Monoidal Categories

Eugene W. Stark

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

March 17, 2025

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

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 $F : C \to D$, 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 $F : C \to D$ 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 $\mathcal{C}$.

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 α, whose components are isomorphisms α (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 α 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 α 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 use 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.

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 α: $T\ o\ (T \times C) \rightarrow T\ o\ (C \times T)$, a *unit object* $\mathcal{I}$ of C, and an isomorphism ι: $T\ (\mathcal{I}, \mathcal{I}) \rightarrow \mathcal{I}$, subject to two axioms: the *pentagon axiom*, which expresses the commutativity of certain pentagonal diagrams involving components of α, 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 *unitor* isomorphisms λ and ϱ and associated axioms involving their components. However, as is shown in [2] and formalized here, the unitors are uniquely determined by α and their values $\lambda(\mathcal{I})$ and $\varrho(\mathcal{I})$ at $\mathcal{I}$, which coincide. Treating λ and ϱ 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 ι.

locale *monoidal-category* =
  category C +
  CC: *product-category C C* +
  CCC: *product-category C CC.comp* +
  T: *binary-endofunctor C T* +
  α: *natural-isomorphism CCC.comp C T.ToTC T.ToCT* α +
  L: *equivalence-functor C C* λ*f*. *T* (*cod* ι, *f*) +
  R: *equivalence-functor C C* λ*f*. *T* (*f*, *cod* ι)
for *C* :: *'a comp* (**infixr** ‹·› *55*)
and *T* :: *'a* ∗ *'a* ⇒ *'a*
and α :: *'a* ∗ *'a* ∗ *'a* ⇒ *'a*
and ι :: *'a* +
assumes *unit-in-hom-ax*: «ι : *T* (*cod* ι, *cod* ι) → *cod* ι»
and *unit-is-iso*: *iso* ι
and *pentagon*: ⟦ *ide a*; *ide b*; *ide c*; *ide d* ⟧ ⟹
    T (*a*, α (*b*, *c*, *d*)) · α (*a*, *T* (*b*, *c*), *d*) · *T* (α (*a*, *b*, *c*), *d*) =
    α (*a*, *b*, *T* (*c*, *d*)) · α (*T* (*a*, *b*), *c*, *d*)
begin

We now define helpful notation and abbreviations to improve readability. We did not define and use the notation ⊗ for the tensor product in the definition of the locale because to define ⊗ 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.

abbreviation *unity* :: *'a* (‹$\mathcal{I}$›)
where *unity* ≡ *cod* ι

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

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

abbreviation *tensor* (**infixr** ‹⊗› *53*)
where *f* ⊗ *g* ≡ *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*. «$f : \mathcal{I} \otimes a \rightarrow a$» $\wedge\ \mathcal{I} \otimes f = (\iota \otimes a) \cdot$ *inv* a[$\mathcal{I}$, $\mathcal{I}$, a]

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

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 *unit-in-hom* [*intro*]:
shows «$\iota : \mathcal{I} \otimes \mathcal{I} \rightarrow \mathcal{I}$»
⟨*proof*⟩

lemma *ide-unity* [*simp*]:
shows *ide* $\mathcal{I}$
⟨*proof*⟩

lemma *tensor-in-hom* [*simp*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$»
shows «$f \otimes g : a \otimes c \rightarrow b \otimes d$»
⟨*proof*⟩

lemma *tensor-in-homI* [*intro*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$» **and** $x = a \otimes c$ **and** $y = b \otimes d$
shows «$f \otimes g : x \rightarrow y$»
⟨*proof*⟩

lemma *arr-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *arr* $(f \otimes g)$
⟨*proof*⟩

lemma *dom-tensor* [*simp*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$»
shows *dom* $(f \otimes g) = a \otimes c$
⟨*proof*⟩

lemma *cod-tensor* [*simp*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$»
shows *cod* $(f \otimes g) = b \otimes d$
⟨*proof*⟩

lemma *tensor-preserves-ide* [*simp*]:
assumes *ide a* **and** *ide b*
shows *ide* $(a \otimes b)$
⟨*proof*⟩

lemma *tensor-preserves-iso* [*simp*]:
assumes *iso f* **and** *iso g*
shows *iso* $(f \otimes g)$
⟨*proof*⟩

lemma *inv-tensor* [*simp*]:
assumes *iso f* **and** *iso g*
shows *inv* $(f \otimes g) =$ *inv f* $\otimes$ *inv g*
⟨*proof*⟩

lemma *interchange*:
assumes *seq h g* **and** *seq h′ g′*
shows $(h \otimes h') \cdot (g \otimes g') = h \cdot g \otimes h' \cdot g'$
⟨*proof*⟩

lemma *α-simp*:
assumes *arr f* **and** *arr g* **and** *arr h*
shows $\alpha\ (f, g, h) = (f \otimes g \otimes h) \cdot$ a[*dom f*, *dom g*, *dom h*]
⟨*proof*⟩

lemma *assoc-in-hom* [*intro*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows «a[a, b, c] : $(a \otimes b) \otimes c \rightarrow a \otimes b \otimes c$»
⟨*proof*⟩

lemma *arr-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *arr* a[a, b, c]
⟨*proof*⟩

lemma *dom-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *dom* a[a, b, c] $= (a \otimes b) \otimes c$
⟨*proof*⟩

lemma *cod-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *cod* a[a, b, c] $= a \otimes b \otimes c$
⟨*proof*⟩

lemma *assoc-naturality*:
assumes *arr f0* **and** *arr f1* **and** *arr f2*
shows a[*cod f0*, *cod f1*, *cod f2*] $\cdot$ ((*f0* $\otimes$ *f1*) $\otimes$ *f2*) =

$(f0 \otimes f1 \otimes f2) \cdot$ a[*dom f0*, *dom f1*, *dom f2*]
⟨*proof*⟩

lemma *iso-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *iso* a[a, b, c]
⟨*proof*⟩

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 by $\otimes$, α, and ι.

lemma *lunit-char*:
assumes *ide a*
shows «l[a] : $\mathcal{I} \otimes a \rightarrow a$» **and** $\mathcal{I} \otimes$ l[a] $= (\iota \otimes a) \cdot$ *inv* a[$\mathcal{I}$, $\mathcal{I}$, a]
and $\exists!f$. «$f : \mathcal{I} \otimes a \rightarrow a$» $\wedge\ \mathcal{I} \otimes f = (\iota \otimes a) \cdot$ *inv* a[$\mathcal{I}$, $\mathcal{I}$, a]
⟨*proof*⟩

lemma *lunit-in-hom* [*intro*]:
assumes *ide a*
shows «l[a] : $\mathcal{I} \otimes a \rightarrow a$»
⟨*proof*⟩

lemma *arr-lunit* [*simp*]:
assumes *ide a*
shows *arr* l[a]
⟨*proof*⟩

lemma *dom-lunit* [*simp*]:
assumes *ide a*
shows *dom* l[a] $= \mathcal{I} \otimes a$
⟨*proof*⟩

lemma *cod-lunit* [*simp*]:
assumes *ide a*
shows *cod* l[a] $= a$
⟨*proof*⟩

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

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

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

lemma *lunit-eqI*:

assumes «$f : \mathcal{I} \otimes a \rightarrow a$» **and** $\mathcal{I} \otimes f = (\iota \otimes a) \cdot inv$ a$[\mathcal{I}, \mathcal{I}, a]$
shows $f =$ l$[a]$
⟨*proof*⟩

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

lemma *runit-char*:
assumes *ide a*
shows «r$[a] : a \otimes \mathcal{I} \rightarrow a$» **and** r$[a] \otimes \mathcal{I} = (a \otimes \iota) \cdot$ a$[a, \mathcal{I}, \mathcal{I}]$
and $\exists!f.$ «$f : a \otimes \mathcal{I} \rightarrow a$» $\wedge f \otimes \mathcal{I} = (a \otimes \iota) \cdot$ a$[a, \mathcal{I}, \mathcal{I}]$
⟨*proof*⟩

lemma *runit-in-hom* [*intro*]:
assumes *ide a*
shows «r$[a] : a \otimes \mathcal{I} \rightarrow a$»
⟨*proof*⟩

lemma *arr-runit* [*simp*]:
assumes *ide a*
shows *arr* r$[a]$
⟨*proof*⟩

lemma *dom-runit* [*simp*]:
assumes *ide a*
shows *dom* r$[a] = a \otimes \mathcal{I}$
⟨*proof*⟩

lemma *cod-runit* [*simp*]:
assumes *ide a*
shows *cod* r$[a] = a$
⟨*proof*⟩

lemma *runit-eqI*:
assumes «$f : a \otimes \mathcal{I} \rightarrow a$» **and** $f \otimes \mathcal{I} = (a \otimes \iota) \cdot$ a$[a, \mathcal{I}, \mathcal{I}]$
shows $f =$ r$[a]$
⟨*proof*⟩

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

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$[cod\ f] \cdot (\mathcal{I} \otimes f) = f \cdot$ l$[dom\ f]$
⟨*proof*⟩

lemma *runit-naturality*:
assumes *arr f*
shows r[*cod f*] · (*f* ⊗ $\mathcal{I}$) = *f* · r[*dom f*]
⟨*proof*⟩

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

abbreviation 𝔩
where 𝔩 *f* ≡ *if arr f then f* · l[*dom f*] *else null*

abbreviation *ϱ*
where *ϱ f* ≡ *if arr f then f* · r[*dom f*] *else null*

lemma 𝔩*-ide-simp*:
assumes *ide a*
shows 𝔩 *a* = l[*a*]
⟨*proof*⟩

lemma *ϱ-ide-simp*:
assumes *ide a*
shows *ϱ a* = r[*a*]
⟨*proof*⟩

end

context *monoidal-category*
begin

sublocale 𝔩: *natural-transformation C C L map* 𝔩
⟨*proof*⟩

sublocale 𝔩: *natural-isomorphism C C L map* 𝔩
⟨*proof*⟩

sublocale *ϱ*: *natural-transformation C C R map ϱ*
⟨*proof*⟩

sublocale *ϱ*: *natural-isomorphism C C R map ϱ*
⟨*proof*⟩

sublocale 𝔩′: *inverse-transformation C C L map* 𝔩 ⟨*proof*⟩
sublocale *ϱ*′: *inverse-transformation C C R map ϱ* ⟨*proof*⟩
sublocale *α*′: *inverse-transformation CCC.comp C T.ToTC T.ToCT α* ⟨*proof*⟩

abbreviation *α*′
where *α*′ ≡ *α*′*.map*

abbreviation *assoc*′ (‹a^{-1}[-, -, -]›)

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

lemma *α′-ide-simp*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $\alpha'\ (a,\ b,\ c) = \mathrm{a}^{-1}[a,\ b,\ c]$
⟨*proof*⟩

lemma *α′-simp*:
assumes *arr f* **and** *arr g* **and** *arr h*
shows $\alpha'\ (f,\ g,\ h) = ((f \otimes g) \otimes h) \cdot \mathrm{a}^{-1}[\mathit{dom}\ f,\ \mathit{dom}\ g,\ \mathit{dom}\ h]$
⟨*proof*⟩

lemma *assoc-inv*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *inverse-arrows* $\mathrm{a}[a,\ b,\ c]\ \mathrm{a}^{-1}[a,\ b,\ c]$
⟨*proof*⟩

lemma *assoc′-in-hom* [*intro*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows «$\mathrm{a}^{-1}[a,\ b,\ c] : a \otimes b \otimes c \rightarrow (a \otimes b) \otimes c$»
⟨*proof*⟩

lemma *arr-assoc′* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *arr* $\mathrm{a}^{-1}[a,\ b,\ c]$
⟨*proof*⟩

lemma *dom-assoc′* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *dom* $\mathrm{a}^{-1}[a,\ b,\ c] = a \otimes b \otimes c$
⟨*proof*⟩

lemma *cod-assoc′* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *cod* $\mathrm{a}^{-1}[a,\ b,\ c] = (a \otimes b) \otimes c$
⟨*proof*⟩

lemma *comp-assoc-assoc′* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $\mathrm{a}[a,\ b,\ c] \cdot \mathrm{a}^{-1}[a,\ b,\ c] = a \otimes (b \otimes c)$
and $\mathrm{a}^{-1}[a,\ b,\ c] \cdot \mathrm{a}[a,\ b,\ c] = (a \otimes b) \otimes c$
⟨*proof*⟩

lemma *assoc′-naturality*:
assumes *arr f0* **and** *arr f1* **and** *arr f2*
shows $((f0 \otimes f1) \otimes f2) \cdot \mathrm{a}^{-1}[\mathit{dom}\ f0,\ \mathit{dom}\ f1,\ \mathit{dom}\ f2] =$
$\mathrm{a}^{-1}[\mathit{cod}\ f0,\ \mathit{cod}\ f1,\ \mathit{cod}\ f2] \cdot (f0 \otimes f1 \otimes f2)$
⟨*proof*⟩

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

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

lemma $\mathfrak{l}'$*-ide-simp*:
assumes *ide a*
shows $\mathfrak{l}'.map\ a = l^{-1}[a]$
⟨*proof*⟩

lemma *lunit-inv*:
assumes *ide a*
shows *inverse-arrows* $l[a]\ l^{-1}[a]$
⟨*proof*⟩

lemma *lunit'-in-hom* [*intro*]:
assumes *ide a*
shows «$l^{-1}[a] : a \rightarrow \mathcal{I} \otimes a$»
⟨*proof*⟩

lemma *comp-lunit-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*⟩

lemma *lunit'-naturality*:
assumes *arr f*
shows $(\mathcal{I} \otimes f) \cdot l^{-1}[dom\ f] = l^{-1}[cod\ f] \cdot f$
⟨*proof*⟩

abbreviation ϱ'
where $\varrho' \equiv \varrho'.map$

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

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

lemma *runit-inv*:
assumes *ide a*
shows *inverse-arrows* $r[a]\ r^{-1}[a]$
⟨*proof*⟩

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

assumes *ide a*
shows «$\mathrm{r}^{-1}[a] : a \to a \otimes \mathcal{I}$»
⟨*proof*⟩

lemma *comp-runit-runit′* [*simp*]:
assumes *ide a*
shows $\mathrm{r}[a] \cdot \mathrm{r}^{-1}[a] = a$
and $\mathrm{r}^{-1}[a] \cdot \mathrm{r}[a] = a \otimes \mathcal{I}$
⟨*proof*⟩

lemma *runit′-naturality*:
assumes *arr f*
shows $(f \otimes \mathcal{I}) \cdot \mathrm{r}^{-1}[dom\ f] = \mathrm{r}^{-1}[cod\ f] \cdot f$
⟨*proof*⟩

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

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

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 $(a \otimes \mathrm{l}[b]) \cdot \mathrm{a}[a, \mathcal{I}, b] = \mathrm{r}[a] \otimes b$
⟨*proof*⟩

lemma *lunit-tensor-gen*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $(a \otimes \mathrm{l}[b \otimes c]) \cdot (a \otimes \mathrm{a}[\mathcal{I}, b, c]) = a \otimes \mathrm{l}[b] \otimes c$
⟨*proof*⟩

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 $\mathrm{l}[a \otimes b] \cdot \mathrm{a}[\mathcal{I}, a, b] = \mathrm{l}[a] \otimes b$
⟨*proof*⟩

lemma *lunit-tensor*:
assumes *ide a* **and** *ide b*
shows $\mathrm{l}[a \otimes b] = (\mathrm{l}[a] \otimes b) \cdot \mathrm{a}^{-1}[\mathcal{I}, a, b]$

⟨*proof*⟩

We next show the corresponding result for the right unitor.

lemma *runit-tensor-gen*:
assumes *ide* a **and** *ide* b **and** *ide* c
shows $\mathrm{r}[a \otimes b] \otimes c = ((a \otimes \mathrm{r}[b]) \otimes c) \cdot (\mathrm{a}[a,\ b,\ \mathcal{I}] \otimes c)$
⟨*proof*⟩

lemma *runit-tensor*:
assumes *ide* a **and** *ide* b
shows $\mathrm{r}[a \otimes b] = (a \otimes \mathrm{r}[b]) \cdot \mathrm{a}[a,\ b,\ \mathcal{I}]$
⟨*proof*⟩

lemma *runit-tensor′*:
assumes *ide* a **and** *ide* b
shows $\mathrm{r}[a \otimes b] \cdot \mathrm{a}^{-1}[a,\ b,\ \mathcal{I}] = a \otimes \mathrm{r}[b]$
⟨*proof*⟩

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

lemma *triangle′*:
assumes *ide* a **and** *ide* b
shows $(a \otimes \mathrm{l}[b]) = (\mathrm{r}[a] \otimes b) \cdot \mathrm{a}^{-1}[a, \mathcal{I},\ b]$
⟨*proof*⟩

lemma *pentagon′*:
assumes *ide* a **and** *ide* b **and** *ide* c **and** *ide* d
shows $((\mathrm{a}^{-1}[a,\ b,\ c] \otimes d) \cdot \mathrm{a}^{-1}[a,\ b \otimes c,\ d]) \cdot (a \otimes \mathrm{a}^{-1}[b,\ c,\ d])$
$= \mathrm{a}^{-1}[a \otimes b,\ c,\ d] \cdot \mathrm{a}^{-1}[a,\ b,\ c \otimes d]$
⟨*proof*⟩

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

lemma *unitor-coincidence*:
shows $\mathrm{l}[\mathcal{I}] = \iota$ **and** $\mathrm{r}[\mathcal{I}] = \iota$
⟨*proof*⟩

lemma *unit-triangle*:
shows $\iota \otimes \mathcal{I} = (\mathcal{I} \otimes \iota) \cdot \mathrm{a}[\mathcal{I}, \mathcal{I}, \mathcal{I}]$
and $(\iota \otimes \mathcal{I}) \cdot \mathrm{a}^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}] = \mathcal{I} \otimes \iota$
⟨*proof*⟩

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

lemma *iso-commuting-with-unit-equals-unity*:
assumes «$f : \mathcal{I} \rightarrow \mathcal{I}$» **and** *iso* f **and** $f \cdot \iota = \iota \cdot (f \otimes f)$
shows $f = \mathcal{I}$
⟨*proof*⟩

end

We now show that the unit ι 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 $C\ T\ \alpha\ \iota$ +
  C_1: *monoidal-category* $C\ T\ \alpha\ \iota_1$
for C :: $'a\ comp$ (**infixr** ‹·› *55*)
and T :: $'a * 'a \Rightarrow 'a$
and α :: $'a * 'a * 'a \Rightarrow 'a$
and ι :: $'a$
and ι_1 :: $'a$
begin

  no-notation C_1.*tensor* (**infixr** ‹⊗› *53*)
  no-notation C_1.*unity* (‹$\mathcal{I}$›)
  no-notation C_1.*lunit* (‹l[-]›)
  no-notation C_1.*runit* (‹r[-]›)
  no-notation C_1.*assoc* (‹a[-, -, -]›)
  no-notation C_1.*assoc'* (‹a^{-1}[-, -, -]›)

  notation C_1.*tensor* (**infixr** ‹$\otimes_1$› *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 «$i : \mathcal{I} \rightarrow \mathcal{I}_1$» **and** *iso* i
  ⟨*proof*⟩

The following is Exercise 2.2.7 of [2].

  lemma *i-maps-*ι*-to-*ι_1:
  shows $i \cdot \iota = \iota_1 \cdot (i \otimes i)$
  ⟨*proof*⟩

  lemma *inv-i-iso-*ι:
  assumes «$f : \mathcal{I} \rightarrow \mathcal{I}_1$» **and** *iso* f **and** $f \cdot \iota = \iota_1 \cdot (f \otimes f)$
  shows «$inv\ i \cdot f : \mathcal{I} \rightarrow \mathcal{I}$» **and** *iso* $(inv\ i \cdot f)$
  and $(inv\ i \cdot f) \cdot \iota = \iota \cdot (inv\ i \cdot f \otimes inv\ i \cdot f)$
  ⟨*proof*⟩

  lemma *unit-unique-upto-unique-iso*:
  shows $\exists!f.$ «$f : \mathcal{I} \rightarrow \mathcal{I}_1$» $\wedge$ *iso* $f \wedge f \cdot \iota = \iota_1 \cdot (f \otimes f)$
  ⟨*proof*⟩

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 comp* (**infixr** ‹·› *55*)
and *tensor* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹⊗› *53*)
and *unity* :: *'a* (‹𝓘›)
and *lunit* :: *'a* ⇒ *'a* (‹l[-]›)
and *runit* :: *'a* ⇒ *'a* (‹r[-]›)
and *assoc* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹a[-, -, -]›) +
assumes *ide-unity* [*simp*]: *ide* 𝓘
and *iso-lunit*: *ide a* ⟹ *iso* l[*a*]
and *iso-runit*: *ide a* ⟹ *iso* r[*a*]
and *iso-assoc*: ⟦ *ide a*; *ide b*; *ide c* ⟧ ⟹ *iso* a[*a*, *b*, *c*]
and *tensor-in-hom* [*simp*]: ⟦ «*f* : *a* → *b*»; «*g* : *c* → *d*» ⟧ ⟹ «*f* ⊗ *g* : *a* ⊗ *c* → *b* ⊗ *d*»
and *tensor-preserves-ide*: ⟦ *ide a*; *ide b* ⟧ ⟹ *ide* (*a* ⊗ *b*)
and *interchange*: ⟦ *seq g f*; *seq g' f'* ⟧ ⟹ (*g* ⊗ *g'*) · (*f* ⊗ *f'*) = *g* · *f* ⊗ *g'* · *f'*
and *lunit-in-hom* [*simp*]: *ide a* ⟹ «l[*a*] : 𝓘 ⊗ *a* → *a*»
and *lunit-naturality*: *arr f* ⟹ l[*cod f*] · (𝓘 ⊗ *f*) = *f* · l[*dom f*]
and *runit-in-hom* [*simp*]: *ide a* ⟹ «r[*a*] : *a* ⊗ 𝓘 → *a*»
and *runit-naturality*: *arr f* ⟹ r[*cod f*] · (*f* ⊗ 𝓘) = *f* · r[*dom f*]
and *assoc-in-hom* [*simp*]:
    ⟦ *ide a*; *ide b*; *ide c* ⟧ ⟹ «a[*a*, *b*, *c*] : (*a* ⊗ *b*) ⊗ *c* → *a* ⊗ *b* ⊗ *c*»
and *assoc-naturality*:
    ⟦ *arr f0*; *arr f1*; *arr f2* ⟧ ⟹ a[*cod f0*, *cod f1*, *cod f2*] · ((*f0* ⊗ *f1*) ⊗ *f2*)
        = (*f0* ⊗ (*f1* ⊗ *f2*)) · a[*dom f0*, *dom f1*, *dom f2*]
and *triangle*: ⟦ *ide a*; *ide b* ⟧ ⟹ (*a* ⊗ l[*b*]) · a[*a*, 𝓘, *b*] = r[*a*] ⊗ *b*
and *pentagon*: ⟦ *ide a*; *ide b*; *ide c*; *ide d* ⟧ ⟹
        (*a* ⊗ a[*b*, *c*, *d*]) · a[*a*, *b* ⊗ *c*, *d*] · (a[*a*, *b*, *c*] ⊗ *d*)
          = a[*a*, *b*, *c* ⊗ *d*] · a[*a* ⊗ *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* 𝓘 *lunit runit assoc*
      ⟨*proof*⟩

end

context *elementary-monoidal-category*
begin

interpretation *CC*: *product-category C C* ⟨*proof*⟩
interpretation *CCC*: *product-category C CC.comp* ⟨*proof*⟩

definition $T :: 'a * 'a \Rightarrow 'a$
where $T\ f \equiv$ *if CC.arr f then* (*fst f* ⊗ *snd f*) *else null*

lemma *T-simp* [*simp*]:
assumes *arr f* **and** *arr g*
shows $T\ (f,\ g) = f \otimes g$
⟨*proof*⟩

lemma *arr-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *arr* $(f \otimes g)$
⟨*proof*⟩

lemma *dom-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *dom* $(f \otimes g) =$ *dom f* ⊗ *dom g*
⟨*proof*⟩

lemma *cod-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *cod* $(f \otimes g) =$ *cod f* ⊗ *cod g*
⟨*proof*⟩

interpretation *T*: *binary-endofunctor C T*
⟨*proof*⟩

lemma *binary-endofunctor-T*:
shows *binary-endofunctor C T* ⟨*proof*⟩

interpretation *ToTC*: *functor CCC.comp C T.ToTC*
⟨*proof*⟩

interpretation *ToCT*: *functor CCC.comp C T.ToCT*
⟨*proof*⟩

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

lemma α*-ide-simp* [*simp*]:

assumes *ide a* **and** *ide b* **and** *ide c*
shows α $(a,\ b,\ c) = \mathrm{a}[a,\ b,\ c]$
⟨*proof*⟩

lemma *arr-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *arr* $\mathrm{a}[a,\ b,\ c]$
⟨*proof*⟩

lemma *dom-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *dom* $\mathrm{a}[a,\ b,\ c] = (a \otimes b) \otimes c$
⟨*proof*⟩

lemma *cod-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *cod* $\mathrm{a}[a,\ b,\ c] = a \otimes b \otimes c$
⟨*proof*⟩

interpretation α: *natural-isomorphism CCC.comp C T.ToTC T.ToCT* α
⟨*proof*⟩

interpretation α': *inverse-transformation CCC.comp C T.ToTC T.ToCT* α ⟨*proof*⟩

interpretation *L*: *functor C C* ‹$\lambda f.\ T\ (\mathcal{I},\ f)$›
⟨*proof*⟩

interpretation *R*: *functor C C* ‹$\lambda f.\ T\ (f,\ \mathcal{I})$›
⟨*proof*⟩

interpretation $\mathfrak{l}$: *natural-isomorphism C C* ‹$\lambda f.\ T\ (\mathcal{I},\ f)$› *map*
‹$\lambda f.$ *if arr f then* $f \cdot \mathrm{l}[dom\ f]$ *else null*›
⟨*proof*⟩

interpretation ϱ: *natural-isomorphism C C* ‹$\lambda f.\ T\ (f,\ \mathcal{I})$› *map*
‹$\lambda f.$ *if arr f then* $f \cdot \mathrm{r}[dom\ f]$ *else null*›
⟨*proof*⟩

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* ‹$\lambda f.\ T\ (\mathcal{I},\ f)$›
⟨*proof*⟩

interpretation *R*: *equivalence-functor C C* ‹$\lambda f.\ T\ (f,\ \mathcal{I})$›
⟨*proof*⟩

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

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

lemma *ι-in-hom*:
shows «$\iota : \mathcal{I} \otimes \mathcal{I} \to \mathcal{I}$»
⟨*proof*⟩

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

interpretation *MC*: *monoidal-category* $C\ T\ \alpha\ \iota$
⟨*proof*⟩

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 *$\mathcal{I}$-agreement*:
shows $MC.unity = \mathcal{I}$
⟨*proof*⟩

lemma *L-agreement*:
shows $MC.L = (\lambda f.\ T\ (\mathcal{I}, f))$
⟨*proof*⟩

lemma *R-agreement*:
shows $MC.R = (\lambda f.\ T\ (f, \mathcal{I}))$
⟨*proof*⟩

We wish to show that the components of the unitors $MC.\mathfrak{l}$ and $MC.\varrho$ 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 $\mathrm{l}[\mathcal{I}] = MC.lunit\ MC.unity$ and $\mathrm{r}[\mathcal{I}] = MC.runit\ MC.unity$, from which it follows by facts already proved for *MC* that both are equal to ι. We then show that for an arbitrary identity a the arrows $\mathrm{l}[a]$ and $\mathrm{r}[a]$ 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 $\mathrm{l}[\mathcal{I}] = \iota$ **and** $\mathrm{r}[\mathcal{I}] = \iota$
⟨*proof*⟩

lemma *lunit-char*:
assumes *ide a*
shows $\mathcal{I} \otimes \mathrm{l}[a] = (\iota \otimes a) \cdot inv\ \mathrm{a}[\mathcal{I}, \mathcal{I}, a]$

⟨*proof*⟩

lemma *runit-char*:
assumes *ide a*
shows $\mathrm{r}[a] \otimes \mathcal{I} = (a \otimes \iota) \cdot \mathrm{a}[a, \mathcal{I}, \mathcal{I}]$
⟨*proof*⟩

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

lemma ϱ-*agreement*:
shows $MC.\varrho = (\lambda f.\ \textit{if arr f then } f \cdot \mathrm{r}[\textit{dom } f]\ \textit{else null})$
⟨*proof*⟩

lemma *lunit-agreement*:
assumes *ide a*
shows $MC.\textit{lunit } a = \mathrm{l}[a]$
⟨*proof*⟩

lemma *runit-agreement*:
assumes *ide a*
shows $MC.\textit{runit } a = \mathrm{r}[a]$
⟨*proof*⟩

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*: $[\![\ \textit{ide } a0;\ \textit{ide } a1;\ \textit{ide } a2\ ]\!] \Longrightarrow \textit{ide } \mathrm{a}[a0,\ a1,\ a2]$
and *strict-lunit*: $\textit{ide } a \Longrightarrow \mathrm{l}[a] = a$
and *strict-runit*: $\textit{ide } a \Longrightarrow \mathrm{r}[a] = a$
begin

lemma *strict-unit*:
shows $\iota = \mathcal{I}$
⟨*proof*⟩

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*⟩

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 T_C α_C ι
for C :: *'a comp*   (**infixr** ‹·› *55*)
and T_C :: $'a * 'a \Rightarrow 'a$
and α_C :: $'a * 'a * 'a \Rightarrow 'a$
and ι :: $'a$
begin

  abbreviation T
  where $T\,f \equiv T_C$ (*snd f*, *fst f*)

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

end

sublocale *opposite-monoidal-category* ⊆ *monoidal-category* C T α ι
⟨*proof*⟩

context *opposite-monoidal-category*
begin

  lemma *lunit-simp*:
  assumes *C.ide a*
  shows *lunit a* = *C.runit a*
    ⟨*proof*⟩

  lemma *runit-simp*:
  assumes *C.ide a*
  shows *runit a* = *C.lunit a*
    ⟨*proof*⟩

end

2.5 Dual Monoidal Category

The *dual* of a monoidal category is obtained by reversing the arrows of the underlying category. The tensor product remains the same, but the associators and unitors are inverted.

locale *dual-monoidal-category* =
  M: *monoidal-category*
begin

sublocale *dual-category C* ⟨*proof*⟩
sublocale *MM*: *product-category comp comp* ⟨*proof*⟩
interpretation *T*: *binary-functor comp comp comp T*
⟨*proof*⟩
interpretation *T*: *binary-endofunctor comp* ⟨*proof*⟩
interpretation *ToTC*: *functor T.CCC.comp comp T.ToTC*
⟨*proof*⟩
interpretation *ToCT*: *functor T.CCC.comp comp T.ToCT*
⟨*proof*⟩
interpretation α: *natural-transformation T.CCC.comp comp T.ToTC T.ToCT* $M.\alpha'$
⟨*proof*⟩
interpretation α: *natural-isomorphism T.CCC.comp comp T.ToTC T.ToCT* $M.\alpha'$
⟨*proof*⟩
interpretation *L*: *equivalence-functor comp comp* ‹*M.tensor* (*cod* (*M.inv* ι))›
⟨*proof*⟩
interpretation *R*: *equivalence-functor comp comp* ‹λf. *M.tensor f* (*cod* (*M.inv* ι))›
⟨*proof*⟩

sublocale *monoidal-category comp T* $M.\alpha'$ ‹*M.inv* ι›
⟨*proof*⟩

lemma *is-monoidal-category*:
shows *monoidal-category comp T* $M.\alpha'$ (*M.inv* ι)
⟨*proof*⟩

no-notation *comp* (**infixr** ‹·› *55*)

lemma *assoc-char*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *assoc a b c* = *M.assoc′ a b c* **and** *assoc′ a b c* = *M.assoc a b c*
⟨*proof*⟩

lemma *lunit-char*:
assumes *ide a*
shows *lunit a* = *M.lunit′ a*
⟨*proof*⟩

lemma *runit-char*:
assumes *ide a*
shows *runit a* = *M.runit′ a*
⟨*proof*⟩

end

2.6 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*
    ⟨*proof*⟩

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*

| $Cod\ \mathbf{r}^{-1}[t] = (Cod\ t \otimes \mathcal{I})$
| $Cod\ \mathbf{a}[t,\ u,\ v] = (Cod\ t \otimes (Cod\ u \otimes Cod\ v))$
| $Cod\ \mathbf{a}^{-1}[t,\ u,\ v] = ((Cod\ t \otimes Cod\ u) \otimes Cod\ v)$

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

abbreviation *Hom* :: *'a term* ⇒ *'a term* ⇒ *'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* ⇒ *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-implies-Arr* [*simp*]:
shows $Ide\ t \Longrightarrow Arr\ t$
⟨*proof*⟩

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

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

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

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

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

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

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

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 $Inv\ \langle f \rangle = \langle C.inv\ f \rangle$
| $Inv\ \boldsymbol{\mathcal{I}} = \boldsymbol{\mathcal{I}}$
| $Inv\ (t \otimes u) = (Inv\ t \otimes Inv\ u)$
| $Inv\ (t \cdot u) = (Inv\ u \cdot Inv\ t)$
| $Inv\ \mathbf{l}[t] = \mathbf{l}^{-1}[Inv\ t]$
| $Inv\ \mathbf{l}^{-1}[t] = \mathbf{l}[Inv\ t]$
| $Inv\ \mathbf{r}[t] = \mathbf{r}^{-1}[Inv\ t]$
| $Inv\ \mathbf{r}^{-1}[t] = \mathbf{r}[Inv\ t]$
| $Inv\ \mathbf{a}[t,\ u,\ v] = \mathbf{a}^{-1}[Inv\ t,\ Inv\ u,\ Inv\ v]$
| $Inv\ \mathbf{a}^{-1}[t,\ u,\ v] = \mathbf{a}[Inv\ t,\ Inv\ u,\ Inv\ v]$

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

lemma *Inv-preserves-Can*:

assumes *Can t*
shows *Can* (*Inv t*) **and** *Dom* (*Inv t*) = *Cod t* **and** *Cod* (*Inv t*) = *Dom t*
⟨*proof*⟩

lemma *Inv-in-Hom* [*simp*]:
assumes *Can t*
shows *Inv t* ∈ *Hom* (*Cod t*) (*Dom t*)
⟨*proof*⟩

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

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

We call a term "diagonal" if it is either $\boldsymbol{\mathcal{I}}$ or it is constructed from arrows of C using only the ⊗ operator associated to the right. Essentially, such terms are lists of arrows of C, where $\boldsymbol{\mathcal{I}}$ represents the empty list and ⊗ is used as the list constructor. We call them "diagonal" because terms can 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* $\boldsymbol{\mathcal{I}}$ = *True*
| *Diag* ⟨*f*⟩ = *C.arr f*
| *Diag* (⟨*f*⟩ ⊗ *u*) = (*C.arr f* ∧ *Diag u* ∧ *u* ≠ $\boldsymbol{\mathcal{I}}$)
| *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* ≠ $\boldsymbol{\mathcal{I}}$
⟨*proof*⟩

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

lemma *Dom-preserves-Diag*:
shows *Diag t* ⟹ *Diag* (*Dom t*)
⟨*proof*⟩

lemma *Cod-preserves-Diag*:
shows *Diag t* ⟹ *Diag* (*Cod t*)
⟨*proof*⟩

lemma *Inv-preserves-Diag*:
assumes *Can t* **and** *Diag t*
shows *Diag* (*Inv t*)
⟨*proof*⟩

The following function defines the "dimension" of a term, which is the number of arrows of (·) it contains. For diagonal terms, this is just the length of the term when regarded as a list of arrows of (·). 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* ⇒ *nat*
where *dim* ⟨*f*⟩ = *1*
| *dim* $\mathcal{I}$ = *0*
| *dim* (*t* ⊗ *u*) = (*dim t* + *dim u*)
| *dim* (*t* · *u*) = *dim t*
| *dim* **l**[*t*] = *dim t*
| *dim* $\mathbf{l}^{-1}$[*t*] = *dim t*
| *dim* **r**[*t*] = *dim t*
| *dim* $\mathbf{r}^{-1}$[*t*] = *dim t*
| *dim* **a**[*t*, *u*, *v*] = *dim t* + *dim u* + *dim v*
| *dim* $\mathbf{a}^{-1}$[*t*, *u*, *v*] = *dim t* + *dim u* + *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** ‹⌊⊗⌋› *53*)
where $\mathcal{I}$ ⌊⊗⌋ *u* = *u*
| *t* ⌊⊗⌋ $\mathcal{I}$ = *t*
| ⟨*f*⟩ ⌊⊗⌋ *u* = ⟨*f*⟩ ⊗ *u*
| (*t* ⊗ *u*) ⌊⊗⌋ *v* = *t* ⌊⊗⌋ (*u* ⌊⊗⌋ *v*)
| *t* ⌊⊗⌋ *u* = *undefined*

lemma *TensorDiag-Prim* [*simp*]:
assumes *t* ≠ $\mathcal{I}$
shows ⟨*f*⟩ ⌊⊗⌋ *t* = ⟨*f*⟩ ⊗ *t*
⟨*proof*⟩

lemma *TensorDiag-term-Unity* [*simp*]:
shows *t* ⌊⊗⌋ $\mathcal{I}$ = *t*
⟨*proof*⟩

lemma *TensorDiag-Diag*:
assumes *Diag* (*t* ⊗ *u*)
shows *t* ⌊⊗⌋ *u* = *t* ⊗ *u*
⟨*proof*⟩

lemma *TensorDiag-preserves-Diag*:
assumes *Diag t* **and** *Diag u*
shows *Diag* (*t* ⌊⊗⌋ *u*)
and *Dom* (*t* ⌊⊗⌋ *u*) = *Dom t* ⌊⊗⌋ *Dom u*

and *Cod* (*t* ⌊⊗⌋ *u*) = *Cod t* ⌊⊗⌋ *Cod u*
⟨*proof*⟩

lemma *TensorDiag-in-Hom*:
assumes *Diag t* **and** *Diag u*
shows *t* ⌊⊗⌋ *u* ∈ *Hom* (*Dom t* ⌊⊗⌋ *Dom u*) (*Cod t* ⌊⊗⌋ *Cod u*)
⟨*proof*⟩

lemma *Dom-TensorDiag*:
assumes *Diag t* **and** *Diag u*
shows *Dom* (*t* ⌊⊗⌋ *u*) = *Dom t* ⌊⊗⌋ *Dom u*
⟨*proof*⟩

lemma *Cod-TensorDiag*:
assumes *Diag t* **and** *Diag u*
shows *Cod* (*t* ⌊⊗⌋ *u*) = *Cod t* ⌊⊗⌋ *Cod u*
⟨*proof*⟩

lemma *not-is-Tensor-TensorDiagE*:
assumes ¬ *is-Tensor* (*t* ⌊⊗⌋ *u*) **and** *Diag t* **and** *Diag u*
and *t* ≠ ***I*** **and** *u* ≠ ***I***
shows *False*
⟨*proof*⟩

lemma *TensorDiag-assoc*:
assumes *Diag t* **and** *Diag u* **and** *Diag v*
shows (*t* ⌊⊗⌋ *u*) ⌊⊗⌋ *v* = *t* ⌊⊗⌋ (*u* ⌊⊗⌋ *v*)
⟨*proof*⟩

lemma *TensorDiag-preserves-Ide*:
assumes *Ide t* **and** *Ide u* **and** *Diag t* **and** *Diag u*
shows *Ide* (*t* ⌊⊗⌋ *u*)
⟨*proof*⟩

lemma *TensorDiag-preserves-Can*:
assumes *Can t* **and** *Can u* **and** *Diag t* **and** *Diag u*
shows *Can* (*t* ⌊⊗⌋ *u*)
⟨*proof*⟩

lemma *Inv-TensorDiag*:
assumes *Can t* **and** *Can u* **and** *Diag t* **and** *Diag u*
shows *Inv* (*t* ⌊⊗⌋ *u*) = *Inv t* ⌊⊗⌋ *Inv u*
⟨*proof*⟩

The following function defines composition for compatible diagonal terms, by "pushing the composition down" to arrows of *C*.

fun *CompDiag* :: *'a term* ⇒ *'a term* ⇒ *'a term* (**infixr** ‹⌊·⌋› *55*)
where ***I*** ⌊·⌋ *u* = *u*
| ⟨*f*⟩ ⌊·⌋ ⟨*g*⟩ = ⟨*f* · *g*⟩

| $(u \otimes v)$ ⌊·⌋ $(w \otimes x) = (u$ ⌊·⌋ $w \otimes v$ ⌊·⌋ $x)$
| t ⌊·⌋ $\boldsymbol{\mathcal{I}} = t$
| t ⌊·⌋ - = *undefined* · *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* ⌊·⌋ *u*)
and *Dom* (*t* ⌊·⌋ *u*) = *Dom u*
and *Cod* (*t* ⌊·⌋ *u*) = *Cod t*
⟨*proof*⟩

lemma *CompDiag-in-Hom*:
assumes *Diag t* **and** *Diag u* **and** *Dom t* = *Cod u*
shows *t* ⌊·⌋ *u* ∈ *Hom* (*Dom u*) (*Cod t*)
⟨*proof*⟩

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

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

lemma *CompDiag-Cod-Diag* [*simp*]:
assumes *Diag t*
shows *Cod t* ⌊·⌋ *t* = *t*
⟨*proof*⟩

lemma *CompDiag-Diag-Dom* [*simp*]:
assumes *Diag t*
shows *t* ⌊·⌋ *Dom t* = *t*
⟨*proof*⟩

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

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

lemma *CompDiag-assoc*:

assumes *Diag t* **and** *Diag u* **and** *Diag v*
and *Dom t* = *Cod u* **and** *Dom u* = *Cod v*
shows (*t* ⌊·⌋ *u*) ⌊·⌋ *v* = *t* ⌊·⌋ (*u* ⌊·⌋ *v*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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

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

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

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* ⌊⊗⌋ *u*) ⌊·⌋ (*v* ⌊⊗⌋ *w*) = (*t* ⌊·⌋ *v*) ⌊⊗⌋ (*u* ⌊·⌋ *w*)
⟨*proof*⟩

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* ⇒ *'a term* (‹⌊-⌋›)
where ⌊⟨*f*⟩⌋ = ⟨*f*⟩
| ⌊***I***⌋ = ***I***
| ⌊*t* ⊗ *u*⌋ = ⌊*t*⌋ ⌊⊗⌋ ⌊*u*⌋
| ⌊*t* · *u*⌋ = ⌊*t*⌋ ⌊·⌋ ⌊*u*⌋

| $\lfloor \mathbf{l}[t] \rfloor = \lfloor t \rfloor$
| $\lfloor \mathbf{l}^{-1}[t] \rfloor = \lfloor t \rfloor$
| $\lfloor \mathbf{r}[t] \rfloor = \lfloor t \rfloor$
| $\lfloor \mathbf{r}^{-1}[t] \rfloor = \lfloor t \rfloor$
| $\lfloor \mathbf{a}[t, u, v] \rfloor = (\lfloor t \rfloor \lfloor\otimes\rfloor \lfloor u \rfloor) \lfloor\otimes\rfloor \lfloor v \rfloor$
| $\lfloor \mathbf{a}^{-1}[t, u, v] \rfloor = \lfloor t \rfloor \lfloor\otimes\rfloor (\lfloor u \rfloor \lfloor\otimes\rfloor \lfloor v \rfloor)$

lemma *Diag-Diagonalize*:
assumes *Arr t*
shows *Diag* $\lfloor t \rfloor$ **and** *Dom* $\lfloor t \rfloor = \lfloor$*Dom t*$\rfloor$ **and** *Cod* $\lfloor t \rfloor = \lfloor$*Cod t*$\rfloor$
⟨*proof*⟩

lemma *Diagonalize-in-Hom*:
assumes *Arr t*
shows $\lfloor t \rfloor \in$ *Hom* $\lfloor$*Dom t*$\rfloor$ $\lfloor$*Cod t*$\rfloor$
⟨*proof*⟩

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

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

lemma *Diagonalize-preserves-Ide*:
assumes *Ide a*
shows *Ide* $\lfloor a \rfloor$
⟨*proof*⟩

The diagonalizations of canonical arrows are identities.

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

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

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

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

assumes *Arr t*
shows $\lfloor\lfloor t\rfloor\rfloor = \lfloor t\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Tensor*:
assumes *Arr t* **and** *Arr u*
shows $\lfloor t \otimes u\rfloor = \lfloor\lfloor t\rfloor \otimes \lfloor u\rfloor\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Tensor-Unity-Arr* [*simp*]:
assumes *Arr u*
shows $\lfloor \boldsymbol{\mathcal{I}} \otimes u\rfloor = \lfloor u\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Tensor-Arr-Unity* [*simp*]:
assumes *Arr t*
shows $\lfloor t \otimes \boldsymbol{\mathcal{I}}\rfloor = \lfloor t\rfloor$
⟨*proof*⟩

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

lemma *Diagonalize-Tensor-Tensor*:
assumes *Arr t* **and** *Arr u* **and** *Arr v*
shows $\lfloor (t \otimes u) \otimes v\rfloor = \lfloor\lfloor t\rfloor \otimes (\lfloor u\rfloor \otimes \lfloor v\rfloor)\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Comp-Cod-Arr*:
assumes *Arr t*
shows $\lfloor \mathit{Cod}\ t \cdot t\rfloor = \lfloor t\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Comp-Arr-Dom*:
assumes *Arr t*
shows $\lfloor t \cdot \mathit{Dom}\ t\rfloor = \lfloor t\rfloor$
⟨*proof*⟩

lemma *Diagonalize-Inv*:
assumes *Can t*
shows $\lfloor \mathit{Inv}\ t\rfloor = \mathit{Inv}\ \lfloor t\rfloor$
⟨*proof*⟩

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 $\lfloor t\rfloor$ are opposite sides of a square whose other sides are certain canonical terms $\mathit{Dom}\ t{\Downarrow} \in \mathit{Hom}\ (\mathit{Dom}\ t)\ \lfloor \mathit{Dom}\ t\rfloor$ and $\mathit{Cod}\ t{\Downarrow} \in \mathit{Hom}\ (\mathit{Cod}\ t)\ \lfloor \mathit{Cod}\ t\rfloor$, where $\mathit{Dom}\ t{\Downarrow}$ and $\mathit{Cod}\ t{\Downarrow}$ 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* $a\ \lfloor a \rfloor$. The auxiliary function *red2* takes a pair $(a,\ b)$ of diagonal identity terms and produces a canonical arrow $a \Downarrow b \in$ *Hom* $(a \otimes b)\ \lfloor a \otimes b \rfloor$. The canonical arrow $a{\downarrow}$ amounts to a "parallel innermost reduction" from a to $\lfloor a \rfloor$, 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 $\lfloor a \rfloor$ 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 $\lfloor a \rfloor$.

fun *red2* (**infixr** ‹⇓› *53*)
where $\mathcal{I} \Downarrow a = \mathbf{l}[a]$
| $\langle f \rangle \Downarrow \mathcal{I} = \mathbf{r}[\langle f \rangle]$
| $\langle f \rangle \Downarrow a = \langle f \rangle \otimes a$
| $(a \otimes b) \Downarrow \mathcal{I} = \mathbf{r}[a \otimes b]$
| $(a \otimes b) \Downarrow c = (a \Downarrow \lfloor b \otimes c \rfloor) \cdot (a \otimes (b \Downarrow c)) \cdot \mathbf{a}[a,\ b,\ c]$
| $a \Downarrow b =$ *undefined*

fun *red* (‹-↓› *[56] 56*)
where $\mathcal{I}{\downarrow} = \mathcal{I}$
| $\langle f \rangle{\downarrow} = \langle f \rangle$
| $(a \otimes b){\downarrow} = ($*if Diag* $(a \otimes b)$ *then* $a \otimes b$ *else* $(\lfloor a \rfloor \Downarrow \lfloor b \rfloor) \cdot (a{\downarrow} \otimes b{\downarrow}))$
| $a{\downarrow} =$ *undefined*

lemma *red-Diag* [*simp*]:
assumes *Diag* a
shows $a{\downarrow} = a$
⟨*proof*⟩

lemma *red2-Diag*:
assumes *Diag* $(a \otimes b)$
shows $a \Downarrow b = a \otimes b$

⟨*proof*⟩

lemma *Can-red2*:
assumes *Ide a* **and** *Diag a* **and** *Ide b* **and** *Diag b*
shows *Can* $(a \Downarrow b)$
and $a \Downarrow b \in Hom\ (a \otimes b)\ \lfloor a \otimes b \rfloor$
⟨*proof*⟩

lemma *red2-in-Hom*:
assumes *Ide a* **and** *Diag a* **and** *Ide b* **and** *Diag b*
shows $a \Downarrow b \in Hom\ (a \otimes b)\ \lfloor a \otimes b \rfloor$
⟨*proof*⟩

lemma *Can-red*:
assumes *Ide a*
shows *Can* $(a\downarrow)$ **and** $a\downarrow \in Hom\ a\ \lfloor a \rfloor$
⟨*proof*⟩

lemma *red-in-Hom*:
assumes *Ide a*
shows $a\downarrow \in Hom\ a\ \lfloor a \rfloor$
⟨*proof*⟩

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

lemma *Diagonalize-red2* [*simp*]:
assumes *Ide a* **and** *Ide b* **and** *Diag a* **and** *Diag b*
shows $\lfloor a \Downarrow b \rfloor = \lfloor a \otimes b \rfloor$
⟨*proof*⟩

end

2.7 Coherence

If D is a monoidal category, then a functor $V\colon C \to D$ 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 α ι +
V: *functor C D V*
for C :: *'c comp* (**infixr** $\langle\cdot_C\rangle$ *55*)
and D :: *'d comp* (**infixr** $\langle\cdot\rangle$ *55*)
and T :: $'d * 'd \Rightarrow 'd$
and α :: $'d * 'd * 'd \Rightarrow 'd$

and ι :: $'d$
and V :: $'c \Rightarrow 'd$
begin

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

notation *unity* (‹$\mathcal{I}$›)
notation *runit* (‹r[-]›)
notation *lunit* (‹l[-]›)
notation *assoc'* (‹a^{-1}[-, -, -]›)
notation *runit'* (‹r^{-1}[-]›)
notation *lunit'* (‹l^{-1}[-]›)

primrec *eval* :: $'c$ *term* $\Rightarrow 'd$ (‹⦃-⦄›)
where ⦃⟨f⟩⦄ = $V\,f$
| ⦃$\mathcal{I}$⦄ = $\mathcal{I}$
| ⦃$t \otimes u$⦄ = ⦃t⦄ $\otimes$ ⦃u⦄
| ⦃$t \cdot u$⦄ = ⦃t⦄ $\cdot$ ⦃u⦄
| ⦃**l**[t]⦄ = $\mathfrak{l}$ ⦃t⦄
| ⦃$\mathbf{l}^{-1}$[t]⦄ = $\mathfrak{l}'$ ⦃t⦄
| ⦃**r**[t]⦄ = ϱ ⦃t⦄
| ⦃$\mathbf{r}^{-1}$[t]⦄ = ϱ' ⦃t⦄
| ⦃**a**[t, u, v]⦄ = α (⦃t⦄, ⦃u⦄, ⦃v⦄)
| ⦃$\mathbf{a}^{-1}$[t, u, v]⦄ = α' (⦃t⦄, ⦃u⦄, ⦃v⦄)

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

lemma *ide-eval-Ide* [*simp*]:
shows *Ide t* $\Longrightarrow$ *ide* ⦃t⦄
⟨*proof*⟩

lemma *eval-in-hom*:
shows *Arr t* $\Longrightarrow$ «⦃t⦄ : ⦃*Dom t*⦄ → ⦃*Cod t*⦄»
⟨*proof*⟩

lemma *arr-eval* [*simp*]:
assumes *Arr f*
shows *arr* ⦃f⦄
⟨*proof*⟩

lemma *dom-eval* [*simp*]:
assumes *Arr f*
shows *dom* ⦃f⦄ = ⦃*Dom f*⦄
⟨*proof*⟩

lemma *cod-eval* [*simp*]:
assumes *Arr f*
shows *cod* ⦃f⦄ = ⦃*Cod f*⦄
⟨*proof*⟩

lemma *eval-Prim* [*simp*]:
assumes *C.arr f*
shows $\{\!\!\{\langle f\rangle\}\!\!\} = V\ f$
⟨*proof*⟩

lemma *eval-Tensor* [*simp*]:
assumes *Arr t* **and** *Arr u*
shows $\{\!\!\{t \otimes u\}\!\!\} = \{\!\!\{t\}\!\!\} \otimes \{\!\!\{u\}\!\!\}$
⟨*proof*⟩

lemma *eval-Comp* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Dom t = Cod u*
shows $\{\!\!\{t \cdot u\}\!\!\} = \{\!\!\{t\}\!\!\} \cdot \{\!\!\{u\}\!\!\}$
⟨*proof*⟩

lemma *eval-Lunit* [*simp*]:
assumes *Arr t*
shows $\{\!\!\{\mathbf{l}[t]\}\!\!\} = \mathrm{l}[\{\!\!\{Cod\ t\}\!\!\}] \cdot (\mathcal{I} \otimes \{\!\!\{t\}\!\!\})$
⟨*proof*⟩

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

lemma *eval-Runit* [*simp*]:
assumes *Arr t*
shows $\{\!\!\{\mathbf{r}[t]\}\!\!\} = \mathrm{r}[\{\!\!\{Cod\ t\}\!\!\}] \cdot (\{\!\!\{t\}\!\!\} \otimes \mathcal{I})$
⟨*proof*⟩

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

lemma *eval-Assoc* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Arr v*
shows $\{\!\!\{\mathbf{a}[t, u, v]\}\!\!\} = \mathrm{a}[cod\ \{\!\!\{t\}\!\!\}, cod\ \{\!\!\{u\}\!\!\}, cod\ \{\!\!\{v\}\!\!\}] \cdot ((\{\!\!\{t\}\!\!\} \otimes \{\!\!\{u\}\!\!\}) \otimes \{\!\!\{v\}\!\!\})$
⟨*proof*⟩

lemma *eval-Assoc′* [*simp*]:
assumes *Arr t* **and** *Arr u* **and** *Arr v*
shows $\{\!\!\{\mathbf{a}^{-1}[t, u, v]\}\!\!\} = \mathrm{a}^{-1}[cod\ \{\!\!\{t\}\!\!\}, cod\ \{\!\!\{u\}\!\!\}, cod\ \{\!\!\{v\}\!\!\}] \cdot (\{\!\!\{t\}\!\!\} \otimes \{\!\!\{u\}\!\!\} \otimes \{\!\!\{v\}\!\!\})$
⟨*proof*⟩

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 $\{\!|\mathbf{l}[a]|\!\} = \mathrm{l}[\{\!|a|\!\}]$
⟨*proof*⟩

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

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

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

lemma *eval-Assoc-Ide* [*simp*]:
assumes *Ide a* **and** *Ide b* **and** *Ide c*
shows $\{\!|\mathbf{a}[a,\ b,\ c]|\!\} = \mathrm{a}[\{\!|a|\!\},\ \{\!|b|\!\},\ \{\!|c|\!\}]$
⟨*proof*⟩

lemma *eval-Assoc′-Ide* [*simp*]:
assumes *Ide a* **and** *Ide b* **and** *Ide c*
shows $\{\!|\mathbf{a}^{-1}[a,\ b,\ c]|\!\} = \mathrm{a}^{-1}[\{\!|a|\!\},\ \{\!|b|\!\},\ \{\!|c|\!\}]$
⟨*proof*⟩

Canonical arrows evaluate to isomorphisms in D, and formal inverses evaluate to inverses in D.

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

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

The operation $\lfloor\cdot\rfloor$ evaluates to composition in D.

lemma *eval-CompDiag*:
assumes *Diag t* **and** *Diag u* **and** *Seq t u*
shows $\{\!|t\ \lfloor\cdot\rfloor\ u|\!\} = \{\!|t|\!\} \cdot \{\!|u|\!\}$
⟨*proof*⟩

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*⟩

lemma *eval-red2-Diag-Unity*:
assumes *Ide a* **and** *Diag a*
shows $\{\!\!\{a \Downarrow \mathcal{I}\}\!\!\} = \mathrm{r}[\{\!\!\{a\}\!\!\}]$
⟨*proof*⟩

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*
⟨*proof*⟩

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* $t \longleftrightarrow \{\!\!\{t\}\!\!\} = inv\ \{\!\!\{Cod\ t{\downarrow}\}\!\!\} \cdot \{\!\!\{\lfloor t\rfloor\}\!\!\} \cdot \{\!\!\{Dom\ t{\downarrow}\}\!\!\}$
⟨*proof*⟩

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*⟩

Some special cases of coherence are readily dispatched.

lemma *coherent-Unity*:
shows *coherent* $\mathcal{I}$
⟨*proof*⟩

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

lemma *coherent-Lunit-Ide*:

assumes *Ide a*
shows *coherent* $\mathbf{l}[a]$
⟨*proof*⟩

lemma *coherent-Runit-Ide*:
assumes *Ide a*
shows *coherent* $\mathbf{r}[a]$
⟨*proof*⟩

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

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

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 $((a \lfloor\otimes\rfloor\ b) \Downarrow c) \cdot ((a \Downarrow b) \otimes c)$ 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 $(a \Downarrow b \lfloor\otimes\rfloor\ c) \cdot (a \otimes b \Downarrow c)$ 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 $\{\!|(a \lfloor\otimes\rfloor\ b) \Downarrow c|\!\} \cdot (\{\!|a \Downarrow b|\!\} \otimes \{\!|c|\!\})$
$= (\{\!|a \Downarrow (b \lfloor\otimes\rfloor\ c)|\!\} \cdot (\{\!|a|\!\} \otimes \{\!|b \Downarrow c|\!\})) \cdot \mathrm{a}[\{\!|a|\!\}, \{\!|b|\!\}, \{\!|c|\!\}]$
⟨*proof*⟩

lemma *coherent-Assoc-Ide*:
assumes *Ide a* **and** *Ide b* **and** *Ide c*
shows *coherent* $\mathbf{a}[a, b, c]$
⟨*proof*⟩

lemma *coherent-Assoc′-Ide*:
assumes *Ide a* **and** *Ide b* **and** *Ide c*
shows *coherent* $\mathbf{a}^{-1}[a, b, c]$
⟨*proof*⟩

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 $\{\!|Cod\ t \Downarrow Cod\ u|\!\} \cdot (\{\!|t|\!\} \otimes \{\!|u|\!\}) = \{\!|t \lfloor\otimes\rfloor\ u|\!\} \cdot \{\!|Dom\ t \Downarrow Dom\ u|\!\}$
⟨*proof*⟩

lemma *Tensor-preserves-coherent*:

assumes *Arr t* **and** *Arr u* **and** *coherent t* **and** *coherent u*
shows *coherent* $(t \otimes u)$
⟨*proof*⟩

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*⟩

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

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

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

Our final corollary expresses coherence in a more "MacLane-like" fashion: parallel canonical arrows are equivalent under evaluation.

corollary *maclane-coherence*:
assumes *Par t u* **and** *Can t* **and** *Can u*
shows $\{\!|t|\!\} = \{\!|u|\!\}$
⟨*proof*⟩

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 φ that expresses the preservation of tensor product and a natural isomorphism ψ that expresses the preservation of the unit object. These natural isomorphisms are subject to coherence conditions; the condition for φ involving the associator and the conditions for ψ involving the unitors. However, as pointed out in [2] (Section 2.4), it is not necessary to take the natural isomorphism ψ 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 ψ 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_DoFF: *composite-functor* $C.CC.comp\ D.CC.comp\ D\ FF.map\ T_D$ +
  φ: *natural-isomorphism* $C.CC.comp\ D\ T_DoFF.map\ FoT_C.map\ \varphi$
for C :: $'c$ *comp* (**infixr** $\langle\cdot_C\rangle$ *55*)
and T_C :: $'c * 'c \Rightarrow 'c$
and α_C :: $'c * 'c * 'c \Rightarrow 'c$
and ι_C :: $'c$
and D :: $'d$ *comp* (**infixr** $\langle\cdot_D\rangle$ *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$

and F :: $'c \Rightarrow 'd$
and φ :: $'c * 'c \Rightarrow 'd$ +
assumes *preserves-unity*: *D.isomorphic D.unity* (F *C.unity*)
and *assoc-coherence*:
$[\![$ *C.ide a*; *C.ide b*; *C.ide c* $]\!] \Longrightarrow$
$F\ (\alpha_C\ (a,\ b,\ c)) \cdot_D \varphi\ (T_C\ (a,\ b),\ c) \cdot_D T_D\ (\varphi\ (a,\ b),\ F\ c)$
$= \varphi\ (a,\ T_C\ (b,\ c)) \cdot_D T_D\ (F\ a,\ \varphi\ (b,\ c)) \cdot_D \alpha_D\ (F\ a,\ F\ b,\ F\ c)$
begin

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 *φ-in-hom*:
assumes *C.ide a* **and** *C.ide b*
shows «$\varphi\ (a,\ b) : F\ a \otimes_D F\ b \rightarrow_D F\ (a \otimes_C b)$»
⟨*proof*⟩

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 ψ is defined by the equation $\mathrm{l}_D[F\ \mathcal{I}_C] = F\ \mathrm{l}_C[\mathcal{I}_C] \cdot_D \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C) \cdot_D (\psi \otimes_D F\ \mathcal{I}_C)$, 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 $\mathrm{l}_D[F\ a] = F\ \mathrm{l}_C[a] \cdot_D \varphi\ (\mathcal{I}_C,\ a) \cdot_D (\psi \otimes_D F\ a)$ 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 $\mathrm{l}_D[F\ \mathcal{I}_C] = F\ \mathrm{l}_C[\mathcal{I}_C] \cdot_D \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C) \cdot_D (\psi \otimes_D F\ \mathcal{I}_C)$ used as the definition of ψ. 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 ι_D to $\iota_D{}'$. We define ψ to be this isomorphism and then use the definition to establish the desired coherence conditions.

abbreviation ι_1
where $\iota_1 \equiv F\ \iota_C \cdot_D \varphi\ (\mathcal{I}_C,\ \mathcal{I}_C)$

lemma *ι_1-in-hom*:
shows «$\iota_1 : F\ \mathcal{I}_C \otimes_D F\ \mathcal{I}_C \rightarrow_D F\ \mathcal{I}_C$»
⟨*proof*⟩

lemma *ι_1-is-iso*:
shows *D.iso* ι_1

⟨*proof*⟩

interpretation D: *monoidal-category-with-alternate-unit* $D\ T_D\ \alpha_D\ \iota_D\ \iota_1$
⟨*proof*⟩

no-notation $D.tensor$ (**infixr** ‹$\otimes_D$› *53*)
notation $D.C_1.tensor$ (**infixr** ‹$\otimes_D$› *53*)
no-notation $D.assoc$ (‹a_D[-, -, -]›)
notation $D.C_1.assoc$ (‹a_D[-, -, -]›)
no-notation $D.assoc'$ (‹$a_D{}^{-1}$[-, -, -]›)
notation $D.C_1.assoc'$ (‹$a_D{}^{-1}$[-, -, -]›)
notation $D.C_1.unity$ (‹$\mathcal{I}_1$›)
notation $D.C_1.lunit$ (‹l_1[-]›)
notation $D.C_1.runit$ (‹r_1[-]›)

lemma $\mathcal{I}_1$-*char* [*simp*]:
shows $\mathcal{I}_1 = F\ \mathcal{I}_C$
⟨*proof*⟩

definition ψ
where $\psi \equiv THE\ \psi.$ «$\psi : \mathcal{I}_D \to_D F\ \mathcal{I}_C$» $\wedge$ $D.iso\ \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$

lemma ψ-*char*:
shows «$\psi : \mathcal{I}_D \to_D F\ \mathcal{I}_C$» **and** $D.iso\ \psi$ **and** $\psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$
and $\exists!\psi.$ «$\psi : \mathcal{I}_D \to_D F\ \mathcal{I}_C$» $\wedge$ $D.iso\ \psi \wedge \psi \cdot_D \iota_D = \iota_1 \cdot_D (\psi \otimes_D \psi)$
⟨*proof*⟩

lemma ψ-*eqI*:
assumes «f: $\mathcal{I}_D \to_D F\ \mathcal{I}_C$» **and** $D.iso\ f$ **and** $f \cdot_D \iota_D = \iota_1 \cdot_D (f \otimes_D f)$
shows $f = \psi$
⟨*proof*⟩

lemma *lunit-coherence1*:
assumes $C.ide\ a$
shows $l_1[F\ a] \cdot_D (\psi \otimes_D F\ a) = l_D[F\ a]$
⟨*proof*⟩

lemma *lunit-coherence2*:
assumes $C.ide\ a$
shows $F\ l_C[a] \cdot_D \varphi\ (\mathcal{I}_C,\ a) = l_1[F\ a]$
⟨*proof*⟩

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 $C.ide\ a$
shows $l_D[F\ a] = F\ l_C[a] \cdot_D \varphi\ (\mathcal{I}_C,\ a) \cdot_D (\psi \otimes_D F\ a)$
⟨*proof*⟩

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* $C\ T_C\ \alpha_C\ \iota_C$ ⟨*proof*⟩
interpretation D': *opposite-monoidal-category* $D\ T_D\ \alpha_D\ \iota_D$ ⟨*proof*⟩
interpretation $T_D{}'oFF$: *composite-functor* $C.CC.comp\ D.CC.comp\ D\ FF.map\ D'.T$ ⟨*proof*⟩
interpretation $FoT_C{}'$: *composite-functor* $C.CC.comp\ C\ D\ C'.T\ F$ ⟨*proof*⟩
interpretation φ': *natural-transformation* $C.CC.comp\ D\ T_D{}'oFF.map\ FoT_C{}'.map$
‹$\lambda f.\ \varphi$ (*snd* f, *fst* f)›
⟨*proof*⟩
interpretation φ': *natural-isomorphism* $C.CC.comp\ D\ T_D{}'oFF.map\ FoT_C{}'.map$
‹$\lambda f.\ \varphi$ (*snd* f, *fst* f)›
⟨*proof*⟩
interpretation F': *monoidal-functor* $C\ C'.T\ C'.\alpha\ \iota_C\ D\ D'.T\ D'.\alpha\ \iota_D\ F$ ‹$\lambda f.\ \varphi$ (*snd* f, *fst* f)›
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *runit-coherence*:
assumes $C.ide\ a$
shows $\mathrm{r}_D[F\ a] = F\ \mathrm{r}_C[a] \cdot_D \varphi\ (a, \mathcal{I}_C) \cdot_D (F\ a \otimes_D \psi)$
⟨*proof*⟩

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 :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and T_C :: $'c * 'c \Rightarrow 'c$
and α_C :: $'c * 'c * 'c \Rightarrow 'c$
and ι_C :: $'c$
and D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and F :: $'c \Rightarrow 'd$ +

assumes *strictly-preserves-ι*: $F\ \iota_C = \iota_D$
and *strictly-preserves-T*: ⟦ $C.arr\ f$; $C.arr\ g$ ⟧ $\Longrightarrow F\ (T_C\ (f,\ g)) = T_D\ (F\ f,\ F\ g)$
and *strictly-preserves-α-ide*: ⟦ $C.ide\ a$; $C.ide\ b$; $C.ide\ c$ ⟧ $\Longrightarrow$
$F\ (\alpha_C\ (a,\ b,\ c)) = \alpha_D\ (F\ a,\ F\ b,\ F\ c)$
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 $F\ (f \otimes_C g) = F\ f \otimes_D F\ g$
⟨*proof*⟩

lemma *strictly-preserves-α*:
assumes $C.arr\ f$ **and** $C.arr\ g$ **and** $C.arr\ h$
shows $F\ (\alpha_C\ (f,\ g,\ h)) = \alpha_D\ (F\ f,\ F\ g,\ F\ h)$
⟨*proof*⟩

lemma *strictly-preserves-unity*:
shows $F\ \mathcal{I}_C = \mathcal{I}_D$
⟨*proof*⟩

lemma *strictly-preserves-assoc*:
assumes $C.arr\ a$ **and** $C.arr\ b$ **and** $C.arr\ c$
shows $F\ \mathrm{a}_C[a,\ b,\ c] = \mathrm{a}_D[F\ a,\ F\ b,\ F\ c]$
⟨*proof*⟩

lemma *strictly-preserves-lunit*:
assumes $C.ide\ a$
shows $F\ \mathrm{l}_C[a] = \mathrm{l}_D[F\ a]$
⟨*proof*⟩

lemma *strictly-preserves-runit*:
assumes $C.ide\ a$
shows $F\ \mathrm{r}_C[a] = \mathrm{r}_D[F\ a]$
⟨*proof*⟩

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 φ. For a strict monoidal functor, this is an identity transformation.

interpretation *FF*: *product-functor C C D D F F* ⟨*proof*⟩
interpretation FoT_C: *composite-functor C.CC.comp C D* T_C *F* ⟨*proof*⟩
interpretation $T_D oFF$: *composite-functor C.CC.comp D.CC.comp D FF.map* T_D ⟨*proof*⟩

lemma *structure-is-trivial*:
shows $T_D oFF.map = FoT_C.map$
⟨*proof*⟩

abbreviation φ **where** $\varphi \equiv T_D oFF.map$

lemma *structure-naturalityisomorphism*:
shows *natural-isomorphism C.CC.comp D* $T_D oFF.map$ $FoT_C.map$ φ
⟨*proof*⟩

end

A strict monoidal functor is a monoidal functor.

sublocale *strict-monoidal-functor* ⊆ *monoidal-functor* $C\ T_C\ \alpha_C\ \iota_C\ D\ T_D\ \alpha_D\ \iota_D\ F\ \varphi$
⟨*proof*⟩

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* $B\ T_B\ \alpha_B\ \iota_B\ D\ T_D\ \alpha_D\ \iota_D\ (G\ o\ F)$
⟨*proof*⟩

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* $C\ T_C\ \alpha_C\ \iota_C$ +
D: *monoidal-category* $D\ T_D\ \alpha_D\ \iota_D$ +
equivalence-of-categories $C\ D\ F\ G\ \eta\ \varepsilon$ +
monoidal-functor $D\ T_D\ \alpha_D\ \iota_D\ C\ T_C\ \alpha_C\ \iota_C\ F\ \varphi$
for C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and T_C :: $'c * 'c \Rightarrow 'c$
and α_C :: $'c * 'c * 'c \Rightarrow 'c$
and ι_C :: $'c$
and D :: *'d comp* (**infixr** ‹$\cdot_D$› *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and F :: $'d \Rightarrow 'c$
and φ :: $'d * 'd \Rightarrow 'c$
and ι :: $'c$
and G :: $'c \Rightarrow 'd$
and η :: $'d \Rightarrow 'd$
and ε :: $'c \Rightarrow 'c$

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 *MonoidalCategory.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* (‹«- : - $\rightarrow_C$ -»›)

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 $equiv\ t\ u \equiv Par\ t\ u \wedge \lfloor t \rfloor = \lfloor u \rfloor$

Arrows of $\mathcal{F}C$ will be the equivalence classes of formal arrows determined by the relation *equiv*. We define here the property of being an equivalence class of the relation *equiv*. Later we show that this property coincides with that of being an arrow of the category that we will construct.

type-synonym $'a\ arr = 'a\ term\ set$
definition ARR **where** $ARR\ f \equiv f \neq \{\} \wedge (\forall t.\ t \in f \longrightarrow f = Collect\ (equiv\ t))$

lemma *not-ARR-empty*:
shows $\neg ARR\ \{\}$
⟨*proof*⟩

lemma *ARR-eqI*:
assumes $ARR\ f$ **and** $ARR\ g$ **and** $f \cap g \neq \{\}$
shows $f = g$
⟨*proof*⟩

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 $Inv\ (Cod\ t\Downarrow) \cdot \lfloor t \rfloor \cdot Dom\ t\Downarrow$. However, the only specific properties of this definition we actually use are the three we have just stated.

definition *norm* (‹‖-‖›)
where $\|t\| = Inv\ (Cod\ t\Downarrow) \cdot \lfloor t \rfloor \cdot Dom\ t\Downarrow$

If t is a formal arrow, then t is equivalent to its normal form.

lemma *equiv-norm-Arr*:
assumes $Arr\ t$
shows $equiv\ \|t\|\ t$
⟨*proof*⟩

Equivalent arrows have identical normal forms.

lemma *norm-respects-equiv*:
assumes $equiv\ t\ u$
shows $\|t\| = \|u\|$
⟨*proof*⟩

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 $Can\ \|t\| \longleftrightarrow Ide\ \lfloor t \rfloor$
⟨*proof*⟩

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 *Can* $\|t\|$
⟨*proof*⟩

lemma *Par-Arr-norm*:
assumes *Arr t*
shows *Par* $\|t\|$ *t*
⟨*proof*⟩

lemma *Diagonalize-norm* [*simp*]:
assumes *Arr t*
shows $\lfloor \|t\| \rfloor = \lfloor t \rfloor$
⟨*proof*⟩

lemma *unique-norm*:
assumes *ARR f*
shows $\exists! t.\ \forall u.\ u \in f \longrightarrow \|u\| = t$
⟨*proof*⟩

lemma *Dom-norm*:
assumes *Arr t*
shows *Dom* $\|t\|$ = *Dom t*
⟨*proof*⟩

lemma *Cod-norm*:
assumes *Arr t*
shows *Cod* $\|t\|$ = *Cod t*
⟨*proof*⟩

lemma *norm-in-Hom*:
assumes *Arr t*
shows $\|t\| \in$ *Hom* (*Dom t*) (*Cod t*)
⟨*proof*⟩

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* ≡ $\|$*SOME t. t* ∈ *f*$\|$

lemma *rep-in-ARR*:
assumes *ARR f*
shows *rep f* ∈ *f*
⟨*proof*⟩

lemma *Arr-rep-ARR*:
assumes *ARR f*
shows *Arr* (*rep f*)

⟨*proof*⟩

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 ¬*Arr t*
shows *mkarr t* = {}
⟨*proof*⟩

lemma *ARR-mkarr*:
assumes *Arr t*
shows *ARR* (*mkarr t*)
⟨*proof*⟩

lemma *mkarr-memb-ARR*:
assumes *ARR f* **and** *t* ∈ *f*
shows *mkarr t* = *f*
⟨*proof*⟩

lemma *mkarr-rep-ARR* [*simp*]:
assumes *ARR f*
shows *mkarr* (*rep f*) = *f*
⟨*proof*⟩

lemma *Arr-in-mkarr*:
assumes *Arr t*
shows *t* ∈ *mkarr t*
⟨*proof*⟩

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* ⟷ *Arr t* ∧ *Arr u* ∧ ‖*t*‖ = ‖*u*‖
⟨*proof*⟩

lemma *norm-norm* [*simp*]:
assumes *Arr t*
shows ‖‖*t*‖‖ = ‖*t*‖
⟨*proof*⟩

lemma *norm-in-ARR*:
assumes *ARR f* **and** *t* ∈ *f*
shows ‖*t*‖ ∈ *f*
⟨*proof*⟩

lemma *norm-rep-ARR* [*simp*]:
assumes *ARR f*

shows $\|rep\ f\| = rep\ f$
⟨*proof*⟩

lemma *norm-memb-eq-rep-ARR*:
assumes *ARR f* **and** $t \in f$
shows *norm t* = *rep f*
⟨*proof*⟩

lemma *rep-mkarr*:
assumes *Arr f*
shows *rep* (*mkarr f*) = $\|f\|$
⟨*proof*⟩

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*
⟨*proof*⟩

We use canonical representatives to lift the formal domain and codomain functions from terms to equivalence classes.

abbreviation *DOM* **where** *DOM f* ≡ *Dom* (*rep f*)
abbreviation *COD* **where** *COD f* ≡ *Cod* (*rep f*)

lemma *DOM-mkarr*:
assumes *Arr t*
shows *DOM* (*mkarr t*) = *Dom t*
⟨*proof*⟩

lemma *COD-mkarr*:
assumes *Arr t*
shows *COD* (*mkarr t*) = *Cod t*
⟨*proof*⟩

A composition operation can now be defined on equivalence classes using the syntactic constructor *Comp*.

definition *comp* (**infixr** ‹·› *55*)
where *comp f g* ≡ (*if ARR f* ∧ *ARR g* ∧ *DOM f* = *COD g*
then mkarr ((*rep f*) · (*rep g*)) *else* {})

We commence the task of showing that the composition *comp* so defined determines a category.

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

notation *in-hom* (‹«- : - → -»›)

The empty set serves as the null for the composition.

lemma *null-char*:
shows $null = \{\}$
⟨*proof*⟩

lemma *ARR-comp*:
assumes $ARR\ f$ **and** $ARR\ g$ **and** $DOM\ f = COD\ g$
shows $ARR\ (f \cdot g)$
⟨*proof*⟩

lemma *DOM-comp* [*simp*]:
assumes $ARR\ f$ **and** $ARR\ g$ **and** $DOM\ f = COD\ g$
shows $DOM\ (f \cdot g) = DOM\ g$
⟨*proof*⟩

lemma *COD-comp* [*simp*]:
assumes $ARR\ f$ **and** $ARR\ g$ **and** $DOM\ f = COD\ g$
shows $COD\ (f \cdot g) = COD\ f$
⟨*proof*⟩

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*⟩

lemma *Comp-in-comp-ARR*:
assumes $ARR\ f$ **and** $ARR\ g$ **and** $DOM\ f = COD\ g$
and $t \in f$ **and** $u \in g$
shows $t \bullet u \in f \cdot g$
⟨*proof*⟩

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** $IDE\ f \equiv ARR\ f \wedge (\forall t.\ t \in f \longrightarrow Ide\ \lfloor t \rfloor) \wedge DOM\ f = COD\ f$

lemma *IDE-implies-ARR*:
assumes $IDE\ f$
shows $ARR\ f$
⟨*proof*⟩

lemma *IDE-mkarr-Ide*:
assumes $Ide\ a$
shows $IDE\ (mkarr\ a)$
⟨*proof*⟩

lemma *IDE-implies-ide*:
assumes $IDE\ a$
shows $ide\ a$
⟨*proof*⟩

lemma *ARR-iff-has-domain*:
shows $ARR\ f \longleftrightarrow domains\ f \neq \{\}$
⟨*proof*⟩

lemma *ARR-iff-has-codomain*:
shows $ARR\ f \longleftrightarrow codomains\ f \neq \{\}$
⟨*proof*⟩

lemma *arr-iff-ARR*:
shows $arr\ f \longleftrightarrow ARR\ f$
⟨*proof*⟩

The arrows of the category are the equivalence classes of formal arrows.

lemma *arr-char*:
shows $arr\ f \longleftrightarrow f \neq \{\} \wedge (\forall t.\ t \in f \longrightarrow f = mkarr\ t)$
⟨*proof*⟩

lemma *seq-char*:
shows $seq\ g\ f \longleftrightarrow g \cdot f \neq null$
⟨*proof*⟩

lemma *seq-char'*:
shows $seq\ g\ f \longleftrightarrow ARR\ f \wedge ARR\ g \wedge DOM\ g = COD\ f$
⟨*proof*⟩

Finally, we can show that the composition *comp* determines a category.

interpretation *category comp*
⟨*proof*⟩

lemma *mkarr-rep* [*simp*]:
assumes *arr f*
shows $mkarr\ (rep\ f) = f$
⟨*proof*⟩

lemma *arr-mkarr* [*simp*]:
assumes *Arr t*
shows $arr\ (mkarr\ t)$
⟨*proof*⟩

lemma *mkarr-memb*:
assumes $t \in f$ **and** *arr f*
shows *Arr t* **and** $mkarr\ t = f$
⟨*proof*⟩

lemma *rep-in-arr* [*simp*]:
assumes *arr f*
shows $rep\ f \in f$
⟨*proof*⟩

lemma *Arr-rep* [*simp*]:
assumes *arr f*
shows *Arr* (*rep f*)
⟨*proof*⟩

lemma *rep-in-Hom*:
assumes *arr f*
shows *rep f* ∈ *Hom* (*DOM f*) (*COD f*)
⟨*proof*⟩

lemma *norm-memb-eq-rep*:
assumes *arr f* **and** *t* ∈ *f*
shows ‖*t*‖ = *rep f*
⟨*proof*⟩

lemma *norm-rep*:
assumes *arr f*
shows ‖*rep f*‖ = *rep f*
⟨*proof*⟩

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* · *mkarr u* = *mkarr* (*t* • *u*)
⟨*proof*⟩

lemma *dom-char*:
shows *dom f* = (*if arr f then mkarr* (*DOM f*) *else null*)
⟨*proof*⟩

lemma *dom-simp*:
assumes *arr f*
shows *dom f* = *mkarr* (*DOM f*)
⟨*proof*⟩

lemma *cod-char*:
shows *cod f* = (*if arr f then mkarr* (*COD f*) *else null*)
⟨*proof*⟩

lemma *cod-simp*:
assumes *arr f*
shows *cod f* = *mkarr* (*COD f*)
⟨*proof*⟩

lemma *Dom-memb*:
assumes *arr f* **and** *t* ∈ *f*
shows *Dom t* = *DOM f*

⟨*proof*⟩

lemma *Cod-memb*:
assumes *arr f* **and** $t \in f$
shows *Cod t* = *COD f*
⟨*proof*⟩

lemma *dom-mkarr* [*simp*]:
assumes *Arr t*
shows *dom* (*mkarr t*) = *mkarr* (*Dom t*)
⟨*proof*⟩

lemma *cod-mkarr* [*simp*]:
assumes *Arr t*
shows *cod* (*mkarr t*) = *mkarr* (*Cod t*)
⟨*proof*⟩

lemma *mkarr-in-hom*:
assumes *Arr t*
shows «*mkarr t* : *mkarr* (*Dom t*) → *mkarr* (*Cod t*)»
⟨*proof*⟩

lemma *DOM-in-dom* [*intro*]:
assumes *arr f*
shows *DOM f* ∈ *dom f*
⟨*proof*⟩

lemma *COD-in-cod* [*intro*]:
assumes *arr f*
shows *COD f* ∈ *cod f*
⟨*proof*⟩

lemma *DOM-dom*:
assumes *arr f*
shows *DOM* (*dom f*) = *DOM f*
⟨*proof*⟩

lemma *DOM-cod*:
assumes *arr f*
shows *DOM* (*cod f*) = *COD f*
⟨*proof*⟩

lemma *memb-equiv*:
assumes *arr f* **and** $t \in f$ **and** $u \in f$
shows *Par t u* **and** $\lfloor t \rfloor = \lfloor u \rfloor$
⟨*proof*⟩

Two arrows can be proved equal by showing that they are parallel and have representatives with identical diagonalizations.

lemma *arr-eqI*:
assumes *par f g* **and** $t \in f$ **and** $u \in g$ **and** $\lfloor t \rfloor = \lfloor u \rfloor$
shows $f = g$
⟨*proof*⟩

lemma *comp-char*:
shows $f \cdot g = (\textit{if seq f g then mkarr (rep f} \bullet \textit{rep g) else null})$
⟨*proof*⟩

The mapping that takes identity terms to their equivalence classes is injective.

lemma *mkarr-inj-on-Ide*:
assumes *Ide t* **and** *Ide u* **and** $\textit{mkarr } t = \textit{mkarr } u$
shows $t = u$
⟨*proof*⟩

lemma *Comp-in-comp* [*intro*]:
assumes *arr f* **and** $g \in \textit{hom (dom g) (dom f)}$ **and** $t \in f$ **and** $u \in g$
shows $t \bullet u \in f \cdot g$
⟨*proof*⟩

An arrow is defined to be "canonical" if some (equivalently, all) its representatives diagonalize to an identity term.

definition *can*
where $\textit{can } f \equiv \textit{arr } f \wedge (\exists t.\ t \in f \wedge \textit{Ide } \lfloor t \rfloor)$

lemma *can-def-alt*:
shows $\textit{can } f \longleftrightarrow \textit{arr } f \wedge (\forall t.\ t \in f \longrightarrow \textit{Ide } \lfloor t \rfloor)$
⟨*proof*⟩

lemma *can-implies-arr*:
assumes *can f*
shows *arr f*
⟨*proof*⟩

The identities of the category are precisely the canonical endo-arrows.

lemma *ide-char*:
shows $\textit{ide } f \longleftrightarrow \textit{can } f \wedge \textit{dom } f = \textit{cod } f$
⟨*proof*⟩

lemma *ide-iff-IDE*:
shows $\textit{ide } a \longleftrightarrow \textit{IDE } a$
⟨*proof*⟩

lemma *ide-mkarr-Ide*:
assumes *Ide a*
shows *ide* (*mkarr a*)
⟨*proof*⟩

lemma *rep-dom*:

assumes *arr f*
shows *rep* (*dom f*) = ‖*DOM f*‖
⟨*proof*⟩

lemma *rep-cod*:
assumes *arr f*
shows *rep* (*cod f*) = ‖*COD f*‖
⟨*proof*⟩

lemma *rep-preserves-seq*:
assumes *seq g f*
shows *Seq* (*rep g*) (*rep f*)
⟨*proof*⟩

lemma *rep-comp*:
assumes *seq g f*
shows *rep* (*g* · *f*) = ‖*rep g* **·** *rep f*‖
⟨*proof*⟩

The equivalence classes of canonical terms are canonical arrows.

lemma *can-mkarr-Can*:
assumes *Can t*
shows *can* (*mkarr t*)
⟨*proof*⟩

lemma *ide-implies-can*:
assumes *ide a*
shows *can a*
⟨*proof*⟩

lemma *Can-rep-can*:
assumes *can f*
shows *Can* (*rep f*)
⟨*proof*⟩

Parallel canonical arrows are identical.

lemma *can-coherence*:
assumes *par f g* **and** *can f* **and** *can g*
shows *f* = *g*
⟨*proof*⟩

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*↓) **·** ⌊*rep f*⌋ **·** *COD f*↓))
⟨*proof*⟩

lemma *inv-mkarr* [*simp*]:
assumes *Can t*

shows *inv* (*mkarr t*) = *mkarr* (*Inv t*)
⟨*proof*⟩

lemma *iso-can*:
assumes *can f*
shows *iso f*
⟨*proof*⟩

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 *can* (*mkcan a b*) **and** «*mkcan a b* : *a* → *b*»
⟨*proof*⟩

lemma *dom-mkcan*:
assumes *ide a* **and** *ide b* **and** ⌊*DOM a*⌋ = ⌊*COD b*⌋
shows *dom* (*mkcan a b*) = *a*
⟨*proof*⟩

lemma *cod-mkcan*:
assumes *ide a* **and** *ide b* **and** ⌊*DOM a*⌋ = ⌊*COD b*⌋
shows *cod* (*mkcan a b*) = *b*
⟨*proof*⟩

lemma *can-coherence'*:
assumes *can f*
shows *mkcan* (*dom f*) (*cod f*) = *f*
⟨*proof*⟩

lemma *Ide-Diagonalize-rep-ide*:
assumes *ide a*
shows *Ide* ⌊*rep a*⌋
⟨*proof*⟩

lemma *Diagonalize-DOM*:
assumes *arr f*
shows ⌊*DOM f*⌋ = *Dom* ⌊*rep f*⌋
⟨*proof*⟩

lemma *Diagonalize-COD*:
assumes *arr f*
shows ⌊*COD f*⌋ = *Cod* ⌊*rep f*⌋
⟨*proof*⟩

lemma *Diagonalize-rep-preserves-seq*:

assumes *seq g f*
shows *Seq* ⌊*rep g*⌋ ⌊*rep f*⌋
⟨*proof*⟩

lemma *Dom-Diagonalize-rep*:
assumes *arr f*
shows *Dom* ⌊*rep f*⌋ = ⌊*rep* (*dom f*)⌋
⟨*proof*⟩

lemma *Cod-Diagonalize-rep*:
assumes *arr f*
shows *Cod* ⌊*rep f*⌋ = ⌊*rep* (*cod f*)⌋
⟨*proof*⟩

lemma *mkarr-Diagonalize-rep*:
assumes *arr f* **and** *Diag* (*DOM f*) **and** *Diag* (*COD f*)
shows *mkarr* ⌊*rep f*⌋ = *f*
⟨*proof*⟩

We define tensor product of arrows via the constructor (⊗) on terms.

definition $tensor_{FMC}$ (**infixr** ‹⊗› *53*)
where *f* ⊗ *g* ≡ (*if arr f* ∧ *arr g then mkarr* (*rep f* ⊗ *rep g*) *else null*)

lemma *arr-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *arr* (*f* ⊗ *g*)
⟨*proof*⟩

lemma *rep-tensor*:
assumes *arr f* **and** *arr g*
shows *rep* (*f* ⊗ *g*) = ‖*rep f* ⊗ *rep g*‖
⟨*proof*⟩

lemma *Par-memb-rep*:
assumes *arr f* **and** *t* ∈ *f*
shows *Par t* (*rep f*)
⟨*proof*⟩

lemma *Tensor-in-tensor* [*intro*]:
assumes *arr f* **and** *arr g* **and** *t* ∈ *f* **and** *u* ∈ *g*
shows *t* ⊗ *u* ∈ *f* ⊗ *g*
⟨*proof*⟩

lemma *DOM-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *DOM* (*f* ⊗ *g*) = *DOM f* ⊗ *DOM g*
⟨*proof*⟩

lemma *COD-tensor* [*simp*]:

assumes *arr f* **and** *arr g*
shows *COD* $(f \otimes g) =$ *COD f* $\otimes$ *COD g*
⟨*proof*⟩

lemma *tensor-in-hom* [*simp*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$»
shows «$f \otimes g : a \otimes c \rightarrow b \otimes d$»
⟨*proof*⟩

lemma *dom-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *dom* $(f \otimes g) =$ *dom f* $\otimes$ *dom g*
⟨*proof*⟩

lemma *cod-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *cod* $(f \otimes g) =$ *cod f* $\otimes$ *cod g*
⟨*proof*⟩

lemma *tensor-mkarr* [*simp*]:
assumes *Arr t* **and** *Arr u*
shows *mkarr t* $\otimes$ *mkarr u* = *mkarr* $(t \otimes u)$
⟨*proof*⟩

lemma *tensor-preserves-ide*:
assumes *ide a* **and** *ide b*
shows *ide* $(a \otimes b)$
⟨*proof*⟩

lemma *tensor-preserves-can*:
assumes *can f* **and** *can g*
shows *can* $(f \otimes g)$
⟨*proof*⟩

lemma *comp-preserves-can*:
assumes *can f* **and** *can g* **and** *dom f* = *cod g*
shows *can* $(f \cdot g)$
⟨*proof*⟩

The remaining structure required of a monoidal category is also defined syntactically.

definition $unity_{FMC}$ :: *'c arr* (‹$\mathcal{I}$›)
where $\mathcal{I}$ = *mkarr* $\boldsymbol{\mathcal{I}}$

definition $lunit_{FMC}$ :: *'c arr* ⇒ *'c arr* (‹l[-]›)
where l[*a*] = *mkarr* **l**[*rep a*]

definition $runit_{FMC}$ :: *'c arr* ⇒ *'c arr* (‹r[-]›)
where r[*a*] = *mkarr* **r**[*rep a*]

definition $assoc_{FMC}$ :: $'c\ arr \Rightarrow 'c\ arr \Rightarrow 'c\ arr \Rightarrow 'c\ arr$ (‹a[-, -, -]›)
where a[a, b, c] = *mkarr* **a**[*rep* a, *rep* b, *rep* c]

lemma *can-lunit*:
assumes *ide* a
shows *can* l[a]
⟨*proof*⟩

lemma *lunit-in-hom*:
assumes *ide* a
shows «l[a] : $\mathcal{I} \otimes a \rightarrow a$»
⟨*proof*⟩

lemma *arr-lunit* [*simp*]:
assumes *ide* a
shows *arr* l[a]
⟨*proof*⟩

lemma *dom-lunit* [*simp*]:
assumes *ide* a
shows *dom* l[a] = $\mathcal{I} \otimes a$
⟨*proof*⟩

lemma *cod-lunit* [*simp*]:
assumes *ide* a
shows *cod* l[a] = a
⟨*proof*⟩

lemma *can-runit*:
assumes *ide* a
shows *can* r[a]
⟨*proof*⟩

lemma *runit-in-hom* [*simp*]:
assumes *ide* a
shows «r[a] : $a \otimes \mathcal{I} \rightarrow a$»
⟨*proof*⟩

lemma *arr-runit* [*simp*]:
assumes *ide* a
shows *arr* r[a]
⟨*proof*⟩

lemma *dom-runit* [*simp*]:
assumes *ide* a
shows *dom* r[a] = $a \otimes \mathcal{I}$
⟨*proof*⟩

lemma *cod-runit* [*simp*]:

assumes *ide a*
shows *cod* r[*a*] = *a*
⟨*proof*⟩

lemma *can-assoc*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *can* a[*a*, *b*, *c*]
⟨*proof*⟩

lemma *assoc-in-hom*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows «a[*a*, *b*, *c*] : $(a \otimes b) \otimes c \rightarrow a \otimes b \otimes c$»
⟨*proof*⟩

lemma *arr-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *arr* a[*a*, *b*, *c*]
⟨*proof*⟩

lemma *dom-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *dom* a[*a*, *b*, *c*] = $(a \otimes b) \otimes c$
⟨*proof*⟩

lemma *cod-assoc* [*simp*]:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *cod* a[*a*, *b*, *c*] = $a \otimes b \otimes c$
⟨*proof*⟩

lemma *ide-unity* [*simp*]:
shows *ide* $\mathcal{I}$
⟨*proof*⟩

lemma *Unity-in-unity* [*simp*]:
shows $\boldsymbol{\mathcal{I}} \in \mathcal{I}$
⟨*proof*⟩

lemma *rep-unity* [*simp*]:
shows *rep* $\mathcal{I} = \|\boldsymbol{\mathcal{I}}\|$
⟨*proof*⟩

lemma *Lunit-in-lunit* [*intro*]:
assumes *arr f* **and** $t \in f$
shows **l**[*t*] ∈ l[*f*]
⟨*proof*⟩

lemma *Runit-in-runit* [*intro*]:
assumes *arr f* **and** $t \in f$
shows **r**[*t*] ∈ r[*f*]

⟨*proof*⟩

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 $\mathbf{a}[t, u, v] \in \mathrm{a}[f, g, h]$
⟨*proof*⟩

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*⟩

lemma *is-elementary-monoidal-category*:
shows *elementary-monoidal-category*
comp $tensor_{FMC}$ $unity_{FMC}$ $lunit_{FMC}$ $runit_{FMC}$ $assoc_{FMC}$
⟨*proof*⟩

abbreviation T_{FMC} **where** $T_{FMC} \equiv EMC.T$
abbreviation α_{FMC} **where** $\alpha_{FMC} \equiv EMC.\alpha$
abbreviation ι_{FMC} **where** $\iota_{FMC} \equiv EMC.\iota$

interpretation *MC*: *monoidal-category comp* T_{FMC} α_{FMC} ι_{FMC}
⟨*proof*⟩

lemma *induces-monoidal-category*:
shows *monoidal-category comp* T_{FMC} α_{FMC} ι_{FMC}
⟨*proof*⟩

end

sublocale *free-monoidal-category* ⊆
elementary-monoidal-category
comp $tensor_{FMC}$ $unity_{FMC}$ $lunit_{FMC}$ $runit_{FMC}$ $assoc_{FMC}$
⟨*proof*⟩

sublocale *free-monoidal-category* ⊆ *monoidal-category comp* T_{FMC} α_{FMC} ι_{FMC}
⟨*proof*⟩

4.2 Proof of Freeness

Now we proceed on to establish the freeness of $\mathcal{F}C$: each functor from C to a monoidal category D extends uniquely to a strict monoidal functor from $\mathcal{F}C$ to D.

context *free-monoidal-category*
begin

lemma *rep-lunit*:
assumes *ide a*

shows *rep* l[*a*] = ‖**l**[*rep a*]‖
⟨*proof*⟩

lemma *rep-runit*:
assumes *ide a*
shows *rep* r[*a*] = ‖**r**[*rep a*]‖
⟨*proof*⟩

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*]‖
⟨*proof*⟩

lemma *mkarr-Unity*:
shows *mkarr* $\boldsymbol{\mathcal{I}}$ = $\mathcal{I}$
⟨*proof*⟩

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* **l**[*t*] = $\mathfrak{l}$ (*mkarr t*)
⟨*proof*⟩

lemma *mkarr-Lunit'*:
assumes *Arr t*
shows *mkarr* $\mathbf{l}^{-1}$[*t*] = $\mathfrak{l}'$ (*mkarr t*)
⟨*proof*⟩

lemma *mkarr-Runit*:
assumes *Arr t*
shows *mkarr* **r**[*t*] = ϱ (*mkarr t*)
⟨*proof*⟩

lemma *mkarr-Runit'*:
assumes *Arr t*
shows *mkarr* $\mathbf{r}^{-1}$[*t*] = ϱ' (*mkarr t*)
⟨*proof*⟩

lemma *mkarr-Assoc*:
assumes *Arr t* **and** *Arr u* **and** *Arr v*
shows *mkarr* **a**[*t*, *u*, *v*] = α (*mkarr t*, *mkarr u*, *mkarr v*)
⟨*proof*⟩

lemma *mkarr-Assoc'*:
assumes *Arr t* **and** *Arr u* **and** *Arr v*
shows *mkarr* $\mathbf{a}^{-1}$[*t*, *u*, *v*] = α' (*mkarr t*, *mkarr u*, *mkarr v*)

⟨proof⟩

Next, we define the "inclusion of generators" functor from C to $\mathcal{F}C$.

definition *inclusion-of-generators*
where *inclusion-of-generators* ≡ $\lambda f.$ *if C.arr f then mkarr* ⟨*f*⟩ *else null*

lemma *inclusion-is-functor*:
shows *functor C comp inclusion-of-generators*
⟨*proof*⟩

end

We now show that, given a functor V from C to a 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* $D\ T_D\ \alpha_D\ \iota_D$ +
evaluation-map $C\ D\ T_D\ \alpha_D\ \iota_D\ V$ +
$\mathcal{F}C$: *free-monoidal-category C*
for C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and D :: *'d comp* (**infixr** ‹$\cdot_D$› *55*)
and T_D :: *'d* ∗ *'d* ⇒ *'d*
and α_D :: *'d* ∗ *'d* ∗ *'d* ⇒ *'d*
and ι_D :: *'d*
and V :: *'c* ⇒ *'d*
begin

notation *eval* (‹⦃-⦄›)

definition *map*
where *map f* ≡ *if* $\mathcal{F}C$*.arr f then* ⦃$\mathcal{F}C$*.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*⦄
⟨*proof*⟩

interpretation *functor* $\mathcal{F}C$*.comp D map*
⟨*proof*⟩

lemma *is-functor*:
shows *functor* $\mathcal{F}C$*.comp D map* ⟨*proof*⟩

interpretation *FF*: *product-functor* $\mathcal{F}C$*.comp* $\mathcal{F}C$*.comp D D map map* ⟨*proof*⟩
interpretation *FoT*: *composite-functor* $\mathcal{F}C$*.CC.comp* $\mathcal{F}C$*.comp D* $\mathcal{F}C.T_{FMC}$ *map* ⟨*proof*⟩
interpretation *ToFF*: *composite-functor* $\mathcal{F}C$*.CC.comp D.CC.comp D FF.map* T_D ⟨*proof*⟩

interpretation *strict-monoidal-functor*
$\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha\ \mathcal{F}C.\iota\ D\ T_D\ \alpha_D\ \iota_D\ map$
⟨*proof*⟩

lemma *is-strict-monoidal-functor*:
shows *strict-monoidal-functor* $\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha\ \mathcal{F}C.\iota\ D\ T_D\ \alpha_D\ \iota_D\ map$
⟨*proof*⟩

end

sublocale *evaluation-functor* ⊆ *strict-monoidal-functor*
$\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}\ D\ T_D\ \alpha_D\ \iota_D\ map$
⟨*proof*⟩

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 $C\ T_C\ \alpha_C\ \iota_C\ D\ T_D\ \alpha_D\ \iota_D\ F$
for C_0 :: $'c_0$ *comp*
and C :: $'c$ *comp* (**infixr** ‹$\cdot_C$› *55*)
and T_C :: $'c * 'c \Rightarrow 'c$
and α_C :: $'c * 'c * 'c \Rightarrow 'c$
and ι_C :: $'c$
and D :: $'d$ *comp* (**infixr** ‹$\cdot_D$› *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and I :: $'c_0 \Rightarrow 'c$
and V :: $'c_0 \Rightarrow 'd$
and F :: $'c \Rightarrow 'd$ +
assumes *is-extension*: $\forall f.\ C_0.arr\ f \longrightarrow F\ (I\ f) = V\ f$

sublocale *evaluation-functor* ⊆
strict-monoidal-extension $C\ \mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha\ \mathcal{F}C.\iota\ D\ T_D\ \alpha_D\ \iota_D$
$\mathcal{F}C.inclusion\text{-}of\text{-}generators\ V\ map$
⟨*proof*⟩

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 +
  $\mathcal{F}C$: *free-monoidal-category C* +
  strict-monoidal-extension C $\mathcal{F}C.comp$ $\mathcal{F}C.T_{FMC}$ $\mathcal{F}C.\alpha$ $\mathcal{F}C.\iota$ D T_D α_D ι_D
      $\mathcal{F}C$*.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 α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and V :: $'c \Rightarrow 'd$
and F :: *'c free-monoidal-category.arr* $\Rightarrow$ *'d*
begin

  lemma *strictly-preserves-everything*:
  shows $C.arr\ f \Longrightarrow F\ (\mathcal{F}C.mkarr\ \langle f\rangle) = V\ f$
  and $F\ (\mathcal{F}C.mkarr\ \boldsymbol{\mathcal{I}}) = \mathcal{I}_D$
  and $[\![\ Arr\ t;\ Arr\ u\ ]\!] \Longrightarrow F\ (\mathcal{F}C.mkarr\ (t \otimes u)) = F\ (\mathcal{F}C.mkarr\ t) \otimes_D F\ (\mathcal{F}C.mkarr\ u)$
  and $[\![\ Arr\ t;\ Arr\ u;\ Dom\ t = Cod\ u\ ]\!] \Longrightarrow$
      $F\ (\mathcal{F}C.mkarr\ (t \cdot u)) = F\ (\mathcal{F}C.mkarr\ t) \cdot_D F\ (\mathcal{F}C.mkarr\ u)$
  and $Arr\ t \Longrightarrow F\ (\mathcal{F}C.mkarr\ \mathbf{l}[t]) = D.\mathfrak{l}\ (F\ (\mathcal{F}C.mkarr\ t))$
  and $Arr\ t \Longrightarrow F\ (\mathcal{F}C.mkarr\ \mathbf{l}^{-1}[t]) = D.\mathfrak{l}'.map\ (F\ (\mathcal{F}C.mkarr\ t))$
  and $Arr\ t \Longrightarrow F\ (\mathcal{F}C.mkarr\ \mathbf{r}[t]) = D.\varrho\ (F\ (\mathcal{F}C.mkarr\ t))$
  and $Arr\ t \Longrightarrow F\ (\mathcal{F}C.mkarr\ \mathbf{r}^{-1}[t]) = D.\varrho'.map\ (F\ (\mathcal{F}C.mkarr\ t))$
  and $[\![\ Arr\ t;\ Arr\ u;\ Arr\ v\ ]\!] \Longrightarrow$
      $F\ (\mathcal{F}C.mkarr\ \mathbf{a}[t,\ u,\ v]) = \alpha_D\ (F\ (\mathcal{F}C.mkarr\ t),\ F\ (\mathcal{F}C.mkarr\ u),\ F\ (\mathcal{F}C.mkarr\ v))$
  and $[\![\ Arr\ t;\ Arr\ u;\ Arr\ v\ ]\!] \Longrightarrow$
      $F\ (\mathcal{F}C.mkarr\ \mathbf{a}^{-1}[t,\ u,\ v])$
        $= D.\alpha'\ (F\ (\mathcal{F}C.mkarr\ t),\ F\ (\mathcal{F}C.mkarr\ u),\ F\ (\mathcal{F}C.mkarr\ v))$
  ⟨*proof*⟩

end

sublocale *evaluation-functor* ⊆ *strict-monoidal-extension-to-free-monoidal-category*
      $C\ D\ T_D\ \alpha_D\ \iota_D\ V\ map$
  ⟨*proof*⟩

context *free-monoidal-category*
begin

The evaluation functor induced by V is the unique strict monoidal extension of V to $\mathcal{F}C$.

  theorem *is-free*:
  assumes *strict-monoidal-extension-to-free-monoidal-category* $C\ D\ T_D\ \alpha_D\ \iota_D\ V\ F$
  shows F = *evaluation-functor.map* $C\ D\ T_D\ \alpha_D\ \iota_D\ V$
  ⟨*proof*⟩

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* ‹$\lambda f.$ *ide f* $\wedge$ *Diag* (*DOM f*)›
⟨*proof*⟩

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* $\lfloor$*rep f*$\rfloor$ *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*⟩

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*⟩

interpretation D: *functor comp* $\mathcal{F}_S C$*.comp* D
⟨*proof*⟩

lemma *diagonalize-is-functor*:
shows *functor comp* $\mathcal{F}_S C$*.comp* D ⟨*proof*⟩

lemma *diagonalize-strict-arr*:
assumes $\mathcal{F}_S C$*.arr f*
shows $D\ f = f$
⟨*proof*⟩

lemma *diagonalize-is-idempotent*:
shows $D\ o\ D = D$
⟨*proof*⟩

lemma *diagonalize-tensor*:
assumes *arr f* **and** *arr g*
shows $D\ (f \otimes g) = D\ (D\ f \otimes D\ g)$

⟨*proof*⟩

lemma *ide-diagonalize-can*:
assumes *can f*
shows *ide* (*D f*)
⟨*proof*⟩

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*↓), determined by reductions of canonical representatives, are the components of a natural isomorphism.

interpretation *S*: *full-inclusion-functor comp* ‹$\lambda f.$ *ide f* ∧ *Diag* (*DOM f*)› ⟨*proof*⟩
interpretation *DoS*: *composite-functor* $\mathcal{F}_S C$*.comp comp* $\mathcal{F}_S C$*.comp* $\mathcal{F}_S C$*.map D*
⟨*proof*⟩
interpretation *SoD*: *composite-functor comp* $\mathcal{F}_S C$*.comp comp D* $\mathcal{F}_S C$*.map* ⟨*proof*⟩

interpretation ν: *transformation-by-components*
comp comp map SoD.map ‹$\lambda a.$ *mkarr* (*DOM a*↓)›
⟨*proof*⟩

interpretation ν: *natural-isomorphism comp comp map SoD.map* ν*.map*
⟨*proof*⟩

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*⟩

interpretation μ: *transformation-by-components*
$\mathcal{F}_S C$*.comp* $\mathcal{F}_S C$*.comp DoS.map* $\mathcal{F}_S C$*.map* ‹$\lambda a.$ *a*›
⟨*proof*⟩

interpretation μ: *natural-isomorphism* $\mathcal{F}_S C$*.comp* $\mathcal{F}_S C$*.comp DoS.map* $\mathcal{F}_S C$*.map* μ*.map*
⟨*proof*⟩

interpretation *equivalence-of-categories* $\mathcal{F}_S C$*.comp comp D* $\mathcal{F}_S C$*.map* ν*.map* μ*.map* ⟨*proof*⟩

We defined the natural isomorphisms μ and ν 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 μ
where $\mu \equiv \lambda f.$ *if* $\mathcal{F}_S C$*.arr f then f else* $\mathcal{F}_S C$*.null*

definition ν
where $\nu \equiv \lambda f.$ *if arr f then mkarr* (*COD f*↓) · *f else null*

lemma μ*-char*:
shows μ*.map* = μ
⟨*proof*⟩

lemma *ν-char*:
shows $\nu.map = \nu$
⟨*proof*⟩

lemma *is-equivalent-to-strict-subcategory*:
shows *equivalence-of-categories* $\mathcal{F}_S C.comp$ *comp D* $\mathcal{F}_S C.map$ ν μ
⟨*proof*⟩

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*
⟨*proof*⟩
interpretation *DoI*: *composite-functor C comp* $\mathcal{F}_S C.comp$ *inclusion-of-generators D* ⟨*proof*⟩

lemma *DoI-eq-I*:
shows *DoI.map* = *inclusion-of-generators*
⟨*proof*⟩

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 comp*
begin

interpretation *D*: *functor* $\mathcal{F}C.comp$ *comp* $\mathcal{F}C.D$
⟨*proof*⟩

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 $f \otimes_S g = \mathcal{F}C.mkarr\ (\lfloor \mathcal{F}C.rep\ f \rfloor\ \lfloor\otimes\rfloor\ \lfloor \mathcal{F}C.rep\ g \rfloor)$
⟨*proof*⟩

lemma *tensor-in-hom* [*simp*]:
assumes «$f : a \rightarrow b$» **and** «$g : c \rightarrow d$»
shows «$f \otimes_S g : a \otimes_S c \rightarrow b \otimes_S d$»

⟨*proof*⟩

lemma *arr-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *arr* $(f \otimes_S g)$
⟨*proof*⟩

lemma *dom-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *dom* $(f \otimes_S g) =$ *dom f* $\otimes_S$ *dom g*
⟨*proof*⟩

lemma *cod-tensor* [*simp*]:
assumes *arr f* **and** *arr g*
shows *cod* $(f \otimes_S g) =$ *cod f* $\otimes_S$ *cod g*
⟨*proof*⟩

lemma *tensor-preserves-ide*:
assumes *ide a* **and** *ide b*
shows *ide* $(a \otimes_S b)$
⟨*proof*⟩

lemma *tensor-tensor*:
assumes *arr f* **and** *arr g* **and** *arr h*
shows $(f \otimes_S g) \otimes_S h = \mathcal{F}C.mkarr\ (\lfloor \mathcal{F}C.rep\ f \rfloor\ \lfloor \otimes \rfloor\ \lfloor \mathcal{F}C.rep\ g \rfloor\ \lfloor \otimes \rfloor\ \lfloor \mathcal{F}C.rep\ h \rfloor)$
and $f \otimes_S g \otimes_S h = \mathcal{F}C.mkarr\ (\lfloor \mathcal{F}C.rep\ f \rfloor\ \lfloor \otimes \rfloor\ \lfloor \mathcal{F}C.rep\ g \rfloor\ \lfloor \otimes \rfloor\ \lfloor \mathcal{F}C.rep\ h \rfloor)$
⟨*proof*⟩

lemma *tensor-assoc*:
assumes *arr f* **and** *arr g* **and** *arr h*
shows $(f \otimes_S g) \otimes_S h = f \otimes_S g \otimes_S h$
⟨*proof*⟩

lemma *arr-unity*:
shows *arr* $\mathcal{I}$
⟨*proof*⟩

lemma *tensor-unity-arr*:
assumes *arr f*
shows $\mathcal{I} \otimes_S f = f$
⟨*proof*⟩

lemma *tensor-arr-unity*:
assumes *arr f*
shows $f \otimes_S \mathcal{I} = f$
⟨*proof*⟩

lemma *assoc-char*:
assumes *ide a* **and** *ide b* **and** *ide c*

shows $\mathrm{a}_S[a,\ b,\ c] = \mathcal{F}C.mkarr\ (\lfloor \mathcal{F}C.rep\ a \rfloor\ \lfloor\otimes\rfloor\ \lfloor \mathcal{F}C.rep\ b \rfloor\ \lfloor\otimes\rfloor\ \lfloor \mathcal{F}C.rep\ c \rfloor)$
⟨*proof*⟩

lemma *assoc-in-hom*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows «$\mathrm{a}_S[a,\ b,\ c] : (a \otimes_S b) \otimes_S c \rightarrow a \otimes_S b \otimes_S c$»
⟨*proof*⟩

The category $\mathcal{F}_S C$ is a monoidal category.

interpretation *EMC*: *elementary-monoidal-category comp tensor*$_S$ $\mathcal{I}$ ‹$\lambda a.\ a$› ‹$\lambda a.\ a$› *assoc*$_S$
⟨*proof*⟩

lemma *is-elementary-monoidal-category*:
shows *elementary-monoidal-category comp tensor*$_S$ $\mathcal{I}$ $(\lambda a.\ a)$ $(\lambda a.\ a)$ *assoc*$_S$ ⟨*proof*⟩

abbreviation T_{FSMC} **where** $T_{FSMC} \equiv EMC.T$
abbreviation α_{FSMC} **where** $\alpha_{FSMC} \equiv EMC.\alpha$
abbreviation ι_{FSMC} **where** $\iota_{FSMC} \equiv EMC.\iota$

lemma *is-monoidal-category*:
shows *monoidal-category comp* T_{FSMC} α_{FSMC} ι_{FSMC}
⟨*proof*⟩

end

sublocale *free-strict-monoidal-category* ⊆
elementary-monoidal-category comp tensor$_S$ $\mathcal{I}$ $\lambda a.\ a$ $\lambda a.\ a$ *assoc*$_S$
⟨*proof*⟩

sublocale *free-strict-monoidal-category* ⊆ *monoidal-category comp* T_{FSMC} α_{FSMC} ι_{FSMC}
⟨*proof*⟩

sublocale *free-strict-monoidal-category* ⊆
strict-monoidal-category comp T_{FSMC} α_{FSMC} ι_{FSMC}
⟨*proof*⟩

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* $\mathcal{F}C.D$
⟨*proof*⟩

interpretation *I*: *composite-functor C* $\mathcal{F}C.comp$ *comp* $\mathcal{F}C.inclusion\text{-}of\text{-}generators$ $\mathcal{F}C.D$
⟨*proof*⟩

definition *inclusion-of-generators*
where *inclusion-of-generators* ≡ $\mathcal{F}C$*.inclusion-of-generators*

lemma *inclusion-is-functor*:
shows *functor C comp inclusion-of-generators*
⟨*proof*⟩

The diagonalization functor is strict monoidal.

interpretation *D*: *strict-monoidal-functor* $\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}$
$comp\ T_{FSMC}\ \alpha_{FSMC}\ \iota_{FSMC}$
$\mathcal{F}C.D$
⟨*proof*⟩

lemma *diagonalize-is-strict-monoidal-functor*:
shows *strict-monoidal-functor* $\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}$
$comp\ T_{FSMC}\ \alpha_{FSMC}\ \iota_{FSMC}$
$\mathcal{F}C.D$
⟨*proof*⟩

interpretation φ: *natural-isomorphism*
$\mathcal{F}C.CC.comp\ comp\ D.T_D oFF.map\ D.FoT_C.map\ D.\varphi$
⟨*proof*⟩

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* $\mathcal{F}C.comp\ \mathcal{F}C.D\ map\ \mathcal{F}C.\nu\ \mathcal{F}C.\mu$
⟨*proof*⟩

interpretation *D*: *monoidal-functor* $\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}$
$comp\ T_{FSMC}\ \alpha_{FSMC}\ \iota_{FSMC}$
$\mathcal{F}C.D\ D.\varphi$
⟨*proof*⟩

interpretation *equivalence-of-monoidal-categories comp* $T_{FSMC}\ \alpha_{FSMC}\ \iota_{FSMC}$
$\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}$
$\mathcal{F}C.D\ D.\varphi\ \mathcal{I}$
$map\ \mathcal{F}C.\nu\ \mathcal{F}C.\mu$
⟨*proof*⟩

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* $T_{FSMC}\ \alpha_{FSMC}\ \iota_{FSMC}$
$\mathcal{F}C.comp\ \mathcal{F}C.T_{FMC}\ \mathcal{F}C.\alpha_{FMC}\ \mathcal{F}C.\iota_{FMC}$
$\mathcal{F}C.D\ D.\varphi$
$map\ \mathcal{F}C.\nu\ \mathcal{F}C.\mu$
⟨*proof*⟩

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 $C\ D\ T_D\ \alpha_D\ \iota_D\ V$ +
$\mathcal{F}C$: *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** $\langle\cdot_C\rangle$ *55*)
and D :: *'d comp* (**infixr** $\langle\cdot_D\rangle$ *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and V :: $'c \Rightarrow 'd$
begin

notation $\mathcal{F}C$.*in-hom* (‹«- : - → -»›)
notation $\mathcal{F}_S C$.*in-hom* (‹«- : - $\rightarrow_S$ -»›)

definition *map*
where *map* ≡ λf. *if* $\mathcal{F}_S C$.*arr f then E.map f else D.null*

interpretation *functor* $\mathcal{F}_S C$.*comp D map*
⟨*proof*⟩

lemma *is-functor*:
shows *functor* $\mathcal{F}_S C$.*comp D map* ⟨*proof*⟩

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 *Can t* ⟹ *D.ide* ⦃*t*⦄
⟨*proof*⟩

lemma *ide-eval-can*:
assumes $\mathcal{F}C$.*can f*
shows *D.ide* (*E.map f*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

end

sublocale *strict-evaluation-functor* ⊆
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*⟩

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 $C\ \mathcal{F}_S C.comp\ \mathcal{F}_S C.T_{FSMC}\ \mathcal{F}_S C.\alpha\ \mathcal{F}_S C.\iota\ D\ T_D\ \alpha_D\ \iota_D$
$\mathcal{F}_S C.inclusion\text{-}of\text{-}generators\ V\ F$
for C :: *'c comp* (**infixr** ‹$\cdot_C$› *55*)
and D :: *'d comp* (**infixr** ‹$\cdot_D$› *55*)
and T_D :: $'d * 'd \Rightarrow 'd$
and α_D :: $'d * 'd * 'd \Rightarrow 'd$
and ι_D :: $'d$
and V :: $'c \Rightarrow 'd$
and F :: *'c free-monoidal-category.arr* $\Rightarrow$ *'d*

sublocale *strict-evaluation-functor* ⊆
strict-monoidal-extension $C\ \mathcal{F}_S C.comp\ \mathcal{F}_S C.T_{FSMC}\ \mathcal{F}_S C.\alpha\ \mathcal{F}_S C.\iota\ D\ T_D\ \alpha_D\ \iota_D$
$\mathcal{F}_S C.inclusion\text{-}of\text{-}generators\ V\ map$
⟨*proof*⟩

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 $F =$ *strict-evaluation-functor.map* $C\ D\ T_D\ \alpha_D\ \iota_D\ V$
⟨*proof*⟩

end

end

Chapter 5

Cartesian Monoidal Category

theory *CartesianMonoidalCategory*
imports *MonoidalCategory Category3.CartesianCategory*
begin

5.1 Symmetric Monoidal Category

locale *symmetric-monoidal-category* =
  monoidal-category C T α ι +
  S: *symmetry-functor C C* +
  ToS: *composite-functor CC.comp CC.comp C S.map T* +
  σ: *natural-isomorphism CC.comp C T ToS.map σ*
for *C* :: *'a comp* (**infixr** ‹·› *55*)
and *T* :: *'a * 'a ⇒ 'a*
and *α* :: *'a * 'a * 'a ⇒ 'a*
and *ι* :: *'a*
and *σ* :: *'a * 'a ⇒ 'a* +
assumes *sym-inverse*: ⟦ *ide a*; *ide b* ⟧ ⟹ *inverse-arrows* (*σ* (*a*, *b*)) (*σ* (*b*, *a*))
and *unitor-coherence*: *ide a* ⟹ l[*a*] · *σ* (*a*, *ℐ*) = r[*a*]
and *assoc-coherence*: ⟦ *ide a*; *ide b*; *ide c* ⟧ ⟹
  α (*b*, *c*, *a*) · *σ* (*a*, *b* ⊗ *c*) · *α* (*a*, *b*, *c*)
    = (*b* ⊗ *σ* (*a*, *c*)) · *α* (*b*, *a*, *c*) · (*σ* (*a*, *b*) ⊗ *c*)
begin

  abbreviation *sym* (‹s[-, -]›)
  where *sym a b* ≡ *σ* (*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 ⇒ 'a ⇒ 'a* (**infixr** ‹⊗› *53*)
and *unity* :: *'a* (‹ℐ›)
and *lunit* :: *'a ⇒ 'a* (‹l[-]›)

and *runit* :: ′a ⇒ ′a (‹r[-]›)
and *assoc* :: ′a ⇒ ′a ⇒ ′a ⇒ ′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 *unitor-coherence*: *ide a* ⟹ l[a] · s[a, 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
⟨*proof*⟩

interpretation *CC*: *product-category C C* ⟨*proof*⟩
sublocale *MC*: *monoidal-category C T α ι*
⟨*proof*⟩

interpretation *S*: *symmetry-functor C C* ⟨*proof*⟩
interpretation *ToS*: *composite-functor CC.comp CC.comp C S.map T* ⟨*proof*⟩

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*⟩

interpretation *symmetric-monoidal-category C T α ι σ*
⟨*proof*⟩

lemma *induces-symmetric-monoidal-category*$_{CMC}$:
shows *symmetric-monoidal-category C T α ι σ*
⟨*proof*⟩

end

context *symmetric-monoidal-category*
begin

interpretation *EMC*: *elementary-monoidal-category C tensor unity lunit runit assoc*
⟨*proof*⟩

lemma *induces-elementary-symmetric-monoidal-category*$_{CMC}$:
shows *elementary-symmetric-monoidal-category*

C tensor unity lunit runit assoc ($\lambda a\ b.\ \sigma\ (a,\ b)$)
⟨*proof*⟩

end

locale *dual-symmetric-monoidal-category* =
M: *symmetric-monoidal-category*
begin

sublocale *dual-monoidal-category C T* α ι ⟨*proof*⟩
interpretation *S*: *symmetry-functor comp comp* ⟨*proof*⟩
interpretation *ToS*: *composite-functor MM.comp MM.comp comp S.map T* ⟨*proof*⟩
sublocale σ': *inverse-transformation M.CC.comp C T M.ToS.map* σ ⟨*proof*⟩
interpretation σ: *natural-transformation MM.comp comp T ToS.map* σ'*.map*
⟨*proof*⟩
interpretation σ: *natural-isomorphism MM.comp comp T ToS.map* σ'*.map*
⟨*proof*⟩

sublocale *symmetric-monoidal-category comp T M.*α' ‹*M.inv* ι› σ'*.map*
⟨*proof*⟩

lemma *is-symmetric-monoidal-category*:
shows *symmetric-monoidal-category comp T M.*α' (*M.inv* ι) σ'*.map*
⟨*proof*⟩

end

5.2 Cartesian Monoidal Category

Here we define "cartesian monoidal category" by imposing additional properties, but not additional structure, on top of "monoidal category". The additional properties are that the unit is a terminal object and that the tensor is a categorical product, with projections defined in terms of unitors, terminators, and tensor. It then follows that the associators are induced by the product structure.

locale *cartesian-monoidal-category* =
monoidal-category C T α ι
for *C* :: *'a comp* (**infixr** ‹·› *55*)
and *T* :: *'a* ∗ *'a* ⇒ *'a*
and α :: *'a* ∗ *'a* ∗ *'a* ⇒ *'a*
and ι :: *'a* +
assumes *terminal-unity*: *terminal* $\mathcal{I}$
and *tensor-is-product*:
⟦*ide a*; *ide b*; «t_a : $a \to \mathcal{I}$»; «t_b : $b \to \mathcal{I}$»⟧ ⟹
has-as-binary-product a b (r[*a*] · (*a* ⊗ t_b)) (l[*b*] · (t_a ⊗ *b*))
begin

sublocale *category-with-terminal-object*
⟨*proof*⟩

lemma *is-category-with-terminal-object*:
shows *category-with-terminal-object C*
⟨*proof*⟩

definition *the-trm* (‹t[-]›)
where *the-trm* ≡ λf. *THE t*. «t : *dom f* → $\mathcal{I}$»

lemma *trm-in-hom* [*intro*]:
assumes *ide a*
shows «t[a] : a → $\mathcal{I}$»
⟨*proof*⟩

lemma *trm-simps* [*simp*]:
assumes *ide a*
shows *arr* t[a] **and** *dom* t[a] = a **and** *cod* t[a] = $\mathcal{I}$
⟨*proof*⟩

interpretation *elementary-category-with-terminal-object C* $\mathcal{I}$ *the-trm*
⟨*proof*⟩

lemma *extends-to-elementary-category-with-terminal-object*$_{CMC}$:
shows *elementary-category-with-terminal-object C* $\mathcal{I}$ *the-trm*
⟨*proof*⟩

definition pr_0 (‹$\mathfrak{p}_0$[-, -]›)
where pr_0 a b ≡ l[b] · (t[a] ⊗ b)

definition pr_1 (‹$\mathfrak{p}_1$[-, -]›)
where pr_1 a b ≡ r[a] · (a ⊗ t[b])

sublocale *ECC*: *elementary-category-with-binary-products C* pr_0 pr_1
⟨*proof*⟩

lemma *induces-elementary-category-with-binary-products*$_{CMC}$:
shows *elementary-category-with-binary-products C* pr_0 pr_1
⟨*proof*⟩

lemma *is-category-with-binary-products*:
shows *category-with-binary-products C*
⟨*proof*⟩

sublocale *category-with-binary-products C*
⟨*proof*⟩

sublocale *ECC*: *elementary-cartesian-category C* pr_0 pr_1 $\mathcal{I}$ *the-trm* ⟨*proof*⟩

lemma *extends-to-elementary-cartesian-category*$_{CMC}$:
shows *elementary-cartesian-category C* pr_0 pr_1 $\mathcal{I}$ *the-trm*
⟨*proof*⟩

lemma *is-cartesian-category*:
shows *cartesian-category C*
⟨*proof*⟩

sublocale *cartesian-category C*
⟨*proof*⟩

abbreviation *dup* (‹d[-]›)
where *dup* ≡ *ECC.dup*

abbreviation *tuple* (‹⟨-, -⟩›)
where ⟨*f*, *g*⟩ ≡ *ECC.tuple f g*

lemma *prod-eq-tensor*:
shows *ECC.prod* = *tensor*
⟨*proof*⟩

lemma *Prod-eq-T*:
shows *ECC.Prod* = *T*
⟨*proof*⟩

lemma *tuple-pr* [*simp*]:
assumes *ide a* **and** *ide b*
shows $\langle \mathfrak{p}_1[a, b], \mathfrak{p}_0[a, b]\rangle = a \otimes b$
⟨*proof*⟩

lemma *tensor-expansion*:
assumes *arr f* **and** *arr g*
shows $f \otimes g = \langle f \cdot \mathfrak{p}_1[dom\ f, dom\ g], g \cdot \mathfrak{p}_0[dom\ f, dom\ g]\rangle$
⟨*proof*⟩

It is somewhat amazing that once the tensor product has been assumed to be a categorical product with the indicated projections, then the associators are forced to be those induced by the categorical product.

lemma *pr-assoc*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $\mathfrak{p}_1[a, b \otimes c] \cdot \mathrm{a}[a, b, c] = \mathfrak{p}_1[a, b] \cdot \mathfrak{p}_1[a \otimes b, c]$
and $\mathfrak{p}_1[b, c] \cdot \mathfrak{p}_0[a, b \otimes c] \cdot \mathrm{a}[a, b, c] = \mathfrak{p}_0[a, b] \cdot \mathfrak{p}_1[a \otimes b, c]$
and $\mathfrak{p}_0[b, c] \cdot \mathfrak{p}_0[a, b \otimes c] \cdot \mathrm{a}[a, b, c] = \mathfrak{p}_0[a \otimes b, c]$
⟨*proof*⟩

lemma *assoc-agreement*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows *ECC.assoc a b c* = $\mathrm{a}[a, b, c]$
⟨*proof*⟩

lemma *lunit-eq*:
assumes *ide a*
shows $\mathfrak{p}_0[\mathcal{I},\ a] = \mathrm{l}[a]$
⟨*proof*⟩

lemma *runit-eq*:
assumes *ide a*
shows $\mathfrak{p}_1[a,\ \mathcal{I}] = \mathrm{r}[a]$
⟨*proof*⟩

lemma *lunit′-as-tuple*:
assumes *ide a*
shows *tuple* t[*a*] *a* = *lunit′ a*
⟨*proof*⟩

lemma *runit′-as-tuple*:
assumes *ide a*
shows *tuple a* t[*a*] = *runit′ a*
⟨*proof*⟩

interpretation *S*: *symmetry-functor C C* ⟨*proof*⟩
interpretation *ToS*: *composite-functor CC.comp CC.comp C S.map T* ⟨*proof*⟩

interpretation *σ*: *natural-transformation CC.comp C T ToS.map ECC.σ*
⟨*proof*⟩

interpretation *σ*: *natural-isomorphism CC.comp C T ToS.map ECC.σ*
⟨*proof*⟩

sublocale *SMC*: *symmetric-monoidal-category C T α ι ECC.σ*
⟨*proof*⟩

end

5.3 Elementary Cartesian Monoidal Category

locale *elementary-cartesian-monoidal-category* =
elementary-monoidal-category C tensor unity lunit runit assoc
for *C* :: *′a comp* (**infixr** ‹·› *55*)
and *tensor* :: *′a ⇒ ′a ⇒ ′a* (**infixr** ‹⊗› *53*)
and *unity* :: *′a* (‹$\mathcal{I}$›)
and *lunit* :: *′a ⇒ ′a* (‹l[-]›)
and *runit* :: *′a ⇒ ′a* (‹r[-]›)
and *assoc* :: *′a ⇒ ′a ⇒ ′a ⇒ ′a* (‹a[-, -, -]›)
and *trm* :: *′a ⇒ ′a* (‹t[-]›)
and *dup* :: *′a ⇒ ′a* (‹d[-]›) +
assumes *trm-in-hom*: *ide a* ⟹ «t[*a*] : *a* → $\mathcal{I}$»
and *trm-unity*: t[$\mathcal{I}$] = $\mathcal{I}$

and *trm-naturality*: *arr f* ⟹ t[*cod f*] · *f* = t[*dom f*]
and *dup-in-hom* [*intro*]: *ide a* ⟹ «d[*a*] : *a* → *a* ⊗ *a*»
and *dup-naturality*: *arr f* ⟹ d[*cod f*] · *f* = (*f* ⊗ *f*) · d[*dom f*]
and *prj0-dup*: *ide a* ⟹ r[*a*] · (*a* ⊗ t[*a*]) · d[*a*] = *a*
and *prj1-dup*: *ide a* ⟹ l[*a*] · (t[*a*] ⊗ *a*) · d[*a*] = *a*
and *tuple-prj*: ⟦ *ide a*; *ide b* ⟧ ⟹ (r[*a*] · (*a* ⊗ t[*b*]) ⊗ l[*b*] · (t[*a*] ⊗ *b*)) · d[*a* ⊗ *b*] = *a* ⊗ *b*

context *cartesian-monoidal-category*
begin

interpretation *elementary-category-with-terminal-object C* $\mathcal{I}$ *the-trm*
⟨*proof*⟩

interpretation *elementary-monoidal-category C tensor unity lunit runit assoc*
⟨*proof*⟩

interpretation *elementary-cartesian-monoidal-category C*
tensor unity lunit runit assoc the-trm dup
⟨*proof*⟩

lemma *induces-elementary-cartesian-monoidal-category*$_{CMC}$:
shows *elementary-cartesian-monoidal-category C tensor* $\mathcal{I}$ *lunit runit assoc the-trm dup*
⟨*proof*⟩

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*] = $\mathcal{I}$
⟨*proof*⟩

lemma *dup-simps* [*simp*]:
assumes *ide a*
shows *arr* d[*a*] **and** *dom* d[*a*] = *a* **and** *cod* d[*a*] = *a* ⊗ *a*
⟨*proof*⟩

interpretation *elementary-category-with-terminal-object C* $\mathcal{I}$ *trm*
⟨*proof*⟩

lemma *is-elementary-category-with-terminal-object*:
shows *elementary-category-with-terminal-object C* $\mathcal{I}$ *trm*
⟨*proof*⟩

interpretation *MC*: *monoidal-category C T* α ι
⟨*proof*⟩

interpretation *ECBP*: *elementary-category-with-binary-products* C
‹$\lambda a\ b.\ \mathrm{l}[b] \cdot (\mathrm{t}[a] \otimes b)$› ‹$\lambda a\ b.\ \mathrm{r}[a] \cdot (a \otimes \mathrm{t}[b])$›
⟨*proof*⟩

lemma *induces-elementary-category-with-binary-products*$_{ECMC}$:
shows *elementary-category-with-binary-products* C
($\lambda a\ b.\ \mathrm{l}[b] \cdot (\mathrm{t}[a] \otimes b)$) ($\lambda a\ b.\ \mathrm{r}[a] \cdot (a \otimes \mathrm{t}[b])$)
⟨*proof*⟩

sublocale *cartesian-monoidal-category* C T α ι
⟨*proof*⟩

lemma *induces-cartesian-monoidal-category*$_{ECMC}$:
shows *cartesian-monoidal-category* C T α ι
⟨*proof*⟩

end

locale *diagonal-functor* =
C: *category* C +
CC: *product-category* C C
for C :: *'a comp*
begin

abbreviation *map*
where *map f* ≡ *if* C.*arr f then* (f, f) *else* CC.*null*

lemma *is-functor*:
shows *functor* C CC.*comp map*
⟨*proof*⟩

sublocale *functor* C CC.*comp map*
⟨*proof*⟩

end

context *cartesian-monoidal-category*
begin

sublocale Δ: *diagonal-functor* C ⟨*proof*⟩
interpretation *To*Δ: *composite-functor* C CC.*comp* C Δ.*map* T ⟨*proof*⟩

sublocale δ: *natural-transformation* C C *map* ‹T *o* Δ.*map*› *dup*
⟨*proof*⟩

end

5.4 Cartesian Monoidal Category from Cartesian Category

A cartesian category extends to a cartesian monoidal category by using the product structure to obtain the various canonical maps.

context *elementary-cartesian-category*
begin

interpretation *CC*: *product-category C C* ⟨*proof*⟩
interpretation *CCC*: *product-category C CC.comp* ⟨*proof*⟩
interpretation *T*: *binary-functor C C C Prod*
⟨*proof*⟩
interpretation *T*: *binary-endofunctor C Prod* ⟨*proof*⟩
interpretation *ToTC*: *functor CCC.comp C T.ToTC*
⟨*proof*⟩
interpretation *ToCT*: *functor CCC.comp C T.ToCT*
⟨*proof*⟩

interpretation α: *natural-isomorphism CCC.comp C T.ToTC T.ToCT* α
⟨*proof*⟩

interpretation *L*: *functor C C* ‹λ*f*. *Prod* (*cod* ι, *f*)›
⟨*proof*⟩
interpretation *L*: *endofunctor C* ‹λ*f*. *Prod* (*cod* ι, *f*)› ⟨*proof*⟩
interpretation 𝔩: *transformation-by-components C C*
‹λ*f*. *Prod* (*cod* ι, *f*)› *map* ‹λ*a*. *pr0* (*cod* ι) *a*›
⟨*proof*⟩
interpretation 𝔩: *natural-isomorphism C C* ‹λ*f*. *Prod* (*cod* ι, *f*)› *map* 𝔩.*map*
⟨*proof*⟩
interpretation *L*: *equivalence-functor C C* ‹λ*f*. *Prod* (*cod* ι, *f*)›
⟨*proof*⟩

interpretation *R*: *functor C C* ‹λ*f*. *Prod* (*f*, *cod* ι)›
⟨*proof*⟩
interpretation *R*: *endofunctor C*‹λ*f*. *Prod* (*f*, *cod* ι)› ⟨*proof*⟩
interpretation ϱ: *transformation-by-components C C*
‹λ*f*. *Prod* (*f*, *cod* ι)› *map* ‹λ*a*. $\mathfrak{p}_1$[*a*, *cod* ι]›
⟨*proof*⟩
interpretation ϱ: *natural-isomorphism C C* ‹λ*f*. *Prod* (*f*, *cod* ι)› *map* ϱ.*map*
⟨*proof*⟩
interpretation *R*: *equivalence-functor C C* ‹λ*f*. *Prod* (*f*, *cod* ι)›
⟨*proof*⟩

interpretation *MC*: *monoidal-category C Prod* α ι
⟨*proof*⟩

lemma *induces-monoidal-category*$_{ECC}$:
shows *monoidal-category C Prod* α ι
⟨*proof*⟩

lemma *unity-agreement*:
shows $MC.unity = \mathbf{1}$
  ⟨*proof*⟩

lemma *assoc-agreement*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $MC.assoc\ a\ b\ c = \mathrm{a}[a,\ b,\ c]$
  ⟨*proof*⟩

lemma *assoc′-agreement*:
assumes *ide a* **and** *ide b* **and** *ide c*
shows $MC.assoc'\ a\ b\ c = \mathrm{a}^{-1}[a,\ b,\ c]$
  ⟨*proof*⟩

lemma *runit-char-eqn*:
assumes *ide a*
shows $\mathrm{r}[a] \otimes \mathbf{1} = (a \otimes \iota) \cdot \mathrm{a}[a,\ \mathbf{1},\ \mathbf{1}]$
  ⟨*proof*⟩

lemma *runit-agreement*:
assumes *ide a*
shows $MC.runit\ a = \mathrm{r}[a]$
  ⟨*proof*⟩

lemma *lunit-char-eqn*:
assumes *ide a*
shows $\mathbf{1} \otimes \mathrm{l}[a] = (\iota \otimes a) \cdot \mathrm{a}^{-1}[\mathbf{1},\ \mathbf{1},\ a]$
⟨*proof*⟩

lemma *lunit-agreement*:
assumes *ide a*
shows $MC.lunit\ a = \mathrm{l}[a]$
  ⟨*proof*⟩

interpretation *CMC*: *cartesian-monoidal-category C Prod* α ι
⟨*proof*⟩

lemma *extends-to-cartesian-monoidal-category*$_{ECC}$:
shows *cartesian-monoidal-category C Prod* α ι
  ⟨*proof*⟩

lemma *trm-agreement*:
assumes *ide a*
shows $CMC.the\text{-}trm\ a = \mathrm{t}[a]$
  ⟨*proof*⟩

lemma *pr-agreement*:
assumes *ide a* **and** *ide b*
shows $CMC.pr_0\ a\ b = \mathfrak{p}_0[a,\ b]$ **and** $CMC.pr_1\ a\ b = \mathfrak{p}_1[a,\ b]$

⟨*proof*⟩

lemma *dup-agreement*:
assumes *ide a*
shows $CMC.dup\ a = \mathrm{d}[a]$
⟨*proof*⟩

end

5.5 Cartesian Monoidal Category from Elementary Cartesian Category

context *elementary-cartesian-category*
begin

interpretation *MC*: *monoidal-category C Prod* α ι
⟨*proof*⟩

lemma *triangle*:
assumes *ide a* **and** *ide b*
shows $(a \otimes \mathrm{l}[b]) \cdot \mathrm{a}[a, \mathbf{1}, b] = \mathrm{r}[a] \otimes b$
⟨*proof*⟩

lemma $\textit{induces-elementary-cartesian-monoidal-category}_{ECC}$:
shows *elementary-cartesian-monoidal-category* $(\cdot)$ *prod* $\mathbf{1}$ *lunit runit assoc trm dup*
⟨*proof*⟩

end

context *cartesian-category*
begin

interpretation *ECC*: *elementary-cartesian-category C*
some-pr0 some-pr1 some-terminal some-terminator
⟨*proof*⟩

lemma $\textit{extends-to-cartesian-monoidal-category}_{CC}$:
shows *cartesian-monoidal-category C ECC.Prod ECC.*α *ECC.*ι
⟨*proof*⟩

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.