*  
**using**  $f$  *ARR-eqI*  $\langle ide\ (mkarr\ (DOM\ f)) \rangle$  *null-char ide-def* **by** *auto*  
**qed**  
**ultimately** **show**  $domains\ f \neq \{\}$   
**using**  $f$  *domains-def not-ARR-empty null-char* **by** *auto*  
**qed**

**lemma** *ARR-iff-has-codomain:*

**shows**  $ARR\ f \iff codomains\ f \neq \{\}$

**proof**

**assume**  $f: codomains\ f \neq \{\}$

**show**  $ARR\ f$  **using**  $f$  *codomains-def comp-def null-char* **by** *auto*

**next**

**assume**  $f: ARR\ f$

**have**  $Ide\ (COD\ f)$

**using**  $f$  *ARR-def* **by** (*simp add: Arr-rep-ARR Arr-implies-Ide-Cod*)

**hence**  $IDE\ (mkarr\ (COD\ f))$  **using** *IDE-mkarr-Ide* **by** *metis*

**hence**  $ide\ (mkarr\ (COD\ f))$  **using** *IDE-implies-ide* **by** *simp*

**moreover** **have**  $mkarr\ (COD\ f) \cdot f = f$

**proof** –

**have**  $1: COD\ f \cdot rep\ f \in mkarr\ (COD\ f) \cdot f$

**using**  $f$  *Comp-in-comp-ARR*

**using** *IDE-implies-ARR Ide-in-Hom rep-in-ARR*  $\langle IDE\ (mkarr\ (COD\ f)) \rangle$

$\langle Ide\ (COD\ f) \rangle$  *Arr-in-mkarr DOM-mkarr*

**by** *fastforce*

```

moreover have  $COD\ f \cdot rep\ f \in f$ 
  using 1 null-char norm-rep-ARR norm-memb-eq-rep-ARR mkarr-memb-ARR
     $\langle ide\ (mkarr\ (COD\ f)) \rangle\ emptyE\ equiv-iff-eq-norm\ mkarr-extensionality\ ide-def$ 
  by metis
ultimately show ?thesis
  using  $f\ ARR-eqI\ \langle ide\ (mkarr\ (COD\ f)) \rangle\ null-char\ ide-def$  by auto
qed
ultimately show  $codomains\ f \neq \{\}$ 
  using  $codomains-def\ f\ not-ARR-empty\ null-char$  by auto
qed

```

```

lemma arr-iff-ARR:
shows  $arr\ f \longleftrightarrow ARR\ f$ 
  using  $arr-def\ ARR-iff-has-domain\ ARR-iff-has-codomain$  by simp

```

The arrows of the category are the equivalence classes of formal arrows.

```

lemma arr-char:
shows  $arr\ f \longleftrightarrow f \neq \{\} \wedge (\forall t. t \in f \longrightarrow f = mkarr\ t)$ 
  using  $arr-iff-ARR\ ARR-def\ mkarr-def$  by simp

```

```

lemma seq-char:
shows  $seq\ g\ f \longleftrightarrow g \cdot f \neq null$ 
proof
  show  $g \cdot f \neq null \implies seq\ g\ f$ 
    using  $comp-def\ null-char\ Comp-in-comp-ARR\ rep-in-ARR\ ARR-mkarr$ 
       $Arr-rep-ARR\ arr-iff-ARR$ 
    by auto
  show  $seq\ g\ f \implies g \cdot f \neq null$ 
    by auto
qed

```

```

lemma seq-char':
shows  $seq\ g\ f \longleftrightarrow ARR\ f \wedge ARR\ g \wedge DOM\ g = COD\ f$ 
proof
  show  $ARR\ f \wedge ARR\ g \wedge DOM\ g = COD\ f \implies seq\ g\ f$ 
    using  $comp-def\ null-char\ Comp-in-comp-ARR\ rep-in-ARR\ ARR-mkarr$ 
       $Arr-rep-ARR\ arr-iff-ARR$ 
    by auto
  have  $\neg (ARR\ f \wedge ARR\ g \wedge DOM\ g = COD\ f) \implies g \cdot f = null$ 
    using  $comp-def\ null-char$  by auto
  thus  $seq\ g\ f \implies ARR\ f \wedge ARR\ g \wedge DOM\ g = COD\ f$ 
    using ext by fastforce
qed

```

Finally, we can show that the composition *comp* determines a category.

```

interpretation category comp
proof
  show  $\bigwedge f. domains\ f \neq \{\} \longleftrightarrow codomains\ f \neq \{\}$ 
    using  $ARR-iff-has-domain\ ARR-iff-has-codomain$  by simp

```

```

show 1:  $\bigwedge f g. g \cdot f \neq \text{null} \implies \text{seq } g f$ 
  using comp-def ARR-comp null-char arr-iff-ARR by metis
fix f g h
show  $\text{seq } h g \implies \text{seq } (h \cdot g) f \implies \text{seq } g f$ 
  using seq-char' by auto
show  $\text{seq } h (g \cdot f) \implies \text{seq } g f \implies \text{seq } h g$ 
  using seq-char' by auto
show  $\text{seq } g f \implies \text{seq } h g \implies \text{seq } (h \cdot g) f$ 
  using seq-char' ARR-comp arr-iff-ARR by auto
show  $\text{seq } g f \implies \text{seq } h g \implies (h \cdot g) \cdot f = h \cdot g \cdot f$ 
  using seq-char comp-assoc by auto
qed

```

```

lemma mkarr-rep [simp]:
assumes arr f
shows  $\text{mkarr } (\text{rep } f) = f$ 
  using assms arr-iff-ARR by simp

```

```

lemma arr-mkarr [simp]:
assumes Arr t
shows  $\text{arr } (\text{mkarr } t)$ 
  using assms by (simp add: ARR-mkarr arr-iff-ARR)

```

```

lemma mkarr-memb:
assumes  $t \in f$  and arr f
shows  $\text{Arr } t$  and  $\text{mkarr } t = f$ 
  using assms arr-char mkarr-extensionality by auto

```

```

lemma rep-in-arr [simp]:
assumes arr f
shows  $\text{rep } f \in f$ 
  using assms by (simp add: rep-in-ARR arr-iff-ARR)

```

```

lemma Arr-rep [simp]:
assumes arr f
shows  $\text{Arr } (\text{rep } f)$ 
  using assms mkarr-memb rep-in-arr by blast

```

```

lemma rep-in-Hom:
assumes arr f
shows  $\text{rep } f \in \text{Hom } (\text{DOM } f) (\text{COD } f)$ 
  using assms by simp

```

```

lemma norm-memb-eq-rep:
assumes arr f and  $t \in f$ 
shows  $\|t\| = \text{rep } f$ 
  using assms arr-iff-ARR norm-memb-eq-rep-ARR by auto

```

```

lemma norm-rep:

```



**assumes**  $arr\ f$   
**shows**  $\|rep\ f\| = rep\ f$   
**using**  $assms\ norm-memb-eq-rep$  **by**  $simp$

Composition, domain, and codomain on arrows reduce to the corresponding syntactic operations on their representative terms.

**lemma**  $comp-mkarr$  [ $simp$ ]:  
**assumes**  $Arr\ t$  **and**  $Arr\ u$  **and**  $Dom\ t = Cod\ u$   
**shows**  $mkarr\ t \cdot mkarr\ u = mkarr\ (t \cdot u)$   
**using**  $assms$   
**by** ( $metis$  ( $no-types$ ,  $lifting$ )  $ARR-mkarr\ ARR-comp\ ARR-def\ Arr-in-mkarr\ COD-mkarr\ Comp-in-comp-ARR\ DOM-mkarr\ mkarr-def$ )

**lemma**  $dom-char$ :  
**shows**  $dom\ f = (if\ arr\ f\ then\ mkarr\ (DOM\ f)\ else\ null)$   
**proof** –  
**have**  $\neg arr\ f \implies ?thesis$   
**using**  $dom-def$  **by** ( $simp\ add: arr-def$ )  
**moreover** **have**  $arr\ f \implies ?thesis$   
**proof** –  
**assume**  $f: arr\ f$   
**have**  $dom\ f = mkarr\ (DOM\ f)$   
**proof** ( $intro\ dom-eqI$ )  
**have**  $1: Ide\ (DOM\ f)$   
**using**  $f\ arr-char$  **by** ( $metis\ Arr-rep\ Arr-implies-Ide-Dom$ )  
**hence**  $2: IDE\ (mkarr\ (DOM\ f))$   
**using**  $IDE-mkarr-Ide$  **by**  $metis$   
**thus**  $ide\ (mkarr\ (DOM\ f))$  **using**  $IDE-implies-ide$  **by**  $simp$   
**moreover** **show**  $seq\ f\ (mkarr\ (DOM\ f))$   
**proof** –  
**have**  $f \cdot mkarr\ (DOM\ f) \neq null$   
**using**  $f\ 1\ 2\ ARR-def\ DOM-mkarr\ IDE-implies-ARR\ Ide-in-Hom\ ARR-comp\ IDE-def\ ARR-iff-has-codomain\ ARR-iff-has-domain\ null-char\ arr-def$   
**by** ( $metis$  ( $mono-tags$ ,  $lifting$ )  $mem-Collect-eq$ )  
**thus**  $?thesis$  **using**  $seq-char$  **by**  $simp$   
**qed**  
**qed**  
**thus**  $?thesis$  **using**  $f$  **by**  $simp$   
**qed**  
**ultimately** **show**  $?thesis$  **by**  $blast$   
**qed**

**lemma**  $dom-simp$ :  
**assumes**  $arr\ f$   
**shows**  $dom\ f = mkarr\ (DOM\ f)$   
**using**  $assms\ dom-char$  **by**  $simp$

**lemma**  $cod-char$ :  
**shows**  $cod\ f = (if\ arr\ f\ then\ mkarr\ (COD\ f)\ else\ null)$

```

proof –
  have  $\neg \text{arr } f \implies ?thesis$ 
    using cod-def by (simp add: arr-def)
  moreover have  $\text{arr } f \implies ?thesis$ 
  proof –
    assume  $f: \text{arr } f$ 
    have  $\text{cod } f = \text{mkarr } (\text{COD } f)$ 
    proof (intro cod-eqI)
      have  $1: \text{Ide } (\text{COD } f)$ 
        using  $f$  arr-char by (metis Arr-rep Arr-implies-Ide-Cod)
      hence  $2: \text{IDE } (\text{mkarr } (\text{COD } f))$ 
        using IDE-mkarr-Ide by metis
      thus  $\text{ide } (\text{mkarr } (\text{COD } f))$  using IDE-implies-ide by simp
      moreover show  $\text{seq } (\text{mkarr } (\text{COD } f)) \ f$ 
      proof –
        have  $\text{mkarr } (\text{COD } f) \cdot f \neq \text{null}$ 
          using  $f$   $1$   $2$  ARR-def DOM-mkarr IDE-implies-ARR Ide-in-Hom ARR-comp IDE-def
            ARR-iff-has-codomain ARR-iff-has-domain null-char arr-def
          by (metis (mono-tags, lifting) mem-Collect-eq)
        thus  $?thesis$  using seq-char by simp
      qed
    qed
  thus  $?thesis$  using  $f$  by simp
qed
ultimately show  $?thesis$  by blast
qed

```

```

lemma cod-simp:
assumes  $\text{arr } f$ 
shows  $\text{cod } f = \text{mkarr } (\text{COD } f)$ 
  using assms cod-char by simp

```

```

lemma Dom-memb:
assumes  $\text{arr } f$  and  $t \in f$ 
shows  $\text{Dom } t = \text{DOM } f$ 
  using assms DOM-mkarr mkarr-extensionality arr-char by fastforce

```

```

lemma Cod-memb:
assumes  $\text{arr } f$  and  $t \in f$ 
shows  $\text{Cod } t = \text{COD } f$ 
  using assms COD-mkarr mkarr-extensionality arr-char by fastforce

```

```

lemma dom-mkarr [simp]:
assumes  $\text{Arr } t$ 
shows  $\text{dom } (\text{mkarr } t) = \text{mkarr } (\text{Dom } t)$ 
  using assms dom-char DOM-mkarr arr-mkarr by auto

```

```

lemma cod-mkarr [simp]:
assumes  $\text{Arr } t$ 

```

**shows**  $\text{cod } (mkarr\ t) = mkarr\ (\text{Cod } t)$   
**using** *assms cod-char COD-mkarr arr-mkarr* **by** *auto*

**lemma** *mkarr-in-hom*:  
**assumes** *Arr t*  
**shows**  $\langle\langle mkarr\ t : mkarr\ (\text{Dom } t) \rightarrow mkarr\ (\text{Cod } t) \rangle\rangle$   
**using** *assms arr-mkarr dom-mkarr cod-mkarr* **by** *auto*

**lemma** *DOM-in-dom* [*intro*]:  
**assumes** *arr f*  
**shows**  $\text{DOM } f \in \text{dom } f$   
**using** *assms dom-char*  
**by** (*metis Arr-in-mkarr mkarr-extensionality ideD(1) ide-dom not-arr-null null-char*)

**lemma** *COD-in-cod* [*intro*]:  
**assumes** *arr f*  
**shows**  $\text{COD } f \in \text{cod } f$   
**using** *assms cod-char*  
**by** (*metis Arr-in-mkarr mkarr-extensionality ideD(1) ide-cod not-arr-null null-char*)

**lemma** *DOM-dom*:  
**assumes** *arr f*  
**shows**  $\text{DOM } (\text{dom } f) = \text{DOM } f$   
**using** *assms Arr-rep Arr-implies-Ide-Dom Ide-implies-Arr dom-char rep-mkarr Par-Arr-norm*  
*Ide-in-Hom*  
**by** *simp*

**lemma** *DOM-cod*:  
**assumes** *arr f*  
**shows**  $\text{DOM } (\text{cod } f) = \text{COD } f$   
**using** *assms Arr-rep Arr-implies-Ide-Cod Ide-implies-Arr cod-char rep-mkarr Par-Arr-norm*  
*Ide-in-Hom*  
**by** *simp*

**lemma** *memb-equiv*:  
**assumes** *arr f* **and**  $t \in f$  **and**  $u \in f$   
**shows**  $\text{Par } t\ u$  **and**  $\lfloor t \rfloor = \lfloor u \rfloor$   
**proof** –  
**show**  $\text{Par } t\ u$   
**using** *assms Cod-memb Dom-memb mkarr-memb(1)* **by** *metis*  
**show**  $\lfloor t \rfloor = \lfloor u \rfloor$   
**using** *assms arr-iff-ARR ARR-def* **by** *auto*  
**qed**

Two arrows can be proved equal by showing that they are parallel and have representatives with identical diagonalizations.

**lemma** *arr-eqI*:  
**assumes**  $\text{par } f\ g$  **and**  $t \in f$  **and**  $u \in g$  **and**  $\lfloor t \rfloor = \lfloor u \rfloor$   
**shows**  $f = g$

**proof** –  
**have**  $Arr\ t \wedge Arr\ u$  **using** *assms mkarr-memb(1)* **by** *blast*  
**moreover have**  $Dom\ t = Dom\ u \wedge Cod\ t = Cod\ u$   
**using** *assms Dom-memb Cod-memb comp-def arr-char comp-arr-dom comp-cod-arr*  
**by** *(metis (full-types))*  
**ultimately have**  $Par\ t\ u$  **by** *simp*  
**thus** *?thesis*  
**using** *assms arr-char* **by** *(metis rep-mkarr rep-in-arr equiv-iff-eq-norm)*  
**qed**

**lemma** *comp-char*:  
**shows**  $f \cdot g = (if\ seq\ f\ g\ then\ mkarr\ (rep\ f \cdot rep\ g)\ else\ null)$   
**using** *comp-def seq-char arr-char* **by** *meson*

The mapping that takes identity terms to their equivalence classes is injective.

**lemma** *mkarr-inj-on-Ide*:  
**assumes**  $Ide\ t$  **and**  $Ide\ u$  **and**  $mkarr\ t = mkarr\ u$   
**shows**  $t = u$   
**using** *assms*  
**by** *(metis (mono-tags, lifting) COD-mkarr Ide-in-Hom mem-Collect-eq)*

**lemma** *Comp-in-comp* [*intro*]:  
**assumes**  $arr\ f$  **and**  $g \in hom\ (dom\ g)\ (dom\ f)$  **and**  $t \in f$  **and**  $u \in g$   
**shows**  $t \cdot u \in f \cdot g$   
**proof** –  
**have**  $ARR\ f$  **using** *assms arr-iff-ARR* **by** *simp*  
**moreover have**  $ARR\ g$  **using** *assms arr-iff-ARR* **by** *auto*  
**moreover have**  $DOM\ f = COD\ g$   
**using** *assms dom-char cod-char mkarr-inj-on-Ide Arr-implies-Ide-Cod Arr-implies-Ide-Dom*  
**by** *force*  
**ultimately show** *?thesis* **using** *assms Comp-in-comp-ARR* **by** *simp*  
**qed**

An arrow is defined to be “canonical” if some (equivalently, all) its representatives diagonalize to an identity term.

**definition** *can*  
**where**  $can\ f \equiv arr\ f \wedge (\exists t. t \in f \wedge Ide\ [t])$

**lemma** *can-def-alt*:  
**shows**  $can\ f \iff arr\ f \wedge (\forall t. t \in f \longrightarrow Ide\ [t])$   
**proof**  
**assume**  $arr\ f \wedge (\forall t. t \in f \longrightarrow Ide\ [t])$   
**thus**  $can\ f$  **using** *can-def arr-char* **by** *fastforce*  
**next**  
**assume**  $f: can\ f$   
**show**  $arr\ f \wedge (\forall t. t \in f \longrightarrow Ide\ [t])$   
**proof** –  
**obtain**  $t$  **where**  $t: t \in f \wedge Ide\ [t]$  **using**  $f$  *can-def* **by** *auto*  
**have**  $ARR\ f$  **using**  $f$  *can-def arr-char ARR-def mkarr-def* **by** *simp*

**hence**  $\forall u. u \in f \longrightarrow \|u\| = \|t\|$  **using**  $t$  *unique-norm* **by** *auto*  
**hence**  $\forall u. u \in f \longrightarrow \lfloor t \rfloor = \lfloor u \rfloor$   
**using**  $t$  **by** (*metis*  $\langle ARR f \rangle$  *equiv-iff-eq-norm arr-iff-ARR mkarr-memb(1)*)  
**hence**  $\forall u. u \in f \longrightarrow Ide \lfloor u \rfloor$   
**using**  $t$  **by** *metis*  
**thus** *?thesis* **using**  $f$  *can-def* **by** *blast*  
**qed**  
**qed**

**lemma** *can-implies-arr*:  
**assumes** *can f*  
**shows** *arr f*  
**using** *assms can-def* **by** *auto*

The identities of the category are precisely the canonical endo-arrows.

**lemma** *ide-char*:  
**shows**  $ide f \longleftrightarrow can f \wedge dom f = cod f$   
**proof**  
**assume**  $f: ide f$   
**show**  $can f \wedge dom f = cod f$   
**using**  $f$  *can-def arr-char dom-char cod-char IDE-def Arr-implies-Ide-Cod can-def-alt*  
*Arr-rep IDE-mkarr-Ide*  
**by** (*metis ideD(1) ideD(3)*)  
**next**  
**assume**  $f: can f \wedge dom f = cod f$   
**show**  $ide f$   
**proof** –  
**have**  $f = dom f$   
**proof** (*intro arr-eqI*)  
**show**  $par f (dom f)$  **using**  $f$  *can-def* **by** *simp*  
**show**  $rep f \in f$  **using**  $f$  *can-def* **by** *simp*  
**show**  $DOM f \in dom f$  **using**  $f$  *can-def* **by** *auto*  
**show**  $\lfloor rep f \rfloor = \lfloor DOM f \rfloor$   
**proof** –  
**have**  $\lfloor rep f \rfloor \in Hom \lfloor DOM f \rfloor \lfloor COD f \rfloor$   
**using**  $f$  *can-def Diagonalize-in-Hom* **by** *simp*  
**moreover** **have**  $Ide \lfloor rep f \rfloor$  **using**  $f$  *can-def-alt rep-in-arr* **by** *simp*  
**ultimately** **show** *?thesis*  
**using**  $f$  *can-def Ide-in-Hom* **by** *simp*  
**qed**  
**qed**  
**thus** *?thesis* **using**  $f$  *can-implies-arr ide-dom [of f]* **by** *auto*  
**qed**  
**qed**

**lemma** *ide-iff-IDE*:  
**shows**  $ide a \longleftrightarrow IDE a$   
**using** *ide-char IDE-def can-def-alt arr-iff-ARR dom-char cod-char mkarr-inj-on-Ide*  
*Arr-implies-Ide-Cod Arr-implies-Ide-Dom Arr-rep*

by auto

**lemma** *ide-mkarr-Ide*:

**assumes** *Ide a*

**shows** *ide (mkarr a)*

**using** *assms IDE-mkarr-Ide ide-iff-IDE* by *simp*

**lemma** *rep-dom*:

**assumes** *arr f*

**shows** *rep (dom f) = ||DOM f||*

**using** *assms dom-simp rep-mkarr Arr-rep Arr-implies-Ide-Dom* by *simp*

**lemma** *rep-cod*:

**assumes** *arr f*

**shows** *rep (cod f) = ||COD f||*

**using** *assms cod-simp rep-mkarr Arr-rep Arr-implies-Ide-Cod* by *simp*

**lemma** *rep-preserves-seq*:

**assumes** *seq g f*

**shows** *Seq (rep g) (rep f)*

**using** *assms Arr-rep dom-char cod-char mkarr-inj-on-Ide Arr-implies-Ide-Dom  
Arr-implies-Ide-Cod*

by *auto*

**lemma** *rep-comp*:

**assumes** *seq g f*

**shows** *rep (g · f) = ||rep g · rep f||*

**proof** –

**have** *rep (g · f) = rep (mkarr (rep g · rep f))*

**using** *assms comp-char* by *metis*

**also have** *... = ||rep g · rep f||*

**using** *assms rep-preserves-seq rep-mkarr* by *simp*

**finally show** *?thesis* by *blast*

**qed**

The equivalence classes of canonical terms are canonical arrows.

**lemma** *can-mkarr-Can*:

**assumes** *Can t*

**shows** *can (mkarr t)*

**using** *assms Arr-in-mkarr Can-implies-Arr Ide-Diagonalize-Can arr-mkarr can-def* by *blast*

**lemma** *ide-implies-can*:

**assumes** *ide a*

**shows** *can a*

**using** *assms ide-char* by *blast*

**lemma** *Can-rep-can*:

**assumes** *can f*

**shows** *Can (rep f)*

**proof** –  
**have**  $Can \parallel rep f \parallel$   
**using** *assms can-def-alt Can-norm-iff-Ide-Diagonalize* **by** *auto*  
**moreover** **have**  $rep f = \parallel rep f \parallel$   
**using** *assms can-implies-arr norm-rep* **by** *simp*  
**ultimately show** *?thesis* **by** *simp*  
**qed**

Parallel canonical arrows are identical.

**lemma** *can-coherence*:  
**assumes** *par f g* **and** *can f* **and** *can g*  
**shows**  $f = g$   
**proof** –  
**have**  $\lfloor rep f \rfloor = \lfloor rep g \rfloor$   
**proof** –  
**have**  $\lfloor rep f \rfloor = \lfloor DOM f \rfloor$   
**using** *assms Ide-Diagonalize-Can Can-rep-can Diagonalize-in-Hom Ide-in-Hom* **by** *force*  
**also** **have**  $\dots = \lfloor DOM g \rfloor$   
**using** *assms dom-char equiv-iff-eq-norm*  
**by** (*metis DOM-in-dom mkarr-memb(1) rep-mkarr arr-dom-iff-arr*)  
**also** **have**  $\dots = \lfloor rep g \rfloor$   
**using** *assms Ide-Diagonalize-Can Can-rep-can Diagonalize-in-Hom Ide-in-Hom* **by** *force*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**hence**  $rep f = rep g$   
**using** *assms rep-in-arr norm-memb-eq-rep equiv-iff-eq-norm*  
**by** (*metis (no-types, lifting) arr-eqI*)  
**thus** *?thesis*  
**using** *assms arr-eqI [of f g] rep-in-arr [of f] rep-in-arr [of g]* **by** *metis*  
**qed**

Canonical arrows are invertible, and their inverses can be obtained syntactically.

**lemma** *inverse-arrows-can*:  
**assumes** *can f*  
**shows** *inverse-arrows f (mkarr (Inv (DOM f $\lfloor$ )  $\cdot$   $\lfloor rep f \rfloor$   $\cdot$  COD f $\rfloor$ ))*  
**proof**  
**let**  $?t = (Inv (DOM f \lfloor) \cdot \lfloor rep f \rfloor \cdot COD f \rfloor)$   
**have**  $1: rep f \in f \wedge Arr (rep f) \wedge Can (rep f) \wedge Ide \lfloor rep f \rfloor$   
**using** *assms can-def-alt rep-in-arr rep-in-arr(1) Can-rep-can* **by** *simp*  
**hence**  $2: \lfloor DOM f \rfloor = \lfloor COD f \rfloor$   
**using** *Diagonalize-in-Hom [of rep f] Ide-in-Hom* **by** *auto*  
**have**  $3: Can ?t$   
**using** *assms 1 2 Can-red Ide-implies-Can Diagonalize-in-Hom Inv-preserves-Can Arr-implies-Ide-Cod Arr-implies-Ide-Dom Diag-Diagonalize*  
**by** *simp*  
**have**  $4: DOM f = Cod ?t$   
**using** *assms can-def Can-red*  
**by** (*simp add: Arr-implies-Ide-Dom Inv-preserves-Can(3)*)  
**have**  $5: COD f = Dom ?t$

```

    using assms can-def Can-red Arr-rep Arr-implies-Ide-Cod by simp
have 6: antipar f (mkarr ?t)
    using assms 3 4 5 dom-char cod-char can-def cod-mkarr dom-mkarr Can-implies-Arr
    by simp
show ide (f · mkarr ?t)
proof –
  have 7: par (f · mkarr ?t) (dom (f · mkarr ?t))
    using assms 6 by auto
  moreover have can (f · mkarr ?t)
  proof –
    have 8: Comp (rep f) ?t ∈ (f · mkarr ?t)
      using assms 1 3 4 6 can-implies-arr Arr-in-mkarr COD-mkarr Comp-in-comp-ARR
      Can-implies-Arr arr-iff-ARR seq-char'
    by meson
  moreover have Can (rep f · ?t)
    using 1 3 7 8 mkarr-memb(1) by (metis Arr.simps(4) Can.simps(4))
  ultimately show ?thesis
    using can-mkarr-Can 7 mkarr-memb(2) by metis
qed
moreover have can (dom (f · mkarr ?t))
  using 7 ide-implies-can by force
ultimately have f · mkarr ?t = dom (f · mkarr ?t)
  using can-coherence by meson
thus ?thesis
  using 7 ide-dom by metis
qed
show ide (mkarr ?t · f)
proof –
  have 7: par (mkarr ?t · f) (cod (mkarr ?t · f))
    using assms 6 by auto
  moreover have can (mkarr ?t · f)
  proof –
    have 8: Comp ?t (rep f) ∈ mkarr ?t · f
      using assms 1 3 6 7 Arr-in-mkarr Comp-in-comp-ARR Can-implies-Arr arr-char
      comp-def
    by meson
  moreover have Can (?t · rep f)
    using 1 3 7 8 mkarr-memb(1) by (metis Arr.simps(4) Can.simps(4))
  ultimately show ?thesis
    using can-mkarr-Can 7 mkarr-memb(2) by metis
qed
moreover have can (cod (mkarr ?t · f))
  using 7 ide-implies-can by force
ultimately have mkarr ?t · f = cod (mkarr ?t · f)
  using can-coherence by meson
thus ?thesis
  using 7 can-implies-arr ide-cod by metis
qed
qed

```



```

lemma inv-mkarr [simp]:
assumes Can t
shows inv (mkarr t) = mkarr (Inv t)
proof –
  have t: Can t ∧ Arr t ∧ Can (Inv t) ∧ Arr (Inv t) ∧ Ide (Dom t) ∧ Ide (Cod t)
    using assms Can-implies-Arr Arr-implies-Ide-Dom Arr-implies-Ide-Cod
      Inv-preserves-Can
    by simp
  have inverse-arrows (mkarr t) (mkarr (Inv t))
proof
  show ide (mkarr t · mkarr (Inv t))
proof –
    have mkarr (Cod t) = mkarr (Comp t (Inv t))
    using t Inv-in-Hom Ide-in-Hom Diagonalize-Inv Diag-Diagonalize Diagonalize-preserves-Can
      by (intro mkarr-eqI, auto)
    also have ... = mkarr t · mkarr (Inv t)
      using t comp-mkarr Inv-in-Hom by simp
    finally have mkarr (Cod t) = mkarr t · mkarr (Inv t)
      by blast
    thus ?thesis using t ide-mkarr-Ide [of Cod t] by simp
  qed
  show ide (mkarr (Inv t) · mkarr t)
proof –
    have mkarr (Dom t) = mkarr (Inv t · t)
    using t Inv-in-Hom Ide-in-Hom Diagonalize-Inv Diag-Diagonalize Diagonalize-preserves-Can
      by (intro mkarr-eqI, auto)
    also have ... = mkarr (Inv t) · mkarr t
      using t comp-mkarr Inv-in-Hom by simp
    finally have mkarr (Dom t) = mkarr (Inv t) · mkarr t
      by blast
    thus ?thesis using t ide-mkarr-Ide [of Dom t] by simp
  qed
qed
thus ?thesis using inverse-unique by auto
qed

```

```

lemma iso-can:
assumes can f
shows iso f
  using assms inverse-arrows-can by auto

```

The following function produces the unique canonical arrow between two given objects, if such an arrow exists.

```

definition mkcan
where mkcan a b = mkarr (Inv (COD b↓) · (DOM a↓))

```

```

lemma can-mkcan:
assumes ide a and ide b and  $[DOM a] = [COD b]$ 

```

**shows**  $\text{can} (\text{mkcan } a \ b)$  **and**  $\llbracket \text{mkcan } a \ b : a \rightarrow b \rrbracket$   
**proof** –  
**show**  $\text{can} (\text{mkcan } a \ b)$   
**using** *assms mkcan-def Arr-rep Arr-implies-Ide-Dom Arr-implies-Ide-Cod Can-red Inv-preserves-Can can-mkarr-Can*  
**by** *simp*  
**show**  $\llbracket \text{mkcan } a \ b : a \rightarrow b \rrbracket$   
**using** *assms mkcan-def Arr-rep Arr-implies-Ide-Dom Arr-implies-Ide-Cod Can-red Inv-in-Hom dom-char [of a] cod-char [of b] mkarr-rep mkarr-in-hom can-implies-arr*  
**by** *auto*  
**qed**

**lemma** *dom-mkcan*:  
**assumes** *ide a and ide b and*  $\llbracket \text{DOM } a \rrbracket = \llbracket \text{COD } b \rrbracket$   
**shows**  $\text{dom} (\text{mkcan } a \ b) = a$   
**using** *assms can-mkcan by blast*

**lemma** *cod-mkcan*:  
**assumes** *ide a and ide b and*  $\llbracket \text{DOM } a \rrbracket = \llbracket \text{COD } b \rrbracket$   
**shows**  $\text{cod} (\text{mkcan } a \ b) = b$   
**using** *assms can-mkcan by blast*

**lemma** *can-coherence'*:  
**assumes** *can f*  
**shows**  $\text{mkcan} (\text{dom } f) (\text{cod } f) = f$   
**proof** –  
**have**  $\text{Ide} \llbracket \text{rep } f \rrbracket$   
**using** *assms Ide-Diagonalize-Can Can-rep-can by simp*  
**hence**  $\text{Dom} \llbracket \text{rep } f \rrbracket = \text{Cod} \llbracket \text{rep } f \rrbracket$   
**using** *Ide-in-Hom by simp*  
**hence**  $\llbracket \text{DOM } f \rrbracket = \llbracket \text{COD } f \rrbracket$   
**using** *assms can-implies-arr Arr-rep Diagonalize-in-Hom by simp*  
**moreover have**  $\text{DOM } f = \text{DOM} (\text{dom } f)$   
**using** *assms can-implies-arr dom-char rep-mkarr Arr-implies-Ide-Dom Ide-implies-Arr Par-Arr-norm [of DOM f] Ide-in-Hom*  
**by** *auto*  
**moreover have**  $\text{COD } f = \text{COD} (\text{cod } f)$   
**using** *assms can-implies-arr cod-char rep-mkarr Arr-implies-Ide-Cod Ide-implies-Arr Par-Arr-norm [of COD f] Ide-in-Hom*  
**by** *auto*  
**ultimately have**  $\text{can} (\text{mkcan} (\text{dom } f) (\text{cod } f)) \wedge \text{par } f (\text{mkcan} (\text{dom } f) (\text{cod } f))$   
**using** *assms can-implies-arr can-mkcan dom-mkcan cod-mkcan by simp*  
**thus ?thesis using** *assms can-coherence by blast*  
**qed**

**lemma** *Ide-Diagonalize-rep-ide*:  
**assumes** *ide a*  
**shows**  $\text{Ide} \llbracket \text{rep } a \rrbracket$   
**using** *assms ide-implies-can can-def-alt rep-in-arr by simp*

```

lemma Diagonalize-DOM:
assumes arr f
shows  $\llbracket DOM f \rrbracket = Dom \llbracket rep f \rrbracket$ 
  using assms Diag-Diagonalize by simp

lemma Diagonalize-COD:
assumes arr f
shows  $\llbracket COD f \rrbracket = Cod \llbracket rep f \rrbracket$ 
  using assms Diag-Diagonalize by simp

lemma Diagonalize-rep-preserves-seq:
assumes seq g f
shows  $Seq \llbracket rep g \rrbracket \llbracket rep f \rrbracket$ 
  using assms Diagonalize-DOM Diagonalize-COD Diag-implies-Arr Diag-Diagonalize(1)
  rep-preserves-seq
  by force

lemma Dom-Diagonalize-rep:
assumes arr f
shows  $Dom \llbracket rep f \rrbracket = \llbracket rep (dom f) \rrbracket$ 
  using assms Diagonalize-rep-preserves-seq [of f dom f] Ide-Diagonalize-rep-ide Ide-in-Hom
  by simp

lemma Cod-Diagonalize-rep:
assumes arr f
shows  $Cod \llbracket rep f \rrbracket = \llbracket rep (cod f) \rrbracket$ 
  using assms Diagonalize-rep-preserves-seq [of cod f f] Ide-Diagonalize-rep-ide Ide-in-Hom
  by simp

lemma mkarr-Diagonalize-rep:
assumes arr f and Diag (DOM f) and Diag (COD f)
shows  $mkarr \llbracket rep f \rrbracket = f$ 
proof –
  have  $mkarr (rep f) = mkarr \llbracket rep f \rrbracket$ 
    using assms rep-in-Hom Diagonalize-in-Hom Diag-Diagonalize Diagonalize-Diag
    by (intro mkarr-eqI, simp-all)
  thus ?thesis using assms mkarr-rep by auto
qed

```

We define tensor product of arrows via the constructor  $(\otimes)$  on terms.

```

definition tensorFMC (infixr  $\otimes$  53)
  where  $f \otimes g \equiv (if\ arr\ f \wedge arr\ g\ then\ mkarr\ (rep\ f \otimes rep\ g)\ else\ null)$ 

```

```

lemma arr-tensor [simp]:
assumes arr f and arr g
shows  $arr (f \otimes g)$ 
  using assms tensorFMC-def arr-mkarr by simp

```

**lemma** *rep-tensor*:  
**assumes** *arr f* **and** *arr g*  
**shows**  $\text{rep } (f \otimes g) = \|\text{rep } f \otimes \text{rep } g\|$   
**using** *assms tensor<sub>FM C</sub>-def rep-mkarr* **by** *simp*

**lemma** *Par-memb-rep*:  
**assumes** *arr f* **and**  $t \in f$   
**shows**  $\text{Par } t (\text{rep } f)$   
**using** *assms mkarr-memb* **apply** *simp*  
**using** *rep-in-Hom Dom-memb Cod-memb* **by** *metis*

**lemma** *Tensor-in-tensor* [*intro*]:  
**assumes** *arr f* **and** *arr g* **and**  $t \in f$  **and**  $u \in g$   
**shows**  $t \otimes u \in f \otimes g$   
**proof** –  
  **have** *equiv*  $(t \otimes u) (\text{rep } f \otimes \text{rep } g)$   
  **proof** –  
    **have** *1*:  $\text{Par } (t \otimes u) (\text{rep } f \otimes \text{rep } g)$   
    **proof** –  
      **have**  $\text{Par } t (\text{rep } f) \wedge \text{Par } u (\text{rep } g)$  **using** *assms Par-memb-rep* **by** *blast*  
      **thus** *?thesis* **by** *simp*  
    **qed**  
  **moreover** **have**  $\lfloor t \otimes u \rfloor = \lfloor \text{rep } f \otimes \text{rep } g \rfloor$   
  **using** *assms 1 equiv-iff-eq-norm rep-mkarr norm-norm mkarr-memb*(2)  
  **by** (*metis Arr.simps*(3) *Diagonalize.simps*(3))  
  **ultimately show** *?thesis* **by** *simp*  
**qed**  
**thus** *?thesis*  
  **using** *assms tensor<sub>FM C</sub>-def mkarr-def* **by** *simp*  
**qed**

**lemma** *DOM-tensor* [*simp*]:  
**assumes** *arr f* **and** *arr g*  
**shows**  $\text{DOM } (f \otimes g) = \text{DOM } f \otimes \text{DOM } g$   
**by** (*metis (no-types, lifting) DOM-mkarr Dom.simps*(3) *mkarr-extensionality arr-char arr-tensor assms*(1) *assms*(2) *tensor<sub>FM C</sub>-def*)

**lemma** *COD-tensor* [*simp*]:  
**assumes** *arr f* **and** *arr g*  
**shows**  $\text{COD } (f \otimes g) = \text{COD } f \otimes \text{COD } g$   
**by** (*metis (no-types, lifting) COD-mkarr Cod.simps*(3) *mkarr-extensionality arr-char arr-tensor assms*(1) *assms*(2) *tensor<sub>FM C</sub>-def*)

**lemma** *tensor-in-hom* [*simp*]:  
**assumes**  $\langle f : a \rightarrow b \rangle$  **and**  $\langle g : c \rightarrow d \rangle$   
**shows**  $\langle f \otimes g : a \otimes c \rightarrow b \otimes d \rangle$   
**proof** –  
  **have**  $f : \text{arr } f \wedge \text{dom } f = a \wedge \text{cod } f = b$  **using** *assms*(1) **by** *auto*  
  **have**  $g : \text{arr } g \wedge \text{dom } g = c \wedge \text{cod } g = d$  **using** *assms*(2) **by** *auto*

```

have  $\text{dom } (f \otimes g) = \text{dom } f \otimes \text{dom } g$ 
  using  $f\ g\ \text{arr-tensor dom-char Tensor-in-tensor [of dom } f\ \text{dom } g\ \text{DOM } f\ \text{DOM } g]$ 
     $\text{DOM-in-dom mkarr-memb}(?)\ \text{DOM-tensor arr-dom-iff-arr}$ 
  by metis
moreover have  $\text{cod } (f \otimes g) = \text{cod } f \otimes \text{cod } g$ 
  using  $f\ g\ \text{arr-tensor cod-char Tensor-in-tensor [of cod } f\ \text{cod } g\ \text{COD } f\ \text{COD } g]$ 
     $\text{COD-in-cod mkarr-memb}(?)\ \text{COD-tensor arr-cod-iff-arr}$ 
  by metis
ultimately show ?thesis using assms arr-tensor by blast
qed

lemma dom-tensor [simp]:
assumes  $\text{arr } f$  and  $\text{arr } g$ 
shows  $\text{dom } (f \otimes g) = \text{dom } f \otimes \text{dom } g$ 
  using assms tensor-in-hom [of f] by blast

lemma cod-tensor [simp]:
assumes  $\text{arr } f$  and  $\text{arr } g$ 
shows  $\text{cod } (f \otimes g) = \text{cod } f \otimes \text{cod } g$ 
  using assms tensor-in-hom [of f] by blast

lemma tensor-mkarr [simp]:
assumes  $\text{Arr } t$  and  $\text{Arr } u$ 
shows  $\text{mkarr } t \otimes \text{mkarr } u = \text{mkarr } (t \otimes u)$ 
  using assms by (meson Tensor-in-tensor arr-char Arr-in-mkarr arr-mkarr arr-tensor)

lemma tensor-preserves-ide:
assumes ide a and ide b
shows ide  $(a \otimes b)$ 
proof –
  have can  $(a \otimes b)$ 
    using assms tensorFM C-def Can-rep-can ide-implies-can can-mkarr-Can by simp
  moreover have  $\text{dom } (a \otimes b) = \text{cod } (a \otimes b)$ 
    using assms tensor-in-hom by simp
  ultimately show ?thesis using ide-char by metis
qed

lemma tensor-preserves-can:
assumes can f and can g
shows can  $(f \otimes g)$ 
  using assms can-implies-arr Can-rep-can tensorFM C-def can-mkarr-Can by simp

lemma comp-preserves-can:
assumes can f and can g and  $\text{dom } f = \text{cod } g$ 
shows can  $(f \cdot g)$ 
proof –
  have  $1: \text{ARR } f \wedge \text{ARR } g \wedge \text{DOM } f = \text{COD } g$ 
    using assms can-implies-arr arr-iff-ARR Arr-implies-Ide-Cod Arr-implies-Ide-Dom
    mkarr-inj-on-Ide cod-char dom-char

```

**by** *simp*  
**hence**  $Can (rep f \cdot rep g)$   
**using** *assms can-implies-arr Can-rep-can* **by** *force*  
**thus** *?thesis*  
**using** *assms 1 can-implies-arr comp-char can-mkarr-Can seq-char'* **by** *simp*  
**qed**

The remaining structure required of a monoidal category is also defined syntactically.

**definition**  $unity_{FMC} :: 'c \text{ arr}$  ( $\mathcal{I}$ )  
**where**  $\mathcal{I} = mkarr \mathcal{I}$

**definition**  $lunit_{FMC} :: 'c \text{ arr} \Rightarrow 'c \text{ arr}$  ( $l[-]$ )  
**where**  $l[a] = mkarr l[rep a]$

**definition**  $runit_{FMC} :: 'c \text{ arr} \Rightarrow 'c \text{ arr}$  ( $r[-]$ )  
**where**  $r[a] = mkarr r[rep a]$

**definition**  $assoc_{FMC} :: 'c \text{ arr} \Rightarrow 'c \text{ arr} \Rightarrow 'c \text{ arr} \Rightarrow 'c \text{ arr}$  ( $a[-, -, -]$ )  
**where**  $a[a, b, c] = mkarr a[rep a, rep b, rep c]$

**lemma** *can-lunit*:  
**assumes** *ide a*  
**shows** *can l[a]*  
**using** *assms lunit<sub>FMC</sub>-def can-mkarr-Can*  
**by** (*simp add: Can-rep-can ide-implies-can*)

**lemma** *lunit-in-hom*:  
**assumes** *ide a*  
**shows**  $\langle l[a] : \mathcal{I} \otimes a \rightarrow a \rangle$   
**proof** –  
**have**  $dom l[a] = \mathcal{I} \otimes a$   
**using** *assms lunit<sub>FMC</sub>-def unity<sub>FMC</sub>-def Ide-implies-Arr dom-mkarr dom-char ten-*  
*sor-mkarr*  
*Arr-rep*  
**by** (*metis Arr.simps(2) Arr.simps(5) Arr-implies-Ide-Dom Dom.simps(5)*  
*ideD(1) ideD(2)*)  
**moreover** **have**  $cod l[a] = a$   
**using** *assms lunit<sub>FMC</sub>-def rep-in-arr(1) cod-mkarr cod-char ideD(3)* **by** *auto*  
**ultimately show** *?thesis*  
**using** *assms arr-cod-iff-arr* **by** (*intro in-homI, fastforce*)  
**qed**

**lemma** *arr-lunit [simp]*:  
**assumes** *ide a*  
**shows** *arr l[a]*  
**using** *assms can-lunit can-implies-arr* **by** *simp*

**lemma** *dom-lunit [simp]*:  
**assumes** *ide a*

**shows**  $\text{dom } l[a] = \mathcal{I} \otimes a$   
**using** *assms lunit-in-hom* **by** *auto*

**lemma** *cod-lunit* [*simp*]:  
**assumes** *ide a*  
**shows**  $\text{cod } l[a] = a$   
**using** *assms lunit-in-hom* **by** *auto*

**lemma** *can-runit*:  
**assumes** *ide a*  
**shows** *can*  $r[a]$   
**using** *assms runit<sub>FM C</sub>-def can-mkarr-Can*  
**by** (*simp add: Can-rep-can ide-implies-can*)

**lemma** *runit-in-hom* [*simp*]:  
**assumes** *ide a*  
**shows**  $\langle r[a] : a \otimes \mathcal{I} \rightarrow a \rangle$   
**proof** –  
**have**  $\text{dom } r[a] = a \otimes \mathcal{I}$   
**using** *assms Arr-rep Arr.simps(2) Arr.simps(7) Arr-implies-Ide-Dom Dom.simps(7)*  
*Ide-implies-Arr dom-mkarr dom-char ideD(1) ideD(2) runit<sub>FM C</sub>-def tensor-mkarr*  
*unity<sub>FM C</sub>-def*  
**by** *metis*  
**moreover** **have**  $\text{cod } r[a] = a$   
**using** *assms runit<sub>FM C</sub>-def rep-in-arr(1) cod-mkarr cod-char ideD(3)* **by** *auto*  
**ultimately show** *?thesis*  
**using** *assms arr-cod-iff-arr* **by** (*intro in-homI, fastforce*)  
**qed**

**lemma** *arr-runit* [*simp*]:  
**assumes** *ide a*  
**shows** *arr*  $r[a]$   
**using** *assms can-runit can-implies-arr* **by** *simp*

**lemma** *dom-runit* [*simp*]:  
**assumes** *ide a*  
**shows**  $\text{dom } r[a] = a \otimes \mathcal{I}$   
**using** *assms runit-in-hom* **by** *blast*

**lemma** *cod-runit* [*simp*]:  
**assumes** *ide a*  
**shows**  $\text{cod } r[a] = a$   
**using** *assms runit-in-hom* **by** *blast*

**lemma** *can-assoc*:  
**assumes** *ide a* **and** *ide b* **and** *ide c*  
**shows** *can*  $a[a, b, c]$   
**using** *assms assoc<sub>FM C</sub>-def can-mkarr-Can*  
**by** (*simp add: Can-rep-can ide-implies-can*)

**lemma** *assoc-in-hom*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\llbracket a[a, b, c] : (a \otimes b) \otimes c \rightarrow a \otimes b \otimes c \rrbracket$   
**proof** –  
    **have**  $\text{dom } a[a, b, c] = (a \otimes b) \otimes c$   
    **proof** –  
        **have**  $\text{dom } a[a, b, c] = \text{mkarr } (\text{Dom } \mathbf{a}[\text{rep } a, \text{rep } b, \text{rep } c])$   
            **using** *assms assoc<sub>FM C</sub>-def rep-in-arr(1) by simp*  
        **also have**  $\dots = \text{mkarr } ((\text{DOM } a \otimes \text{DOM } b) \otimes \text{DOM } c)$   
            **by** *simp*  
        **also have**  $\dots = (a \otimes b) \otimes c$   
            **by** (*metis mkarr-extensionality arr-tensor assms dom-char*  
                *ideD(1) ideD(2) not-arr-null null-char tensor-mkarr*)  
    **finally show** *?thesis by blast*  
**qed**  
**moreover have**  $\text{cod } a[a, b, c] = a \otimes b \otimes c$   
**proof** –  
    **have**  $\text{cod } a[a, b, c] = \text{mkarr } (\text{Cod } \mathbf{a}[\text{rep } a, \text{rep } b, \text{rep } c])$   
        **using** *assms assoc<sub>FM C</sub>-def rep-in-arr(1) by simp*  
    **also have**  $\dots = \text{mkarr } (\text{COD } a \otimes \text{COD } b \otimes \text{COD } c)$   
        **by** *simp*  
    **also have**  $\dots = a \otimes b \otimes c$   
        **by** (*metis mkarr-extensionality arr-tensor assms(1) assms(2) assms(3) cod-char*  
            *ideD(1) ideD(3) not-arr-null null-char tensor-mkarr*)  
    **finally show** *?thesis by blast*  
**qed**  
**moreover have**  $\text{arr } a[a, b, c]$   
    **using** *assms assoc<sub>FM C</sub>-def rep-in-arr(1) arr-mkarr by simp*  
**ultimately show** *?thesis by auto*  
**qed**

**lemma** *arr-assoc [simp]*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\text{arr } a[a, b, c]$   
    **using** *assms can-assoc can-implies-arr by simp*

**lemma** *dom-assoc [simp]*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\text{dom } a[a, b, c] = (a \otimes b) \otimes c$   
    **using** *assms assoc-in-hom by blast*

**lemma** *cod-assoc [simp]*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\text{cod } a[a, b, c] = a \otimes b \otimes c$   
    **using** *assms assoc-in-hom by blast*

**lemma** *ide-unity [simp]*:  
**shows** *ide I*



**using** *unity<sub>PMC-def</sub> Arr.simps(2) Dom.simps(2) arr-mkarr dom-mkarr ide-dom*  
**by** *metis*

**lemma** *Unity-in-unity [simp]:*  
**shows**  $\mathcal{I} \in \mathcal{I}$   
**using** *unity<sub>PMC-def</sub> Arr-in-mkarr* **by** *simp*

**lemma** *rep-unity [simp]:*  
**shows**  $\text{rep } \mathcal{I} = \|\mathcal{I}\|$   
**using** *unity<sub>PMC-def</sub> rep-mkarr* **by** *simp*

**lemma** *Lunit-in-lunit [intro]:*  
**assumes** *arr f* **and**  $t \in f$   
**shows**  $\mathbb{l}[t] \in \mathbb{l}[f]$   
**proof** –  
**have**  $\text{Arr } t \wedge \text{Arr } (\text{rep } f) \wedge \text{Dom } t = \text{DOM } f \wedge \text{Cod } t = \text{COD } f \wedge [t] = [\text{rep } f]$   
**using** *assms*  
**by** (*metis mkarr-memb(1) mkarr-memb(2) rep-mkarr rep-in-arr(1) equiv-iff-eq-norm norm-rep*)  
**thus** *?thesis*  
**by** (*simp add: mkarr-def lunit<sub>PMC-def</sub>*)  
**qed**

**lemma** *Runit-in-runit [intro]:*  
**assumes** *arr f* **and**  $t \in f$   
**shows**  $\mathbb{r}[t] \in \mathbb{r}[f]$   
**proof** –  
**have**  $\text{Arr } t \wedge \text{Arr } (\text{rep } f) \wedge \text{Dom } t = \text{DOM } f \wedge \text{Cod } t = \text{COD } f \wedge [t] = [\text{rep } f]$   
**using** *assms*  
**by** (*metis mkarr-memb(1) mkarr-memb(2) rep-mkarr rep-in-arr(1) equiv-iff-eq-norm norm-rep*)  
**thus** *?thesis*  
**by** (*simp add: mkarr-def runit<sub>PMC-def</sub>*)  
**qed**

**lemma** *Assoc-in-assoc [intro]:*  
**assumes** *arr f* **and** *arr g* **and** *arr h*  
**and**  $t \in f$  **and**  $u \in g$  **and**  $v \in h$   
**shows**  $\mathbb{a}[t, u, v] \in \mathbb{a}[f, g, h]$   
**proof** –  
**have**  $\text{Arr } t \wedge \text{Arr } (\text{rep } f) \wedge \text{Dom } t = \text{DOM } f \wedge \text{Cod } t = \text{COD } f \wedge [t] = [\text{rep } f]$   
**using** *assms*  
**by** (*metis mkarr-memb(1) rep-mkarr rep-in-arr(1) equiv-iff-eq-norm mkarr-memb(2) norm-rep*)  
**moreover** **have**  $\text{Arr } u \wedge \text{Arr } (\text{rep } g) \wedge \text{Dom } u = \text{DOM } g \wedge \text{Cod } u = \text{COD } g \wedge [u] = [\text{rep } g]$   
**using** *assms*  
**by** (*metis mkarr-memb(1) rep-mkarr rep-in-arr(1) equiv-iff-eq-norm mkarr-memb(2) norm-rep*)

**moreover have**  $Arr\ v \wedge Arr\ (rep\ h) \wedge Dom\ v = DOM\ h \wedge Cod\ v = COD\ h \wedge$   
 $\quad [v] = [rep\ h]$   
**using** *assms*  
**by** (*metis mkarr-memb(1) rep-mkarr rep-in-arr(1) equiv-iff-eq-norm mkarr-memb(2)*  
*norm-rep*)  
**ultimately show** *?thesis*  
**using** *assoc\_FMC-def mkarr-def* **by** *simp*  
**qed**

At last, we can show that we've constructed a monoidal category.

**interpretation** *EMC: elementary-monoidal-category*  
 $comp\ tensor_{FMC}\ unity_{FMC}\ lunit_{FMC}\ runit_{FMC}\ assoc_{FMC}$

**proof**

**show** *ide I* **using** *ide-unity* **by** *auto*  
**show**  $\bigwedge a. ide\ a \implies \llbracket l[a] : \mathcal{I} \otimes a \rightarrow a \rrbracket$  **by** *auto*  
**show**  $\bigwedge a. ide\ a \implies \llbracket r[a] : a \otimes \mathcal{I} \rightarrow a \rrbracket$  **by** *auto*  
**show**  $\bigwedge a. ide\ a \implies iso\ l[a]$  **using** *can-lunit iso-can* **by** *auto*  
**show**  $\bigwedge a. ide\ a \implies iso\ r[a]$  **using** *can-runit iso-can* **by** *auto*  
**show**  $\bigwedge a\ b\ c. \llbracket ide\ a; ide\ b; ide\ c \rrbracket \implies \llbracket a[a, b, c] : (a \otimes b) \otimes c \rightarrow a \otimes b \otimes c \rrbracket$  **by** *auto*  
**show**  $\bigwedge a\ b\ c. \llbracket ide\ a; ide\ b; ide\ c \rrbracket \implies iso\ a[a, b, c]$  **using** *can-assoc iso-can* **by** *auto*  
**show**  $\bigwedge a\ b. \llbracket ide\ a; ide\ b \rrbracket \implies ide\ (a \otimes b)$  **using** *tensor-preserves-ide* **by** *auto*  
**fix** *f a b g c d*  
**show**  $\llbracket \llbracket f : a \rightarrow b \rrbracket; \llbracket g : c \rightarrow d \rrbracket \rrbracket \implies \llbracket f \otimes g : a \otimes c \rightarrow b \otimes d \rrbracket$   
**using** *tensor-in-hom* **by** *auto*  
**next**

Naturality of left unitor.

**fix** *f*  
**assume** *f: arr f*  
**show**  $l[cod\ f] \cdot (\mathcal{I} \otimes f) = f \cdot l[dom\ f]$   
**proof** (*intro arr-eqI*)  
**show** *par* ( $l[cod\ f] \cdot (\mathcal{I} \otimes f)$ ) ( $f \cdot l[dom\ f]$ )  
**using** *f* **by** *simp*  
**show**  $l[COD\ f] \cdot (\mathcal{I} \otimes rep\ f) \in l[cod\ f] \cdot (\mathcal{I} \otimes f)$   
**using** *f* **by** *fastforce*  
**show**  $rep\ f \cdot l[DOM\ f] \in f \cdot l[dom\ f]$   
**using** *f* **by** *fastforce*  
**show**  $l[COD\ f] \cdot (\mathcal{I} \otimes rep\ f) = [rep\ f \cdot l[DOM\ f]]$   
**using** *f* **by** (*simp add: Diag-Diagonalize(1) Diagonalize-DOM Diagonalize-COD*)  
**qed**

Naturality of right unitor.

**show**  $r[cod\ f] \cdot (f \otimes \mathcal{I}) = f \cdot r[dom\ f]$   
**proof** (*intro arr-eqI*)  
**show** *par* ( $r[cod\ f] \cdot (f \otimes \mathcal{I})$ ) ( $f \cdot r[dom\ f]$ )  
**using** *f* **by** *simp*  
**show**  $r[COD\ f] \cdot (rep\ f \otimes \mathcal{I}) \in r[cod\ f] \cdot (f \otimes \mathcal{I})$   
**using** *f* **by** *fastforce*  
**show**  $rep\ f \cdot r[DOM\ f] \in f \cdot r[dom\ f]$

```

    using f by fastforce
  show [r[COD f] · (rep f ⊗ I)] = [rep f · r[DOM f]]
    using f by (simp add: Diag-Diagonalize(1) Diagonalize-DOM Diagonalize-COD)
qed
next

```

Naturality of associator.

```

fix f0 :: 'c arr and f1 f2
assume f0: arr f0 and f1: arr f1 and f2: arr f2
show a[cod f0, cod f1, cod f2] · ((f0 ⊗ f1) ⊗ f2)
  = (f0 ⊗ f1 ⊗ f2) · a[dom f0, dom f1, dom f2]
proof (intro arr-eqI)
  show 1: par (a[cod f0, cod f1, cod f2] · ((f0 ⊗ f1) ⊗ f2))
    ((f0 ⊗ f1 ⊗ f2) · a[dom f0, dom f1, dom f2])
    using f0 f1 f2 by force
  show a[rep (cod f0), rep (cod f1), rep (cod f2)] · ((rep f0 ⊗ rep f1) ⊗ rep f2)
    ∈ a[cod f0, cod f1, cod f2] · ((f0 ⊗ f1) ⊗ f2)
    using f0 f1 f2 by fastforce
  show (rep f0 ⊗ rep f1 ⊗ rep f2) · a[rep (dom f0), rep (dom f1), rep (dom f2)]
    ∈ (f0 ⊗ f1 ⊗ f2) · a[dom f0, dom f1, dom f2]
    using f0 f1 f2 by fastforce
  show [a[rep (cod f0), rep (cod f1), rep (cod f2)] · ((rep f0 ⊗ rep f1) ⊗ rep f2)]
    = [(rep f0 ⊗ rep f1 ⊗ rep f2) · a[rep (dom f0), rep (dom f1), rep (dom f2)]]
  proof -
    have [a[rep (cod f0), rep (cod f1), rep (cod f2)] · ((rep f0 ⊗ rep f1) ⊗ rep f2)]
      = [rep f0] [⊗] [rep f1] [⊗] [rep f2]
    proof -
      have b0: [rep (cod f0)] = Cod [rep f0]
        using f0 Cod-Diagonalize-rep by simp
      have b1: [rep (cod f1)] = Cod [rep f1]
        using f1 Cod-Diagonalize-rep by simp
      have b2: [rep (cod f2)] = Cod [rep f2]
        using f2 Cod-Diagonalize-rep by simp
      have [a[rep (cod f0), rep (cod f1), rep (cod f2)] · ((rep f0 ⊗ rep f1) ⊗ rep f2)]
        = ([rep (cod f0)] [⊗] [rep (cod f1)] [⊗] [rep (cod f2)]) [·]
          ([rep f0] [⊗] [rep f1] [⊗] [rep f2])
        using f0 f1 f2 using Diag-Diagonalize(1) TensorDiag-assoc by auto
      also have ... = [rep (cod f0)] [·] [rep f0] [⊗]
        [rep (cod f1)] [·] [rep f1] [⊗] [rep (cod f2)] [·] [rep f2]
    proof -
      have Seq [rep (cod f0)] [rep f0] ∧ Seq [rep (cod f1)] [rep f1] ∧
        Seq [rep (cod f2)] [rep f2]
    using f0 f1 f2 rep-in-Hom Diagonalize-in-Hom Dom-Diagonalize-rep Cod-Diagonalize-rep
    by auto
    thus ?thesis
    using f0 f1 f2 b0 b1 b2 TensorDiag-in-Hom TensorDiag-preserves-Diag
      Diag-Diagonalize Arr-implies-Ide-Dom Arr-implies-Ide-Cod
      CompDiag-TensorDiag
    by simp

```

**qed**  
**also have** ... =  $\llbracket \text{rep } f0 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f1 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f2 \rrbracket$   
**proof** –  
**have**  $\llbracket \text{rep } (\text{cod } f0) \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } f0 \rrbracket = \llbracket \text{rep } f0 \rrbracket$   
**using**  $f0\ b0\ \text{CompDiag-Cod-Diag}$  [of  $\llbracket \text{rep } f0 \rrbracket$ ] *Diag-Diagonalize*  
**by simp**  
**moreover have**  $\llbracket \text{rep } (\text{cod } f1) \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } f1 \rrbracket = \llbracket \text{rep } f1 \rrbracket$   
**using**  $f1\ b1\ \text{CompDiag-Cod-Diag}$  [of  $\llbracket \text{rep } f1 \rrbracket$ ] *Diag-Diagonalize*  
**by simp**  
**moreover have**  $\llbracket \text{rep } (\text{cod } f2) \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } f2 \rrbracket = \llbracket \text{rep } f2 \rrbracket$   
**using**  $f2\ b2\ \text{CompDiag-Cod-Diag}$  [of  $\llbracket \text{rep } f2 \rrbracket$ ] *Diag-Diagonalize*  
**by simp**  
**ultimately show** *?thesis* **by argo**  
**qed**  
**finally show** *?thesis* **by blast**  
**qed**  
**also have** ... =  $\llbracket (\text{rep } f0 \otimes \text{rep } f1 \otimes \text{rep } f2) \cdot \mathbf{a}[\text{rep } (\text{dom } f0), \text{rep } (\text{dom } f1), \text{rep } (\text{dom } f2)] \rrbracket$   
**proof** –  
**have**  $a0: \llbracket \text{rep } (\text{dom } f0) \rrbracket = \text{Dom } \llbracket \text{rep } f0 \rrbracket$   
**using**  $f0\ \text{Dom-Diagonalize-rep}$  **by simp**  
**have**  $a1: \llbracket \text{rep } (\text{dom } f1) \rrbracket = \text{Dom } \llbracket \text{rep } f1 \rrbracket$   
**using**  $f1\ \text{Dom-Diagonalize-rep}$  **by simp**  
**have**  $a2: \llbracket \text{rep } (\text{dom } f2) \rrbracket = \text{Dom } \llbracket \text{rep } f2 \rrbracket$   
**using**  $f2\ \text{Dom-Diagonalize-rep}$  **by simp**  
**have**  $\llbracket (\text{rep } f0 \otimes \text{rep } f1 \otimes \text{rep } f2) \cdot \mathbf{a}[\text{rep } (\text{dom } f0), \text{rep } (\text{dom } f1), \text{rep } (\text{dom } f2)] \rrbracket$   
=  $(\llbracket \text{rep } f0 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f1 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f2 \rrbracket) \llbracket \cdot \rrbracket$   
 $(\llbracket \text{rep } (\text{dom } f0) \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } (\text{dom } f1) \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } (\text{dom } f2) \rrbracket)$   
**using**  $f0\ f1\ f2\ \text{using}$  *Diag-Diagonalize(1)* *TensorDiag-assoc* **by auto**  
**also have** ... =  $\llbracket \text{rep } f0 \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } (\text{dom } f0) \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f1 \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } (\text{dom } f1) \rrbracket \llbracket \otimes \rrbracket$   
 $\llbracket \text{rep } f2 \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } (\text{dom } f2) \rrbracket$   
**proof** –  
**have**  $\text{Seq } \llbracket \text{rep } f0 \rrbracket \llbracket \text{rep } (\text{dom } f0) \rrbracket \wedge \text{Seq } \llbracket \text{rep } f1 \rrbracket \llbracket \text{rep } (\text{dom } f1) \rrbracket \wedge$   
 $\text{Seq } \llbracket \text{rep } f2 \rrbracket \llbracket \text{rep } (\text{dom } f2) \rrbracket$   
**using**  $f0\ f1\ f2\ \text{rep-in-Hom}$  *Diagonalize-in-Hom* *Dom-Diagonalize-rep* *Cod-Diagonalize-rep*  
**by auto**  
**thus** *?thesis*  
**using**  $f0\ f1\ f2\ a0\ a1\ a2\ \text{TensorDiag-in-Hom}$  *TensorDiag-preserves-Diag*  
*Diag-Diagonalize* *Arr-implies-Ide-Dom* *Arr-implies-Ide-Cod*  
*CompDiag-TensorDiag*  
**by force**  
**qed**  
**also have** ... =  $\llbracket \text{rep } f0 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f1 \rrbracket \llbracket \otimes \rrbracket \llbracket \text{rep } f2 \rrbracket$   
**proof** –  
**have**  $\llbracket \text{rep } f0 \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } (\text{dom } f0) \rrbracket = \llbracket \text{rep } f0 \rrbracket$   
**using**  $f0\ a0\ \text{CompDiag-Diag-Dom}$  [of *Diagonalize (rep f0)*] *Diag-Diagonalize*  
**by simp**  
**moreover have**  $\llbracket \text{rep } f1 \rrbracket \llbracket \cdot \rrbracket \llbracket \text{rep } (\text{dom } f1) \rrbracket = \llbracket \text{rep } f1 \rrbracket$   
**using**  $f1\ a1\ \text{CompDiag-Diag-Dom}$  [of *Diagonalize (rep f1)*] *Diag-Diagonalize*

```

    by simp
    moreover have [rep f2] [·] [rep (dom f2)] = [rep f2]
      using f2 a2 CompDiag-Diag-Dom [of Diagonalize (rep f2)] Diag-Diagonalize
      by simp
    ultimately show ?thesis by argo
  qed
  finally show ?thesis by argo
  qed
  finally show ?thesis by blast
  qed
  qed
  next

```

Tensor preserves composition (interchange).

```

fix f g f' g'
show [ seq g f; seq g' f' ] ==> (g ⊗ g') · (f ⊗ f') = g · f ⊗ g' · f'
proof -
  assume gf: seq g f
  assume gf': seq g' f'
  show ?thesis
  proof (intro arr-eqI)
    show par ((g ⊗ g') · (f ⊗ f')) (g · f ⊗ g' · f')
      using gf gf' by fastforce
    show (rep g ⊗ rep g') · (rep f ⊗ rep f') ∈ (g ⊗ g') · (f ⊗ f')
      using gf gf' by force
    show rep g · rep f ⊗ rep g' · rep f' ∈ g · f ⊗ g' · f'
      using gf gf'
      by (meson Comp-in-comp-ARR Tensor-in-tensor rep-in-arr seqE seq-char')
    show [(rep g ⊗ rep g') · (rep f ⊗ rep f')] = [rep g · rep f ⊗ rep g' · rep f']
    proof -
      have [(rep g ⊗ rep g') · (rep f ⊗ rep f')]
        = ([rep g] [⊗] [rep g']) [·] ([rep f] [⊗] [rep f'])
      by auto
      also have ... = [rep g] [·] [rep f] [⊗] [rep g'] [·] [rep f']
      using gf gf' Arr-rep Diagonalize-rep-preserves-seq
        CompDiag-TensorDiag [of [rep g] [rep g'] [rep f] [rep f']]
        Diag-Diagonalize Diagonalize-DOM Diagonalize-COD
      by force
      also have ... = [rep g · rep f ⊗ rep g' · rep f']
      by auto
    finally show ?thesis by blast
  qed
  qed
  qed
  next

```

The triangle.

```

fix a b
assume a: ide a

```

```

assume  $b$ : ide  $b$ 
show  $(a \otimes l[b]) \cdot a[a, \mathcal{I}, b] = r[a] \otimes b$ 
proof –
  have par  $((a \otimes l[b]) \cdot a[a, \mathcal{I}, b]) (r[a] \otimes b)$ 
    using  $a$   $b$  by simp
  moreover have can  $((a \otimes l[b]) \cdot a[a, \mathcal{I}, b])$ 
    using  $a$   $b$  ide-implies-can comp-preserves-can tensor-preserves-can can-assoc can-lunit
    by simp
  moreover have can  $(r[a] \otimes b)$ 
    using  $a$   $b$  ide-implies-can can-runit tensor-preserves-can by simp
  ultimately show ?thesis using can-coherence by blast
qed
next

```

The pentagon.

```

fix  $a$   $b$   $c$   $d$ 
assume  $a$ : ide  $a$ 
assume  $b$ : ide  $b$ 
assume  $c$ : ide  $c$ 
assume  $d$ : ide  $d$ 
show  $(a \otimes a[b, c, d]) \cdot a[a, b \otimes c, d] \cdot (a[a, b, c] \otimes d)$ 
   $= a[a, b, c \otimes d] \cdot a[a \otimes b, c, d]$ 
proof –
  let ?LHS  $= (a \otimes a[b, c, d]) \cdot a[a, b \otimes c, d] \cdot (a[a, b, c] \otimes d)$ 
  let ?RHS  $= a[a, b, c \otimes d] \cdot a[a \otimes b, c, d]$ 
  have par ?LHS ?RHS
    using  $a$   $b$   $c$   $d$  can-assoc tensor-preserves-ide by auto
  moreover have can ?LHS
    using  $a$   $b$   $c$   $d$  ide-implies-can comp-preserves-can tensor-preserves-can can-assoc
    tensor-preserves-ide
    by simp
  moreover have can ?RHS
    using  $a$   $b$   $c$   $d$  comp-preserves-can tensor-preserves-can can-assoc tensor-in-hom
    tensor-preserves-ide
    by simp
  ultimately show ?thesis using can-coherence by blast
qed
qed

```

**lemma** *is-elementary-monoidal-category*:

**shows** *elementary-monoidal-category*

*comp*  $T_{FMC}$  *unity*  $T_{FMC}$  *lunit*  $T_{FMC}$  *runit*  $T_{FMC}$  *assoc*  $T_{FMC}$

..

**abbreviation**  $T_{FMC}$  **where**  $T_{FMC} \equiv EMC.T$

**abbreviation**  $\alpha_{FMC}$  **where**  $\alpha_{FMC} \equiv EMC.\alpha$

**abbreviation**  $\iota_{FMC}$  **where**  $\iota_{FMC} \equiv EMC.\iota$

**interpretation**  $MC$ : *monoidal-category* *comp*  $T_{FMC}$   $\alpha_{FMC}$   $\iota_{FMC}$

```

using EMC.induces-monoidal-category by auto

lemma induces-monoidal-category:
shows monoidal-category comp TFMC αFMC ℓFMC
  ..

end

sublocale free-monoidal-category ⊆
  elementary-monoidal-category
  comp tensorFMC unityFMC lunitFMC runitFMC assocFMC
using is-elementary-monoidal-category by auto

sublocale free-monoidal-category ⊆ monoidal-category comp TFMC αFMC ℓFMC
using induces-monoidal-category by auto

```

## 4.2 Proof of Freeness

Now we proceed on to establish the freeness of  $\mathcal{FC}$ : each functor from  $C$  to a monoidal category  $D$  extends uniquely to a strict monoidal functor from  $\mathcal{FC}$  to  $D$ .

```

context free-monoidal-category
begin

lemma rep-lunit:
assumes ide a
shows rep l[a] = ||l[rep a]||
  using assms Lunit-in-lunit [of a rep a] rep-in-arr norm-memb-eq-rep [of l[a]]
  by simp

lemma rep-runit:
assumes ide a
shows rep r[a] = ||r[rep a]||
  using assms Runit-in-runit [of a rep a] rep-in-arr norm-memb-eq-rep [of r[a]]
  by simp

lemma rep-assoc:
assumes ide a and ide b and ide c
shows rep a[a, b, c] = ||a[rep a, rep b, rep c]||
  using assms Assoc-in-assoc [of a b c rep a rep b rep c] rep-in-arr
  norm-memb-eq-rep [of a[a, b, c]]
  by simp

lemma mkarr-Unity:
shows mkarr I = I
  using unityFMC-def by simp

```

The unitors and associator were given syntactic definitions in terms of corresponding terms, but these were only for the special case of identity arguments (*i.e.* the components

of the natural transformations). We need to show that  $mkarr$  gives the correct result for *all* terms.

**lemma** *mkarr-Lunit*:

**assumes** *Arr t*

**shows**  $mkarr \mathbb{1}[t] = \mathbb{1} (mkarr t)$

**proof** –

**have**  $mkarr \mathbb{1}[t] = mkarr (t \cdot \mathbb{1}[\|Dom\ t\|])$

**using** *assms Arr-implies-Ide-Dom Ide-in-Hom Diagonalize-preserves-Ide  
Diag-Diagonalize Par-Arr-norm*

**by** (*intro mkarr-eqI simp-all*)

**also have**  $\dots = mkarr t \cdot mkarr \mathbb{1}[\|Dom\ t\|]$

**using** *assms Arr-implies-Ide-Dom Par-Arr-norm Ide-in-Hom* **by** *simp*

**also have**  $\dots = mkarr t \cdot \mathbb{1}[dom (mkarr t)]$

**proof** –

**have**  $arr \mathbb{1}[mkarr (Dom\ t)]$

**using** *assms Arr-implies-Ide-Dom ide-mkarr-Ide* **by** *simp*

**moreover have**  $\mathbb{1}[\|Dom\ t\|] \in \mathbb{1}[mkarr (Dom\ t)]$

**using** *assms Arr-implies-Ide-Dom Lunit-in-lunit rep-mkarr  
rep-in-arr [of mkarr (Dom t)]*

**by** *simp*

**ultimately show** *?thesis*

**using** *assms mkarr-memb(2)* **by** *simp*

**qed**

**also have**  $\dots = \mathbb{1} (mkarr t)$

**using** *assms Arr-implies-Ide-Dom ide-mkarr-Ide lunit-agreement* **by** *simp*

**finally show** *?thesis* **by** *blast*

**qed**

**lemma** *mkarr-Lunit'*:

**assumes** *Arr t*

**shows**  $mkarr \mathbb{1}^{-1}[t] = \mathbb{1}' (mkarr t)$

**proof** –

**have**  $mkarr \mathbb{1}^{-1}[t] = mkarr (\mathbb{1}^{-1}[\|Cod\ t\|] \cdot t)$

**using** *assms Arr-implies-Ide-Cod Ide-in-Hom Diagonalize-preserves-Ide  
Diag-Diagonalize Par-Arr-norm*

**by** (*intro mkarr-eqI simp-all*)

**also have**  $\dots = mkarr \mathbb{1}^{-1}[\|Cod\ t\|] \cdot mkarr t$

**using** *assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm* **by** *simp*

**also have**  $\dots = mkarr (Inv \mathbb{1}[\|Cod\ t\|]) \cdot mkarr t$

**proof** –

**have**  $mkarr \mathbb{1}^{-1}[\|Cod\ t\|] = mkarr (Inv \mathbb{1}[\|Cod\ t\|])$

**using** *assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm Inv-in-Hom  
Ide-implies-Can norm-preserves-Can Diagonalize-Inv Diagonalize-preserves-Ide*

**by** (*intro mkarr-eqI, simp-all*)

**thus** *?thesis* **by** *argo*

**qed**

**also have**  $\dots = \mathbb{1}' (cod (mkarr t)) \cdot mkarr t$

**proof** –

**have**  $mkarr (Inv \mathbb{1}[\|Cod\ t\|]) \cdot mkarr t = lunit' (cod (mkarr t)) \cdot mkarr t$



**using** *assms* *Arr-implies-Ide-Cod rep-mkarr Par-Arr-norm inv-mkarr*  
*norm-preserves-Can Ide-implies-Can lunit-agreement l'-ide-simp*  
*Can-implies-Arr arr-mkarr cod-mkarr ide-cod lunit<sub>FMC</sub>-def*  
**by** (*metis* (*no-types, lifting*) *Can.simps(5)*)  
**also have** ... =  $l'(\text{cod } (mkarr\ t)) \cdot mkarr\ t$   
**using** *assms* *l'-ide-simp arr-mkarr ide-cod* **by** *presburger*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**also have** ... =  $l'(mkarr\ t)$   
**using** *assms* *l'.is-natural-2* [*of mkarr t*] **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**

**lemma** *mkarr-Runit*:  
**assumes** *Arr t*  
**shows**  $mkarr\ r[t] = \varrho(mkarr\ t)$   
**proof** –  
**have**  $mkarr\ r[t] = mkarr\ (t \cdot r[\|Dom\ t\|])$   
**proof** –  
**have**  $\neg\ Diag\ (Dom\ t \otimes \mathcal{I})$  **by** (*cases Dom t*) *simp-all*  
**thus** *?thesis*  
**using** *assms* *Par-Arr-norm Arr-implies-Ide-Dom Ide-in-Hom Diag-Diagonalize*  
*Diagonalize-preserves-Ide*  
**by** (*intro mkarr-eqI*) *simp-all*  
**qed**  
**also have** ... =  $mkarr\ t \cdot mkarr\ r[\|Dom\ t\|]$   
**using** *assms* *Arr-implies-Ide-Dom Par-Arr-norm Ide-in-Hom* **by** *simp*  
**also have** ... =  $mkarr\ t \cdot r[dom\ (mkarr\ t)]$   
**proof** –  
**have**  $arr\ r[mkarr\ (Dom\ t)]$   
**using** *assms* *Arr-implies-Ide-Dom ide-mkarr-Ide* **by** *simp*  
**moreover have**  $r[\|Dom\ t\|] \in r[mkarr\ (Dom\ t)]$   
**using** *assms* *Arr-implies-Ide-Dom Runit-in-runit rep-mkarr*  
*rep-in-arr* [*of mkarr (Dom t)*]  
**by** *simp*  
**moreover have**  $mkarr\ (Dom\ t) = mkarr\ \|Dom\ t\|$   
**using** *assms* *mkarr-rep rep-mkarr arr-mkarr Ide-implies-Arr Arr-implies-Ide-Dom*  
**by** *metis*  
**ultimately show** *?thesis*  
**using** *assms* *mkarr-memb(2)* **by** *simp*  
**qed**  
**also have** ... =  $\varrho(mkarr\ t)$   
**using** *assms* *Arr-implies-Ide-Dom ide-mkarr-Ide runit-agreement* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**

**lemma** *mkarr-Runit'*:  
**assumes** *Arr t*  
**shows**  $mkarr\ r^{-1}[t] = \varrho'(mkarr\ t)$

```

proof –
  have  $mkarr\ r^{-1}[t] = mkarr\ (r^{-1}[\|Cod\ t\|]) \cdot t$ 
  proof –
    have  $\neg\ Diag\ (Cod\ t \otimes \mathcal{I})$  by  $(cases\ Cod\ t)\ simp\ all$ 
    thus ?thesis
    using assms Par-Arr-norm Arr-implies-Ide-Cod Ide-in-Hom
      Diagonalize-preserves-Ide Diag-Diagonalize
    by  $(intro\ mkarr\ eqI)\ simp\ all$ 
  qed
  also have  $\dots = mkarr\ r^{-1}[\|Cod\ t\|] \cdot mkarr\ t$ 
    using assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm by simp
  also have  $\dots = mkarr\ (Inv\ r[\|Cod\ t\|]) \cdot mkarr\ t$ 
  proof –
    have  $mkarr\ (Runit'\ (norm\ (Cod\ t))) = mkarr\ (Inv\ (Runit\ (norm\ (Cod\ t))))$ 
    using assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm Inv-in-Hom
      Ide-implies-Can norm-preserves-Can Diagonalize-Inv Diagonalize-preserves-Ide
    by  $(intro\ mkarr\ eqI)\ simp\ all$ 
    thus ?thesis by argo
  qed
  also have  $\dots = \varrho'\ (cod\ (mkarr\ t)) \cdot mkarr\ t$ 
  proof –
    have  $mkarr\ (Inv\ r[\|Cod\ t\|]) \cdot mkarr\ t = runit'\ (cod\ (mkarr\ t)) \cdot mkarr\ t$ 
    using assms Arr-implies-Ide-Cod rep-mkarr inv-mkarr norm-preserves-Can
      Ide-implies-Can runit-agreement Can-implies-Arr arr-mkarr cod-mkarr
      ide-cod runit_FMC-def
    by  $(metis\ (no-types,\ lifting)\ Can.simps(7))$ 
    also have  $\dots = \varrho'\ (cod\ (mkarr\ t)) \cdot mkarr\ t$ 
  proof –
    have  $runit'\ (cod\ (mkarr\ t)) = \varrho'\ (cod\ (mkarr\ t))$ 
    using assms  $\varrho'$ -ide-simp arr-mkarr ide-cod by blast
    thus ?thesis by argo
  qed
  finally show ?thesis by blast
  qed
  also have  $\dots = \varrho'\ (mkarr\ t)$ 
    using assms  $\varrho'$ -is-natural-2  $[of\ mkarr\ t]$  by simp
  finally show ?thesis by blast
  qed

```

**lemma** *mkarr-Assoc*:

**assumes** *Arr t and Arr u and Arr v*

**shows**  $mkarr\ \mathbf{a}[t, u, v] = \alpha\ (mkarr\ t, mkarr\ u, mkarr\ v)$

**proof** –

**have**  $mkarr\ \mathbf{a}[t, u, v] = mkarr\ ((t \otimes u \otimes v) \cdot \mathbf{a}[\|Dom\ t\|, \|Dom\ u\|, \|Dom\ v\|])$

**using** *assms Arr-implies-Ide-Dom Arr-implies-Ide-Cod Ide-in-Hom*

*Diag-Diagonalize Diagonalize-preserves-Ide TensorDiag-preserves-Ide*

*TensorDiag-preserves-Diag TensorDiag-assoc Par-Arr-norm*

**by**  $(intro\ mkarr\ eqI, simp\ all)$

**also have**  $\dots = \alpha\ (mkarr\ t, mkarr\ u, mkarr\ v)$

**using** *assms Arr-implies-Ide-Dom rep-mkarr Ide-in-Hom assoc<sub>FM C</sub>-def*  
*Par-Arr-norm [of Dom t] Par-Arr-norm [of Dom u] Par-Arr-norm [of Dom v]*  
*α-simp*  
**by** *simp*  
**finally show** *?thesis by blast*  
**qed**

**lemma** *mkarr-Assoc'*:

**assumes** *Arr t and Arr u and Arr v*

**shows** *mkarr a<sup>-1</sup>[t, u, v] = α' (mkarr t, mkarr u, mkarr v)*

**proof** –

**have** *mkarr a<sup>-1</sup>[t, u, v] = mkarr (a<sup>-1</sup>[|| Cod t||, || Cod u||, || Cod v||] · (t ⊗ u ⊗ v))*

**using** *assms Par-Arr-norm Arr-implies-Ide-Cod Ide-in-Hom Diag-Diagonalize*

*TensorDiag-preserves-Diag CompDiag-Cod-Diag [of [t] [⊗] [u] [⊗] [v]]*

**by** *(intro mkarr-eqI, simp-all)*

**also have** *... = mkarr a<sup>-1</sup>[|| Cod t||, || Cod u||, || Cod v||] · mkarr (t ⊗ u ⊗ v)*

**using** *assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm by simp*

**also have** *... = mkarr (Inv a[|| Cod t||, || Cod u||, || Cod v||]) · mkarr (t ⊗ u ⊗ v)*

**proof** –

**have** *mkarr a<sup>-1</sup>[|| Cod t||, || Cod u||, || Cod v||] =*

*mkarr (Inv a[|| Cod t||, || Cod u||, || Cod v||])*

**using** *assms Arr-implies-Ide-Cod Ide-in-Hom Par-Arr-norm Inv-in-Hom Ide-implies-Can*

*norm-preserves-Can Diagonalize-Inv Diagonalize-preserves-Ide*

**by** *(intro mkarr-eqI, simp-all)*

**thus** *?thesis by argo*

**qed**

**also have** *... = inv (mkarr a[|| Cod t||, || Cod u||, || Cod v||]) · mkarr (t ⊗ u ⊗ v)*

**using** *assms Arr-implies-Ide-Cod Ide-implies-Can norm-preserves-Can by simp*

**also have** *... = α' (mkarr t, mkarr u, mkarr v)*

**proof** –

**have** *mkarr (a<sup>-1</sup>[Inv || Cod t||, Inv || Cod u||, Inv || Cod v||] · (Cod t ⊗ Cod u ⊗ Cod v))*

*= mkarr a<sup>-1</sup>[Inv || Cod t||, Inv || Cod u||, Inv || Cod v||]*

**using** *assms Arr-implies-Ide-Cod Inv-in-Hom norm-preserves-Can Diagonalize-Inv*

*Ide-implies-Can Diag-Diagonalize Ide-in-Hom Diagonalize-preserves-Ide*

*Par-Arr-norm TensorDiag-preserves-Diag*

*CompDiag-Cod-Diag [of [Cod t] [⊗] [Cod u] [⊗] [Cod v]]*

**by** *(intro mkarr-eqI) simp-all*

**thus** *?thesis*

**using** *assms Arr-implies-Ide-Cod rep-mkarr assoc<sub>FM C</sub>-def α'.map-simp by simp*

**qed**

**finally show** *?thesis by blast*

**qed**

Next, we define the “inclusion of generators” functor from  $C$  to  $\mathcal{F}C$ .

**definition** *inclusion-of-generators*

**where** *inclusion-of-generators ≡ λf. if C.arr f then mkarr ⟨f⟩ else null*

**lemma** *inclusion-is-functor:*

**shows** *functor C comp inclusion-of-generators*

```

unfolding inclusion-of-generators-def
apply unfold-locales
  apply auto[4]
by (elim C.seqE, simp, intro mkarr-eqI, auto)

```

**end**

We now show that, given a functor  $V$  from  $C$  to a monoidal category  $D$ , the evaluation map that takes formal arrows of the monoidal language of  $C$  to arrows of  $D$  induces a strict monoidal functor from  $\mathcal{F}C$  to  $D$ .

```

locale evaluation-functor =
  C: category C +
  D: monoidal-category D TD αD ιD +
  evaluation-map C D TD αD ιD V +
  FC: free-monoidal-category C
for C :: 'c comp      (infixr ·C 55)
and D :: 'd comp      (infixr ·D 55)
and TD :: 'd * 'd ⇒ 'd
and αD :: 'd * 'd * 'd ⇒ 'd
and ιD :: 'd
and V :: 'c ⇒ 'd
begin

```

```

  notation eval      ({|-|})

```

```

  definition map
  where map f ≡ if FC.arr f then {|FC.rep f|} else D.null

```

It follows from the coherence theorem that a formal arrow and its normal form always have the same evaluation.

```

lemma eval-norm:
assumes Arr t
shows {|{|t|}|} = {|t|}
  using assms FC.Par-Arr-norm FC.Diagonalize-norm coherence canonical-factorization
  by simp

```

```

interpretation functor FC.comp D map

```

**proof**

```

  fix f
  show ¬FC.arr f ⇒ map f = D.null using map-def by simp
  assume f: FC.arr f
  show D.arr (map f) using f map-def FC.arr-char by simp
  show D.dom (map f) = map (FC.dom f)
    using f map-def eval-norm FC.rep-dom Arr-implies-Ide-Dom by auto
  show D.cod (map f) = map (FC.cod f)
    using f map-def eval-norm FC.rep-cod Arr-implies-Ide-Cod by auto
  next
  fix f g
  assume fg: FC.seq g f

```

**show**  $\text{map } (\mathcal{F}C.\text{comp } g f) = D (\text{map } g) (\text{map } f)$   
**using**  $\text{fg map-def } \mathcal{F}C.\text{rep-comp } \mathcal{F}C.\text{rep-preserves-seq eval-norm}$  **by** *auto*  
**qed**

**lemma** *is-functor*:  
**shows** *functor*  $\mathcal{F}C.\text{comp } D \text{ map } ..$

**interpretation**  $FF$ : *product-functor*  $\mathcal{F}C.\text{comp } \mathcal{F}C.\text{comp } D D \text{ map } \text{map } ..$   
**interpretation**  $FoT$ : *composite-functor*  $\mathcal{F}C.CC.\text{comp } \mathcal{F}C.\text{comp } D \mathcal{F}C.T_{FMC} \text{ map } ..$   
**interpretation**  $ToFF$ : *composite-functor*  $\mathcal{F}C.CC.\text{comp } D.CC.\text{comp } D FF.\text{map } T_D ..$

**interpretation** *strict-monoidal-functor*  
 $\mathcal{F}C.\text{comp } \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha \mathcal{F}C.\iota D T_D \alpha_D \iota_D \text{ map}$

**proof**

**show**  $\text{map } \mathcal{F}C.\iota = \iota_D$   
**using**  $\mathcal{F}C.\iota\text{-def } \mathcal{F}C.\text{lunit-agreement map-def } \mathcal{F}C.\text{rep-lunit } \mathcal{F}C.\text{Arr-rep [of } \mathcal{I}]$   
 $\text{eval-norm } \mathcal{F}C.\text{lunit-agreement } D.\text{unit-or-coincidence } D.\text{comp-cod-arr } D.\iota\text{-in-hom}$   
**by** *auto*  
**show**  $\bigwedge f g. \llbracket \mathcal{F}C.\text{arr } f; \mathcal{F}C.\text{arr } g \rrbracket \implies$   
 $\text{map } (\mathcal{F}C.\text{tensor } f g) = D.\text{tensor } (\text{map } f) (\text{map } g)$   
**using**  $\text{map-def } \mathcal{F}C.\text{rep-tensor } \mathcal{F}C.\text{Arr-rep eval-norm}$  **by** *simp*  
**show**  $\bigwedge a b c. \llbracket \mathcal{F}C.\text{ide } a; \mathcal{F}C.\text{ide } b; \mathcal{F}C.\text{ide } c \rrbracket \implies$   
 $\text{map } (\mathcal{F}C.\text{assoc } a b c) = D.\text{assoc } (\text{map } a) (\text{map } b) (\text{map } c)$   
**using**  $\text{map-def } \mathcal{F}C.\text{assoc}_{FMC}\text{-def } \mathcal{F}C.\text{rep-mkarr eval-norm}$  **by** *auto*  
**qed**

**lemma** *is-strict-monoidal-functor*:  
**shows** *strict-monoidal-functor*  $\mathcal{F}C.\text{comp } \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha \mathcal{F}C.\iota D T_D \alpha_D \iota_D \text{ map}$   
 $..$

**end**

**sublocale** *evaluation-functor*  $\subseteq$  *strict-monoidal-functor*  
 $\mathcal{F}C.\text{comp } \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha_{FMC} \mathcal{F}C.\iota_{FMC} D T_D \alpha_D \iota_D \text{ map}$   
**using** *is-strict-monoidal-functor* **by** *auto*

The final step in proving freeness is to show that the evaluation functor is the *unique* strict monoidal extension of the functor  $V$  to  $\mathcal{F}C$ . This is done by induction, exploiting the syntactic construction of  $\mathcal{F}C$ .

To ease the statement and proof of the result, we define a locale that expresses that  $F$  is a strict monoidal extension to monoidal category  $C$ , of a functor  $V$  from  $C_0$  to a monoidal category  $D$ , along a functor  $I$  from  $C_0$  to  $C$ .

**locale** *strict-monoidal-extension* =  
 $C_0$ : *category*  $C_0$  +  
 $C$ : *monoidal-category*  $C T_C \alpha_C \iota_C$  +  
 $D$ : *monoidal-category*  $D T_D \alpha_D \iota_D$  +  
 $I$ : *functor*  $C_0 C I$  +  
 $V$ : *functor*  $C_0 D V$  +

```

    strict-monoidal-functor C T_C α_C ι_C D T_D α_D ι_D F
  for C_0 :: 'c_0 comp
  and C :: 'c comp      (infixr ·_C 55)
  and T_C :: 'c * 'c ⇒ 'c
  and α_C :: 'c * 'c * 'c ⇒ 'c
  and ι_C :: 'c
  and D :: 'd comp      (infixr ·_D 55)
  and T_D :: 'd * 'd ⇒ 'd
  and α_D :: 'd * 'd * 'd ⇒ 'd
  and ι_D :: 'd
  and I :: 'c_0 ⇒ 'c
  and V :: 'c_0 ⇒ 'd
  and F :: 'c ⇒ 'd +
  assumes is-extension: ∀ f. C_0.arr f → F (I f) = V f

  sublocale evaluation-functor ⊆
    strict-monoidal-extension C FC.comp FC.T_FMC FC.α FC.ι D T_D α_D ι_D
    FC.inclusion-of-generators V map

  proof –
    interpret inclusion: functor C FC.comp FC.inclusion-of-generators
    using FC.inclusion-is-functor by auto
    show strict-monoidal-extension C FC.comp FC.T_FMC FC.α FC.ι D T_D α_D ι_D
    FC.inclusion-of-generators V map

    apply unfold-locales
    using map-def FC.rep-mkarr eval-norm FC.inclusion-of-generators-def by simp
  qed

```

A special case of interest is a strict monoidal extension to  $\mathcal{F}C$ , of a functor  $V$  from a category  $C$  to a monoidal category  $D$ , along the inclusion of generators from  $C$  to  $\mathcal{F}C$ . The evaluation functor induced by  $V$  is such an extension.

```

  locale strict-monoidal-extension-to-free-monoidal-category =
    C: category C +
    monoidal-language C +
    FC: free-monoidal-category C +
    strict-monoidal-extension C FC.comp FC.T_FMC FC.α FC.ι D T_D α_D ι_D
    FC.inclusion-of-generators V F

  for C :: 'c comp      (infixr ·_C 55)
  and D :: 'd comp      (infixr ·_D 55)
  and T_D :: 'd * 'd ⇒ 'd
  and α_D :: 'd * 'd * 'd ⇒ 'd
  and ι_D :: 'd
  and V :: 'c ⇒ 'd
  and F :: 'c free-monoidal-category.arr ⇒ 'd
  begin

    lemma strictly-preserves-everything:
    shows C.arr f ⇒ F (FC.mkarr ⟨f⟩) = V f
    and F (FC.mkarr  $\mathcal{I}$ ) =  $\mathcal{I}_D$ 
    and [ Arr t; Arr u ] ⇒ F (FC.mkarr (t ⊗ u)) = F (FC.mkarr t) ⊗_D F (FC.mkarr u)
  end

```

**and**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Dom } t = \text{Cod } u \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } (t \cdot u)) = F (\mathcal{F}C.\text{mkarr } t) \cdot_D F (\mathcal{F}C.\text{mkarr } u)$   
**and**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{l}[t]) = D.\mathbf{l} (F (\mathcal{F}C.\text{mkarr } t))$   
**and**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{l}^{-1}[t]) = D.\mathbf{l}'.\text{map} (F (\mathcal{F}C.\text{mkarr } t))$   
**and**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{r}[t]) = D.\mathbf{r} (F (\mathcal{F}C.\text{mkarr } t))$   
**and**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{r}^{-1}[t]) = D.\mathbf{r}'.\text{map} (F (\mathcal{F}C.\text{mkarr } t))$   
**and**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Arr } v \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } \mathbf{a}[t, u, v]) = \alpha_D (F (\mathcal{F}C.\text{mkarr } t), F (\mathcal{F}C.\text{mkarr } u), F (\mathcal{F}C.\text{mkarr } v))$   
**and**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Arr } v \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } \mathbf{a}^{-1}[t, u, v])$   
 $= D.\alpha' (F (\mathcal{F}C.\text{mkarr } t), F (\mathcal{F}C.\text{mkarr } u), F (\mathcal{F}C.\text{mkarr } v))$   
**proof** –  
**show**  $C.\text{arr } f \implies F (\mathcal{F}C.\text{mkarr } \langle f \rangle) = V f$   
**using** *is-extension*  $\mathcal{F}C.\text{inclusion-of-generators-def}$  **by** *simp*  
**show**  $F (\mathcal{F}C.\text{mkarr } \mathcal{I}) = \mathcal{I}_D$   
**using**  $\mathcal{F}C.\text{mkarr-Unity}$   $\mathcal{F}C.\iota\text{-def}$  *strictly-preserves-unity*  $\mathcal{F}C.\mathcal{I}\text{-agreement}$  **by** *auto*  
**show** *tensor-case*:  
 $\bigwedge t u. \llbracket \text{Arr } t; \text{Arr } u \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } (t \otimes u)) = F (\mathcal{F}C.\text{mkarr } t) \otimes_D F (\mathcal{F}C.\text{mkarr } u)$   
**proof** –  
**fix**  $t u$   
**assume**  $t: \text{Arr } t$  **and**  $u: \text{Arr } u$   
**have**  $F (\mathcal{F}C.\text{mkarr } (t \otimes u)) = F (\mathcal{F}C.\text{tensor } (\mathcal{F}C.\text{mkarr } t) (\mathcal{F}C.\text{mkarr } u))$   
**using**  $t u$   $\mathcal{F}C.\text{tensor-mkarr}$   $\mathcal{F}C.\text{arr-mkarr}$  **by** *simp*  
**also have**  $\dots = F (\mathcal{F}C.\text{mkarr } t) \otimes_D F (\mathcal{F}C.\text{mkarr } u)$   
**using**  $t u$   $\mathcal{F}C.\text{arr-mkarr}$  *strictly-preserves-tensor* **by** *blast*  
**finally show**  $F (\mathcal{F}C.\text{mkarr } (t \otimes u)) = F (\mathcal{F}C.\text{mkarr } t) \otimes_D F (\mathcal{F}C.\text{mkarr } u)$   
**by** *fast*  
**qed**  
**show**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Dom } t = \text{Cod } u \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } (t \cdot u)) = F (\mathcal{F}C.\text{mkarr } t) \cdot_D F (\mathcal{F}C.\text{mkarr } u)$   
**proof** –  
**fix**  $t u$   
**assume**  $t: \text{Arr } t$  **and**  $u: \text{Arr } u$  **and**  $tu: \text{Dom } t = \text{Cod } u$   
**show**  $F (\mathcal{F}C.\text{mkarr } (t \cdot u)) = F (\mathcal{F}C.\text{mkarr } t) \cdot_D F (\mathcal{F}C.\text{mkarr } u)$   
**proof** –  
**have**  $F (\mathcal{F}C.\text{mkarr } (t \cdot u)) = F (\mathcal{F}C.\text{mkarr } t \cdot \mathcal{F}C.\text{mkarr } u)$   
**using**  $t u tu$   $\mathcal{F}C.\text{comp-mkarr}$  **by** *simp*  
**also have**  $\dots = F (\mathcal{F}C.\text{mkarr } t) \cdot_D F (\mathcal{F}C.\text{mkarr } u)$   
**using**  $t u tu$   $\mathcal{F}C.\text{arr-mkarr}$  **by** *fastforce*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**qed**  
**show**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{l}[t]) = D.\mathbf{l} (F (\mathcal{F}C.\text{mkarr } t))$   
**using**  $\mathcal{F}C.\text{mkarr-Lunit}$  *Arr-implies-Ide-Dom*  $\mathcal{F}C.\text{ide-mkarr-Ide}$  *strictly-preserves-lunit*  
**by** *simp*  
**show**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{r}[t]) = D.\mathbf{r} (F (\mathcal{F}C.\text{mkarr } t))$   
**using**  $\mathcal{F}C.\text{mkarr-Runit}$  *Arr-implies-Ide-Dom*  $\mathcal{F}C.\text{ide-mkarr-Ide}$  *strictly-preserves-runit*  
**by** *simp*

**show**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Arr } v \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } \mathbf{a}[t, u, v])$   
 $= \alpha_D (F (\mathcal{F}C.\text{mkarr } t), F (\mathcal{F}C.\text{mkarr } u), F (\mathcal{F}C.\text{mkarr } v))$   
**using**  $\mathcal{F}C.\text{mkarr-Assoc strictly-preserves-assoc } \mathcal{F}C.\text{ide-mkarr-Ide tensor-case}$   
**by** *simp*  
**show**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{l}^{-1}[t]) = D.\mathbf{l}'.\text{map } (F (\mathcal{F}C.\text{mkarr } t))$   
**proof** –  
**assume**  $t: \text{Arr } t$   
**have**  $F (\mathcal{F}C.\text{mkarr } \mathbf{l}^{-1}[t]) = F (\mathcal{F}C.\text{lunit}' (\mathcal{F}C.\text{mkarr } (\text{Cod } t))) \cdot_D F (\mathcal{F}C.\text{mkarr } t)$   
**using**  $t \mathcal{F}C.\text{mkarr-Lunit}' \text{Arr-implies-Ide-Cod } \mathcal{F}C.\text{ide-mkarr-Ide } \mathcal{F}C.\mathbf{l}'.\text{map-simp}$   
 $\mathcal{F}C.\text{comp-cod-arr}$   
**by** *simp*  
**also have**  $\dots = D.\text{lunit}' (D.\text{cod } (F (\mathcal{F}C.\text{mkarr } t))) \cdot_D F (\mathcal{F}C.\text{mkarr } t)$   
**using**  $t \text{Arr-implies-Ide-Cod } \mathcal{F}C.\text{ide-mkarr-Ide strictly-preserves-lunit}$   
 $\text{preserves-inv}$   
**by** *simp*  
**also have**  $\dots = D.\mathbf{l}'.\text{map } (F (\mathcal{F}C.\text{mkarr } t))$   
**using**  $t D.\mathbf{l}'.\text{map-simp } D.\text{comp-cod-arr}$  **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**show**  $\text{Arr } t \implies F (\mathcal{F}C.\text{mkarr } \mathbf{r}^{-1}[t]) = D.\mathbf{r}'.\text{map } (F (\mathcal{F}C.\text{mkarr } t))$   
**proof** –  
**assume**  $t: \text{Arr } t$   
**have**  $F (\mathcal{F}C.\text{mkarr } \mathbf{r}^{-1}[t]) = F (\mathcal{F}C.\text{runit}' (\mathcal{F}C.\text{mkarr } (\text{Cod } t))) \cdot_D F (\mathcal{F}C.\text{mkarr } t)$   
**using**  $t \mathcal{F}C.\text{mkarr-Runit}' \text{Arr-implies-Ide-Cod } \mathcal{F}C.\text{ide-mkarr-Ide } \mathcal{F}C.\mathbf{r}'.\text{map-simp}$   
 $\mathcal{F}C.\text{comp-cod-arr}$   
**by** *simp*  
**also have**  $\dots = D.\text{runit}' (D.\text{cod } (F (\mathcal{F}C.\text{mkarr } t))) \cdot_D F (\mathcal{F}C.\text{mkarr } t)$   
**using**  $t \text{Arr-implies-Ide-Cod } \mathcal{F}C.\text{ide-mkarr-Ide strictly-preserves-runit}$   
 $\text{preserves-inv}$   
**by** *simp*  
**also have**  $\dots = D.\mathbf{r}'.\text{map } (F (\mathcal{F}C.\text{mkarr } t))$   
**using**  $t D.\mathbf{r}'.\text{map-simp } D.\text{comp-cod-arr}$  **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**show**  $\llbracket \text{Arr } t; \text{Arr } u; \text{Arr } v \rrbracket \implies$   
 $F (\mathcal{F}C.\text{mkarr } \mathbf{a}^{-1}[t, u, v])$   
 $= D.\mathbf{a}'.\text{map } (F (\mathcal{F}C.\text{mkarr } t), F (\mathcal{F}C.\text{mkarr } u), F (\mathcal{F}C.\text{mkarr } v))$   
**proof** –  
**assume**  $t: \text{Arr } t$  **and**  $u: \text{Arr } u$  **and**  $v: \text{Arr } v$   
**have**  $F (\mathcal{F}C.\text{mkarr } \mathbf{a}^{-1}[t, u, v]) =$   
 $F (\mathcal{F}C.\text{assoc}' (\mathcal{F}C.\text{mkarr } (\text{Cod } t)) (\mathcal{F}C.\text{mkarr } (\text{Cod } u)) (\mathcal{F}C.\text{mkarr } (\text{Cod } v))) \cdot_D$   
 $(F (\mathcal{F}C.\text{mkarr } t) \otimes_D F (\mathcal{F}C.\text{mkarr } u) \otimes_D F (\mathcal{F}C.\text{mkarr } v))$   
**using**  $t u v \mathcal{F}C.\text{mkarr-Assoc}' \text{Arr-implies-Ide-Cod } \mathcal{F}C.\text{ide-mkarr-Ide } \mathcal{F}C.\mathbf{a}'.\text{map-simp}$   
 $\text{tensor-case } \mathcal{F}C.\text{iso-assoc}$   
**by** *simp*  
**also have**  $\dots = D.\text{assoc}' (D.\text{cod } (F (\mathcal{F}C.\text{mkarr } t))) (D.\text{cod } (F (\mathcal{F}C.\text{mkarr } u)))$   
 $(D.\text{cod } (F (\mathcal{F}C.\text{mkarr } v))) \cdot_D$   
 $(F (\mathcal{F}C.\text{mkarr } t) \otimes_D F (\mathcal{F}C.\text{mkarr } u) \otimes_D F (\mathcal{F}C.\text{mkarr } v))$



```

    using t u v FC.ide-mkarr-Ide Arr-implies-Ide-Cod preserves-inv FC.iso-assoc
      strictly-preserves-assoc
      [of FC.mkarr (Cod t) FC.mkarr (Cod u) FC.mkarr (Cod v)]
    by simp
    also have ... = D.alpha'.map (F (FC.mkarr t), F (FC.mkarr u), F (FC.mkarr v))
      using t u v D.alpha'.map-simp by simp
    finally show ?thesis by blast
  qed
qed

end

sublocale evaluation-functor  $\subseteq$  strict-monoidal-extension-to-free-monoidal-category
  C D T_D alpha_D l_D V map
..

context free-monoidal-category
begin

  The evaluation functor induced by  $V$  is the unique strict monoidal extension of  $V$  to
   $FC$ .

  theorem is-free:
  assumes strict-monoidal-extension-to-free-monoidal-category C D T_D alpha_D l_D V F
  shows  $F = \text{evaluation-functor.map } C D T_D \alpha_D l_D V$ 
  proof -
    interpret  $F$ : strict-monoidal-extension-to-free-monoidal-category C D T_D alpha_D l_D V F
      using assms by auto
    interpret  $E$ : evaluation-functor C D T_D alpha_D l_D V ..
    have Ide-case:  $\bigwedge a. \text{Ide } a \implies F (\text{mkarr } a) = E.\text{map } (\text{mkarr } a)$ 
    proof -
      fix  $a$ 
      show  $\text{Ide } a \implies F (\text{mkarr } a) = E.\text{map } (\text{mkarr } a)$ 
        using E.strictly-preserves-everything F.strictly-preserves-everything Ide-implies-Arr
        by (induct a) auto
    qed
    show ?thesis
  proof
    fix  $f$ 
    have  $\neg \text{arr } f \implies F f = E.\text{map } f$ 
      using E.is-extensional F.is-extensional by simp
    moreover have  $\text{arr } f \implies F f = E.\text{map } f$ 
    proof -
      assume  $f$ : arr f
      have  $\text{Arr } (\text{rep } f) \wedge f = \text{mkarr } (\text{rep } f)$  using f mkarr-rep by simp
      moreover have  $\bigwedge t. \text{Arr } t \implies F (\text{mkarr } t) = E.\text{map } (\text{mkarr } t)$ 
      proof -
        fix  $t$ 
        show  $\text{Arr } t \implies F (\text{mkarr } t) = E.\text{map } (\text{mkarr } t)$ 
          using Ide-case E.strictly-preserves-everything F.strictly-preserves-everything

```

```

      Arr-implies-Ide-Dom Arr-implies-Ide-Cod
    by (induct t) auto
  qed
  ultimately show  $F f = E.map f$  by metis
  qed
  ultimately show  $F f = E.map f$  by blast
  qed
  qed
end

```

### 4.3 Strict Subcategory

```

context free-monoidal-category
begin

```

In this section we show that  $\mathcal{F}C$  is monoidally equivalent to its full subcategory  $\mathcal{F}_S C$  whose objects are the equivalence classes of diagonal identity terms, and that this subcategory is the free strict monoidal category generated by  $C$ .

```

interpretation  $\mathcal{F}_S C$ : full-subcategory comp  $\langle \lambda f. ide f \wedge Diag (DOM f) \rangle$ 
  by (unfold-locales) auto

```

The mapping defined on equivalence classes by diagonalizing their representatives is a functor from the free monoidal category to the subcategory  $\mathcal{F}_S C$ .

```

definition  $D$ 
  where  $D \equiv \lambda f. if\ arr\ f\ then\ mkarr\ [rep\ f]\ else\ \mathcal{F}_S C.null$ 

```

The arrows of  $\mathcal{F}_S C$  are those equivalence classes whose canonical representative term has diagonal formal domain and codomain.

```

lemma strict-arr-char:
shows  $\mathcal{F}_S C.arr\ f \longleftrightarrow arr\ f \wedge Diag\ (DOM\ f) \wedge Diag\ (COD\ f)$ 
proof
  show  $arr\ f \wedge Diag\ (DOM\ f) \wedge Diag\ (COD\ f) \implies \mathcal{F}_S C.arr\ f$ 
    using  $\mathcal{F}_S C.arr-char\ DOM-dom\ DOM-cod$  by simp
  show  $\mathcal{F}_S C.arr\ f \implies arr\ f \wedge Diag\ (DOM\ f) \wedge Diag\ (COD\ f)$ 
    using  $\mathcal{F}_S C.arr-char\ Arr-rep\ Arr-implies-Ide-Cod\ Ide-implies-Arr\ DOM-dom\ DOM-cod$ 
    by force
qed

```

Alternatively, the arrows of  $\mathcal{F}_S C$  are those equivalence classes that are preserved by diagonalization of representatives.

```

lemma strict-arr-char':
shows  $\mathcal{F}_S C.arr\ f \longleftrightarrow arr\ f \wedge D\ f = f$ 
proof
  fix  $f$ 
  assume  $f: \mathcal{F}_S C.arr\ f$ 
  show  $arr\ f \wedge D\ f = f$ 
proof

```

```

show arr f using f  $\mathcal{F}_S C$ .arr-char by blast
show  $D f = f$ 
  using f strict-arr-char mkarr-Diagonalize-rep D-def by simp
qed
next
assume f: arr f  $\wedge$   $D f = f$ 
show  $\mathcal{F}_S C$ .arr f
proof -
  have arr f using f by simp
  moreover have  $Diag (DOM f)$ 
  proof -
    have  $DOM f = DOM (mkarr [rep f])$  using f D-def by auto
    also have ... =  $Dom \parallel [rep f] \parallel$ 
      using f Arr-rep Diagonalize-in-Hom rep-mkarr by simp
    also have ... =  $Dom [rep f]$ 
      using f Arr-rep Diagonalize-in-Hom Par-Arr-norm [of [rep f]] by force
    finally have  $DOM f = Dom [rep f]$  by blast
    thus ?thesis using f Arr-rep Diag-Diagonalize Dom-preserves-Diag by metis
  qed
  moreover have  $Diag (COD f)$ 
  proof -
    have  $COD f = COD (mkarr [rep f])$  using f D-def by auto
    also have ... =  $Cod \parallel [rep f] \parallel$ 
      using f Arr-rep Diagonalize-in-Hom rep-mkarr by simp
    also have ... =  $Cod [rep f]$ 
      using f Arr-rep Diagonalize-in-Hom Par-Arr-norm [of [rep f]] by force
    finally have  $COD f = Cod [rep f]$  by blast
    thus ?thesis using f Arr-rep Diag-Diagonalize Cod-preserves-Diag by metis
  qed
  ultimately show ?thesis using strict-arr-char by auto
qed
qed

```

**interpretation**  $D$ : functor comp  $\mathcal{F}_S C$ .comp  $D$

```

proof -
  have 1:  $\bigwedge f. arr f \implies \mathcal{F}_S C$ .arr (D f)
    unfolding strict-arr-char D-def
    using arr-mkarr Diagonalize-in-Hom Arr-rep rep-mkarr Par-Arr-norm
      Arr-implies-Ide-Dom Arr-implies-Ide-Cod Diag-Diagonalize
    by force
  show functor comp  $\mathcal{F}_S C$ .comp  $D$ 
  proof
    show  $\bigwedge f. \neg arr f \implies D f = \mathcal{F}_S C$ .null using D-def by simp
    show  $\bigwedge f. arr f \implies \mathcal{F}_S C$ .arr (D f) by fact
    show  $\bigwedge f. arr f \implies \mathcal{F}_S C$ .dom (D f) =  $D (dom f)$ 
      using D-def Diagonalize-in-Hom  $\mathcal{F}_S C$ .dom-char  $\mathcal{F}_S C$ .arr-char
        rep-mkarr rep-dom Arr-implies-Ide-Dom Arr-implies-Ide-Cod
        Diagonalize-preserves-Ide ide-mkarr-Ide Diag-Diagonalize Dom-norm
      by simp
  qed

```

```

show 2:  $\bigwedge f. \text{arr } f \implies \mathcal{F}_S C. \text{cod } (D f) = D (\text{cod } f)$ 
  using D-def Diagonalize-in-Hom  $\mathcal{F}_S C. \text{cod-char } \mathcal{F}_S C. \text{arr-char}$ 
    rep-mkarr rep-cod Arr-implies-Ide-Dom Arr-implies-Ide-Cod
    Diagonalize-preserves-Ide ide-mkarr-Ide Diag-Diagonalize Dom-norm
  by simp
fix f g
assume fg: seq g f
hence fg': arr f  $\wedge$  arr g  $\wedge$  dom g = cod f by blast
show  $D (g \cdot f) = \mathcal{F}_S C. \text{comp } (D g) (D f)$ 
proof –
  have seq:  $\mathcal{F}_S C. \text{seq } (\text{mkarr } [\text{rep } g]) (\text{mkarr } [\text{rep } f])$ 
  proof –
    have 3:  $\mathcal{F}_S C. \text{arr } (\text{mkarr } [\text{rep } g]) \wedge \mathcal{F}_S C. \text{arr } (\text{mkarr } [\text{rep } f])$ 
      using fg' 1 arr-char D-def by force
    moreover have  $\mathcal{F}_S C. \text{dom } (\text{mkarr } [\text{rep } g]) = \mathcal{F}_S C. \text{cod } (\text{mkarr } [\text{rep } f])$ 
      using fg' 2 3  $\mathcal{F}_S C. \text{dom-char rep-in-Hom mkarr-in-hom D-def}$ 
      Dom-Diagonalize-rep Diag-implies-Arr Diag-Diagonalize(1)  $\mathcal{F}_S C. \text{arr-char}$ 
    by force
    ultimately show ?thesis using  $\mathcal{F}_S C. \text{seqI}$  by auto
  qed
have  $\text{mkarr } [\text{rep } (g \cdot f)] = \mathcal{F}_S C. \text{comp } (\text{mkarr } [\text{rep } g]) (\text{mkarr } [\text{rep } f])$ 
proof –
  have Seq:  $\text{Seq } [\text{rep } g] [\text{rep } f]$ 
    using fg rep-preserves-seq Diagonalize-in-Hom by force
  hence 4:  $[\text{rep } g] \cdot [\text{rep } f] \in \text{Hom } [DOM f] [COD g]$ 
    using fg' Seq Diagonalize-in-Hom by auto
  have  $\mathcal{F}_S C. \text{comp } (\text{mkarr } [\text{rep } g]) (\text{mkarr } [\text{rep } f]) = \text{mkarr } [\text{rep } g] \cdot \text{mkarr } [\text{rep } f]$ 
    using seq  $\mathcal{F}_S C. \text{comp-char } \mathcal{F}_S C. \text{seq-char}$  by meson
  also have  $\dots = \text{mkarr } ([\text{rep } g] \cdot [\text{rep } f])$ 
    using Seq comp-mkarr by fastforce
  also have  $\dots = \text{mkarr } [\text{rep } (g \cdot f)]$ 
  proof (intro mkarr-eqI)
    show  $\text{Par } ([\text{rep } g] \cdot [\text{rep } f]) [\text{rep } (g \cdot f)]$ 
      using fg 4 rep-in-Hom rep-preserves-seq rep-in-Hom Diagonalize-in-Hom
      Par-Arr-norm
    apply (elim seqE, auto)
    by (simp-all add: rep-comp)
  show  $[[\text{rep } g] \cdot [\text{rep } f]] = [[\text{rep } (g \cdot f)]]$ 
    using fg rep-preserves-seq norm-in-Hom Diag-Diagonalize Diagonalize-Diag
    apply auto
    by (simp add: rep-comp)
  qed
finally show ?thesis by blast
qed
thus ?thesis using fg D-def by auto
qed
qed
qed

```

**lemma** *diagonalize-is-functor*:  
**shows** *functor comp*  $\mathcal{F}_S C.comp D ..$

**lemma** *diagonalize-strict-arr*:  
**assumes**  $\mathcal{F}_S C.arr f$   
**shows**  $D f = f$   
**using** *assms arr-char D-def strict-arr-char Arr-rep Arr-implies-Ide-Dom Ide-implies-Arr*  
*mkarr-Diagonalize-rep [of f]*  
**by** *auto*

**lemma** *diagonalize-is-idempotent*:  
**shows**  $D o D = D$   
**unfolding** *D-def*  
**using** *D.is-extensional  $\mathcal{F}_S C.null-char Arr-rep Diagonalize-in-Hom mkarr-Diagonalize-rep$*   
*strict-arr-char rep-mkarr*  
**by** *fastforce*

**lemma** *diagonalize-tensor*:  
**assumes** *arr f and arr g*  
**shows**  $D (f \otimes g) = D (D f \otimes D g)$   
**unfolding** *D-def*  
**using** *assms strict-arr-char rep-in-Hom Diagonalize-in-Hom tensor-mkarr rep-tensor*  
*Diagonalize-in-Hom rep-mkarr Diagonalize-norm Diagonalize-Tensor*  
**by** *force*

**lemma** *ide-diagonalize-can*:  
**assumes** *can f*  
**shows** *ide (D f)*  
**using** *assms D-def Can-rep-can Ide-Diagonalize-Can ide-mkarr-Ide can-implies-arr*  
**by** *simp*

We next show that the diagonalization functor and the inclusion of the full subcategory  $\mathcal{F}_S C$  underlie an equivalence of categories. The arrows  $mkarr (DOM a \downarrow)$ , determined by reductions of canonical representatives, are the components of a natural isomorphism.

**interpretation** *S*: *full-inclusion-functor comp*  $\langle \lambda f. ide f \wedge Diag (DOM f) \rangle ..$

**interpretation** *DoS*: *composite-functor*  $\mathcal{F}_S C.comp comp \mathcal{F}_S C.comp \mathcal{F}_S C.map D$

..

**interpretation** *SoD*: *composite-functor*  $comp \mathcal{F}_S C.comp comp D \mathcal{F}_S C.map ..$

**interpretation** *v*: *transformation-by-components*  
 $comp comp map SoD.map \langle \lambda a. mkarr (DOM a \downarrow) \rangle$

**proof**

**fix** *a*

**assume** *a: ide a*

**show**  $\langle mkarr (DOM a \downarrow) : map a \rightarrow SoD.map a \rangle$

**proof** –

**have**  $\langle mkarr (DOM a \downarrow) : a \rightarrow mkarr [DOM a] \rangle$

**using** *a Arr-implies-Ide-Dom red-in-Hom dom-char [of a]* **by** *auto*

```

moreover have map a = a
  using a map-simp by simp
moreover have SoD.map a = mkarr [DOM a]
  using a D.preserves-ide  $\mathcal{F}_S C.ideD$   $\mathcal{F}_S C.map-simp$  D-def Ide-Diagonalize-rep-ide
    Ide-in-Hom Diagonalize-in-Hom
  by force
ultimately show ?thesis by simp
qed
next
fix f
assume f: arr f
show mkarr (DOM (cod f)↓) · map f = SoD.map f · mkarr (DOM (dom f)↓)
proof –
  have SoD.map f · mkarr (DOM (dom f)↓) = mkarr [rep f] · mkarr (DOM f↓)
    using f DOM-dom D.preserves-arr  $\mathcal{F}_S C.map-simp$  D-def by simp
  also have ... = mkarr ([rep f] · DOM f↓)
    using f Diagonalize-in-Hom red-in-Hom comp-mkarr Arr-implies-Ide-Dom
    by simp
  also have ... = mkarr (COD f↓ · rep f)
proof (intro mkarr-eqI)
  show Par ([rep f] · DOM f↓) (COD f↓ · rep f)
    using f Diagonalize-in-Hom red-in-Hom Arr-implies-Ide-Dom Arr-implies-Ide-Cod
    by simp
  show [[rep f] · DOM f↓] = [COD f↓ · rep f]
proof –
  have [[rep f] · DOM f↓] = [rep f] [·] [DOM f↓]
    using f by simp
  also have ... = [rep f]
    using f Arr-implies-Ide-Dom Can-red Ide-Diagonalize-Can [of DOM f↓]
      Diag-Diagonalize CompDiag-Diag-Ide
    by force
  also have ... = [COD f↓] [·] [rep f]
    using f Arr-implies-Ide-Cod Can-red Ide-Diagonalize-Can [of COD f↓]
      Diag-Diagonalize CompDiag-Diag-Ide
    by force
  also have ... = [COD f↓ · rep f]
    by simp
  finally show ?thesis by blast
qed
qed
also have ... = mkarr (COD f↓) · mkarr (rep f)
  using f comp-mkarr rep-in-Hom red-in-Hom Arr-implies-Ide-Cod by blast
also have ... = mkarr (DOM (cod f)↓) · map f
  using f DOM-cod by simp
finally show ?thesis by blast
qed
qed

```

**interpretation**  $\nu$ : natural-isomorphism comp comp map SoD.map  $\nu$ .map

**apply** *unfold-locales*  
**using**  $\nu$ .*map-simp-ide rep-in-Hom Arr-implies-Ide-Dom Can-red can-mkarr-Can iso-can*  
**by** *simp*

The restriction of the diagonalization functor to the subcategory  $\mathcal{F}_S C$  is the identity.

**lemma** *DoS-eq- $\mathcal{F}_S C$ :*

**shows**  $DoS.map = \mathcal{F}_S C.map$

**proof**

**fix**  $f$

**have**  $\neg \mathcal{F}_S C.arr f \implies DoS.map f = \mathcal{F}_S C.map f$

**using** *DoS.is-extensional  $\mathcal{F}_S C.map-def$  by simp*

**moreover have**  $\mathcal{F}_S C.arr f \implies DoS.map f = \mathcal{F}_S C.map f$

**using**  *$\mathcal{F}_S C.map-simp strict-arr-char Diagonalize-Diag D-def mkarr-Diagonalize-rep$*

**by** *simp*

**ultimately show**  $DoS.map f = \mathcal{F}_S C.map f$  **by** *blast*

**qed**

**interpretation**  $\mu$ : *transformation-by-components*

$\mathcal{F}_S C.comp \mathcal{F}_S C.comp DoS.map \mathcal{F}_S C.map \langle \lambda a. a \rangle$

**using**  *$\mathcal{F}_S C.ideD \mathcal{F}_S C.map-simp DoS-eq- $\mathcal{F}_S C \mathcal{F}_S C.map-simp \mathcal{F}_S C.comp-cod-arr \mathcal{F}_S C.comp-arr-dom$$*

**apply** *unfold-locales*

**by** *(intro  $\mathcal{F}_S C.in-homI$ ) auto*

**interpretation**  $\mu$ : *natural-isomorphism  $\mathcal{F}_S C.comp \mathcal{F}_S C.comp DoS.map \mathcal{F}_S C.map \mu.map$*

**apply** *unfold-locales using  $\mu.map-simp-ide \mathcal{F}_S C.ide-is-iso$  by simp*

**interpretation** *equivalence-of-categories  $\mathcal{F}_S C.comp comp D \mathcal{F}_S C.map \nu.map \mu.map ..$*

We defined the natural isomorphisms  $\mu$  and  $\nu$  by giving their components (*i.e.* their values at objects). However, it is helpful in exporting these facts to have simple characterizations of their values for all arrows.

**definition**  $\mu$

**where**  $\mu \equiv \lambda f. \text{if } \mathcal{F}_S C.arr f \text{ then } f \text{ else } \mathcal{F}_S C.null$

**definition**  $\nu$

**where**  $\nu \equiv \lambda f. \text{if } arr f \text{ then } mkarr (COD \downarrow) \cdot f \text{ else } null$

**lemma**  *$\mu$ -char:*

**shows**  $\mu.map = \mu$

**proof** *(intro NaturalTransformation.eqI)*

**show** *natural-transformation  $\mathcal{F}_S C.comp \mathcal{F}_S C.comp DoS.map \mathcal{F}_S C.map \mu.map ..$*

**have** *natural-transformation  $\mathcal{F}_S C.comp \mathcal{F}_S C.comp \mathcal{F}_S C.map \mathcal{F}_S C.map \mathcal{F}_S C.map$*

**using** *DoS.as-nat-trans.natural-transformation-axioms DoS-eq- $\mathcal{F}_S C$  by simp*

**moreover have**  $\mathcal{F}_S C.map = \mu$  **unfolding**  *$\mu-def$  using  $\mathcal{F}_S C.map-def$  by blast*

**ultimately show** *natural-transformation  $\mathcal{F}_S C.comp \mathcal{F}_S C.comp DoS.map \mathcal{F}_S C.map \mu$*

**using**  *$\mathcal{F}_S C.as-nat-trans.natural-transformation-axioms DoS-eq- $\mathcal{F}_S C$  by simp$*

**show**  $\bigwedge a. \mathcal{F}_S C.ide a \implies \mu.map a = \mu a$

**using**  *$\mu.map-simp-ide \mathcal{F}_S C.ideD \mu-def$  by simp*

**qed**

**lemma**  *$\nu$ -char*:  
**shows**  $\nu.map = \nu$   
**unfolding**  $\nu.map-def$   $\nu-def$  **using** *map-simp* *DOM-cod* **by** *fastforce*

**lemma** *is-equivalent-to-strict-subcategory*:  
**shows** *equivalence-of-categories*  $\mathcal{F}_S C.comp$   $comp$   $D$   $\mathcal{F}_S C.map$   $\nu$   $\mu$   
**proof** –  
**have** *equivalence-of-categories*  $\mathcal{F}_S C.comp$   $comp$   $D$   $\mathcal{F}_S C.map$   $\nu.map$   $\mu.map$  ..  
**thus** *equivalence-of-categories*  $\mathcal{F}_S C.comp$   $comp$   $D$   $\mathcal{F}_S C.map$   $\nu$   $\mu$   
**using**  *$\nu$ -char*  *$\mu$ -char* **by** *simp*  
**qed**

The inclusion of generators functor from  $C$  to  $\mathcal{F}C$  corestricts to a functor from  $C$  to  $\mathcal{F}_S C$ .

**interpretation** *I*: *functor*  $C$  *comp* *inclusion-of-generators*  
**using** *inclusion-is-functor* **by** *auto*  
**interpretation** *DoI*: *composite-functor*  $C$  *comp*  $\mathcal{F}_S C.comp$  *inclusion-of-generators*  $D$  ..

**lemma** *DoI-eq-I*:  
**shows**  $DoI.map = inclusion-of-generators$   
**proof**  
**fix**  $f$   
**have**  $\neg C.arr\ f \implies DoI.map\ f = inclusion-of-generators\ f$   
**using** *DoI.is-extensionsal* *I.is-extensionsal*  *$\mathcal{F}_S C.null-char$*  **by** *blast*  
**moreover** **have**  $C.arr\ f \implies DoI.map\ f = inclusion-of-generators\ f$   
**proof** –  
**assume**  $f: C.arr\ f$   
**have**  $DoI.map\ f = D\ (inclusion-of-generators\ f)$  **using**  $f$  **by** *simp*  
**also** **have**  $\dots = inclusion-of-generators\ f$   
**proof** –  
**have**  $\mathcal{F}_S C.arr\ (inclusion-of-generators\ f)$   
**using**  $f$  *arr-mkarr* *rep-mkarr* *Par-Arr-norm* [*of*  $\langle f \rangle$ ] *strict-arr-char*  
*inclusion-of-generators-def*  
**by** *simp*  
**thus** *?thesis* **using**  $f$  *strict-arr-char'* **by** *blast*  
**qed**  
**finally** **show**  $DoI.map\ f = inclusion-of-generators\ f$  **by** *blast*  
**qed**  
**ultimately** **show**  $DoI.map\ f = inclusion-of-generators\ f$  **by** *blast*  
**qed**

**end**

Next, we show that the subcategory  $\mathcal{F}_S C$  inherits monoidal structure from the ambient category  $\mathcal{F}C$ , and that this monoidal structure is strict.

**locale** *free-strict-monoidal-category* =  
*monoidal-language*  $C$  +  
 $\mathcal{F}C$ : *free-monoidal-category*  $C$  +



*full-subcategory*  $\mathcal{F}C.comp \lambda f. \mathcal{F}C.ide f \wedge Diag (\mathcal{F}C.DOM f)$   
**for**  $C :: 'c \text{ comp}$   
**begin**

**interpretation**  $D$ : *functor*  $\mathcal{F}C.comp \text{ comp } \mathcal{F}C.D$   
**using**  $\mathcal{F}C.diagonalize-is-functor$  **by** *auto*

**notation**  $comp$  (infixr  $\cdot_S$  55)

**definition**  $tensor_S$  (infixr  $\otimes_S$  53)  
**where**  $f \otimes_S g \equiv \mathcal{F}C.D (\mathcal{F}C.tensor f g)$

**definition**  $assoc_S$  ( $a_S[-, -, -]$ )  
**where**  $assoc_S a b c \equiv a \otimes_S b \otimes_S c$

**lemma** *tensor-char*:  
**assumes**  $arr f$  **and**  $arr g$   
**shows**  $f \otimes_S g = \mathcal{F}C.mkarr ([\mathcal{F}C.rep f] [\otimes] [\mathcal{F}C.rep g])$   
**unfolding**  $\mathcal{F}C.D-def$   $tensor_S-def$   
**using**  $assms arr-char \mathcal{F}C.rep-tensor$  **by** *simp*

**lemma** *tensor-in-hom* [*simp*]:  
**assumes**  $\langle f : a \rightarrow b \rangle$  **and**  $\langle g : c \rightarrow d \rangle$   
**shows**  $\langle f \otimes_S g : a \otimes_S c \rightarrow b \otimes_S d \rangle$   
**unfolding**  $tensor_S-def$   
**using**  $assms D.preserves-hom D.preserves-arr arr-char in-hom-char$  **by** *simp*

**lemma** *arr-tensor* [*simp*]:  
**assumes**  $arr f$  **and**  $arr g$   
**shows**  $arr (f \otimes_S g)$   
**using**  $assms arr-iff-in-hom [of f] arr-iff-in-hom [of g] tensor-in-hom$  **by** *blast*

**lemma** *dom-tensor* [*simp*]:  
**assumes**  $arr f$  **and**  $arr g$   
**shows**  $dom (f \otimes_S g) = dom f \otimes_S dom g$   
**using**  $assms arr-iff-in-hom [of f] arr-iff-in-hom [of g] tensor-in-hom$  **by** *blast*

**lemma** *cod-tensor* [*simp*]:  
**assumes**  $arr f$  **and**  $arr g$   
**shows**  $cod (f \otimes_S g) = cod f \otimes_S cod g$   
**using**  $assms arr-iff-in-hom [of f] arr-iff-in-hom [of g] tensor-in-hom$  **by** *blast*

**lemma** *tensor-preserves-ide*:  
**assumes**  $ide a$  **and**  $ide b$   
**shows**  $ide (a \otimes_S b)$   
**using**  $assms tensor_S-def D.preserves-ide \mathcal{F}C.tensor-preserves-ide ide-char$   
**by** *fastforce*

**lemma** *tensor-tensor*:

**assumes**  $arr\ f$  **and**  $arr\ g$  **and**  $arr\ h$   
**shows**  $(f \otimes_S g) \otimes_S h = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**and**  $f \otimes_S g \otimes_S h = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**proof** –  
**show**  $(f \otimes_S g) \otimes_S h = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**proof** –  
**have**  $(f \otimes_S g) \otimes_S h = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ (f \otimes_S g)]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**using** *assms*  $Diag$ - $Diagonalize$   $TensorDiag$ - $preserves$ - $Diag$   $Diag$ - $implies$ - $Arr$   
 $\mathcal{F}C.COD$ - $mkarr$   $\mathcal{F}C.DOM$ - $mkarr$   $\mathcal{F}C.strict$ - $arr$ - $char$   $tensor$ - $char$   
**by** *simp*  
**also have**  
 $\dots = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ (\mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]))]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**using** *assms*  $arr$ - $char$   $tensor$ - $char$  **by** *simp*  
**also have**  $\dots = \mathcal{F}C.mkarr\ ([[\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**using** *assms*  $\mathcal{F}C.rep$ - $mkarr$   $TensorDiag$ - $in$ - $Hom$   $Diag$ - $Diagonalize$   
 $TensorDiag$ - $preserves$ - $Diag$   $arr$ - $char$   
**by** *force*  
**also have**  $\dots = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**using** *assms*  $Diag$ - $Diagonalize$   $TensorDiag$ - $preserves$ - $Diag$   $TensorDiag$ - $assoc$   $arr$ - $char$   
**by** *force*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**show**  $f \otimes_S g \otimes_S h = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h])$   
**proof** –  
**have**  $\dots = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [[[\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h]]])$   
**using** *assms*  $Diag$ - $Diagonalize$   $TensorDiag$ - $preserves$ - $Diag$   $arr$ - $char$  **by** *force*  
**also have**  $\dots = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ ([\mathcal{F}C.rep\ (\mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ g]\ [\otimes]\ [\mathcal{F}C.rep\ h]))])$   
**using** *assms*  $\mathcal{F}C.rep$ - $mkarr$   $TensorDiag$ - $in$ - $Hom$   $Diag$ - $Diagonalize$   $arr$ - $char$  **by** *force*  
**also have**  $\dots = \mathcal{F}C.mkarr\ ([\mathcal{F}C.rep\ f]\ [\otimes]\ [\mathcal{F}C.rep\ (g \otimes_S h)])$   
**using** *assms*  $tensor$ - $char$  **by** *simp*  
**also have**  $\dots = f \otimes_S g \otimes_S h$   
**using** *assms*  $Diag$ - $Diagonalize$   $TensorDiag$ - $preserves$ - $Diag$   $Diag$ - $implies$ - $Arr$   
 $\mathcal{F}C.COD$ - $mkarr$   $\mathcal{F}C.DOM$ - $mkarr$   $\mathcal{F}C.strict$ - $arr$ - $char$   $tensor$ - $char$   
**by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *tensor-assoc*:

**assumes**  $arr\ f$  **and**  $arr\ g$  **and**  $arr\ h$   
**shows**  $(f \otimes_S g) \otimes_S h = f \otimes_S g \otimes_S h$   
**using** *assms*  $tensor$ - $tensor$  **by** *presburger*

**lemma** *arr-unity*:

**shows**  $arr\ \mathcal{I}$   
**using**  $\mathcal{F}C.rep$ - $unity$   $\mathcal{F}C.Par$ - $Arr$ - $norm$   $\mathcal{F}C.\mathcal{I}$ - $agreement$   $\mathcal{F}C.strict$ - $arr$ - $char$  **by** *force*

**lemma** *tensor-unity-arr*:  
**assumes** *arr f*  
**shows**  $\mathcal{I} \otimes_S f = f$   
**using** *assms arr-unity tensor-char FC.strict-arr-char FC.mkarr-Diagonalize-rep*  
**by** *simp*

**lemma** *tensor-arr-unity*:  
**assumes** *arr f*  
**shows**  $f \otimes_S \mathcal{I} = f$   
**using** *assms arr-unity tensor-char FC.strict-arr-char FC.mkarr-Diagonalize-rep*  
**by** *simp*

**lemma** *assoc-char*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $a_S[a, b, c] = \mathcal{F}C.mkarr ([\mathcal{F}C.rep\ a] [\otimes] [\mathcal{F}C.rep\ b] [\otimes] [\mathcal{F}C.rep\ c])$   
**using** *assms tensor-tensor(2) assoc\_S-def ideD(1)* **by** *simp*

**lemma** *assoc-in-hom*:  
**assumes** *ide a and ide b and ide c*  
**shows**  $\langle a_S[a, b, c] : (a \otimes_S b) \otimes_S c \rightarrow a \otimes_S b \otimes_S c \rangle$   
**using** *assms tensor-preserves-ide ideD tensor-assoc assoc\_S-def*  
**by** (*metis (no-types, lifting) ide-in-hom*)

The category  $\mathcal{F}_S C$  is a monoidal category.

**interpretation** *EMC*: *elementary-monoidal-category comp tensor\_S I*  $\langle \lambda a. a \rangle \langle \lambda a. a \rangle$  *assoc\_S*  
**proof**

**show** *ide I*  
**using** *ide-char arr-char FC.rep-mkarr FC.Dom-norm FC.Cod-norm FC.I-agreement*  
**by** *auto*  
**show**  $\bigwedge a. \text{ide } a \implies \text{iso } a$   
**using** *ide-char arr-char iso-char* **by** *auto*  
**show**  $\bigwedge f\ a\ b\ g\ c\ d. [\text{in-hom } a\ b\ f; \text{in-hom } c\ d\ g] \implies \text{in-hom } (a \otimes_S c)\ (b \otimes_S d)\ (f \otimes_S g)$   
**using** *tensor-in-hom* **by** *blast*  
**show**  $\bigwedge a\ b. [\text{ide } a; \text{ide } b] \implies \text{ide } (a \otimes_S b)$   
**using** *tensor-preserves-ide* **by** *blast*  
**show**  $\bigwedge a\ b\ c. [\text{ide } a; \text{ide } b; \text{ide } c] \implies \text{iso } a_S[a, b, c]$   
**using** *tensor-preserves-ide ide-is-iso assoc\_S-def* **by** *presburger*  
**show**  $\bigwedge a\ b\ c. [\text{ide } a; \text{ide } b; \text{ide } c] \implies \langle a_S[a, b, c] : (a \otimes_S b) \otimes_S c \rightarrow a \otimes_S b \otimes_S c \rangle$   
**using** *assoc-in-hom* **by** *blast*  
**show**  $\bigwedge a\ b. [\text{ide } a; \text{ide } b] \implies (a \otimes_S b) \cdot_S a_S[a, \mathcal{I}, b] = a \otimes_S b$   
**using** *ide-def tensor-unity-arr assoc\_S-def ideD(1) tensor-preserves-ide comp-ide-self*  
**by** *simp*  
**show**  $\bigwedge f. \text{arr } f \implies \text{cod } f \cdot_S (\mathcal{I} \otimes_S f) = f \cdot_S \text{dom } f$   
**using** *tensor-unity-arr comp-arr-dom comp-cod-arr* **by** *presburger*  
**show**  $\bigwedge f. \text{arr } f \implies \text{cod } f \cdot_S (f \otimes_S \mathcal{I}) = f \cdot_S \text{dom } f$   
**using** *tensor-arr-unity comp-arr-dom comp-cod-arr* **by** *presburger*  
**next**  
**fix** *a*  
**assume** *a: ide a*

```

show « $a : \mathcal{I} \otimes_S a \rightarrow a$ »
  using  $a$  tensor-unity-arr ide-in-hom [of a] by fast
show « $a : a \otimes_S \mathcal{I} \rightarrow a$ »
  using  $a$  tensor-arr-unity ide-in-hom [of a] by fast
next
fix  $f g f' g'$ 
assume  $fg: seq g f$ 
assume  $fg': seq g' f'$ 
show  $(g \otimes_S g') \cdot_S (f \otimes_S f') = g \cdot_S f \otimes_S g' \cdot_S f'$ 
proof –
  have  $A: \mathcal{F}C.seq g f$  and  $B: \mathcal{F}C.seq g' f'$ 
    using  $fg fg'$  seq-char by auto
  have  $(g \otimes_S g') \cdot_S (f \otimes_S f') = \mathcal{F}C.D ((g \otimes_S g') \cdot (f \otimes_S f'))$ 
    using  $A B$  tensor_S-def by fastforce
  also have  $\dots = \mathcal{F}C.D (g \cdot f \otimes_S g' \cdot f')$ 
    using  $A B \mathcal{F}C.interchange \mathcal{F}C.T-simp \mathcal{F}C.seqE$  by metis
  also have  $\dots = \mathcal{F}C.D (g \cdot f) \otimes_S \mathcal{F}C.D (g' \cdot f')$ 
    using  $A B$  tensor_S-def \mathcal{F}C.T-simp \mathcal{F}C.seqE \mathcal{F}C.diagonalize-tensor arr-char
    by (metis (no-types, lifting) D.preserves-reflects-arr)
  also have  $\dots = \mathcal{F}C.D g \cdot_S \mathcal{F}C.D f \otimes_S \mathcal{F}C.D g' \cdot_S \mathcal{F}C.D f'$ 
    using  $A B$  by simp
  also have  $\dots = g \cdot_S f \otimes_S g' \cdot_S f'$ 
    using  $fg fg' \mathcal{F}C.diagonalize-strict-arr$  by (elim seqE, simp)
  finally show ?thesis by blast
qed
next
fix  $f0 f1 f2$ 
assume  $f0: arr f0$  and  $f1: arr f1$  and  $f2: arr f2$ 
show  $a_S[cod f0, cod f1, cod f2] \cdot_S ((f0 \otimes_S f1) \otimes_S f2)$ 
   $= (f0 \otimes_S f1 \otimes_S f2) \cdot_S a_S[dom f0, dom f1, dom f2]$ 
  using  $f0 f1 f2$  assoc_S-def tensor-assoc dom-tensor cod-tensor arr-tensor
  comp-cod-arr [of f0 \otimes_S f1 \otimes_S f2 cod f0 \otimes_S cod f1 \otimes_S cod f2]
  comp-arr-dom [of f0 \otimes_S f1 \otimes_S f2 dom f0 \otimes_S dom f1 \otimes_S dom f2]
  by presburger
next
fix  $a b c d$ 
assume  $a: ide a$  and  $b: ide b$  and  $c: ide c$  and  $d: ide d$ 
show  $(a \otimes_S a_S[b, c, d]) \cdot_S a_S[a, b \otimes_S c, d] \cdot_S (a_S[a, b, c] \otimes_S d)$ 
   $= a_S[a, b, c \otimes_S d] \cdot_S a_S[a \otimes_S b, c, d]$ 
  unfolding assoc_S-def
  using  $a b c d$  tensor-assoc tensor-preserves-ide ideD tensor-in-hom
  comp-arr-dom [of a \otimes_S b \otimes_S c \otimes_S d]
  by simp
qed

```

**lemma** *is-elementary-monoidal-category:*

**shows** *elementary-monoidal-category comp tensor\_S \mathcal{I} (\lambda a. a) (\lambda a. a) assoc\_S ..*

**abbreviation**  $T_{FSMC}$  **where**  $T_{FSMC} \equiv EMC.T$

**abbreviation**  $\alpha_{FSMC}$  **where**  $\alpha_{FSMC} \equiv EMC.\alpha$   
**abbreviation**  $\iota_{FSMC}$  **where**  $\iota_{FSMC} \equiv EMC.\iota$

**lemma** *is-monoidal-category*:  
**shows** *monoidal-category comp*  $T_{FSMC} \alpha_{FSMC} \iota_{FSMC}$   
**using** *EMC.induces-monoidal-category by auto*

**end**

**sublocale** *free-strict-monoidal-category*  $\subseteq$   
*elementary-monoidal-category comp tensor<sub>S</sub> I λa. a λa. a assoc<sub>S</sub>*  
**using** *is-elementary-monoidal-category by auto*

**sublocale** *free-strict-monoidal-category*  $\subseteq$  *monoidal-category comp*  $T_{FSMC} \alpha_{FSMC} \iota_{FSMC}$   
**using** *is-monoidal-category by auto*

**sublocale** *free-strict-monoidal-category*  $\subseteq$   
*strict-monoidal-category comp*  $T_{FSMC} \alpha_{FSMC} \iota_{FSMC}$   
**using** *tensor-preserves-ide lunit-agreement runit-agreement α-ide-simp assoc<sub>S</sub>-def*  
**by** *unfold-locales auto*

**context** *free-strict-monoidal-category*  
**begin**

The inclusion of generators functor from  $C$  to  $\mathcal{F}_S C$  is the composition of the inclusion of generators from  $C$  to  $\mathcal{F}C$  and the diagonalization functor, which projects  $\mathcal{F}C$  to  $\mathcal{F}_S C$ . As the diagonalization functor is the identity map on the image of  $C$ , the composite functor amounts to the corestriction to  $\mathcal{F}_S C$  of the inclusion of generators of  $\mathcal{F}C$ .

**interpretation** *D: functor*  $\mathcal{F}C.comp comp \mathcal{F}C.D$   
**using** *FC.diagonalize-is-functor by auto*

**interpretation** *I: composite-functor*  $C \mathcal{F}C.comp comp \mathcal{F}C.inclusion-of-generators \mathcal{F}C.D$   
**proof** –

**interpret** *functor*  $C \mathcal{F}C.comp \mathcal{F}C.inclusion-of-generators$   
**using** *FC.inclusion-is-functor by blast*  
**show** *composite-functor*  $C \mathcal{F}C.comp comp \mathcal{F}C.inclusion-of-generators \mathcal{F}C.D ..$   
**qed**

**definition** *inclusion-of-generators*  
**where** *inclusion-of-generators*  $\equiv \mathcal{F}C.inclusion-of-generators$

**lemma** *inclusion-is-functor*:  
**shows** *functor*  $C comp inclusion-of-generators$   
**using** *FC.DoI-eq-I I.functor-axioms inclusion-of-generators-def*  
**by** *auto*

The diagonalization functor is strict monoidal.

**interpretation** *D: strict-monoidal-functor*  $\mathcal{F}C.comp \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha_{FMC} \mathcal{F}C.\iota_{FMC}$   
 $comp T_{FSMC} \alpha_{FSMC} \iota_{FSMC}$

$FC.D$

**proof**  
**show**  $FC.D\ FC.\iota = \iota$   
**proof** –  
**have**  $FC.D\ FC.\iota = FC.mkarr\ [FC.rep\ FC.\iota]$   
**unfolding**  $FC.D-def$  **using**  $FC.\iota-in-hom$  **by** *auto*  
**also have**  $\dots = FC.mkarr\ [1[[\mathcal{I}]]]$   
**using**  $FC.\iota-def\ FC.rep-unity\ FC.rep-lunit\ FC.Par-Arr-norm\ FC.Diagonalize-norm$   
**by** *auto*  
**also have**  $\dots = \iota$   
**using**  $FC.unity_{FMC}-def\ FC.\mathcal{I}-agreement\ \iota-def$  **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**  
**show**  $\bigwedge f\ g.\ [FC.arr\ f;\ FC.arr\ g] \implies$   
 $FC.D\ (FC.tensor\ f\ g) = tensor\ (FC.D\ f)\ (FC.D\ g)$   
**proof** –  
**fix**  $f\ g$   
**assume**  $f: FC.arr\ f$  **and**  $g: FC.arr\ g$   
**have**  $fg: arr\ (FC.D\ f) \wedge arr\ (FC.D\ g)$   
**using**  $f\ g\ D.preserves-arr$  **by** *blast*  
**have**  $FC.D\ (FC.tensor\ f\ g) = f \otimes_S g$   
**using**  $tensor_S-def$  **by** *simp*  
**also have**  $f \otimes_S g = FC.D\ (f \otimes g)$   
**using**  $f\ g\ tensor_S-def$  **by** *simp*  
**also have**  $\dots = FC.D\ f \otimes_S FC.D\ g$   
**using**  $f\ g\ fg\ tensor_S-def\ FC.T-simp\ FC.diagonalize-tensor\ arr-char$   
**by** *(metis (no-types, lifting))*  
**also have**  $\dots = tensor\ (FC.D\ f)\ (FC.D\ g)$   
**using**  $fg\ T-simp$  **by** *simp*  
**finally show**  $FC.D\ (FC.tensor\ f\ g) = tensor\ (FC.D\ f)\ (FC.D\ g)$   
**by** *blast*  
**qed**  
**show**  $\bigwedge a\ b\ c.\ [FC.ide\ a;\ FC.ide\ b;\ FC.ide\ c] \implies$   
 $FC.D\ (FC.assoc\ a\ b\ c) = assoc\ (FC.D\ a)\ (FC.D\ b)\ (FC.D\ c)$   
**proof** –  
**fix**  $a\ b\ c$   
**assume**  $a: FC.ide\ a$  **and**  $b: FC.ide\ b$  **and**  $c: FC.ide\ c$   
**have**  $abc: ide\ (FC.D\ a) \wedge ide\ (FC.D\ b) \wedge ide\ (FC.D\ c)$   
**using**  $a\ b\ c\ D.preserves-ide$  **by** *blast*  
**have**  $abc': FC.ide\ (FC.D\ a) \wedge FC.ide\ (FC.D\ b) \wedge FC.ide\ (FC.D\ c)$   
**using**  $a\ b\ c\ D.preserves-ide\ ide-char$  **by** *simp*  
**have**  $1: \bigwedge f\ g.\ FC.arr\ f \implies FC.arr\ g \implies f \otimes_S g = FC.D\ (f \otimes g)$   
**using**  $tensor_S-def$  **by** *simp*  
**have**  $2: \bigwedge f.\ ide\ f \implies FC.ide\ f$  **using** *ide-char* **by** *blast*  
**have**  $assoc\ (FC.D\ a)\ (FC.D\ b)\ (FC.D\ c) = FC.D\ a \otimes_S FC.D\ b \otimes_S FC.D\ c$   
**using**  $abc\ \alpha-ide-simp\ assoc_S-def$  **by** *simp*  
**also have**  $\dots = FC.D\ a \otimes_S FC.D\ (FC.D\ b \otimes FC.D\ c)$   
**using**  $abc'\ 1$  **by** *auto*  
**also have**  $\dots = FC.D\ a \otimes_S FC.D\ (b \otimes c)$

```

    using b c FC.diagonalize-tensor by force
  also have ... = FC.D (FC.D a  $\otimes$  FC.D (b  $\otimes$  c))
    using 1 b c abc D.preserves-ide FC.tensor-preserves-ide ide-char
    by simp
  also have ... = FC.D (a  $\otimes$  b  $\otimes$  c)
    using a b c FC.diagonalize-tensor by force
  also have ... = FC.D a[a, b, c]
  proof -
    have FC.can a[a, b, c] using a b c FC.can-assoc by simp
    hence FC.ide (FC.D a[a, b, c])
      using a b c FC.ide-diagonalize-can by simp
    moreover have FC.cod (FC.D a[a, b, c]) = FC.D (a  $\otimes$  b  $\otimes$  c)
      using a b c FC.assoc-in-hom D.preserves-hom
      by (metis (no-types, lifting) cod-char in-homE)
    ultimately show ?thesis by simp
  qed
  also have ... = FC.D (FC.assoc a b c)
    using a b c by simp
  finally show FC.D (FC.assoc a b c) = assoc (FC.D a) (FC.D b) (FC.D c)
    by blast
  qed
  qed

```

**lemma** *diagonalize-is-strict-monoidal-functor*:

**shows** *strict-monoidal-functor*  $FC.comp$   $FC.T_{FMC}$   $FC.\alpha_{FMC}$   $FC.\iota_{FMC}$   
 $comp$   $T_{FSMC}$   $\alpha_{FSMC}$   $\iota_{FSMC}$   
 $FC.D$

..

**interpretation**  $\varphi$ : *natural-isomorphism*

$FC.CC.comp$   $comp$   $D.T_D oFF.map$   $D.FoT_C.map$   $D.\varphi$

**using** *D.structure-is-natural-isomorphism* **by** *simp*

The diagonalization functor is part of a monoidal equivalence between the free monoidal category and the subcategory  $\mathcal{F}_S C$ .

**interpretation**  $E$ : *equivalence-of-categories*  $comp$   $FC.comp$   $FC.D$   $map$   $FC.\nu$   $FC.\mu$

**using** *FC.is-equivalent-to-strict-subcategory* **by** *auto*

**interpretation**  $D$ : *monoidal-functor*  $FC.comp$   $FC.T_{FMC}$   $FC.\alpha_{FMC}$   $FC.\iota_{FMC}$

$comp$   $T_{FSMC}$   $\alpha_{FSMC}$   $\iota_{FSMC}$

$FC.D$   $D.\varphi$

**using** *D.monoidal-functor-axioms* **by** *metis*

**interpretation** *equivalence-of-monoidal-categories*  $comp$   $T_{FSMC}$   $\alpha_{FSMC}$   $\iota_{FSMC}$

$FC.comp$   $FC.T_{FMC}$   $FC.\alpha_{FMC}$   $FC.\iota_{FMC}$

$FC.D$   $D.\varphi$   $\mathcal{I}$

$map$   $FC.\nu$   $FC.\mu$

..

The category  $\mathcal{F}C$  is monoidally equivalent to its subcategory  $\mathcal{F}_S C$ .

**theorem** *monoidally-equivalent-to-free-monoidal-category:*  
**shows** *equivalence-of-monoidal-categories comp*  $T_{FSMC} \alpha_{FSMC} \iota_{FSMC}$   
 $FC.comp \ FC.T_{FMC} \ FC.\alpha_{FMC} \ FC.\iota_{FMC}$   
 $FC.D \ D.\varphi$   
 $map \ FC.\nu \ FC.\mu$   
..  
**end**

We next show that the evaluation functor induced on the free monoidal category generated by  $C$  by a functor  $V$  from  $C$  to a strict monoidal category  $D$  restricts to a strict monoidal functor on the subcategory  $\mathcal{F}_S C$ .

**locale** *strict-evaluation-functor* =  
 $D$ : *strict-monoidal-category*  $D \ T_D \ \alpha_D \ \iota_D$  +  
*evaluation-map*  $C \ D \ T_D \ \alpha_D \ \iota_D \ V$  +  
 $FC$ : *free-monoidal-category*  $C$  +  
 $E$ : *evaluation-functor*  $C \ D \ T_D \ \alpha_D \ \iota_D \ V$  +  
 $\mathcal{F}_S C$ : *free-strict-monoidal-category*  $C$   
**for**  $C$  :: ' $c$  comp (infixr  $\cdot_C$  55)  
**and**  $D$  :: ' $d$  comp (infixr  $\cdot_D$  55)  
**and**  $T_D$  :: ' $d$  \* ' $d$   $\Rightarrow$  ' $d$   
**and**  $\alpha_D$  :: ' $d$  \* ' $d$  \* ' $d$   $\Rightarrow$  ' $d$   
**and**  $\iota_D$  :: ' $d$   
**and**  $V$  :: ' $c$   $\Rightarrow$  ' $d$   
**begin**

**notation**  $FC.in-hom$  (« - : -  $\rightarrow$  - »)  
**notation**  $\mathcal{F}_S C.in-hom$  (« - : -  $\rightarrow_S$  - »)

**definition** *map*  
**where**  $map \equiv \lambda f. \text{if } \mathcal{F}_S C.arr \ f \ \text{then } E.map \ f \ \text{else } D.null$

**interpretation** *functor*  $\mathcal{F}_S C.comp \ D \ map$   
**unfolding** *map-def*  
**apply** *unfold-locales*  
**apply** *simp*  
**using**  $\mathcal{F}_S C.arr-char \ E.preserves-arr$   
**apply** *simp*  
**using**  $\mathcal{F}_S C.arr-char \ \mathcal{F}_S C.dom-char \ E.preserves-dom$   
**apply** *simp*  
**using**  $\mathcal{F}_S C.arr-char \ \mathcal{F}_S C.cod-char \ E.preserves-cod$   
**apply** *simp*  
**using**  $\mathcal{F}_S C.arr-char \ \mathcal{F}_S C.dom-char \ \mathcal{F}_S C.cod-char \ \mathcal{F}_S C.comp-char \ E.preserves-comp$   
**by** (*elim*  $\mathcal{F}_S C.seqE$ , *auto*)

**lemma** *is-functor:*  
**shows** *functor*  $\mathcal{F}_S C.comp \ D \ map$  ..



Every canonical arrow is an equivalence class of canonical terms. The evaluations in  $D$  of all such terms are identities, due to the strictness of  $D$ .

**lemma** *ide-eval-Can*:

**shows**  $\text{Can } t \Longrightarrow D.\text{ide } \{t\}$

**proof** (*induct t*)

**show**  $\bigwedge x. \text{Can } \langle x \rangle \Longrightarrow D.\text{ide } \{\langle x \rangle\}$  **by** *simp*

**show**  $\text{Can } \mathcal{I} \Longrightarrow D.\text{ide } \{\mathcal{I}\}$  **by** *simp*

**show**  $\bigwedge t1\ t2. \llbracket \text{Can } t1 \Longrightarrow D.\text{ide } \{t1\}; \text{Can } t2 \Longrightarrow D.\text{ide } \{t2\}; \text{Can } (t1 \otimes t2) \rrbracket \Longrightarrow D.\text{ide } \{t1 \otimes t2\}$

**by** *simp*

**show**  $\bigwedge t1\ t2. \llbracket \text{Can } t1 \Longrightarrow D.\text{ide } \{t1\}; \text{Can } t2 \Longrightarrow D.\text{ide } \{t2\}; \text{Can } (t1 \cdot t2) \rrbracket \Longrightarrow D.\text{ide } \{t1 \cdot t2\}$

**proof** –

**fix**  $t1\ t2$

**assume**  $t1: \text{Can } t1 \Longrightarrow D.\text{ide } \{t1\}$

**and**  $t2: \text{Can } t2 \Longrightarrow D.\text{ide } \{t2\}$

**and**  $t12: \text{Can } (t1 \cdot t2)$

**show**  $D.\text{ide } \{t1 \cdot t2\}$

**using**  $t1\ t2\ t12$  *Can-implies-Arr eval-in-hom [of t1] eval-in-hom [of t2] D.comp-ide-arr*  
**by** *fastforce*

**qed**

**show**  $\bigwedge t. (\text{Can } t \Longrightarrow D.\text{ide } \{t\}) \Longrightarrow \text{Can } \mathbf{l}[t] \Longrightarrow D.\text{ide } \{\mathbf{l}[t]\}$

**using** *D.strict-lunit* **by** *simp*

**show**  $\bigwedge t. (\text{Can } t \Longrightarrow D.\text{ide } \{t\}) \Longrightarrow \text{Can } \mathbf{l}^{-1}[t] \Longrightarrow D.\text{ide } \{\mathbf{l}^{-1}[t]\}$

**using** *D.strict-lunit* **by** *simp*

**show**  $\bigwedge t. (\text{Can } t \Longrightarrow D.\text{ide } \{t\}) \Longrightarrow \text{Can } \mathbf{r}[t] \Longrightarrow D.\text{ide } \{\mathbf{r}[t]\}$

**using** *D.strict-runit* **by** *simp*

**show**  $\bigwedge t. (\text{Can } t \Longrightarrow D.\text{ide } \{t\}) \Longrightarrow \text{Can } \mathbf{r}^{-1}[t] \Longrightarrow D.\text{ide } \{\mathbf{r}^{-1}[t]\}$

**using** *D.strict-runit* **by** *simp*

**fix**  $t1\ t2\ t3$

**assume**  $t1: \text{Can } t1 \Longrightarrow D.\text{ide } \{t1\}$

**and**  $t2: \text{Can } t2 \Longrightarrow D.\text{ide } \{t2\}$

**and**  $t3: \text{Can } t3 \Longrightarrow D.\text{ide } \{t3\}$

**show**  $\text{Can } \mathbf{a}[t1, t2, t3] \Longrightarrow D.\text{ide } \{\mathbf{a}[t1, t2, t3]\}$

**proof** –

**assume**  $\text{Can } \mathbf{a}[t1, t2, t3]$

**hence**  $t123: D.\text{ide } \{t1\} \wedge D.\text{ide } \{t2\} \wedge D.\text{ide } \{t3\}$

**using**  $t1\ t2\ t3$  **by** *simp*

**have**  $\{\mathbf{a}[t1, t2, t3]\} = \{t1\} \otimes_D \{t2\} \otimes_D \{t3\}$

**using**  $t123$  *D.strict-assoc D.assoc-in-hom [of {t1} {t2} {t3}]* **apply** *simp*

**by** (*elim D.in-homE, simp*)

**thus** *?thesis* **using**  $t123$  **by** *simp*

**qed**

**show**  $\text{Can } \mathbf{a}^{-1}[t1, t2, t3] \Longrightarrow D.\text{ide } \{\mathbf{a}^{-1}[t1, t2, t3]\}$

**proof** –

**assume**  $\text{Can } \mathbf{a}^{-1}[t1, t2, t3]$

**hence**  $t123: \text{Can } t1 \wedge \text{Can } t2 \wedge \text{Can } t3 \wedge D.\text{ide } \{t1\} \wedge D.\text{ide } \{t2\} \wedge D.\text{ide } \{t3\}$

**using**  $t1\ t2\ t3$  **by** *simp*

**have**  $\{\mathbf{a}^{-1}[t1, t2, t3]\}$

```

      = D.inv aD[D.cod {t1}, D.cod {t2}, D.cod {t3}] ·D ({t1} ⊗D {t2} ⊗D {t3})
    using t123 eval-Assoc' [of t1 t2 t3] Can-implies-Arr by simp
  also have ... = {t1} ⊗D {t2} ⊗D {t3}
  proof -
    have D.dom aD[{t1}, {t2}, {t3}] = {t1} ⊗D {t2} ⊗D {t3}
    proof -
      have D.dom aD[{t1}, {t2}, {t3}] = D.cod aD[{t1}, {t2}, {t3}]
      using t123 D.strict-assoc by simp
      also have ... = {t1} ⊗D {t2} ⊗D {t3}
      using t123 by simp
      finally show ?thesis by blast
    qed
  thus ?thesis
    using t123 D.strict-assoc D.comp-arr-dom by auto
  qed
  finally have {a-1[t1, t2, t3]} = {t1} ⊗D {t2} ⊗D {t3} by blast
  thus ?thesis using t123 by auto
  qed
  qed

```

**lemma** *ide-eval-can*:

**assumes**  $\mathcal{F}C.can\ f$

**shows**  $D.ide\ (E.map\ f)$

**proof** –

have  $f = \mathcal{F}C.mkarr\ (\mathcal{F}C.rep\ f)$

using *assms*  $\mathcal{F}C.can-implies-arr\ \mathcal{F}C.mkarr-rep$  by *blast*

**moreover** have 1:  $Can\ (\mathcal{F}C.rep\ f)$

using *assms*  $\mathcal{F}C.Can-rep-can$  by *simp*

**moreover** have  $D.ide\ \{\mathcal{F}C.rep\ f\}$

using *assms* *ide-eval-Can* by (*simp* *add: 1*)

**ultimately** show ?thesis using *assms*  $\mathcal{F}C.can-implies-arr\ E.map-def$  by *force*

**qed**

Diagonalization transports formal arrows naturally along reductions, which are canonical terms and therefore evaluate to identities of  $D$ . It follows that the evaluation in  $D$  of a formal arrow is equal to the evaluation of its diagonalization.

**lemma** *map-diagonalize*:

**assumes**  $f: \mathcal{F}C.arr\ f$

**shows**  $E.map\ (\mathcal{F}C.D\ f) = E.map\ f$

**proof** –

**interpret** *EQ*: *equivalence-of-categories*

$\mathcal{F}_S C.comp\ \mathcal{F}C.comp\ \mathcal{F}C.D\ \mathcal{F}_S C.map\ \mathcal{F}C.\nu\ \mathcal{F}C.\mu$

using  $\mathcal{F}C.is-equivalent-to-strict-subcategory$  by *auto*

have 1:  $\mathcal{F}C.seq\ (\mathcal{F}_S C.map\ (\mathcal{F}C.D\ f))\ (\mathcal{F}C.\nu\ (\mathcal{F}C.dom\ f))$

**proof**

show  $\langle\langle \mathcal{F}C.\nu\ (\mathcal{F}C.dom\ f) : \mathcal{F}C.dom\ f \rightarrow \mathcal{F}C.D\ (\mathcal{F}C.dom\ f) \rangle\rangle$

using  $f\ \mathcal{F}_S C.map-simp\ EQ.F.preserves-arr$

by (*intro*  $\mathcal{F}C.in-homI$ , *simp-all*)

show  $\langle\langle \mathcal{F}_S C.map\ (\mathcal{F}C.D\ f) : \mathcal{F}C.D\ (\mathcal{F}C.dom\ f) \rightarrow \mathcal{F}C.cod\ (\mathcal{F}C.D\ f) \rangle\rangle$

```

    using f  $\mathcal{F}_S C$ .map-simp  $\mathcal{F}C$ .arr-iff-in-hom  $EQ.F$ .preserves-hom  $\mathcal{F}_S C$ .arr-char
       $\mathcal{F}_S C$ .in-hom-char [of  $\mathcal{F}C.D$  f  $\mathcal{F}C.D$  ( $\mathcal{F}C.dom$  f)  $\mathcal{F}C.D$  ( $\mathcal{F}C.cod$  f)]
    by (intro  $\mathcal{F}C.in-homI$ , auto)
qed
have  $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.cod$  f))  $\cdot_D$   $E.map$  f =
   $E.map$  ( $\mathcal{F}C.D$  f)  $\cdot_D$   $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.dom$  f))
proof -
  have  $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.cod$  f))  $\cdot_D$   $E.map$  f =  $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.cod$  f)  $\cdot$  f)
    using f by simp
  also have ... =  $E.map$  ( $\mathcal{F}C.D$  f  $\cdot$   $\mathcal{F}C.\nu$  ( $\mathcal{F}C.dom$  f))
    using f  $EQ.\eta$ .naturality  $\mathcal{F}_S C$ .map-simp  $EQ.F$ .preserves-arr by simp
  also have ... =  $E.map$  ( $\mathcal{F}_S C$ .map ( $\mathcal{F}C.D$  f))  $\cdot_D$   $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.dom$  f))
    using f 1  $E.as-nat-trans$ .preserves-comp-2  $EQ.F$ .preserves-arr  $\mathcal{F}_S C$ .map-simp
    by (metis (no-types, lifting))
  also have ... =  $E.map$  ( $\mathcal{F}C.D$  f)  $\cdot_D$   $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.dom$  f))
    using f  $EQ.F$ .preserves-arr  $\mathcal{F}_S C$ .map-simp by simp
  finally show ?thesis by blast
qed
moreover have  $\bigwedge a. \mathcal{F}C.ide$  a  $\implies D.ide$  ( $E.map$  ( $\mathcal{F}C.\nu$  a))
  using  $\mathcal{F}C.\nu$ -def  $\mathcal{F}C$ .Arr-rep Arr-implies-Ide-Cod Can-red  $\mathcal{F}C$ .can-mkarr-Can
    ide-eval-can
  by (metis (no-types, lifting)  $EQ.\eta$ .preserves-reflects-arr  $\mathcal{F}C.seqE$ 
     $\mathcal{F}C.comp$ -preserves-can  $\mathcal{F}C.ideD(1)$   $\mathcal{F}C.ide$ -implies-can)
moreover have  $D.cod$  ( $E.map$  f) =  $D.dom$  ( $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.cod$  f)))
  using f  $E$ .preserves-hom  $EQ.\eta$ .preserves-hom by simp
moreover have  $D.dom$  ( $E.map$  ( $\mathcal{F}C.D$  f)) =  $D.cod$  ( $E.map$  ( $\mathcal{F}C.\nu$  ( $\mathcal{F}C.dom$  f)))
  using f 1  $E$ .preserves-seq  $EQ.F$ .preserves-arr  $\mathcal{F}_S C$ .map-simp by auto
ultimately show ?thesis
  using f  $D.comp$ -arr-dom  $D.ideD$   $D.arr$ -dom-iff-arr  $E.as-nat-trans$ .is-natural-2
  by (metis  $E$ .preserves-cod  $\mathcal{F}C.ide$ -cod  $\mathcal{F}C.ide$ -dom)
qed

lemma strictly-preserves-tensor:
  assumes  $\mathcal{F}_S C$ .arr f and  $\mathcal{F}_S C$ .arr g
  shows  $map$  ( $\mathcal{F}_S C$ .tensor f g) =  $map$  f  $\otimes_D$   $map$  g
proof -
  have 1:  $\mathcal{F}C$ .arr (f  $\otimes$  g)
    using assms  $\mathcal{F}_S C$ .arr-char  $\mathcal{F}C$ .tensor-in-hom by auto
  have 2:  $\mathcal{F}_S C$ .arr ( $\mathcal{F}_S C$ .tensor f g)
    using assms  $\mathcal{F}_S C$ .tensor-in-hom [of f g]  $\mathcal{F}_S C$ .T-simp by fastforce
  have  $map$  ( $\mathcal{F}_S C$ .tensor f g) =  $E.map$  (f  $\otimes$  g)
proof -
  have  $map$  ( $\mathcal{F}_S C$ .tensor f g) =  $map$  (f  $\otimes_S$  g)
    using assms  $\mathcal{F}_S C$ .T-simp by simp
  also have ... =  $map$  ( $\mathcal{F}C.D$  (f  $\otimes$  g))
    using assms  $\mathcal{F}C$ .tensorFMC-def  $\mathcal{F}_S C$ .tensorS-def  $\mathcal{F}_S C$ .arr-char by force
  also have ... =  $E.map$  (f  $\otimes$  g)
proof -
  interpret Diag: functor  $\mathcal{F}C$ .comp  $\mathcal{F}_S C$ .comp  $\mathcal{F}C.D$ 

```

```

    using FC.diagonalize-is-functor by auto
  show ?thesis
    using assms 1 map-diagonalize [of f ⊗ g] Diag.preserves-arr map-def by simp
qed
finally show ?thesis by blast
qed
thus ?thesis
  using assms  $\mathcal{F}_S C$ .arr-char E.strictly-preserves-tensor map-def by simp
qed

```

**lemma** *is-strict-monoidal-functor*:

**shows** *strict-monoidal-functor  $\mathcal{F}_S C$ .comp  $\mathcal{F}_S C$ . $T_{FSMC}$   $\mathcal{F}_S C$ . $\alpha$   $\mathcal{F}_S C$ . $\iota$   $D$   $T_D$   $\alpha_D$   $\iota_D$  map*

**proof**

show  $\bigwedge f g. \mathcal{F}_S C$ .arr  $f \implies \mathcal{F}_S C$ .arr  $g \implies \text{map } (\mathcal{F}_S C$ .tensor  $f g) = \text{map } f \otimes_D \text{map } g$

using *strictly-preserves-tensor* by fast

show  $\text{map } \mathcal{F}_S C$ . $\iota = \iota_D$

using  *$\mathcal{F}_S C$ .arr-unity  $\mathcal{F}_S C$ . $\iota$ -def map-def E.map-def  $\mathcal{F}_C$ .rep-mkarr E.eval-norm  $D$ .strict-unit*  
by auto

fix  $a b c$

assume  $a: \mathcal{F}_S C$ .ide  $a$  and  $b: \mathcal{F}_S C$ .ide  $b$  and  $c: \mathcal{F}_S C$ .ide  $c$

show  $\text{map } (\mathcal{F}_S C$ .assoc  $a b c) = \text{a}_D[\text{map } a, \text{map } b, \text{map } c]$

**proof** –

have  $\text{map } (\mathcal{F}_S C$ .assoc  $a b c) = \text{map } a \otimes_D \text{map } b \otimes_D \text{map } c$

using  $a b c$   *$\mathcal{F}_S C$ . $\alpha$ -def  $\mathcal{F}_S C$ .assoc<sub>S</sub>-def  $\mathcal{F}_S C$ .arr-tensor  $\mathcal{F}_S C$ . $T$ -simp  $\mathcal{F}_S C$ .ide $D(1)$*   
*strictly-preserves-tensor  $\mathcal{F}_S C$ . $\alpha$ -ide-simp*

by presburger

also have  $\dots = \text{a}_D[\text{map } a, \text{map } b, \text{map } c]$

using  $a b c$   *$D$ .strict-assoc  $D$ .assoc-in-hom [of map  $a$  map  $b$  map  $c$ ]* by auto

finally show ?thesis by blast

qed

qed

end

**sublocale** *strict-evaluation-functor*  $\subseteq$

*strict-monoidal-functor  $\mathcal{F}_S C$ .comp  $\mathcal{F}_S C$ . $T_{FSMC}$   $\mathcal{F}_S C$ . $\alpha$   $\mathcal{F}_S C$ . $\iota$   $D$   $T_D$   $\alpha_D$   $\iota_D$  map*

using *is-strict-monoidal-functor* by auto

**locale** *strict-monoidal-extension-to-free-strict-monoidal-category* =

*C*: category  $C$  +

*monoidal-language*  $C$  +

*$\mathcal{F}_S C$* : free-strict-monoidal-category  $C$  +

*strict-monoidal-extension*  $C$   *$\mathcal{F}_S C$ .comp  $\mathcal{F}_S C$ . $T_{FSMC}$   $\mathcal{F}_S C$ . $\alpha$   $\mathcal{F}_S C$ . $\iota$   $D$   $T_D$   $\alpha_D$   $\iota_D$*   
 *$\mathcal{F}_S C$ .inclusion-of-generators  $V F$*

**for**  $C :: 'c$  comp (infixr  $\cdot_C$  55)

**and**  $D :: 'd$  comp (infixr  $\cdot_D$  55)

**and**  $T_D :: 'd * 'd \Rightarrow 'd$

**and**  $\alpha_D :: 'd * 'd * 'd \Rightarrow 'd$

**and**  $\iota_D :: 'd$

```

and  $V :: 'c \Rightarrow 'd$ 
and  $F :: 'c \text{ free-monoidal-category.arr} \Rightarrow 'd$ 

sublocale strict-evaluation-functor  $\subseteq$ 
  strict-monoidal-extension  $C \mathcal{F}_S C.comp \mathcal{F}_S C.T_{FSMC} \mathcal{F}_S C.\alpha \mathcal{F}_S C.\iota D T_D \alpha_D \iota_D$ 
   $\mathcal{F}_S C.inclusion-of-generators$   $V \text{ map}$ 

proof –
  interpret  $V$ : functor  $C \mathcal{F}_S C.comp \mathcal{F}_S C.inclusion-of-generators$ 
  using  $\mathcal{F}_S C.inclusion-is-functor$  by auto
  show strict-monoidal-extension  $C \mathcal{F}_S C.comp \mathcal{F}_S C.T_{FSMC} \mathcal{F}_S C.\alpha \mathcal{F}_S C.\iota D T_D \alpha_D \iota_D$ 
   $\mathcal{F}_S C.inclusion-of-generators$   $V \text{ map}$ 

  proof
  show  $\forall f. C.arr f \longrightarrow \text{map } (\mathcal{F}_S C.inclusion-of-generators f) = V f$ 
  using  $V.preserves-arr$   $E.is-extension$  map-def  $\mathcal{F}_S C.inclusion-of-generators-def$  by simp
  qed
qed

context free-strict-monoidal-category
begin

```

We now have the main result of this section: the evaluation functor on  $\mathcal{F}_S C$  induced by a functor  $V$  from  $C$  to a strict monoidal category  $D$  is the unique strict monoidal extension of  $V$  to  $\mathcal{F}_S C$ .

```

theorem is-free:
assumes strict-monoidal-category  $D T_D \alpha_D \iota_D$ 
and strict-monoidal-extension-to-free-strict-monoidal-category  $C D T_D \alpha_D \iota_D V F$ 
shows  $F = \text{strict-evaluation-functor.map } C D T_D \alpha_D \iota_D V$ 
proof –
  interpret  $D$ : strict-monoidal-category  $D T_D \alpha_D \iota_D$ 
  using assms(1) by auto

```

Let  $F$  be a given extension of  $V$  to a strict monoidal functor defined on  $\mathcal{F}_S C$ .

```

interpret  $F$ : strict-monoidal-extension-to-free-strict-monoidal-category
   $C D T_D \alpha_D \iota_D V F$ 
using assms(2) by auto

```

Let  $E_S$  be the evaluation functor from  $\mathcal{F}_S C$  to  $D$  induced by  $V$ . Then  $E_S$  is also a strict monoidal extension of  $V$ .

```

interpret  $E_S$ : strict-evaluation-functor  $C D T_D \alpha_D \iota_D V ..$ 

```

Let  $D$  be the strict monoidal functor  $\mathcal{F}C.D$  that projects  $\mathcal{F}C$  to the subcategory  $\mathcal{F}_S C$ .

```

interpret  $D$ : functor  $\mathcal{F}C.comp \text{comp } \mathcal{F}C.D$ 
using  $\mathcal{F}C.diagonalize-is-functor$  by auto
interpret  $D$ : strict-monoidal-functor  $\mathcal{F}C.comp \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha \mathcal{F}C.\iota$ 
  comp  $T_{FSMC} \alpha \iota$ 
   $\mathcal{F}C.D$ 
using diagonalize-is-strict-monoidal-functor by blast

```

The composite functor  $F \circ D$  is also an extension of  $V$  to a strict monoidal functor on  $\mathcal{F}C$ .

```

interpret FoD: composite-functor  $\mathcal{F}C.comp \text{ comp } D \mathcal{F}C.D F ..$ 
interpret FoD: strict-monoidal-functor
            $\mathcal{F}C.comp \mathcal{F}C.T_{FMC} \mathcal{F}C.\alpha \mathcal{F}C.\iota D T_D \alpha_D \iota_D \langle F \circ \mathcal{F}C.D \rangle$ 
using  $D.strict\text{-monoidal-functor-axioms } F.strict\text{-monoidal-functor-axioms}$ 
            $strict\text{-monoidal-functors-compose}$ 
by fast
interpret FoD: strict-monoidal-extension-to-free-monoidal-category
            $C D T_D \alpha_D \iota_D V FoD.map$ 
proof
show  $\forall f. C.arr f \longrightarrow FoD.map (FC.inclusion\text{-of-generators } f) = V f$ 
proof –
  have  $\bigwedge f. C.arr f \implies FoD.map (FC.inclusion\text{-of-generators } f) = V f$ 
proof –
  fix  $f$ 
  assume  $f: C.arr f$ 
  have  $FoD.map (FC.inclusion\text{-of-generators } f)$ 
         $= F (FC.D (FC.inclusion\text{-of-generators } f))$ 
  using  $f$  by simp
  also have  $... = F (inclusion\text{-of-generators } f)$ 
  using  $f$   $\mathcal{F}C.strict\text{-arr-char' } F.I.preserves\text{-arr } inclusion\text{-of-generators-def}$  by simp
  also have  $... = V f$ 
  using  $f$   $F.is\text{-extension}$  by simp
  finally show  $FoD.map (FC.inclusion\text{-of-generators } f) = V f$ 
  by blast
qed
thus ?thesis by blast
qed
qed

```

By the freeness of  $\mathcal{F}C$ , we have that  $F \circ D$  is equal to the evaluation functor  $E_S.E.map$  induced by  $V$  on  $\mathcal{F}C$ . Moreover,  $E_S.map$  coincides with  $E_S.E.map$  on  $\mathcal{F}_S C$  and  $F \circ D$  coincides with  $F$  on  $\mathcal{F}_S C$ . Therefore,  $F$  coincides with  $E$  on their common domain  $\mathcal{F}_S C$ , showing  $F = E_S.map$ .

```

have  $\bigwedge f. arr f \implies F f = E_S.map f$ 
using  $\mathcal{F}C.strict\text{-arr-char' } \mathcal{F}C.is\text{-free [of } D] E_S.E.evaluation\text{-functor-axioms}$ 
            $FoD.strict\text{-monoidal-extension-to-free-monoidal-category-axioms } E_S.map\text{-def}$ 
by simp
moreover have  $\bigwedge f. \neg arr f \implies F f = E_S.map f$ 
using  $F.is\text{-extensional } E_S.is\text{-extensional } arr\text{-char}$  by auto
ultimately show  $F = E_S.map$  by blast
qed

```

end

end

## Chapter 5

# Cartesian Monoidal Category

```
theory CartesianMonoidalCategory
imports MonoidalCategory Category3.CartesianCategory
begin

  locale symmetric-monoidal-category =
    monoidal-category C T  $\alpha$   $\iota$  +
    S: symmetry-functor C C +
    ToS: composite-functor CC.comp CC.comp C S.map T +
     $\sigma$ : natural-isomorphism CC.comp C T ToS.map  $\sigma$ 
  for C :: 'a comp (infixr · 55)
  and T :: 'a * 'a  $\Rightarrow$  'a
  and  $\alpha$  :: 'a * 'a * 'a  $\Rightarrow$  'a
  and  $\iota$  :: 'a
  and  $\sigma$  :: 'a * 'a  $\Rightarrow$  'a +
  assumes sym-inverse:  $\llbracket$  ide a; ide b  $\rrbracket \Longrightarrow$  inverse-arrows ( $\sigma$  (a, b)) ( $\sigma$  (b, a))
  and unitor-coherence: ide a  $\Longrightarrow$  l[a] ·  $\sigma$  (a,  $\mathcal{I}$ ) = r[a]
  and assoc-coherence:  $\llbracket$  ide a; ide b; ide c  $\rrbracket \Longrightarrow$ 
     $\alpha$  (b, c, a) ·  $\sigma$  (a, b  $\otimes$  c) ·  $\alpha$  (a, b, c)
    = (b  $\otimes$   $\sigma$  (a, c)) ·  $\alpha$  (b, a, c) · ( $\sigma$  (a, b)  $\otimes$  c)

  begin

    abbreviation sym (s[-, -])
    where sym a b  $\equiv$   $\sigma$  (a, b)

  end

  locale elementary-symmetric-monoidal-category =
    elementary-monoidal-category C tensor unity lunit runit assoc
  for C :: 'a comp (infixr · 55)
  and tensor :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixr  $\otimes$  53)
  and unity :: 'a ( $\mathcal{I}$ )
  and lunit :: 'a  $\Rightarrow$  'a (l[-])
  and runit :: 'a  $\Rightarrow$  'a (r[-])
  and assoc :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a (a[-, -, -])
```

**and** *sym* :: 'a ⇒ 'a ⇒ 'a      (s[-, -]) +  
**assumes** *sym-in-hom*: [ ide a; ide b ] ⇒ «s[a, b] : a ⊗ b → b ⊗ a»  
**and** *sym-naturality*: [ arr f; arr g ] ⇒ s[cod f, cod g] · (f ⊗ g) = (g ⊗ f) · s[dom f, dom g]  
**and** *sym-inverse*: [ ide a; ide b ] ⇒ *inverse-arrows* s[a, b] s[b, a]  
**and** *unit-or-coherence*: ide a ⇒ l[a] · s[a,  $\mathcal{I}$ ] = r[a]  
**and** *assoc-coherence*: [ ide a; ide b; ide c ] ⇒  
     a[b, c, a] · s[a, b ⊗ c] · a[a, b, c]  
     = (b ⊗ s[a, c]) · a[b, a, c] · (s[a, b] ⊗ c)

**begin**

**lemma** *sym-simps* [*simp*]:  
**assumes** *ide a* **and** *ide b*  
**shows** *arr* s[a, b]  
**and** *dom* s[a, b] = a ⊗ b  
**and** *cod* s[a, b] = b ⊗ a  
**using** *assms sym-in-hom* **by** *auto*

**interpretation** *monoidal-category* C T α ι  
**using** *induces-monoidal-category* **by** *simp*  
**interpretation** *CC*: *product-category* C C ..

**interpretation** *S*: *symmetry-functor* C C ..  
**interpretation** *ToS*: *composite-functor* CC.comp CC.comp C S.map T ..

**definition** σ :: 'a \* 'a ⇒ 'a  
**where** σ f ≡ if CC.arr f then s[cod (fst f), cod (snd f)] · (fst f ⊗ snd f) else null

**interpretation** σ: *natural-isomorphism* CC.comp C T ToS.map σ

**proof** –

**interpret** σ: *transformation-by-components* CC.comp C T ToS.map λa. s[fst a, snd a]  
**apply** *unfold-locales*  
**using** *sym-in-hom sym-naturality* **by** *auto*  
**interpret** σ: *natural-isomorphism* CC.comp C T ToS.map σ.map  
**apply** *unfold-locales*  
**using** *sym-inverse σ.map-simp-ide* **by** *auto*  
**have** σ = σ.map  
**using** σ-def σ.map-def *sym-naturality* **by** *fastforce*  
**thus** *natural-isomorphism* CC.comp C T ToS.map σ  
**using** σ.*natural-isomorphism-axioms* **by** *presburger*

**qed**

**interpretation** *symmetric-monoidal-category* C T α ι σ

**proof**

**show** ∧ a b. [ ide a; ide b ] ⇒ *inverse-arrows* (σ (a, b)) (σ (b, a))

**proof** –

**fix** a b

**assume** a: *ide a* **and** b: *ide b*

**show** *inverse-arrows* (σ (a, b)) (σ (b, a))

**using** a b *sym-inverse comp-arr-dom σ-def* **by** *auto*



qed

**show**  $\bigwedge a. \text{ide } a \implies \text{local.lunit } a \cdot \sigma (a, \text{local.unity}) = \text{local.runit } a$

**proof** –

**fix**  $a$

**assume**  $a: \text{ide } a$

**show**  $\text{local.lunit } a \cdot \sigma (a, \text{local.unity}) = \text{local.runit } a$

**using**  $a$  *lunit-agreement* *I-agreement* *sym-in-hom* *comp-arr-dom* [of  $s[a, \mathcal{I}]$ ]  
*unit-coherence* *runit-agreement*  $\sigma$ -def

**by** *simp*

qed

**show**  $\bigwedge a b c. \llbracket \text{ide } a; \text{ide } b; \text{ide } c \rrbracket \implies$

$\text{local.assoc } b c a \cdot \sigma (a, \text{local.tensor } b c) \cdot \text{local.assoc } a b c$

$= \text{local.tensor } b (\sigma (a, c)) \cdot \text{local.assoc } b a c \cdot \text{local.tensor } (\sigma (a, b)) c$

**proof** –

**fix**  $a b c$

**assume**  $a: \text{ide } a$  **and**  $b: \text{ide } b$  **and**  $c: \text{ide } c$

**show**  $\text{local.assoc } b c a \cdot \sigma (a, \text{local.tensor } b c) \cdot \text{local.assoc } a b c$

$= \text{local.tensor } b (\sigma (a, c)) \cdot \text{local.assoc } b a c \cdot \text{local.tensor } (\sigma (a, b)) c$

**using**  $a b c$  *sym-in-hom* *tensor-preserves-ide*  $\sigma$ -def *assoc-coherence*  
*comp-arr-dom* *comp-cod-arr*

**by** *simp*

qed

qed

**lemma** *induces-symmetric-monoidal-category*:

**shows** *symmetric-monoidal-category*  $C T \alpha \iota \sigma$

..

end

**context** *symmetric-monoidal-category*

**begin**

**interpretation** *elementary-monoidal-category*  $C$  *tensor* *unity* *lunit* *runit* *assoc*

**using** *induces-elementary-monoidal-category* **by** *auto*

**lemma** *induces-elementary-symmetric-monoidal-category*:

**shows** *elementary-symmetric-monoidal-category*

$C$  *tensor* *unity* *lunit* *runit* *assoc*  $(\lambda a b. \sigma (a, b))$

**using**  $\sigma$ .*naturality* *unit-coherence* *assoc-coherence* *sym-inverse*

**by** *unfold-locales* *auto*

end

**locale** *diagonal-functor* =

$C$ : *category*  $C$  +

$CC$ : *product-category*  $C C$

```

for  $C :: 'a \text{ comp}$ 
begin

  abbreviation  $\text{map}$ 
  where  $\text{map } f \equiv \text{if } C.\text{arr } f \text{ then } (f, f) \text{ else } CC.\text{null}$ 

  lemma  $\text{is-functor}$ :
  shows  $\text{functor } C \text{ } CC.\text{comp } \text{map}$ 
  using  $\text{map-def by unfold-locales auto}$ 

  sublocale  $\text{functor } C \text{ } CC.\text{comp } \text{map}$ 
  using  $\text{is-functor by simp}$ 

end

locale  $\text{cartesian-monoidal-category} =$ 
   $\text{monoidal-category } C \text{ } T \text{ } \alpha \text{ } \iota \text{ } +$ 
   $\Omega$ :  $\text{constant-functor } C \text{ } C \text{ } \mathcal{I} \text{ } +$ 
   $\Delta$ :  $\text{diagonal-functor } C \text{ } +$ 
   $\tau$ :  $\text{natural-transformation } C \text{ } C \text{ } \text{map } \Omega.\text{map } \tau \text{ } +$ 
   $\delta$ :  $\text{natural-transformation } C \text{ } C \text{ } \text{map } \langle T \circ \Delta.\text{map} \rangle \delta$ 
for  $C :: 'a \text{ comp}$  (infixr · 55)
and  $T :: 'a * 'a \Rightarrow 'a$ 
and  $\alpha :: 'a * 'a * 'a \Rightarrow 'a$ 
and  $\iota :: 'a$ 
and  $\delta :: 'a \Rightarrow 'a$  (d[-])
and  $\tau :: 'a \Rightarrow 'a$  (t[-]) +
assumes  $\text{trm-unity: } t[\mathcal{I}] = \mathcal{I}$ 
and  $\text{pr0-dup: } \text{ide } a \Longrightarrow r[a] \cdot (a \otimes t[a]) \cdot \delta \ a = a$ 
and  $\text{pr1-dup: } \text{ide } a \Longrightarrow l[a] \cdot (t[a] \otimes a) \cdot \delta \ a = a$ 
and  $\text{tuple-pr: } \llbracket \text{ide } a; \text{ide } b \rrbracket \Longrightarrow (r[a] \cdot (a \otimes t[b]) \otimes l[b] \cdot (t[a] \otimes b)) \cdot d[a \otimes b] = a \otimes b$ 

locale  $\text{elementary-cartesian-monoidal-category} =$ 
   $\text{elementary-monoidal-category } C \text{ } \text{tensor } \text{unity } \text{lunit } \text{runit } \text{assoc}$ 
for  $C :: 'a \text{ comp}$  (infixr · 55)
and  $\text{tensor} :: 'a \Rightarrow 'a \Rightarrow 'a$  (infixr  $\otimes$  53)
and  $\text{unity} :: 'a$  ( $\mathcal{I}$ )
and  $\text{lunit} :: 'a \Rightarrow 'a$  (l[-])
and  $\text{runit} :: 'a \Rightarrow 'a$  (r[-])
and  $\text{assoc} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$  (a[-, -, -])
and  $\text{trm} :: 'a \Rightarrow 'a$  (t[-])
and  $\text{dup} :: 'a \Rightarrow 'a$  (d[-]) +
assumes  $\text{trm-in-hom [intro]: } \text{ide } a \Longrightarrow \langle t[a] : a \rightarrow \mathcal{I} \rangle$ 
and  $\text{trm-unity: } t[\mathcal{I}] = \mathcal{I}$ 
and  $\text{trm-naturality: } \text{arr } f \Longrightarrow t[\text{cod } f] \cdot f = t[\text{dom } f]$ 
and  $\text{dup-in-hom [intro]: } \text{ide } a \Longrightarrow \langle d[a] : a \rightarrow a \otimes a \rangle$ 
and  $\text{dup-naturality: } \text{arr } f \Longrightarrow d[\text{cod } f] \cdot f = (f \otimes f) \cdot d[\text{dom } f]$ 
and  $\text{prj0-dup: } \text{ide } a \Longrightarrow r[a] \cdot (a \otimes t[a]) \cdot d[a] = a$ 
and  $\text{prj1-dup: } \text{ide } a \Longrightarrow l[a] \cdot (t[a] \otimes a) \cdot d[a] = a$ 

```

**and** *tuple-prj*:  $\llbracket \text{ide } a; \text{ide } b \rrbracket \implies (r[a] \cdot (a \otimes t[b]) \otimes l[b] \cdot (t[a] \otimes b)) \cdot d[a \otimes b] = a \otimes b$

**context** *cartesian-monoidal-category*

**begin**

**lemma** *terminal-unity*:

**shows** *terminal*  $\mathcal{I}$

**proof**

**show** *ide*  $\mathcal{I}$  **by** *simp*

**show**  $\bigwedge a. \text{ide } a \implies \exists ! f. \langle f : a \rightarrow \mathcal{I} \rangle$

**proof**

**fix**  $a$

**assume**  $a: \text{ide } a$

**show**  $\langle \tau a : a \rightarrow \mathcal{I} \rangle$  **using**  $a$  **by** *auto*

**show**  $\bigwedge f. \langle f : a \rightarrow \mathcal{I} \rangle \implies f = \tau a$

**using** *trm-unity*  $\tau$ .*naturality* *comp-cod-arr* **by** *fastforce*

**qed**

**qed**

**lemma** *trm-is-terminal-arr*:

**assumes** *ide*  $a$

**shows** *terminal-arr*  $t[a]$

**using** *assms* *terminal-unity* **by** *simp*

**interpretation** *elementary-monoidal-category*  $C$  *tensor* *unity* *lunit* *runit* *assoc*

**using** *induces-elementary-monoidal-category* **by** *simp*

**interpretation** *elementary-cartesian-monoidal-category*  $C$  *tensor* *unity* *lunit* *runit* *assoc*  $\tau$   $\delta$

**proof**

**show**  $\bigwedge a. \text{ide } a \implies \langle t[a] : a \rightarrow \mathcal{I} \rangle$

**using** *ι-in-hom* **by** *force*

**show**  $t[\mathcal{I}] = \mathcal{I}$

**using**  $\tau$ .*preserves-hom* *ι-in-hom* *ide-unity* *trm-is-terminal-arr* *terminal-unity*

**by** (*intro* *terminal-arr-unique*) *auto*

**show**  $\bigwedge f. \text{arr } f \implies t[\text{cod } f] \cdot f = t[\text{dom } f]$

**using**  $\tau$ .*naturality* *comp-cod-arr* **by** *simp*

**show**  $\bigwedge a. \text{ide } a \implies \langle d[a] : a \rightarrow a \otimes a \rangle$

**by** *auto*

**show**  $\bigwedge f. \text{arr } f \implies d[\text{cod } f] \cdot f = (f \otimes f) \cdot d[\text{dom } f]$

**using**  $\delta$ .*naturality* **by** *simp*

**show**  $\bigwedge a. \text{ide } a \implies l[a] \cdot (t[a] \otimes a) \cdot d[a] = a$

**using** *pr1-dup* *lunit-in-hom* **by** *simp*

**show**  $\bigwedge a. \text{ide } a \implies r[a] \cdot (a \otimes t[a]) \cdot d[a] = a$

**using** *pr0-dup* *runit-in-hom* **by** *simp*

**show**  $\bigwedge a0 a1. \llbracket \text{ide } a0; \text{ide } a1 \rrbracket \implies$

$$(r[a0] \cdot (a0 \otimes t[a1]) \otimes l[a1] \cdot (t[a0] \otimes a1)) \cdot d[a0 \otimes a1] \\ = a0 \otimes a1$$

**using** *tuple-pr* **by** *simp*

**qed**

**lemma** *induces-elementary-cartesian-monoidal-category*:  
**shows** *elementary-cartesian-monoidal-category C tensor I lunit runit assoc τ δ*  
 ..

**end**

**context** *elementary-cartesian-monoidal-category*  
**begin**

**lemma** *trm-simps [simp]*:  
**assumes** *ide a*  
**shows** *arr t[a] and dom t[a] = a and cod t[a] = I*  
**using** *assms trm-in-hom by auto*

**lemma** *dup-simps [simp]*:  
**assumes** *ide a*  
**shows** *arr d[a] and dom d[a] = a and cod d[a] = a ⊗ a*  
**using** *assms dup-in-hom by auto*

**definition**  $\tau :: 'a \Rightarrow 'a$   
**where**  $\tau f \equiv \text{if arr } f \text{ then } t[\text{dom } f] \text{ else null}$

**definition**  $\delta :: 'a \Rightarrow 'a$   
**where**  $\delta f \equiv \text{if arr } f \text{ then } d[\text{cod } f] \cdot f \text{ else null}$

**interpretation** *CC: product-category C C ..*

**interpretation** *MC: monoidal-category C T α ι*  
**using** *induces-monoidal-category by auto*

**interpretation** *I: constant-functor C C MC.unify*  
**by** *unfold-locales auto*

**interpretation**  $\Delta$ : *diagonal-functor C ..*

**interpretation** *D: composite-functor C CC.comp C Δ.map T ..*

**interpretation**  $\tau$ : *natural-transformation C C map I.map τ*  
**using** *trm-naturality I.map-def τ-def I-agreement comp-cod-arr*  
**by** *unfold-locales auto*

**interpretation**  $\delta$ : *natural-transformation C C map D.map δ*  
**using** *dup-naturality δ-def comp-arr-dom*  
**by** *unfold-locales auto*

**interpretation** *MC: cartesian-monoidal-category C T α ι δ τ*

**proof**

**show**  $\tau \text{ MC.unify} = \text{MC.unify}$   
**using** *I-agreement trm-unify τ-def by simp*  
**show**  $\bigwedge a. \text{ide } a \implies \text{MC.runit } a \cdot \text{MC.tensor } a (\tau a) \cdot \delta a = a$   
**using** *runit-agreement τ-def δ-def prj0-dup comp-arr-dom by auto*  
**show**  $\bigwedge a. \text{ide } a \implies \text{MC.lunit } a \cdot \text{MC.tensor } (\tau a) a \cdot \delta a = a$

```

using lunit-agreement  $\tau$ -def  $\delta$ -def prj1-dup comp-arr-dom by auto
show  $\bigwedge a b. \llbracket \text{ide } a; \text{ide } b \rrbracket \implies$ 
       $MC.tensor (MC.runit a \cdot MC.tensor a (\tau b)) (MC.lunit b \cdot MC.tensor (\tau a) b) \cdot$ 
       $\delta (MC.tensor a b) = MC.tensor a b$ 
proof –
  fix a b
  assume a: ide a and b: ide b
  have seq r[a] (a  $\otimes$  t[b])
    by (metis a b arr-tensor cod-tensor ide-char in-homE runit-in-hom seqI trm-simps(1,3))
  moreover have seq l[b] (t[a]  $\otimes$  b)
    by (metis a b arr-tensor cod-tensor ide-char in-homE lunit-in-hom seqI trm-simps(1,3))
  ultimately show  $MC.tensor (MC.runit a \cdot MC.tensor a (\tau b))$ 
       $(MC.lunit b \cdot MC.tensor (\tau a) b) \cdot \delta (MC.tensor a b) =$ 
       $MC.tensor a b$ 
  using a b lunit-agreement runit-agreement unitor-coincidence  $\tau$ -def  $\delta$ -def comp-arr-dom
      tensor-preserves-ide tuple-prj T-def
  by auto
qed
qed

```

```

lemma induces-cartesian-monoidal-category:
shows cartesian-monoidal-category C T  $\alpha$   $\iota$   $\delta$   $\tau$ 
  ..

```

**end**

A cartesian category extends to a cartesian monoidal category by using the product structure to obtain the various canonical maps.

```

context cartesian-category
begin

```

```

interpretation C: elementary-cartesian-category C pr0 pr1 1 trm
  using extends-to-elementary-cartesian-category by simp
interpretation CC: product-category C C ..
interpretation CCC: product-category C CC.comp ..
interpretation T: binary-functor C C C Prod
  using binary-functor-Prod by simp
interpretation T: binary-endofunctor C Prod ..
interpretation ToTC: functor CCC.comp C T.ToTC
  using T.functor-ToTC by auto
interpretation ToCT: functor CCC.comp C T.ToCT
  using T.functor-ToCT by auto

interpretation  $\alpha$ : transformation-by-components CCC.comp C T.ToTC T.ToCT
       $\langle \lambda abc. \text{assoc } (fst\ abc) (fst\ (snd\ abc)) (snd\ (snd\ abc)) \rangle$ 

```

```

proof
show  $\bigwedge abc. CCC.ide\ abc \implies$ 
       $\langle \text{assoc } (fst\ abc) (fst\ (snd\ abc)) (snd\ (snd\ abc)) \rangle : T.ToTC\ abc \rightarrow T.ToCT\ abc$ 
  using CCC.ide-char CC.ide-char CCC.arr-char CC.arr-char T.ToTC-def T.ToCT-def

```

**by** *auto*  
**show**  $\bigwedge f. CCC.arr\ f \implies$   
 $assoc\ (fst\ (CCC.cod\ f))\ (fst\ (snd\ (CCC.cod\ f)))\ (snd\ (snd\ (CCC.cod\ f)))$   
 $\cdot\ T.ToTC\ f =$   
 $T.ToCT\ f$   
 $\cdot\ assoc\ (fst\ (CCC.dom\ f))\ (fst\ (snd\ (CCC.dom\ f)))\ (snd\ (snd\ (CCC.dom\ f)))$   
**using** *CCC.arr-char CC.arr-char CCC.dom-char CCC.cod-char T.ToTC-def T.ToCT-def*  
*assoc-naturality*  
**by** *simp blast*  
**qed**

**interpretation**  $\alpha$ : *natural-isomorphism CCC.comp C T.ToTC T.ToCT  $\alpha.map$*   
**proof**  
**show**  $\bigwedge a. CCC.ide\ a \implies iso\ (\alpha.map\ a)$   
**using** *CCC.ide-char CC.ide-char  $\alpha.map-simp-ide\ inverse-arrows-assoc$*  **by** *auto*  
**qed**

**interpretation**  $L$ : *functor C C  $\langle \lambda f. Prod\ (cod\ \iota, f) \rangle$*   
**using**  *$\iota$ -is-terminal-arr T.fixing-ide-gives-functor-1* **by** *simp*  
**interpretation**  $L$ : *endofunctor C  $\langle \lambda f. Prod\ (cod\ \iota, f) \rangle$  ..*  
**interpretation**  $l$ : *transformation-by-components C C*  
 $\langle \lambda f. Prod\ (cod\ \iota, f) \rangle\ map\ \langle \lambda a. pr0\ (cod\ \iota)\ a \rangle$   
**using**  *$\iota$ -is-terminal-arr*  
**by** *unfold-locales auto*  
**interpretation**  $l$ : *natural-isomorphism C C  $\langle \lambda f. Prod\ (cod\ \iota, f) \rangle\ map\ l.map$*   
**using**  *$l.map-simp-ide\ inverse-arrows-lunit\ ide-some-terminal$*   
**by** *unfold-locales auto*  
**interpretation**  $L$ : *equivalence-functor C C  $\langle \lambda f. Prod\ (cod\ \iota, f) \rangle$*   
**using**  *$l.natural-isomorphism-axioms\ naturally-isomorphic-def$*   
*L.isomorphic-to-identity-is-equivalence*  
**by** *blast*

**interpretation**  $R$ : *functor C C  $\langle \lambda f. Prod\ (f, cod\ \iota) \rangle$*   
**using**  *$\iota$ -is-terminal-arr T.fixing-ide-gives-functor-2* **by** *simp*  
**interpretation**  $R$ : *endofunctor C  $\langle \lambda f. Prod\ (f, cod\ \iota) \rangle$  ..*  
**interpretation**  $q$ : *transformation-by-components C C*  
 $\langle \lambda f. Prod\ (f, cod\ \iota) \rangle\ map\ \langle \lambda a. pr1\ a\ (cod\ \iota) \rangle$   
**using**  *$\iota$ -is-terminal-arr*  
**by** *unfold-locales auto*  
**interpretation**  $q$ : *natural-isomorphism C C  $\langle \lambda f. Prod\ (f, cod\ \iota) \rangle\ map\ q.map$*   
**using**  *$q.map-simp-ide\ inverse-arrows-runit\ ide-some-terminal$*   
**by** *unfold-locales auto*  
**interpretation**  $R$ : *equivalence-functor C C  $\langle \lambda f. Prod\ (f, cod\ \iota) \rangle$*   
**using**  *$q.natural-isomorphism-axioms\ naturally-isomorphic-def$*   
*R.isomorphic-to-identity-is-equivalence*  
**by** *blast*

**interpretation**  $C$ : *monoidal-category C Prod  $\alpha.map\ \iota$*   
**using**  *$ide-some-terminal\ \iota-is-iso\ pentagon\ comp-assoc$*

**by** *unfold-locales auto*

**interpretation**  $\Omega$ : *constant-functor C C C.unity*  
**using** *C.ide-unity*  
**by** *unfold-locales auto*

**interpretation**  $\tau$ : *natural-transformation C C map  $\Omega$ .map trm*  
**using** *C.unity-def  $\Omega$ .map-def ide-some-terminal trm-naturality comp-cod-arr trm-in-hom*  
**apply** *unfold-locales*  
**using** *trm-def*  
**apply** *auto[1]*  
**apply** *fastforce*  
**apply** *fastforce*  
**apply** *(metis in-homE trm-eqI trm-in-hom cod-pr0 dom-dom)*  
**by** *(metis trm-eqI trm-in-hom dom-dom map-simp)*

**interpretation**  $\Delta$ : *functor C CC.comp Diag*  
**using** *functor-Diag* **by** *simp*

**interpretation**  $\Pi o \Delta$ : *composite-functor C CC.comp C Diag Prod ..*

**interpretation** *natural-transformation C C map  $\langle Prod o Diag \rangle$  dup*  
**using** *dup-is-natural-transformation* **by** *simp*

**lemma** *unity-agreement*:  
**shows** *C.unity = 1*  
**using** *C.unity-def ide-some-terminal* **by** *simp*

**lemma** *assoc-agreement*:  
**assumes** *ide a and ide b and ide c*  
**shows** *C.assoc a b c = assoc a b c*  
**using** *assms assoc-def  $\alpha$ .map-simp-ide* **by** *simp*

**lemma** *assoc'-agreement*:  
**assumes** *ide a and ide b and ide c*  
**shows** *C.assoc' a b c = assoc' a b c*  
**using** *assms inverse-arrows-assoc inverse-unique* **by** *auto*

**lemma** *runit-char-eqn*:  
**assumes** *ide a*  
**shows** *prod (runit a) 1 = prod a  $\iota$  · assoc a 1 1*  
**using** *assms ide-one assoc-def comp-assoc prod-tuple comp-cod-arr*  
**by** *(intro pr-joint-monic [of a 1 prod (runit a) 1 prod a  $\iota$  · assoc a 1 1]) auto*

**lemma** *runit-agreement*:  
**assumes** *ide a*  
**shows** *runit a = C.runit a*  
**using** *assms unity-agreement assoc-agreement C.runit-char(2) runit-char-eqn*  
*ide-some-terminal*  
**by** *(intro C.runit-eqI) auto*

**lemma** *lunit-char-eqn*:  
**assumes** *ide a*  
**shows**  $\text{prod } \mathbf{1} \text{ (lunit } a) = \text{prod } \iota \ a \cdot \text{assoc}' \ \mathbf{1} \ \mathbf{1} \ a$   
**proof** (*intro pr-joint-monic [of  $\mathbf{1} \ a \ \text{prod } \mathbf{1} \ \text{(lunit } a) \ \text{prod } \iota \ a \cdot \text{assoc}' \ \mathbf{1} \ \mathbf{1} \ a]$* )  
  **show**  $\text{seq} \ \text{(lunit } a) \ \text{(local.prod } \mathbf{1} \ \text{(lunit } a))$   
  **using** *assms ide-one by simp*  
  **show**  $\text{lunit } a \cdot \text{prod } \mathbf{1} \ \text{(lunit } a) = \text{lunit } a \cdot \text{prod } \iota \ a \cdot \text{assoc}' \ \mathbf{1} \ \mathbf{1} \ a$   
  **using** *assms ide-one assoc'-def comp-assoc prod-tuple comp-cod-arr by simp*  
  **show**  $\text{pr1 } \mathbf{1} \ a \cdot \text{prod } \mathbf{1} \ \text{(lunit } a) = \text{pr1 } \mathbf{1} \ a \cdot \text{prod } \iota \ a \cdot \text{assoc}' \ \mathbf{1} \ \mathbf{1} \ a$   
  **using** *assms ide-one assoc'-def comp-cod-arr prod-tuple pr-naturality*  
  **apply** *simp*  
  **by** (*metis trm-eqI cod-pr0 cod-pr1 comp-in-homI' ide-prod pr-simps(1,3-6) pr1-in-hom'*)  
**qed**

**lemma** *lunit-agreement*:  
**assumes** *ide a*  
**shows**  $\text{lunit } a = C.\text{lunit } a$   
  **using** *assms unity-agreement assoc'-agreement C.lunit-char(2) lunit-char-eqn*  
  *ide-some-terminal*  
  **by** (*intro C.lunit-eqI auto*)

**interpretation** *C*: *cartesian-monoidal-category C Prod  $\alpha.\text{map } \iota \ \text{dup trm}$*   
**proof**

**show**  $\text{trm } C.\text{unity} = C.\text{unity}$   
  **by** (*simp add: C.unity-def ide-some-terminal trm-one*)  
  **show**  $\bigwedge a. \text{ide } a \implies C.\text{runit } a \cdot C.\text{tensor } a \ \text{t}[a] \cdot \text{dup } a = a$   
  **using** *comp-runit-term-dup runit-agreement by simp*  
  **show**  $\bigwedge a. \text{ide } a \implies C.\text{lunit } a \cdot C.\text{tensor } \text{t}[a] \ a \cdot \text{dup } a = a$   
  **using** *comp-lunit-term-dup lunit-agreement by auto*  
  **show**  $\bigwedge a \ b. [\text{ide } a; \text{ide } b]$   
   $\implies C.\text{tensor} \ (C.\text{runit } a \cdot C.\text{tensor } a \ \text{t}[b])$   
   $\quad (C.\text{lunit } b \cdot C.\text{tensor } \text{t}[a] \ b) \cdot \text{dup} \ (C.\text{tensor } a \ b) =$   
   $C.\text{tensor } a \ b$

**proof** –

**fix**  $a \ b$   
  **assume**  $a: \text{ide } a$  **and**  $b: \text{ide } b$   
  **have**  $C.\text{tensor} \ (C.\text{runit } a \cdot C.\text{tensor } a \ \text{t}[b])$   
   $\quad (C.\text{lunit } b \cdot C.\text{tensor } \text{t}[a] \ b) \cdot \text{dup} \ (C.\text{tensor } a \ b) =$   
   $\text{prod} \ (C.\text{runit } a \cdot \text{prod } a \ \text{t}[b]) \ (C.\text{lunit } b \cdot \text{prod } \text{t}[a] \ b) \cdot \text{dup} \ (\text{prod } a \ b)$   
  **using**  $a \ b$  **by** *simp*  
  **also have**  $\dots = \text{tuple} \ ((C.\text{runit } a \cdot \text{prod } a \ \text{t}[b]) \cdot \text{prod } a \ b)$   
   $\quad ((C.\text{lunit } b \cdot \text{prod } \text{t}[a] \ b) \cdot \text{prod } a \ b)$   
  **using**  $a \ b$  *ide-one trm-in-hom [of a] trm-in-hom [of b] unity-agreement prod-tuple*  
  **by** *fastforce*  
  **also have**  $\dots = \text{tuple} \ (C.\text{runit } a \cdot \text{prod } a \ \text{t}[b] \cdot \text{prod } a \ b)$   
   $\quad (C.\text{lunit } b \cdot \text{prod } \text{t}[a] \ b \cdot \text{prod } a \ b)$   
  **using** *comp-assoc by simp*  
  **also have**  $\dots = \text{tuple} \ (C.\text{runit } a \cdot \text{prod } a \ \text{t}[b]) \ (C.\text{lunit } b \cdot \text{prod } \text{t}[a] \ b)$



```

    using a b comp-arr-dom by simp
  also have ... = tuple (runit a · prod a t[b]) (lunit b · prod t[a] b)
    using a b lunit-agreement runit-agreement by simp
  also have ... = tuple (pr1 a b) (pr0 a b)
  proof -
    have runit a · prod a t[b] = pr1 a b
      using a b pr-naturality(2) trm-in-hom [of b]
    by (metis cod-pr1 comp-ide-arr ide-char in-homE pr-simps(4,6) seqI)
  moreover have lunit b · prod t[a] b = pr0 a b
    using a b pr-naturality(1) [of b b b t[a] a 1] trm-in-hom [of a] comp-cod-arr
    by (metis cod-pr0 ide-char in-homE pr-simps(1))
  ultimately show ?thesis by simp
qed
also have ... = prod a b
  using a b by simp
finally show C.tensor (C.runit a · C.tensor a t[b])
  (C.lunit b · C.tensor t[a] b) · dup (C.tensor a b) =
  C.tensor a b
  by auto
qed
qed

```

**lemma** *extends-to-cartesian-monoidal-category*:  
**shows** *cartesian-monoidal-category C Prod α.map ι dup trm*  
 ..

**end**

In a *cartesian-monoidal-category*, the monoidal structure is given by a categorical product and terminal object, so that the underlying category is cartesian.

**context** *cartesian-monoidal-category*  
**begin**

**definition** *pr0*  $(p_0[-, -])$   
**where**  $p_0[a, b] \equiv \text{if } \text{ide } a \wedge \text{ide } b \text{ then } r[a] \cdot (a \otimes t[b]) \text{ else null}$

**definition** *pr1*  $(p_1[-, -])$   
**where**  $p_1[a, b] \equiv \text{if } \text{ide } a \wedge \text{ide } b \text{ then } l[b] \cdot (t[a] \otimes b) \text{ else null}$

**lemma** *pr-in-hom [intro]*:  
**assumes** *ide a0 and ide a1*  
**shows**  $\langle p_0[a_0, a_1] : a_0 \otimes a_1 \rightarrow a_0 \rangle$   
**and**  $\langle p_1[a_0, a_1] : a_0 \otimes a_1 \rightarrow a_1 \rangle$   
**proof** -  
**show**  $\langle p_0[a_0, a_1] : a_0 \otimes a_1 \rightarrow a_0 \rangle$   
**unfolding** *pr0-def*  
**using** *assms runit-in-hom* **by** *fastforce*  
**show**  $\langle p_1[a_0, a_1] : a_0 \otimes a_1 \rightarrow a_1 \rangle$   
**unfolding** *pr1-def*

**using** *assms lunit-in-hom* **by** *fastforce*  
**qed**

**lemma** *pr-simps [simp]*:  
**assumes** *ide a0* **and** *ide a1*  
**shows** *arr p0[a0, a1]* **and** *dom p0[a0, a1] = a0 ⊗ a1* **and** *cod p0[a0, a1] = a0*  
**and** *arr p1[a0, a1]* **and** *dom p1[a0, a1] = a0 ⊗ a1* **and** *cod p1[a0, a1] = a1*  
**using** *assms pr-in-hom(1-2)* **by** *blast+*

**interpretation** *P: composite-functor CC.comp C CC.comp T Δ.map ..*

**interpretation** *ECC: elementary-cartesian-category C pr1 pr0 I τ*

**proof**

**show**  $\bigwedge a b. \llbracket \text{ide } a; \text{ide } b \rrbracket \implies \text{span } p_0[a, b] \text{ } p_1[a, b]$

**by** *simp*

**show**  $\bigwedge a b. \llbracket \text{ide } a; \text{ide } b \rrbracket \implies \text{cod } p_1[a, b] = b$

**by** *simp*

**show**  $\bigwedge a b. \llbracket \text{ide } a; \text{ide } b \rrbracket \implies \text{cod } p_0[a, b] = a$

**by** *simp*

**show** *ide I*

**by** *simp*

**show**  $\bigwedge a. \text{ide } a \implies \langle \tau \ a : a \rightarrow I \rangle$

**by** *auto*

**show**  $\bigwedge a f. \llbracket \text{ide } a; \langle f : a \rightarrow I \rangle \rrbracket \implies f = t[a]$

**using**  $\langle \bigwedge a. \text{ide } a \implies \langle t[a] : a \rightarrow I \rangle \rangle$  *terminalE terminal-unity* **by** *blast*

**show**  $\bigwedge a b. \neg (\text{ide } a \wedge \text{ide } b) \implies p_1[a, b] = \text{null}$

**using** *pr1-def* **by** *auto*

**show**  $\bigwedge a b. \neg (\text{ide } a \wedge \text{ide } b) \implies p_0[a, b] = \text{null}$

**using** *pr0-def* **by** *auto*

**show**  $\bigwedge f g. \text{span } f g \implies \exists ! l. p_0[\text{cod } f, \text{cod } g] \cdot l = f \wedge p_1[\text{cod } f, \text{cod } g] \cdot l = g$

**proof** –

**fix** *f g*

**assume** *fg: span f g*

**let** *?l = (f ⊗ g) · d[dom f]*

**have**  $p_0[\text{cod } f, \text{cod } g] \cdot ?l = f$

**proof** –

**have**  $p_0[\text{cod } f, \text{cod } g] \cdot ?l = (r[\text{cod } f] \cdot (\text{cod } f \otimes t[\text{cod } g])) \cdot (f \otimes g) \cdot d[\text{dom } f]$

**using** *fg pr0-def* **by** *simp*

**also have**  $\dots = r[\text{cod } f] \cdot ((\text{cod } f \otimes t[\text{cod } g]) \cdot (f \otimes g)) \cdot d[\text{dom } f]$

**using** *comp-assoc* **by** *simp*

**also have**  $\dots = r[\text{cod } f] \cdot (f \otimes t[\text{dom } f]) \cdot d[\text{dom } f]$

**using** *fg interchange comp-cod-arr τ.naturality* **by** *simp*

**also have**  $\dots = r[\text{cod } f] \cdot ((\text{cod } f \otimes t[\text{cod } f]) \cdot (f \otimes f)) \cdot d[\text{dom } f]$

**using** *fg interchange comp-cod-arr τ.naturality* **by** *simp*

**also have**  $\dots = r[\text{cod } f] \cdot (\text{cod } f \otimes t[\text{cod } f]) \cdot (f \otimes f) \cdot d[\text{dom } f]$

**using** *comp-assoc* **by** *simp*

**also have**  $\dots = (r[\text{cod } f] \cdot (\text{cod } f \otimes t[\text{cod } f]) \cdot d[\text{cod } f]) \cdot f$

**using** *fg δ.naturality comp-assoc* **by** *simp*

**also have**  $\dots = f$

```

    using fg pr0-dup comp-cod-arr by simp
    finally show ?thesis by blast
qed
moreover have p1[cod f, cod g] · ?l = g
proof -
  have p1[cod f, cod g] · ?l = l[cod g] · ((t[cod f] ⊗ cod g) · (f ⊗ g)) · d[dom g]
    using fg pr1-def comp-assoc by simp
  also have ... = l[cod g] · (t[dom f] ⊗ g) · d[dom g]
    using fg interchange comp-cod-arr τ.naturality
    by simp
  also have ... = l[cod g] · ((t[cod g] ⊗ cod g) · (g ⊗ g)) · d[dom g]
    using fg interchange comp-cod-arr τ.naturality by simp
  also have ... = l[cod g] · (t[cod g] ⊗ cod g) · (g ⊗ g) · d[dom g]
    using comp-assoc by simp
  also have ... = (l[cod g] · (t[cod g] ⊗ cod g) · d[cod g]) · g
    using fg δ.naturality comp-assoc by simp
  also have ... = g
    using fg pr1-dup comp-cod-arr by simp
  finally show ?thesis by blast
qed
moreover have ∧l. [ p0[cod f, cod g] · l = f; p1[cod f, cod g] · l = g ] ⇒ l = ?l
proof -
  fix l
  assume f: p0[cod f, cod g] · l = f and g: p1[cod f, cod g] · l = g
  have l: «l : dom f → cod f ⊗ cod g»
    using f g fg
    by (metis arr-iff-in-hom dom-comp ide-cod pr-simps(5) seqE)
  have ?l = (p0[cod f, cod g] · l ⊗ p1[cod f, cod g] · l) · d[dom f]
    using f g by simp
  also have ... = ((p0[cod f, cod g] ⊗ p1[cod f, cod g]) · (l ⊗ l)) · d[dom f]
    using fg f g interchange by simp
  also have ... = (p0[cod f, cod g] ⊗ p1[cod f, cod g]) · (l ⊗ l) · d[dom f]
    using comp-assoc by simp
  also have ... = ((p0[cod f, cod g] ⊗ p1[cod f, cod g]) · d[cod f ⊗ cod g]) · l
    using l δ.naturality [of l] comp-assoc by auto
  also have ... = (cod f ⊗ cod g) · l
    using f g fg tuple-pr pr0-def pr1-def by auto
  also have ... = l
    using l comp-cod-arr by auto
  finally show l = ?l by simp
qed
ultimately show ∃!l. p0[cod f, cod g] · l = f ∧ p1[cod f, cod g] · l = g
  by auto
qed
qed
lemma extends-to-elementary-cartesian-category:
shows elementary-cartesian-category C pr1 pr0 I τ
..

```

```
lemma is-cartesian-category:  
shows cartesian-category C  
  using ECC.is-cartesian-category by simp
```

```
end
```

```
end
```

# Bibliography

- [1] J. Bénabou. Catégories avec multiplication. *C. R. Acad. Sci. Paris*, 258:1887 – 1890, 1963.
- [2] P. Etingof, S. Gelaki, D. Nikshych, and V. Ostrik. *Tensor Categories*, volume 205 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2015.
- [3] G. M. Kelly. On MacLane’s conditions for coherence of natural associativities, commutativities, etc. *Journal of Algebra*, 1:397 – 402, 1964.
- [4] S. MacLane. Natural associativity and commutativity. *Rice. Univ. Stud.*, 49:28 – 46, 1963.
- [5] S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.
- [6] 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.