# Monoidal Categories 

Eugene W. Stark<br>Department of Computer Science<br>Stony Brook University<br>Stony Brook, New York 11794 USA

September 13, 2023


#### 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

1 Introduction ..... 3
2 Monoidal Category ..... 6
2.1 Monoidal Category ..... 6
2.2 Elementary Monoidal Category ..... 18
2.3 Strict Monoidal Category ..... 22
2.4 Opposite Monoidal Category ..... 23
2.5 Monoidal Language ..... 23
2.6 Coherence ..... 35
3 Monoidal Functor ..... 42
3.1 Strict Monoidal Functor ..... 45
4 The Free Monoidal Category ..... 48
4.1 Syntactic Construction ..... 48
4.2 Proof of Freeness ..... 64
4.3 Strict Subcategory ..... 69
5 Cartesian Monoidal Category ..... 78
5.1 Symmetric Monoidal Category ..... 78
5.2 Cartesian Monoidal Category ..... 80
5.3 Elementary Cartesian Monoidal Category ..... 82
5.4 Cartesian Monoidal Category from Cartesian Category ..... 85
5.5 Cartesian Monoidal Category from Elementary Cartesian Category ..... 87

## Chapter 1

## Introduction

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

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

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

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

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

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<br>imports Category3.EquivalenceOfCategories<br>begin

### 2.1 Monoidal Category

A typical textbook presentation defines a monoidal category to be a category $C$ equipped with (among other things) a binary "tensor product" functor $\otimes: C \times C \rightarrow C$ and an "associativity" natural isomorphism $\alpha$, whose components are isomorphisms $\alpha(a, b, c)$ : $(a \otimes b) \otimes c \rightarrow a \otimes(b \otimes c)$ for objects $a, b$, and $c$ of $C$. This way of saying things avoids an explicit definition of the functors that are the domain and codomain of $\alpha$ and, in particular, what category serves as the domain of these functors. The domain category is in fact the product category $C \times C \times C$ and the domain and codomain of $\alpha$ are the functors $T o(T \times C): C \times C \times C \rightarrow C$ and $T o(C \times T): C \times C \times C \rightarrow C$. In a formal development, though, we can't gloss over the fact that $C \times C \times C$ has to mean either $C \times(C \times C)$ or $(C \times C) \times C$, which are not formally identical, and that associativities are somehow involved in the definitions of the functors $T o(T \times C)$ and $T o(C \times T)$. Here we 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 $\alpha: T o(T \times C) \rightarrow T o(C \times T)$, a unit object $\mathcal{I}$ of $C$, and an isomorphism $\iota: T(\mathcal{I}, \mathcal{I}) \rightarrow \mathcal{I}$, subject to two axioms: the pentagon axiom, which expresses the commutativity of certain pentagonal diagrams involving components of $\alpha$, and the left and right unit axioms, which state that the endofunctors $T(\mathcal{I},-)$ and $T(-$,
$\mathcal{I})$ are equivalences of categories. This definition is formalized in the monoidal-category locale.

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

```
locale monoidal-category \(=\)
    category \(C+\)
    CC: product-category \(C\) C +
    CCC: product-category C CC.comp +
    T: binary-endofunctor \(C T+\)
    a: natural-isomorphism CCC.comp C T.ToTC T.ToCT \(\alpha+\)
    \(L\) : equivalence-functor \(C C \lambda f . T(\operatorname{cod} \iota, f)+\)
    \(R\) : equivalence-functor \(C C \lambda f . T(f, \operatorname{cod} \iota)\)
for \(C::\) 'a comp (infixr - 55)
and \(T:: ' a * ' a \Rightarrow{ }^{\prime} a\)
and \(\alpha::{ }^{\prime} a *{ }^{\prime} a *{ }^{\prime} a \Rightarrow{ }^{\prime} a\)
and \(\iota::{ }^{\prime} a+\)
assumes unit-in-hom-ax: «८:T( \(\operatorname{cod} \iota, \operatorname{cod} \iota) \rightarrow \operatorname{cod} \iota »\)
and unit-is-iso: iso ८
and pentagon: \(\llbracket\) ide a; ide \(b\); ide \(c\); ide \(d \rrbracket \Longrightarrow\)
                                    \(T(a, \alpha(b, c, d)) \cdot \alpha(a, T(b, c), d) \cdot T(\alpha(a, b, c), d)=\)
                                    \(\alpha(a, b, T(c, d)) \cdot \alpha(T(a, b), c, d)\)
```


## begin

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

```
abbreviation unity \(::{ }^{\prime} a(\mathcal{I})\)
where unity \(\equiv \operatorname{cod} \iota\)
abbreviation \(L::{ }^{\prime} a \Rightarrow^{\prime} a\)
where \(L f \equiv T(\mathcal{I}, f)\)
abbreviation \(R::^{\prime} a \Rightarrow{ }^{\prime} a\)
where \(R f \equiv T(f, \mathcal{I})\)
abbreviation tensor \(\quad(\) infixr \(\otimes 53)\)
where \(f \otimes g \equiv T(f, g)\)
abbreviation assoc \(\quad(\mathrm{a}[-,-,-])\)
```

where $\mathrm{a}[a, b, c] \equiv \alpha(a, b, c)$
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 \(f . « f: \mathcal{I} \otimes a \rightarrow a » \wedge \mathcal{I} \otimes f=(\iota \otimes a) \cdot \operatorname{inv} \mathrm{a}[\mathcal{I}, \mathcal{I}, a]\)
definition runit (r[-])
where runit \(a \equiv T H E f . « f: a \otimes \mathcal{I} \rightarrow a » \wedge f \otimes \mathcal{I}=(a \otimes \iota) \cdot \mathrm{a}[a, \mathcal{I}, \mathcal{I}]\)
```

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}->\mathcal{I}»
    <proof>
```

```
lemma ide-unity [simp]:
```

lemma ide-unity [simp]:
shows ide I
shows ide I
\langleproof\rangle

```
    \langleproof\rangle
```

lemma tensor-in-hom [simp]:
assumes $« f: a \rightarrow b »$ and $« g: c \rightarrow d »$
shows $« f \otimes g: a \otimes c \rightarrow b \otimes d »$
$\langle p r o o f\rangle$
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 $« f \otimes g: x \rightarrow y$ »
$\langle p r o o f\rangle$
lemma arr-tensor [simp]:
assumes $\operatorname{arr} f$ and $\operatorname{arr} g$
shows $\operatorname{arr}(f \otimes g)$
$\langle p r o o f\rangle$
lemma dom-tensor [simp]:
assumes $« f: a \rightarrow b »$ and $« g: c \rightarrow d »$
shows $\operatorname{dom}(f \otimes g)=a \otimes c$
$\langle p r o o f\rangle$
lemma cod-tensor [simp]:
assumes $« f: a \rightarrow b »$ and $« g: c \rightarrow d »$
shows $\operatorname{cod}(f \otimes g)=b \otimes d$
$\langle p r o o f\rangle$
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 $(f \otimes g)$
$\langle$ proof〉
lemma inv－tensor［simp］：
assumes iso $f$ and iso $g$
shows $\operatorname{inv}(f \otimes g)=\operatorname{inv} f \otimes i n v g$
$\langle$ proof $\rangle$
lemma interchange：
assumes seq $h g$ and seq $h^{\prime} g^{\prime}$
shows $\left(h \otimes h^{\prime}\right) \cdot\left(g \otimes g^{\prime}\right)=h \cdot g \otimes h^{\prime} \cdot g^{\prime}$
$\langle p r o o f\rangle$
lemma $\alpha$－simp：
assumes $\operatorname{arr} f$ and $\operatorname{arr} g$ and $\operatorname{arr} h$
shows $\alpha(f, g, h)=(f \otimes g \otimes h) \cdot \mathrm{a}[\operatorname{dom} f, \operatorname{dom} g, \operatorname{dom} h]$〈proof〉
lemma assoc－in－hom［intro］：
assumes $i d e ~ a ~ a n d ~ i d e ~ b ~ a n d ~ i d e ~ c ~$
shows «a $[a, b, c]:(a \otimes b) \otimes c \rightarrow a \otimes b \otimes c »$
$\langle p r o o f\rangle$
lemma arr－assoc［simp］：
assumes ide $a$ and ide $b$ and ide $c$
shows arr a $[a, b, c]$
$\langle p r o o f\rangle$
lemma dom－assoc［simp］：
assumes ide $a$ and ide $b$ and ide $c$
shows dom $\mathrm{a}[a, b, c]=(a \otimes b) \otimes c$
$\langle p r o o f\rangle$
lemma cod－assoc［simp］：
assumes ide $a$ and ide $b$ and ide $c$
shows cod $\mathrm{a}[a, b, c]=a \otimes b \otimes c$
$\langle p r o o f\rangle$
lemma assoc－naturality：
assumes arr f0 and arr f1 and arr f2
shows a［cod f0，cod f1，cod f2］$\cdot((f 0 \otimes f 1) \otimes f 2)=$

```
    (f0\otimesf1\otimesf2) · 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 $1[a]$ of the left unitor．It follows that $1[a]$ ，as given by our definition using definite description，satisfies this characteristic equation and is therefore uniquely determined by by $\otimes, \alpha$ ，and $\iota$ ．

```
lemma lunit-char:
assumes ide a
shows \(« 1[a]: \mathcal{I} \otimes a \rightarrow a »\) and \(\mathcal{I} \otimes 1[a]=(\iota \otimes a) \cdot i n v \mathrm{a}[\mathcal{I}, \mathcal{I}, a]\)
and \(\exists\) !f. \(« f: \mathcal{I} \otimes a \rightarrow a » \wedge \mathcal{I} \otimes f=(\iota \otimes a) \cdot\) inv \(\mathrm{a}[\mathcal{I}, \mathcal{I}, a]\)
〈proof〉
lemma lunit-in-hom [intro]:
assumes ide a
shows \(« 1[a]: \mathcal{I} \otimes a \rightarrow a »\)
    〈proof〉
lemma arr-lunit [simp]:
assumes ide a
shows arr \(1[a]\)
    \(\langle p r o o f\rangle\)
```

lemma dom-lunit [simp]:
assumes ide a
shows dom $\mathrm{l}[a]=\mathcal{I} \otimes a$
$\langle p r o o f\rangle$
lemma cod-lunit [simp]:
assumes ide a
shows $\operatorname{cod} l[a]=a$
$\langle p r o o f\rangle$

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

```
lemma iso-lunit [simp]:
assumes ide a
shows iso \(1[a]\)
    〈proof〉
```

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

```
assumes \(« f: \mathcal{I} \otimes a \rightarrow a »\) and \(\mathcal{I} \otimes f=(\iota \otimes a) \cdot \operatorname{inv} \mathrm{a}[\mathcal{I}, \mathcal{I}, a]\)
shows \(f=\mathrm{l}[a]\)
\(\langle\) proof \(\rangle\)
```

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

```
lemma runit-char:
assumes ide a
shows \(« \mathrm{r}[a]: a \otimes \mathcal{I} \rightarrow a »\) and \(\mathrm{r}[a] \otimes \mathcal{I}=(a \otimes \iota) \cdot \mathrm{a}[a, \mathcal{I}, \mathcal{I}]\)
and \(\exists!f . « f: a \otimes \mathcal{I} \rightarrow a » \wedge f \otimes \mathcal{I}=(a \otimes \iota) \cdot \mathrm{a}[a, \mathcal{I}, \mathcal{I}]\)
\(\langle p r o o f\rangle\)
lemma runit-in-hom [intro]:
assumes ide a
shows \(« \mathrm{r}[a]: a \otimes \mathcal{I} \rightarrow a »\)
    \(\langle p r o o f\rangle\)
lemma arr-runit [simp]:
assumes ide a
shows arr r \([a]\)
    \(\langle p r o o f\rangle\)
lemma dom-runit [simp]:
assumes ide a
shows dom \(\mathrm{r}[a]=a \otimes \mathcal{I}\)
    \(\langle p r o o f\rangle\)
lemma cod-runit [simp]:
assumes ide a
shows \(\operatorname{cod} \mathrm{r}[a]=a\)
    \(\langle p r o o f\rangle\)
lemma runit-eqI:
assumes \(« f: a \otimes \mathcal{I} \rightarrow a »\) and \(f \otimes \mathcal{I}=(a \otimes \iota) \cdot \mathrm{a}[a, \mathcal{I}, \mathcal{I}]\)
shows \(f=\mathrm{r}[a]\)
〈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 $\operatorname{arr} f$
shows $\mathrm{l}[\operatorname{cod} f] \cdot(\mathcal{I} \otimes f)=f \cdot \mathrm{l}[\operatorname{dom} f]$
〈proof〉
lemma runit－naturality：
assumes $\operatorname{arr} f$
shows $\mathrm{r}[\operatorname{cod} f] \cdot(f \otimes \mathcal{I})=f \cdot \mathrm{r}[\operatorname{dom} f]$
〈proof〉
The next two definitions extend the unitors to all arrows，not just identities．Un－ fortunately，the traditional symbol $\lambda$ for the left unitor is already reserved for a higher purpose，so we have to make do with a poor substitute．

```
abbreviation l
where l f}\equiv\mathrm{ if arr f then f · l[dom f] else null
abbreviation \varrho
where \varrho f}\equiv\mathrm{ if arr f then f r r[dom f] else null
lemma l-ide-simp:
assumes ide a
shows }\mathfrak{l}a=1[a
    <proof>
    lemma \varrho-ide-simp:
    assumes ide a
    shows \varrho a r 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\rangle
sublocale @: natural-transformation C C R map @
<proof>
sublocale \varrho: natural-isomorphism C C R map \varrho
    <proof>
sublocale l': inverse-transformation C C L map l <proof\rangle
sublocale \varrho': inverse-transformation C C R map \varrho \langleproof\rangle
sublocale \mp@subsup{\alpha}{}{\prime}:\mathrm{ inverse-transformation CCC.comp C T.ToTC T.ToCT 人 <proof }\rangle
abbreviation }\mp@subsup{\alpha}{}{\prime
where }\mp@subsup{\alpha}{}{\prime}\equiv\mp@subsup{\alpha}{}{\prime}.ma
abbreviation assoc'( (a
```

where $\mathrm{a}^{-1}[a, b, c] \equiv$ inv $\mathrm{a}[a, b, c]$
lemma $\alpha^{\prime}$-ide-simp:
assumes ide $a$ and ide $b$ and ide $c$
shows $\alpha^{\prime}(a, b, c)=\mathrm{a}^{-1}[a, b, c]$
$\langle p r o o f\rangle$
lemma $\alpha^{\prime}$-simp:
assumes $\operatorname{arr} f$ and $\operatorname{arr} g$ and $\operatorname{arr} h$
shows $\alpha^{\prime}(f, g, h)=((f \otimes g) \otimes h) \cdot \mathrm{a}^{-1}[\operatorname{dom} f, \operatorname{dom} g, \operatorname{dom} h]$
$\langle p r o o f\rangle$
lemma assoc-inv:
assumes ide $a$ and ide $b$ and ide $c$
shows inverse-arrows $\mathrm{a}[a, b, c] \mathrm{a}^{-1}[a, b, c]$
$\langle p r o o f\rangle$
lemma assoc'-in-hom [intro]:
assumes ide $a$ and ide $b$ and ide $c$
shows « $\mathrm{a}^{-1}[a, b, c]: a \otimes b \otimes c \rightarrow(a \otimes b) \otimes c »$
$\langle p r o o f\rangle$
lemma arr-assoc ${ }^{\prime}[$ simp $]$ :
assumes ide $a$ and ide $b$ and ide $c$
shows arr $\mathrm{a}^{-1}[a, b, c]$
$\langle p r o o f\rangle$
lemma dom-assoc ${ }^{\prime}[$ simp]:
assumes ide $a$ and ide $b$ and ide $c$
shows dom $\mathrm{a}^{-1}[a, b, c]=a \otimes b \otimes c$
$\langle p r o o f\rangle$
lemma cod-assoc ${ }^{\prime}$ [simp]:
assumes ide $a$ and ide $b$ and ide c
shows $\operatorname{cod} \mathrm{a}^{-1}[a, b, c]=(a \otimes b) \otimes c$
〈proof〉
lemma comp-assoc-assoc ${ }^{\prime}$ [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 $\mathrm{a}^{-1}[a, b, c] \cdot \mathrm{a}[a, b, c]=(a \otimes b) \otimes c$
$\langle p r o o f\rangle$
lemma assoc'-naturality:
assumes arr f0 and arr f1 and arr f2
shows $((f 0 \otimes f 1) \otimes f 2) \cdot \mathrm{a}^{-1}[\operatorname{dom} f 0$, dom $f 1$, dom f2 $]=$ $\mathrm{a}^{-1}[\operatorname{cod} f 0, \operatorname{cod} f 1, \operatorname{cod} f 2] \cdot(f 0 \otimes f 1 \otimes f 2)$
$\langle p r o o f\rangle$

```
abbreviation }\mp@subsup{\mathfrak{l}}{}{\prime
where }\mp@subsup{\mathfrak{l}}{}{\prime}\equiv\mp@subsup{\mathfrak{l}}{}{\prime}.ma
abbreviation lunit' (l-1[-])
where l}\mp@subsup{l}{}{-1}[a]\equiv\operatorname{inv l[a]
lemma l'-ide-simp:
assumes ide a
shows l'.map a= 1-1 [a]
    \langleproof\rangle
lemma lunit-inv:
assumes ide a
shows inverse-arrows l[a] 1-1[a]
    <proof\rangle
lemma lunit'-in-hom [intro]:
assumes ide a
shows «1 }\mp@subsup{1}{}{-1}[a]:a->\mathcal{I}\otimesa
    \langleproof\rangle
lemma comp-lunit-lunit' [simp]:
assumes ide a
shows }\textrm{l}[a]\cdot\mp@subsup{\textrm{l}}{}{-1}[a]=
and \mp@subsup{1}{}{-1}[a]\cdot\textrm{l}[a]=\mathcal{I}\otimesa
<proof>
lemma lunit'-naturality:
assumes arr f
shows }(\mathcal{I}\otimesf)\cdot\mp@subsup{l}{}{-1}[\operatorname{dom}f]=\mp@subsup{1}{}{-1}[\operatorname{cod}f]\cdot
    \langleproof\rangle
abbreviation \varrho}\mp@subsup{\varrho}{}{\prime
where }\mp@subsup{\varrho}{}{\prime}\equiv\mp@subsup{\varrho}{}{\prime}.ma
abbreviation runit' (r}\mp@subsup{}{}{-1}[-]
where r }\mp@subsup{}{}{-1}[a]\equivinv r [a
lemma \varrho'-ide-simp:
assumes ide a
shows \varrho}\mp@subsup{\varrho}{}{\prime}.map a= \mp@subsup{r}{}{-1}[a
    \langleproof\rangle
lemma runit-inv:
assumes ide a
shows inverse-arrows r[a] r}\mp@subsup{}{}{-1}[a
    \langleproof\rangle
lemma runit'-in-hom [intro]:
```

```
assumes ide a
shows « \(\mathrm{r}^{-1}[a]: a \rightarrow a \otimes \mathcal{I}\) »
    〈proof〉
lemma comp-runit-runit' \([\) simp \(]\) :
assumes ide a
shows \(\mathrm{r}[a] \cdot \mathrm{r}^{-1}[a]=a\)
and \(\mathrm{r}^{-1}[a] \cdot \mathrm{r}[a]=a \otimes \mathcal{I}\)
\(\langle\) proof \(\rangle\)
lemma runit'-naturality:
assumes \(\operatorname{arr} f\)
shows \((f \otimes \mathcal{I}) \cdot \mathrm{r}^{-1}[\operatorname{dom} f]=\mathrm{r}^{-1}[\operatorname{cod} f] \cdot f\)
    \(\langle p r o o f\rangle\)
```

lemma lunit-commutes-with- $L$ :
assumes ide a
shows $\mathrm{l}[\mathcal{I} \otimes a]=\mathcal{I} \otimes 1[a]$
〈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 $(a \otimes \mathrm{l}[b]) \cdot \mathrm{a}[a, \mathcal{I}, b]=\mathrm{r}[a] \otimes b$
$\langle p r o o f\rangle$
lemma lunit－tensor－gen：
assumes ide $a$ and ide $b$ and ide $c$
shows $(a \otimes 1[b \otimes c]) \cdot(a \otimes \mathrm{a}[\mathcal{I}, b, c])=a \otimes 1[b] \otimes c$
〈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 $\mathrm{l}[a \otimes b]=(\mathrm{l}[a] \otimes b) \cdot \mathrm{a}^{-1}[\mathcal{I}, a, b]$

$$
\langle p r o o f\rangle
$$

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 $\mathrm{r}[a \otimes b]=(a \otimes \mathrm{r}[b]) \cdot \mathrm{a}[a, b, \mathcal{I}]$
$\langle p r o o f\rangle$
lemma runit－tensor＇：
assumes ide $a$ and ide $b$
shows $\mathrm{r}[a \otimes b] \cdot \mathrm{a}^{-1}[a, b, \mathcal{I}]=a \otimes \mathrm{r}[b]$
$\langle p r o o f\rangle$
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 $i d e b$ and ide $c$ and ide $d$
shows $\left(\left(\mathrm{a}^{-1}[a, b, c] \otimes d\right) \cdot \mathrm{a}^{-1}[a, b \otimes c, d]\right) \cdot\left(a \otimes \mathrm{a}^{-1}[b, c, d]\right)$ $=\mathrm{a}^{-1}[a \otimes b, c, d] \cdot \mathrm{a}^{-1}[a, b, c \otimes d]$
〈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 $1[\mathcal{I}]=\iota$ and $\mathrm{r}[\mathcal{I}]=\iota$
$\langle$ proof〉
lemma unit－triangle：
shows $\iota \otimes \mathcal{I}=(\mathcal{I} \otimes \iota) \cdot \mathrm{a}[\mathcal{I}, \mathcal{I}, \mathcal{I}]$
and $(\iota \otimes \mathcal{I}) \cdot \mathrm{a}^{-1}[\mathcal{I}, \mathcal{I}, \mathcal{I}]=\mathcal{I} \otimes \iota$
$\langle$ proof $\rangle$
The only isomorphism that commutes with $\iota$ 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 $\iota$ of a monoidal category is unique up to a unique iso－ morphism（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 \(T::^{\prime} a *{ }^{\prime} a \Rightarrow{ }^{\prime} a\)
and \(\alpha::{ }^{\prime} a *{ }^{\prime} a *{ }^{\prime} a \Rightarrow{ }^{\prime} a\)
and \(\iota::{ }^{\prime} a\)
and \(\iota_{1}::\) ' \(a\)
begin
```

```
no-notation \(C_{1}\).tensor \((\mathbf{i n f i x r} \otimes 53)\)
no-notation \(C_{1}\).unity ( \(\mathcal{I}\) )
no-notation \(C_{1}\).lunit (l[-])
no-notation \(C_{1}\).runit ( \(\mathrm{r}[-]\) )
no-notation \(C_{1}\).assoc (a[-, -, -])
no-notation \(C_{1} \cdot \operatorname{assoc}^{\prime}\left(\mathrm{a}^{-1}[-,-,-]\right)\)
notation \(C_{1}\).tensor \(\quad\left(\mathbf{i n f i x r} \otimes_{1} 53\right)\)
notation \(C_{1}\).unity \(\quad\left(\mathcal{I}_{1}\right)\)
notation \(C_{1}\).lunit \(\quad\left(l_{1}[-]\right)\)
notation \(C_{1}\).runit \(\quad\left(\mathrm{r}_{1}[-]\right)\)
notation \(C_{1}\).assoc \(\quad\left(\mathrm{a}_{1}[-,-,-]\right)\)
notation \(C_{1}\).assoc \({ }^{\prime} \quad\left(\mathrm{a}_{1}^{-1}[-,-,-]\right)\)
definition \(i\)
where \(i \equiv \mathrm{l}\left[\mathcal{I}_{1}\right] \cdot\) inv \(\mathrm{r}_{1}[\mathcal{I}]\)
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- \(\iota\)-to- \(\iota_{1}\) :
shows \(i \cdot \iota=\iota_{1} \cdot(i \otimes i)\)
〈proof〉
lemma inv-i-iso-ı:
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 \((\operatorname{inv} i \cdot f) \cdot \iota=\iota \cdot(\operatorname{inv} i \cdot f \otimes \operatorname{inv} i \cdot f)\)
〈proof〉
lemma unit-unique-upto-unique-iso:
shows \(\exists!f . 《 f: \mathcal{I} \rightarrow \mathcal{I}_{1} » \wedge\) iso \(f \wedge f \cdot \iota=\iota_{1} \cdot(f \otimes f)\)
\(\langle p r o o f\rangle\)
```

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 \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathbf{i n f i x r} \otimes 53)\)
and unity \(::{ }^{\prime} a\)
    (I)
and lunit \(::{ }^{\prime} a \Rightarrow\) ' \(a\)
and runit \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad\) (r[-])
and assoc :: ' \(a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{a}[-,-,-])+\)
assumes ide-unity [simp]: ide \(\mathcal{I}\)
and iso-lunit: ide \(a \Longrightarrow\) iso \(1[a]\)
and iso-runit: ide \(a \Longrightarrow\) iso \(\mathrm{r}[a]\)
and iso-assoc: \(\llbracket\) ide \(a\); ide \(b\); ide \(c \rrbracket \Longrightarrow\) iso \(\mathrm{a}[a, b, c]\)
and tensor-in-hom \([\) simp \(]: \llbracket « f: a \rightarrow b » ; « g: c \rightarrow d » \rrbracket \Longrightarrow « f \otimes g: a \otimes c \rightarrow b \otimes d »\)
and tensor-preserves-ide: \(\llbracket\) ide \(a\); ide \(b \rrbracket \Longrightarrow\) ide \((a \otimes b)\)
and interchange: \(\llbracket \operatorname{seq} g f ; \operatorname{seq} g^{\prime} f^{\prime} \rrbracket \Longrightarrow\left(g \otimes g^{\prime}\right) \cdot\left(f \otimes f^{\prime}\right)=g \cdot f \otimes g^{\prime} \cdot f^{\prime}\)
and lunit-in-hom \([\) simp \(]\) : ide \(a \Longrightarrow « 1[a]: \mathcal{I} \otimes a \rightarrow a »\)
and lunit-naturality: arr \(f \Longrightarrow \mathrm{l}[\operatorname{cod} f] \cdot(\mathcal{I} \otimes f)=f \cdot \mathrm{l}[\operatorname{dom} f]\)
and runit-in-hom \([\) simp \(]\) : ide \(a \Longrightarrow 《 \mathrm{r}[a]: a \otimes \mathcal{I} \rightarrow a »\)
and runit-naturality: arr \(f \Longrightarrow \mathrm{r}[\operatorname{cod} f] \cdot(f \otimes \mathcal{I})=f \cdot \mathrm{r}[\operatorname{dom} f]\)
and assoc-in-hom [simp]:
    【ide \(a\); ide \(b ;\) ide \(c \rrbracket \Longrightarrow « \mathrm{a}[a, b, c]:(a \otimes b) \otimes c \rightarrow a \otimes b \otimes c »\)
and assoc-naturality:
    \(\llbracket \operatorname{arr} f 0 ; \operatorname{arr} f 1 ; \operatorname{arr} f 2 \rrbracket \Longrightarrow \mathrm{a}[\operatorname{cod} f 0, \operatorname{cod} f 1, \operatorname{cod} f 2] \cdot((f 0 \otimes f 1) \otimes f 2)\)
                        \(=(f 0 \otimes(f 1 \otimes f 2)) \cdot \mathrm{a}[\operatorname{dom} f 0\), dom f1, dom f2 \(]\)
and triangle: \(\llbracket\) ide \(a\); ide \(b \rrbracket \Longrightarrow(a \otimes 1[b]) \cdot \mathrm{a}[a, \mathcal{I}, b]=\mathrm{r}[a] \otimes b\)
and pentagon: 【ide a; ide \(b\); ide \(c\); ide \(d \rrbracket \Longrightarrow\)
    \((a \otimes \mathrm{a}[b, c, d]) \cdot \mathrm{a}[a, b \otimes c, d] \cdot(\mathrm{a}[a, b, c] \otimes d)\)
\(\quad=\mathrm{a}[a, b, c \otimes d] \cdot \mathrm{a}[a \otimes b, c, d]\)
```

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

```
context monoidal-category
begin
    lemma induces-elementary-monoidal-category:
    shows elementary-monoidal-category C tensor }\mathcal{I}\mathrm{ lunit runit assoc
        <proof\rangle
```

end

```
context elementary-monoidal-category
begin
interpretation CC: product-category C C \langleproof\rangle
interpretation CCC: product-category C CC.comp \langleproof\rangle
definition T :: ' }a*\mathrm{ *' }a=>\mp@subsup{|}{}{\prime}
where Tf \equivif CC.arr f then (fst f\otimes snd f) else null
lemma T-simp [simp]:
assumes arr f and arr g
shows}T(f,g)=f\otimes
    <proof>
lemma arr-tensor [simp]:
assumes arr f and arr g
shows arr (f\otimesg)
    <proof\rangle
lemma dom-tensor [simp]:
assumes arr f and arr g
shows dom (f\otimesg) = dom f\otimes\operatorname{dom}g
    <proof\rangle
lemma cod-tensor [simp]:
assumes arr f and arr g
shows cod (f\otimesg) = \operatorname{cod}f\otimes\operatorname{cod}g
    <proof>
interpretation T: binary-endofunctor C T
    <proof\rangle
lemma binary-endofunctor-T:
shows binary-endofunctor C T \langleproof\rangle
interpretation ToTC: functor CCC.comp C T.ToTC
    <proof\rangle
interpretation ToCT: functor CCC.comp C T.ToCT
    <proof>
definition \alpha
where \alphaf\equiv if CCC.arr f
    then (fst f \otimes (fst (snd f)\otimes snd (snd f))).
        a[dom (fst f), dom (fst (snd f)), dom (snd (snd f))]
    else null
lemma \alpha-ide-simp [simp]:
```

```
assumes ide \(a\) and ide \(b\) and ide \(c\)
shows \(\alpha(a, b, c)=\mathrm{a}[a, b, c]\)
    \(\langle p r o o f\rangle\)
lemma arr-assoc [simp]:
assumes ide \(a\) and ide \(b\) and ide \(c\)
shows arr a \([a, b, c]\)
    \(\langle p r o o f\rangle\)
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 \(\operatorname{cod} \mathrm{a}[a, b, c]=a \otimes b \otimes c\)
    \(\langle p r o o f\rangle\)
```

interpretation $\alpha$ : natural-isomorphism CCC.comp C T.ToTC T.ToCT $\alpha$
〈proof〉
interpretation $\alpha^{\prime}$ : inverse-transformation CCC.comp C T.ToTC T.ToCT $\alpha\langle p r o o f\rangle$
interpretation $L$ : functor $C C\langle\lambda f . T(\mathcal{I}, f)\rangle$
$\langle p r o o f\rangle$
interpretation $R$ : functor $C C\langle\lambda f . T(f, \mathcal{I})\rangle$
〈proof〉
interpretation $\mathfrak{l}$ : natural-isomorphism $C C\langle\lambda f . T(\mathcal{I}, f)\rangle$ map
$\langle\lambda f$. if arr $f$ then $f \cdot \mathrm{l}[$ dom $f]$ else null $\rangle$
$\langle p r o o f\rangle$
interpretation $\varrho$ : natural-isomorphism $C C\langle\lambda f . T(f, \mathcal{I})\rangle$ map
$\langle\lambda f$. if arr $f$ then $f \cdot \mathrm{r}[\operatorname{dom} f]$ else null $\rangle$
$\langle p r o o f\rangle$

The endofunctors $\lambda f . T(\mathcal{I}, f)$ and $\lambda f . T(f, \mathcal{I})$ are equivalence functors，due to the existence of the unitors．
interpretation $L$ ：equivalence－functor $C C\langle\lambda f . T(\mathcal{I}, f)\rangle$
〈proof〉
interpretation $R$ ：equivalence－functor $C C\langle\lambda f . T(f, \mathcal{I})\rangle$
〈proof〉
To complete an interpretation of the monoidal－category locale，we define $\iota \equiv l[\mathcal{I}]$ ．We could also have chosen $\iota \equiv \varrho[\mathcal{I}]$ as the two are equal，though to prove that requires some work yet．

```
definition \(\iota\)
where \(\iota \equiv l[\mathcal{I}]\)
lemma \(\iota\)-in-hom:
shows «८: \(\mathcal{I} \otimes \mathcal{I} \rightarrow \mathcal{I}\) »
    \(\langle p r o o f\rangle\)
```

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 $M C$ 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 interpre－ tation 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 $=(\lambda f . T(f, \mathcal{I}))$
$\langle p r o o f\rangle$

We wish to show that the components of the unitors MC．l and MC．＠defined in the induced interpretation $M C$ 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 $M C$ 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 $M C$ that both are equal to $\iota$ ．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 $1[\mathcal{I}]=\iota$ and $r[\mathcal{I}]=\iota$
〈proof〉
lemma lunit－char：
assumes ide a
shows $\mathcal{I} \otimes \mathrm{l}[a]=(\iota \otimes a) \cdot \operatorname{inv} \mathrm{a}[\mathcal{I}, \mathcal{I}, a]$

```
    <proof\rangle
    lemma runit-char:
    assumes ide a
    shows r[a]\otimes\mathcal{I}=(a\otimes\iota)\cdot\textrm{a}[a,\mathcal{I},\mathcal{I}]
        \langleproof\rangle
    lemma l-agreement:
    shows MC.l = (\lambdaf. if arr f then f | l [dom f] else null )
    <proof>
    lemma \varrho-agreement:
    shows MC.\varrho = (\lambdaf. if arr f then f r r[dom f] else null)
    <proof>
    lemma lunit-agreement:
    assumes ide a
    shows MC.lunit a = [ [a]
        <proof>
    lemma runit-agreement:
    assumes ide a
    shows MC.runit a = 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: 【ide a0; ide a1; ide a2 】 \(\Longrightarrow i d e \mathrm{a}[a 0, a 1, a 2]\)
and strict-lunit: ide \(a \Longrightarrow \mathrm{l}[a]=a\)
and strict-runit: ide \(a \Longrightarrow \mathrm{r}[a]=a\)
begin
    lemma strict-unit:
    shows \(\iota=\mathcal{I}\)
        \(\langle p r o o f\rangle\)
    lemma tensor-assoc [simp]:
    assumes arr \(f 0\) and \(\operatorname{arr} f 1\) and arr \(f 2\)
    shows \((f 0 \otimes f 1) \otimes f 2=f 0 \otimes f 1 \otimes f 2\)
    \(\langle p r o o f\rangle\)
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 T 的 \iota
for C :: 'a comp (infixr - 55)
and T}\mp@subsup{T}{C}{}::' 'a*'a=>''
and \mp@subsup{\alpha}{C}{}::' 'a*'a*' a=>''a
and \iota:: 'a
begin
    abbreviation T
    where Tf\equiv T
    abbreviation \alpha
    where \alphaf\equivC.\alpha'}(\mathrm{ snd (snd f), fst (snd f), fst f)
end
sublocale opposite-monoidal-category \subseteqmonoidal-category C T \alpha \iota
<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 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 \(=\)
```

datatype (discs-sels) 't term $=$
Prim 't
Prim 't
$(\langle-\rangle)$
$(\langle-\rangle)$
Unity ( ( )
Unity ( ( )
| Tensor 't term 't term (infixr $\otimes 53)$
| Tensor 't term 't term (infixr $\otimes 53)$
| Comp 't term 't term (infixr •55)
| Comp 't term 't term (infixr •55)
| Lunit 't term (l[-])
| Lunit 't term (l[-])
| Lunit' 't term ( $\left.\mathbf{l}^{-1}[-]\right)$
| Lunit' 't term ( $\left.\mathbf{l}^{-1}[-]\right)$
| Runit 't term (r[-])
| Runit 't term (r[-])
| Runit' 't term ( $\left.\mathbf{r}^{-1}[-]\right)$
| Runit' 't term ( $\left.\mathbf{r}^{-1}[-]\right)$
| Assoc 't term 't term 't term (a[-, -, -])
| Assoc 't term 't term 't term (a[-, -, -])
| Assoc' 't term 't term 't term ( $\left.\mathbf{a}^{-1}[-,-,-]\right)$
| Assoc' 't term 't term 't term ( $\left.\mathbf{a}^{-1}[-,-,-]\right)$
lemma not-is-Tensor-Unity:
shows ᄀ is-Tensor Unity
<proof\rangle

```

We define formal domain and codomain functions on terms.
```

primrec Dom :: 'a term $\Rightarrow$ 'a term
where $\operatorname{Dom}\langle f\rangle=\langle C . d o m f\rangle$
$\mid \operatorname{Dom} \boldsymbol{I}=\boldsymbol{I}$
$\mid \operatorname{Dom}(t \otimes u)=(\operatorname{Dom} t \otimes \operatorname{Dom} u)$
$\operatorname{Dom}(t \cdot u)=\operatorname{Dom} u$
| Dom $\mathbf{1}[t]=(\boldsymbol{\mathcal { I }} \otimes \operatorname{Dom} t)$
| Dom $\mathbf{l}^{-1}[t]=\operatorname{Dom} t$
| Dom $\mathbf{r}[t]=(\operatorname{Dom} t \otimes \boldsymbol{I})$
| Dom $\mathbf{r}^{-1}[t]=\operatorname{Dom} t$
$\operatorname{Dom} \mathbf{a}[t, u, v]=((\operatorname{Dom} t \otimes \operatorname{Dom} u) \otimes \operatorname{Dom} v)$
$\mid \operatorname{Dom} \mathbf{a}^{-1}[t, u, v]=(\operatorname{Dom} t \otimes(\operatorname{Dom} u \otimes \operatorname{Dom} v))$
primrec Cod :: 'a term $\Rightarrow$ 'a term
where $\operatorname{Cod}\langle f\rangle=\langle C \cdot \operatorname{cod} f\rangle$
$\operatorname{Cod} \boldsymbol{I}=\boldsymbol{I}$
$\operatorname{Cod}(t \otimes u)=(\operatorname{Cod} t \otimes \operatorname{Cod} u)$
$\operatorname{Cod}(t \cdot u)=\operatorname{Cod} t$
$\mid \operatorname{Cod} \mathbf{l}[t]=\operatorname{Cod} t$
$\mid \operatorname{Cod} \mathbf{1}^{-1}[t]=(\boldsymbol{\mathcal { I }} \otimes \operatorname{Cod} t)$
$\mid \operatorname{Cod} \mathbf{r}[t]=\operatorname{Cod} t$
$\operatorname{Cod} \mathbf{r}^{-1}[t]=(\operatorname{Cod} t \otimes \boldsymbol{I})$
$\operatorname{Cod} \mathbf{a}[t, u, v]=(\operatorname{Cod} t \otimes(\operatorname{Cod} u \otimes \operatorname{Cod} v))$
$\mid \operatorname{Cod} \mathbf{a}^{-1}[t, u, v]=((\operatorname{Cod} t \otimes \operatorname{Cod} u) \otimes \operatorname{Cod} v)$

```

A term is a "formal arrow" if it is constructed from arrows of \(C\) in such a way that composition is applied only to formally composable pairs of terms.
primrec Arr :: 'a term \(\Rightarrow\) bool
where \(\operatorname{Arr}\langle f\rangle=C . a r r f\)
\[
\begin{aligned}
& \operatorname{Arr} \boldsymbol{\mathcal { I }}=\operatorname{True} \\
& \operatorname{Arr}(t \otimes u)=(\operatorname{Arr} t \wedge \operatorname{Arr} u) \\
& \operatorname{Arr}(t \cdot u)=(\operatorname{Arr} t \wedge \operatorname{Arr} u \wedge \operatorname{Dom} t=\operatorname{Cod} u) \\
& \operatorname{Arr} \mathbf{l}[t]=\operatorname{Arr} t \\
& \operatorname{Arr} \mathbf{l}^{-1}[t]=\operatorname{Arr} t \\
& \operatorname{Arr} \mathbf{r}[t]=\operatorname{Arr} t \\
& \operatorname{Arr} \mathbf{r}^{-1}[t]=\operatorname{Arr} t \\
& \operatorname{Arr} \mathbf{a}[t, u, v]=(\operatorname{Arr} t \wedge \operatorname{Arr} u \wedge \operatorname{Arr} v) \\
& \operatorname{Arr} \mathbf{a}^{-1}[t, u, v]=(\operatorname{Arr} t \wedge \operatorname{Arr} u \wedge \operatorname{Arr} v)
\end{aligned}
\]
abbreviation Par ：：＇a term \(\Rightarrow\)＇a term \(\Rightarrow\) bool
where Par \(t u \equiv \operatorname{Arr} t \wedge \operatorname{Arr} u \wedge \operatorname{Dom} t=\operatorname{Dom} u \wedge \operatorname{Cod} t=\operatorname{Cod} u\)
abbreviation Seq ：：＇a term \(\Rightarrow\)＇a term \(\Rightarrow\) bool
where Seq \(t u \equiv \operatorname{Arr} t \wedge \operatorname{Arr} u \wedge \operatorname{Dom} t=\operatorname{Cod} u\)
abbreviation Hom ：：＇\(a\) term \(\Rightarrow\)＇\(a\) term \(\Rightarrow\)＇a term set
where Hom a \(b \equiv\{t\) ．Arr \(t \wedge \operatorname{Dom} t=a \wedge \operatorname{Cod} t=b\}\)
A term is a＂formal identity＂if it is constructed from identity arrows of \(C\) and \(\boldsymbol{\mathcal { I }}\) using only the \(\otimes\) operator．
```

primrec Ide :: 'a term $\Rightarrow$ bool
where Ide $\langle f\rangle=$ C.ide $f$
| Ide $\boldsymbol{\mathcal { I }}=$ True
| Ide $(t \otimes u)=($ Ide $t \wedge$ Ide $u)$
| Ide $(t \cdot u)=$ False
| Ide $\mathbf{1}[t]=$ 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)
    \(\langle p r o o f\rangle\)
lemma Arr-implies-Ide-Cod:
shows Arr \(t \Longrightarrow\) Ide (Cod \(t\) )
    \(\langle p r o o f\rangle\)
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 \(\operatorname{Can}\langle f\rangle=C\).ide \(f\)
| Can \(\boldsymbol{\mathcal { I }}=\) True
\(\mid C a n(t \otimes u)=(C a n t \wedge C a n u)\)
\(\mid C a n(t \cdot u)=(C a n t \wedge C a n ~ u \wedge \operatorname{Dom} t=C o d u)\)
| Can \(\mathbf{1}[t]=\) Can \(t\)
| Can \(\mathbf{1}^{-1}[t]=\) Can \(t\)
| Can \(\mathbf{r}[t]=\) Can \(t\)
| Can \(\mathbf{r}^{-1}[t]=\) Can \(t\)
| Can \(\mathbf{a}[t, u, v]=(C a n t \wedge C a n u \wedge C a n v)\)
| Can \(\mathbf{a}^{-1}[t, u, v]=(C a n t \wedge C a n u \wedge C a n v)\)
lemma Ide-implies-Can:
shows Ide \(t \Longrightarrow C a n t\)
\(\langle p r o o f\rangle\)
lemma Can-implies-Arr:
shows \(\operatorname{Can} t \Longrightarrow A r r t\)
\(\langle p r o o f\rangle\)
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 $\langle f\rangle=\langle C . i n v f\rangle$
$\mid \operatorname{Inv} \boldsymbol{\mathcal { I }}=\boldsymbol{\mathcal { I }}$
$\mid \operatorname{Inv}(t \otimes u)=(\operatorname{Inv} t \otimes \operatorname{Inv} u)$
$\mid \operatorname{Inv}(t \cdot u)=(\operatorname{Inv} u \cdot \operatorname{Inv} t)$
| Inv $\mathbf{1}[t]=\mathbf{1}^{-1}[\operatorname{Inv} t]$
$\mid \operatorname{Inv} \mathbf{1}^{-1}[t]=\mathbf{l}[\operatorname{Inv} t]$
$\mid \operatorname{Inv} \mathbf{r}[t]=\mathbf{r}^{-1}[\operatorname{Inv} t]$
$\mid \operatorname{Inv} \mathbf{r}^{-1}[t]=\mathbf{r}[\operatorname{Inv} t]$
$\mid \operatorname{Inv} \mathbf{a}[t, u, v]=\mathbf{a}^{-1}[\operatorname{Inv} t, \operatorname{Inv} u, \operatorname{Inv} v]$
$\mid \operatorname{Inv} \mathbf{a}^{-1}[t, u, v]=\mathbf{a}[\operatorname{Inv} t, \operatorname{Inv} u, \operatorname{Inv} v]$
lemma Inv-preserves-Ide:
shows Ide $t \Longrightarrow$ Ide (Inv $t$ )
$\langle p r o o f\rangle$
lemma Inv-preserves-Can:
assumes Can t
shows Can $(\operatorname{Inv} t)$ and $\operatorname{Dom}(\operatorname{Inv} t)=\operatorname{Cod} t$ and $\operatorname{Cod}(\operatorname{Inv} t)=\operatorname{Dom} t$
〈proof〉
lemma Inv-in-Hom [simp]:
assumes Can t
shows Inv $t \in \operatorname{Hom}(\operatorname{Cod} t)(\operatorname{Dom} t)$
$\langle p r o o f\rangle$

```
```

lemma Inv-Ide [simp]:
assumes Ide a
shows Inv $a=a$
$\langle p r o o f\rangle$

```
lemma Inv-Inv [simp]:
assumes Can \(t\)
shows Inv \((\operatorname{Inv} t)=t\)
    \(\langle p r o o f\rangle\)

We call a term＂diagonal＂if it is either \(\mathcal{I}\) or it is constructed from arrows of \(C\) using only the \(\otimes\) operator associated to the right．Essentially，such terms are lists of arrows of \(C\) ，where \(\mathcal{I}\) represents the empty list and \(\otimes\) is used as the list constructor．We call them ＂diagonal＂because terms can 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 $\Rightarrow$ bool
where Diag $\mathcal{I}=$ True
$\mid \operatorname{Diag}\langle f\rangle=C . a r r f$
$\operatorname{Diag}(\langle f\rangle \otimes u)=(C \cdot \operatorname{arr} f \wedge \operatorname{Diag} u \wedge u \neq \mathcal{I})$
| Diag - = False
lemma Diag-TensorE:
assumes Diag (Tensor t u)
shows $\langle u n$-Prim $t\rangle=t$ and C.arr $(u n$-Prim $t)$ and Diag $t$ and Diag $u$ and $u \neq \mathcal{I}$
〈proof〉
lemma Diag-implies-Arr:
shows Diag $t \Longrightarrow$ Arr $t$
$\langle p r o o f\rangle$
lemma Dom-preserves-Diag:
shows Diag $t \Longrightarrow$ Diag $($ Dom $t)$
〈proof〉
lemma Cod-preserves-Diag:
shows Diag $t \Longrightarrow \operatorname{Diag}(\operatorname{Cod} t)$
$\langle$ proof $\rangle$
lemma Inv-preserves-Diag:
assumes Can $t$ and Diag $t$
shows Diag (Inv t)
$\langle p r o o f\rangle$

```

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

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

```

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 $\lfloor\otimes\rfloor 53)$
where $\mathcal{I}\lfloor\otimes\rfloor u=u$
$\mid t\lfloor\otimes\rfloor \mathcal{I}=t$
$\mid\langle f\rangle\lfloor\otimes\rfloor u=\langle f\rangle \otimes u$
$\mid(t \otimes u)\lfloor\otimes\rfloor v=t\lfloor\otimes\rfloor(u\lfloor\otimes\rfloor v)$
$\mid t\lfloor\otimes\rfloor u=$ undefined

```
lemma TensorDiag-Prim [simp]:
assumes \(t \neq \mathcal{I}\)
shows \(\langle f\rangle\lfloor\otimes\rfloor t=\langle f\rangle \otimes t\)
    \(\langle p r o o f\rangle\)
lemma TensorDiag-term-Unity [simp]:
shows \(t\lfloor\otimes\rfloor \mathcal{I}=t\)
    \(\langle\) proof \(\rangle\)
lemma TensorDiag-Diag:
assumes \(\operatorname{Diag}(t \otimes u)\)
shows \(t\lfloor\otimes\rfloor u=t \otimes u\)
    \(\langle\) proof \(\rangle\)
```

lemma TensorDiag-preserves-Diag:
assumes Diag $t$ and Diag u
shows $\operatorname{Diag}(t\lfloor\otimes\rfloor u)$
and $\operatorname{Dom}(t\lfloor\otimes\rfloor u)=\operatorname{Dom} t\lfloor\otimes\rfloor \operatorname{Dom} u$
and $\operatorname{Cod}(t\lfloor\otimes\rfloor u)=\operatorname{Cod} t\lfloor\otimes\rfloor \operatorname{Cod} u$
$\langle p r o o f\rangle$
lemma TensorDiag-in-Hom:
assumes Diag $t$ and Diag u
shows $t\lfloor\otimes\rfloor u \in \operatorname{Hom}(\operatorname{Dom} t\lfloor\otimes\rfloor \operatorname{Dom} u)(\operatorname{Cod} t\lfloor\otimes\rfloor \operatorname{Cod} u)$
$\langle$ proof $\rangle$

```
lemma Dom-TensorDiag:
```

assumes Diag t and Diag u
shows Dom ( }t\lfloor\otimes\rflooru)=\operatorname{Dom}t\lfloor\otimes\rfloor\operatorname{Dom}
<proof\rangle
lemma Cod-TensorDiag:
assumes Diag t and Diag u
shows Cod (t \lfloor\otimes\rflooru)= Cod t \lfloor\otimes\rfloorCod u
<proof>
lemma not-is-Tensor-TensorDiagE:
assumes \neg is-Tensor (t\lfloor\otimes\rflooru) and Diag t and Diag u
and }t\not=\mathcal{I}\mathrm{ and }u\not=\mathcal{I
shows False
<proof>
lemma TensorDiag-assoc:
assumes Diag t and Diag u and Diag v
shows }(t\lfloor\otimes\rflooru)\lfloor\otimes\rfloorv=t\lfloor<br>\rfloor(u\lfloor\otimes\rfloorv
<proof>
lemma TensorDiag-preserves-Ide:
assumes Ide t and Ide u and Diag t and Diag u
shows Ide ( }t<br>otimes\rflooru
\langleproof\rangle
lemma TensorDiag-preserves-Can:
assumes Can t and Can u and Diag t and Diag u
shows Can (t \lfloor\otimes\rflooru)
<proof>
lemma Inv-TensorDiag:
assumes Can t and Can u and Diag t and Diag u
shows Inv (t \lfloor\otimes\rflooru) = Inv t \lfloor\otimes\rfloorInv u
<proof>

```

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

fun CompDiag :: 'a term $\Rightarrow{ }^{\prime}$ 'a term $\Rightarrow$ ' $a$ term $\quad(\mathbf{i n f i x r}\lfloor\cdot\rfloor 55)$
where $\mathcal{I}\lfloor\cdot\rfloor u=u$
$\mid\langle f\rangle\lfloor\cdot\rfloor\langle g\rangle=\langle f \cdot g\rangle$
$\mid(u \otimes v)\lfloor\cdot\rfloor(w \otimes x)=(u\lfloor\cdot\rfloor w \otimes v\lfloor\cdot\rfloor x)$
$\mid t\lfloor\cdot\rfloor \mathcal{I}=t$
| $t\lfloor\cdot\rfloor-=$ undefined $\cdot$ undefined

```

Note that the last clause above is not relevant to diagonal terms. We have chosen a provably non-diagonal value in order to validate associativity.
lemma CompDiag-preserves-Diag:
assumes Diag \(t\) and Diag \(u\) and \(\operatorname{Dom} t=\operatorname{Cod} u\)
shows Diag ( \(t\lfloor\cdot\rfloor u)\)
```

and Dom (t \lfloor\cdot\rflooru) = Dom u
and Cod (t \lfloor\cdot\rflooru) = Cod t
<proof>
lemma CompDiag-in-Hom:
assumes Diagt and Diag u and Dom t = Cod u
shows t \lfloor•\rflooru f Hom (Dom u)(Cod t)
\langleproof\rangle
lemma Dom-CompDiag:
assumes Diagt and Diag u and Dom t = Cod u
shows Dom (t \lfloor\cdot\rflooru) = Dom u
<proof>
lemma Cod-CompDiag:
assumes Diagt and Diag u and Dom t = Cod u
shows Cod (t L\cdot\rflooru) = Cod t
\langleproof\rangle
lemma CompDiag-Cod-Diag [simp]:
assumes Diag t
shows Cod t L.\rfloort=t
<proof>
lemma CompDiag-Diag-Dom [simp]:
assumes Diag t
shows t\lfloor\cdot\rfloorDom t=t
<proof\rangle
lemma CompDiag-Ide-Diag [simp]:
assumes Diag t and Ide a and Dom a = Cod t
shows a \.\rfloort=t
\langleproof\rangle
lemma CompDiag-Diag-Ide [simp]:
assumes Diag t and Ide a and Dom t = Cod a
shows t \lfloor`\rfloora=t
\langleproof\rangle
lemma CompDiag-assoc:
assumes Diagt and Diag u and Diag v
and Dom t=Cod u and Dom u=Codv
shows (t \lfloor\cdot\rflooru) \.\rfloorv=t L.\rfloor(u\lfloor\cdot\rfloorv)
<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 \(t=\operatorname{Cod} u\)
shows Can（ \(t\lfloor\cdot\rfloor u\) ）
〈proof〉
lemma Inv－CompDiag：
assumes Can \(t\) and Can \(u\) and Diag \(t\) and Diag \(u\) and Dom \(t=C o d u\)
shows Inv \((t\lfloor\cdot\rfloor u)=\operatorname{Inv} u\lfloor\cdot\rfloor\) Inv \(t\)
\(\langle p r o o f\rangle\)
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 \(t\lfloor\cdot\rfloor\) Inv \(t=\operatorname{Cod} t\)
\(\langle p r o o f\rangle\)
lemma CompDiag－Inv－Can［simp］：
assumes Can \(t\) and Diag \(t\)
shows Inv \(t\lfloor\cdot\rfloor t=\operatorname{Dom} t\)
〈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 \(u w\)
\(\operatorname{shows}(t\lfloor\otimes\rfloor u)\lfloor\cdot\rfloor(v\lfloor\otimes\rfloor w)=(t\lfloor\cdot\rfloor v)\lfloor\otimes\rfloor(u\lfloor\cdot\rfloor w)\)
〈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 $\Rightarrow$ 'a term ( $\lfloor-\rfloor)$
where $\lfloor\langle f\rangle\rfloor=\langle f\rangle$
$\lfloor\boldsymbol{I}\rfloor=\boldsymbol{I}$
$\lfloor\lfloor t \otimes u\rfloor=\lfloor t\rfloor\lfloor\otimes\rfloor\lfloor u\rfloor$
$\lfloor\lfloor t \cdot u\rfloor=\lfloor t\rfloor\lfloor\cdot\rfloor\lfloor u\rfloor$
$\mid\lfloor 1[t]\rfloor=\lfloor t\rfloor$
$\mid\left\lfloor\mathbf{l}^{-1}[t]\right\rfloor=\lfloor t\rfloor$
$\lfloor\lfloor\mathbf{r}[t]\rfloor=\lfloor t\rfloor$
$\mid\left\lfloor\mathbf{r}^{-1}[t]\right\rfloor=\lfloor t\rfloor$
$\mid\lfloor\mathbf{a}[t, u, v]\rfloor=(\lfloor t\rfloor\lfloor\otimes\rfloor\lfloor u\rfloor)\lfloor\otimes\rfloor\lfloor v\rfloor$
$\left\lfloor\left\lfloor\mathbf{a}^{-1}[t, u, v\rfloor\right\rfloor=\lfloor t\rfloor\lfloor\otimes\rfloor(\lfloor u\rfloor\lfloor\otimes\rfloor\lfloor v\rfloor)\right.$

```
lemma Diag－Diagonalize：
assumes Arr \(t\)

```

<proof>
lemma Diagonalize-in-Hom:
assumes Arr t
shows \lfloort\rfloor\inHom \lfloorDom t\rfloor\lfloorCod t\rfloor
\langleproof\rangle
lemma Diagonalize-Dom:
assumes Arr t

```

```

    \langleproof\rangle
    lemma Diagonalize-Cod:
assumes Arr t
shows \lfloorCod t\rfloor= Cod \lfloort\rfloor
\langleproof\rangle
lemma Diagonalize-preserves-Ide:
assumes Ide a
shows Ide \lfloora\rfloor
<proof>
The diagonalizations of canonical arrows are identities.
lemma Ide-Diagonalize-Can:
assumes Can t
shows Ide \lfloort\rfloor
<proof\rangle
lemma Diagonalize-preserves-Can:
assumes Cant
shows Can \lfloort\rfloor
\langleproof\rangle
lemma Diagonalize-Diag [simp]:
assumes Diag t
shows \lfloort\rfloor=t
<proof>
lemma Diagonalize-Diagonalize [simp]:
assumes Arr t
shows \lfloor\lfloort\rfloor\rfloor= \t\rfloor
<proof>
lemma Diagonalize-Tensor:
assumes Arr t and Arr u
shows \lfloort \otimesu\rfloor=\lfloor\lfloort\rfloor\otimes\lflooru\rfloor\rfloor
\langleproof\rangle

```
```

lemma Diagonalize-Tensor-Unity-Arr [simp]:
assumes Arr u
shows $\lfloor\mathcal{I} \otimes u\rfloor=\lfloor u\rfloor$
$\langle$ proof $\rangle$
lemma Diagonalize-Tensor-Arr-Unity [simp]:
assumes Arr $t$
shows $\lfloor t \otimes \mathcal{I}\rfloor=\lfloor t\rfloor$
$\langle p r o o f\rangle$

```
```

lemma Diagonalize-Tensor-Prim-Arr [simp]:

```
lemma Diagonalize-Tensor-Prim-Arr [simp]:
assumes arr \(f\) and Arr \(u\) and \(\lfloor u\rfloor \neq\) Unity
assumes arr \(f\) and Arr \(u\) and \(\lfloor u\rfloor \neq\) Unity
shows \(\lfloor\langle f\rangle \otimes u\rfloor=\langle f\rangle \otimes\lfloor u\rfloor\)
shows \(\lfloor\langle f\rangle \otimes u\rfloor=\langle f\rangle \otimes\lfloor u\rfloor\)
    〈proof〉
    〈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 \(\lfloor\operatorname{Cod} t \cdot t\rfloor=\lfloor t\rfloor\)
〈proof〉
lemma Diagonalize-Comp-Arr-Dom:
assumes Arr \(t\)
shows \(\lfloor t \cdot \operatorname{Dom} t\rfloor=\lfloor t\rfloor\)
〈proof〉
lemma Diagonalize-Inv:
assumes Can t
shows \(\lfloor\operatorname{Inv} t\rfloor=\operatorname{Inv}\lfloor t\rfloor\)
\(\langle p r o o f\rangle\)
```

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 Dom $t \downarrow \in \operatorname{Hom}(\operatorname{Dom} t)\lfloor\operatorname{Dom} t\rfloor$ and $\operatorname{Cod} t \downarrow \in \operatorname{Hom}(\operatorname{Cod} t)\lfloor\operatorname{Cod} t\rfloor$ ，where Dom $\downarrow \downarrow$ and $C o d ~ t \downarrow$ 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 $a \downarrow \in H o m$ $a\lfloor a\rfloor$ ．The auxiliary function red2 takes a pair $(a, b)$ of diagonal identity terms and produces a canonical arrow $a \Downarrow b \in \operatorname{Hom}(a \otimes b)\lfloor a \otimes b\rfloor$ ．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 nonstructural, "operational" approach to defining a reduction from $a$ to $\lfloor a\rfloor$.

```
fun red2
(infixr \Downarrow 53)
where \mathcal{I}\Downarrow a=\mathbf{l}[a]
    | \langlef\rangle\Downarrow\mathcal{I}=\mathbf{r}[\langlef\rangle]
    | \langlef\rangle\Downarrow | = <f\rangle\otimesa
    |}(a\otimesb)\Downarrow\mathcal{I}=\mathbf{r}[a\otimesb
    | (a\otimesb)\Downarrow c=(a\Downarrow\lfloorb\otimesc\rfloor)\cdot(a\otimes(b\Downarrowc))\cdot\mathbf{a}[a,b,c]
    | a\Downarrowb= undefined
fun red (-\downarrow[56] 56)
where }\mathcal{I}\downarrow=\mathcal{I
    | \langlef\rangle\downarrow=\langlef\rangle
    | (a\otimesb)\downarrow =(if Diag (a\otimesb) then a \otimesb else (\lfloora\rfloor\Downarrow bb\rfloor) \cdot (a\downarrow\otimesb\downarrow))
    | a\downarrow= undefined
lemma red-Diag [simp]:
assumes Diag a
shows }a\downarrow=
    <proof\rangle
lemma red2-Diag:
assumes Diag(a\otimesb)
shows }a\Downarrowb=a\otimes
<proof>
lemma Can-red2:
assumes Ide a and Diag a and Ide b and Diag b
shows Can ( }a\Downarrowb
and a\Downarrowb\inHom (a\otimesb)\lfloora\otimesb\rfloor
<proof>
lemma red2-in-Hom:
```

```
assumes Ide a and Diag a and Ide b and Diag b
shows }a\Downarrowb\in\operatorname{Hom}(a\otimesb)\lfloora\otimesb
    \langleproof\rangle
lemma Can-red:
assumes Ide a
shows Can (a\downarrow) and a\downarrow\inHom a \lfloora\rfloor
<proof>
lemma red-in-Hom:
assumes Ide a
shows }a\downarrow\inHom a \a
    \langleproof\rangle
lemma Diagonalize-red [simp]:
assumes Ide a
shows \lfloora\downarrow\rfloor=\lfloora\rfloor
    <proof\rangle
lemma Diagonalize-red2 [simp]:
assumes Ide a and Ide b and Diag a and Diag b
shows \lfloora\Downarrowb\rfloor=\lfloora\otimesb\rfloor
    <proof\rangle
end
```


### 2.6 Coherence

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

```
locale evaluation-map \(=\)
    monoidal-language \(C+\)
    monoidal-category \(D T \alpha \iota+\)
    \(V\) : functor \(C D V\)
for \(C::\) ' \(c\) comp (infixr \(\cdot C\) 55)
and \(D::\) 'd comp
    (infixr • 55)
and \(T::{ }^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota::{ }^{\prime} d\)
and \(V::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
begin
```

no-notation C.in-hom
$(«-:-\rightarrow-»)$
notation unity
(I)
notation runit (r[-])
notation lunit
([]-])

```
notation assoc \({ }^{\prime}\)
\(\left(\mathrm{a}^{-1}[-,-,-]\right)\)
notation runit \({ }^{\prime}\)
notation lunit \({ }^{\prime}\)
( \(\mathrm{r}^{-1}[-]\) )
\(\left(1^{-1}[-]\right)\)
primrec eval :: 'c term \(\Rightarrow{ }^{\prime} d\)
where \(\{\langle\rho\rangle\}=V f\)
    \(\mid\{\mathcal{I}\}=\mathcal{I}\)
    \(\mid\{t \otimes u\}=\{t\} \otimes\{u\}\)
    \(\mid\{t \cdot u\}=\{t\} \cdot\{u\}\)
    | \(\{1[t]\}=\mathfrak{l}\{t\}\)
    \(\mid\left\{1^{-1}[t]\right\}=\mathfrak{r}^{\prime}\{t\}\)
    \(\mid\{\mathbf{r}[t]\}=\varrho\{t\}\)
    \(\left\{\mathbf{r}^{-1}[t]\right\}=\varrho^{\prime}\{t\}\)
    \(\{\mathbf{a}[t, u, v]\}=\alpha(\{t\},\{u\},\{v\})\)
    \(\mid\left\{\mathbf{a}^{-1}[t, u, v]\right\}=\alpha^{\prime}(\{t\},\{u\},\{v\})\)
```

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} }->{\mathrm{ Cod t}»
    <proof>
```

lemma arr-eval [simp]:
assumes Arr $f$
shows arr $\{f\}$
〈proof〉
lemma dom-eval [simp]:
assumes Arr $f$
shows $\operatorname{dom}\{f\}=\{\operatorname{Dom} f\}$
〈proof〉
lemma cod-eval [simp]:
assumes Arr $f$
shows cod $\{f\}=\{\operatorname{Cod} f\}$
$\langle p r o o f\rangle$
lemma eval-Prim [simp]:
assumes C.arr $f$
shows $\{\langle f\rangle\}=V f$
$\langle p r o o f\rangle$
lemma eval-Tensor [simp]:
assumes $\operatorname{Arr} t$ and $\operatorname{Arr} u$
shows $\{t \otimes u\}=\{t\} \otimes\{u\}$

```
<proof\rangle
```

```
lemma eval-Comp [simp]:
assumes Arr \(t\) and Arr \(u\) and Dom \(t=\operatorname{Cod} u\)
shows \(\{t \cdot u\}=\{t\} \cdot\{u\}\)
    〈proof〉
```

lemma eval-Lunit [simp]:
assumes Arr $t$
shows $\{\mathbf{l}[t]\}=1[\{\operatorname{Cod} t\}] \cdot(\mathcal{I} \otimes\{t\})$
$\langle p r o o f\rangle$
lemma eval-Lunit' [simp]:
assumes Arr $t$
shows $\left\{\mathbf{1}^{-1}[t]\right\}=\mathrm{l}^{-1}[\{\operatorname{Cod} t\}] \cdot\{t\}$
〈proof〉
lemma eval-Runit [simp]:
assumes Arr $t$
shows $\{\mathbf{r}[t]\}=\mathrm{r}[\{\operatorname{Cod} t\}] \cdot(\{t\} \otimes \mathcal{I})$
$\langle$ proof $\rangle$
lemma eval-Runit' [simp]:
assumes Arr $t$
shows $\left\{\mathbf{r}^{-1}[t]\right\}=\mathrm{r}^{-1}[\{\operatorname{Cod} t\}] \cdot\{t\}$
$\langle p r o o f\rangle$
lemma eval-Assoc [simp]:
assumes Arr $t$ and Arr $u$ and Arr $v$
shows $\{\mathbf{a}[t, u, v]\}=\mathrm{a}[\operatorname{cod}\{t\}, \operatorname{cod}\{u\}, \operatorname{cod}\{v\}] \cdot((\{t\} \otimes\{u\}) \otimes\{v\})$
〈proof〉
lemma eval-Assoc ${ }^{\prime}$ [simp]:
assumes Arr $t$ and Arr $u$ and Arr $v$
shows $\left\{\mathbf{a}^{-1}[t, u, v]\right\}=\mathrm{a}^{-1}[\operatorname{cod}\{t\}, \operatorname{cod}\{u\}, \operatorname{cod}\{v\}] \cdot(\{t\} \otimes\{u\} \otimes\{v\})$
〈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 $\{\mathbf{l}[a]\}=1[\{a\}]$
〈proof〉
lemma eval－Lunit＇－Ide［simp］：
assumes Ide a
shows $\left\{\mathbf{l}^{-1}[a]\right\}=\mathbf{l}^{-1}[\{a\}]$
〈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 \(\left\{\mathbf{r}^{-1}[a]\right\}=\mathrm{r}^{-1}[\{a\}]\)
    \(\langle p r o o f\rangle\)
```

lemma eval-Assoc-Ide [simp]:
assumes Ide $a$ and Ide $b$ and Ide $c$
shows $\{\mathbf{a}[a, b, c]\}=\mathbf{a}[\{a\},\{b\},\{c\}\}$
$\langle$ proof〉
lemma eval-Assoc'-Ide [simp]:
assumes Ide $a$ and Ide $b$ and Ide $c$
shows $\left\{\mathbf{a}^{-1}[a, b, c]\right\}=\mathrm{a}^{-1}[\{a\},\{b\},\{c\}]$
〈proof〉

Canonical arrows evaluate to isomorphisms in $D$ ，and formal inverses evaluate to inverses in $D$ ．
lemma iso-eval-Can:
shows $C a n t \Longrightarrow$ iso $\{t\}$
$\langle p r o o f\rangle$
lemma eval-Inv-Can:
shows $\operatorname{Can} t \Longrightarrow\{\operatorname{Inv} t\}=\operatorname{inv}\{t\}$
〈proof〉
The operation $\lfloor\cdot\rfloor$ evaluates to composition in $D$.
lemma eval-CompDiag:
assumes Diag $t$ and Diag $u$ and $\operatorname{Seq} t u$
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 \(\{a \Downarrow \mathcal{I}\}=\operatorname{r}[\{a\}]\)
    〈proof〉
```

Define a formal arrow t to be＂coherent＂if the square formed by $t,\lfloor t\rfloor$ and the reductions Dom $\downarrow \downarrow$ 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 $t \equiv\{\operatorname{Cod} \downarrow \downarrow\} \cdot\{t\}=\{\lfloor t\rfloor\} \cdot\{\operatorname{Dom} t \downarrow\}$
Diagonal arrows are coherent，since for such arrows $t$ the reductions Dom $\downarrow \downarrow$ and Cod $\downarrow$ 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 $\operatorname{Dom} t \downarrow$ ，diagonalization $\lfloor t\rfloor$ ，and inverse reduction $\operatorname{Inv}$（Cod $\downarrow \downarrow$ ）．This will later allow us to use the term $\operatorname{Inv}(\operatorname{Cod} t \downarrow) \cdot\lfloor t\rfloor \cdot \operatorname{Dom} t \downarrow$ as a normal form for $t$ ．
lemma canonical－factorization：
assumes Arr $t$
shows coherent $t \longleftrightarrow\{t\}=\operatorname{inv}\{\operatorname{Cod} \downarrow \downarrow\} \cdot\{\lfloor t\rfloor\} \cdot\{\operatorname{Dom} \downarrow \downarrow\}$
〈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 $\mathcal{I}$
$\langle p r o o f\rangle$
lemma coherent-Prim:
assumes Arr $\langle f\rangle$
shows coherent $\langle f\rangle$
$\langle$ proof $\rangle$
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 1-1 [a]
    <proof>
lemma coherent-Runit'-Ide:
assumes Ide a
shows coherent (\mp@subsup{\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 $a \wedge$ Diag $a$ and Ide $b \wedge$ Diag $b$ and Ide $c \wedge$ Diag $c$
shows $\{(a\lfloor\otimes\rfloor b) \Downarrow c\} \cdot(\{a \Downarrow b\} \otimes\{c\})$
$=(\{a \Downarrow(b\lfloor\otimes\rfloor c)\} \cdot(\{a\} \otimes\{b \Downarrow c\})) \cdot \mathrm{a}[\{a\},\{b\},\{c\}]$
$\langle p r o o f\rangle$
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 $\mathbf{a}^{-1}[a, b, c]$
$\langle p r o o f\rangle$
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 $\{\operatorname{Cod} t \Downarrow \operatorname{Cod} u\} \cdot(\{t\} \otimes\{u\})=\{t\lfloor\otimes\rfloor u\} \cdot\{\operatorname{Dom} t \Downarrow \operatorname{Dom} u\}$
〈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 $t=C o d u$
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 \(\{t\}=\{u\}\)
    \(\langle p r o o f\rangle\)
```

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 $\{t\}=\{u\}$
$\langle p r o o f\rangle$
end
end

## Chapter 3

## Monoidal Functor

theory MonoidalFunctor<br>imports MonoidalCategory<br>begin

A monoidal functor is a functor $F$ between monoidal categories $C$ and $D$ that preserves the monoidal structure up to isomorphism. The traditional definition assumes a monoidal functor to be equipped with two natural isomorphisms, a natural isomorphism $\varphi$ that expresses the preservation of tensor product and a natural isomorphism $\psi$ that expresses the preservation of the unit object. These natural isomorphisms are subject to coherence conditions; the condition for $\varphi$ involving the associator and the conditions for $\psi$ involving the unitors. However, as pointed out in [2] (Section 2.4), it is not necessary to take the natural isomorphism $\psi$ as given, since the mere assumption that $F \mathcal{I}_{C}$ is isomorphic to $\mathcal{I}_{D}$ is sufficient for there to be a canonical definition of $\psi$ from which the coherence conditions can be derived. This leads to a more economical definition of monoidal functor, which is the one we adopt here.

```
locale monoidal-functor \(=\)
    \(C\) : monoidal-category \(C T_{C} \alpha_{C} \iota_{C}+\)
    D: monoidal-category \(D T_{D} \alpha_{D} \iota_{D}+\)
    functor CD \(F+\)
    \(C C\) : product-category \(C C+\)
    DD: product-category \(D D+\)
    FF: product-functor C CDDFF+
    Fo \(T_{C}\) : composite-functor C.CC.comp \(C D T_{C} F+\)
    \(T_{D}\) oFF: composite-functor C.CC.comp D.CC.comp D FF.map \(T_{D}+\)
    \(\varphi\) : natural-isomorphism C.CC.comp D \(T_{D}\) oFF.map Fo \(T_{C} . m a p ~ \varphi ~\)
for \(C::\) 'c comp
                    (infixr \(\cdot{ }_{C} 55\) )
and \(T_{C}::{ }^{\prime} c *^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\alpha_{C}::{ }^{\prime} c *{ }^{\prime} c *{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\iota_{C}::{ }^{\prime} c\)
and \(D::\) 'd comp (infixr \(\cdot{ }_{D}\) 55)
and \(T_{D}:: ' d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}:: ' d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
```

```
and \(F::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
and \(\varphi::{ }^{\prime} c *^{\prime} c \Rightarrow{ }^{\prime} d+\)
assumes preserves-unity: D.isomorphic D.unity (F C.unity)
and assoc-coherence:
    【C.ide a; C.ide b; C.ide c \(\rrbracket \Longrightarrow\)
        \(F\left(\alpha_{C}(a, b, c)\right) \cdot{ }_{D} \varphi\left(T_{C}(a, b), c\right) \cdot{ }_{D} T_{D}(\varphi(a, b), F c)\)
        \(=\varphi\left(a, T_{C}(b, c)\right) \cdot{ }_{D} T_{D}(F a, \varphi(b, c)) \cdot{ }_{D} \alpha_{D}(F a, F b, F c)\)
begin
\begin{tabular}{ll} 
notation C．tensor & \(\left(\mathbf{i n f i x r} \otimes_{C} 53\right)\) \\
and C．unity & \(\left(\mathcal{I}_{C}\right)\) \\
and C．lunit & \(\left(\mathrm{l}_{C}[-]\right)\) \\
and C．runit & \(\left(\mathrm{r}_{C}[-]\right)\) \\
and C．assoc & \(\left(\mathrm{a}_{C}[-,-,-]\right)\) \\
and D．tensor & \(\left(\mathbf{i n f i x r} \otimes_{D} 53\right)\) \\
and D．unity & \(\left(\mathcal{I}_{D}\right)\) \\
and D．lunit & \(\left(\mathrm{l}_{D}[-]\right)\) \\
and D．runit & \(\left(\mathrm{r}_{D}[-]\right)\) \\
and D．assoc & \(\left(\mathrm{a}_{D}[-,-,-]\right)\)
\end{tabular}
lemma \(\varphi\)-in-hom:
assumes C.ide \(a\) and C.ide \(b\)
shows \(« \varphi(a, b): F a \otimes_{D} F b \rightarrow_{D} F\left(a \otimes_{C} b\right) »\)
    〈proof〉
```

We wish to exhibit a canonical definition of an isomorphism $\psi \in D . \operatorname{hom} \mathcal{I}_{D}\left(F \mathcal{I}_{C}\right)$ that satisfies certain coherence conditions that involve the left and right unitors．In［2］， the isomorphism $\psi$ is defined by the equation $l_{D}\left[F \mathcal{I}_{C}\right]=F l_{C}\left[\mathcal{I}_{C}\right] \cdot{ }_{D} \varphi\left(\mathcal{I}_{C}, \mathcal{I}_{C}\right) \cdot{ }_{D}$ $\left(\psi \otimes_{D} F \mathcal{I}_{C}\right)$ ，which suffices for the definition because the functor $-\otimes_{D} F \mathcal{I}_{C}$ is fully faithful．It is then asserted（Proposition 2．4．3）that the coherence condition $l_{D}[F a]=$ $F l_{C}[a] \cdot{ }_{D} \varphi\left(\mathcal{I}_{C}, a\right) \cdot{ }_{D}\left(\psi \otimes_{D} F a\right)$ is satisfied for any object $a$ of $C$ ，as well as the corresponding condition for the right unitor．However，the proof is left as an exercise （Exercise 2．4．4）．The organization of the presentation suggests that that one should derive the general coherence condition from the special case $l_{D}\left[F \mathcal{I}_{C}\right]=F l_{C}\left[\mathcal{I}_{C}\right] \cdot{ }_{D} \varphi$ $\left(\mathcal{I}_{C}, \mathcal{I}_{C}\right) \cdot D\left(\psi \otimes_{D} F \mathcal{I}_{C}\right)$ used as the definition of $\psi$ ．However，I did not see how to do it that way，so I used a different approach．The isomorphism $\iota_{D}{ }^{\prime} \equiv F \iota_{C}{ }^{\circ} D \varphi\left(\mathcal{I}_{C}\right.$ ， $\left.\mathcal{I}_{C}\right)$ serves as an alternative unit for the monoidal category $D$ ．There is consequently a unique isomorphism that maps $\iota_{D}$ to $\iota_{D}{ }^{\prime}$ ．We define $\psi$ to be this isomorphism and then use the definition to establish the desired coherence conditions．

```
abbreviation \(\iota_{1}\)
where \(\iota_{1} \equiv F \iota_{C} \cdot D \varphi\left(\mathcal{I}_{C}, \mathcal{I}_{C}\right)\)
lemma \(\iota_{1}\)-in-hom:
shows « \(\iota_{1}: F \mathcal{I}_{C} \otimes_{D} F \mathcal{I}_{C} \rightarrow_{D} F \mathcal{I}_{C}\) »
    \(\langle p r o o f\rangle\)
```

lemma $\iota_{1}$-is-iso:
shows D.iso $\iota_{1}$

```
<proof\rangle
```

interpretation $D$ ：monoidal－category－with－alternate－unit $D T_{D} \alpha_{D} \iota_{D} \iota_{1}$〈proof〉

```
no-notation \(D\). tensor (infixr \(\left.\otimes_{D} 53\right)\)
notation D. \(C_{1}\).tensor (infixr \(\otimes_{D} 53\) )
no-notation D.assoc \(\quad\left(\mathrm{a}_{D}[-,-,-]\right)\)
notation D.C \(C_{1}\).assoc \(\quad\left(\mathrm{a}_{D}[-,-,-]\right)\)
no-notation D.assoc \({ }^{\prime} \quad\left(\mathrm{a}^{-1}[-,-,-]\right)\)
notation D.C \(C_{1} \cdot\) assoc \(^{\prime} \quad\left(\mathrm{a}_{D}{ }^{-1}[-,-,-]\right)\)
notation \(D . C_{1}\).unity \(\quad\left(\mathcal{I}_{1}\right)\)
notation D. \(C_{1}\).lunit \(\quad\left(l_{1}[-]\right)\)
notation D.C. . runit \(\quad\left(\mathrm{r}_{1}[-]\right)\)
lemma \(\mathcal{I}_{1}\)-char \([\) simp \(]\) :
shows \(\mathcal{I}_{1}=F \mathcal{I}_{C}\)
    〈proof〉
```

definition $\psi$
where $\psi \equiv$ THE $\psi . 《 \psi: \mathcal{I}_{D} \rightarrow_{D} F \mathcal{I}_{C} » \wedge$ D.iso $\psi \wedge \psi \cdot{ }_{D} \iota_{D}=\iota_{1} \cdot{ }_{D}\left(\psi \otimes_{D} \psi\right)$
lemma $\psi$-char:
shows $« \psi: \mathcal{I}_{D} \rightarrow_{D} F \mathcal{I}_{C}$ and D.iso $\psi$ and $\psi \cdot_{D} \iota_{D}=\iota_{1} \cdot_{D}\left(\psi \otimes_{D} \psi\right)$
and $\exists!\psi . 《 \psi: \mathcal{I}_{D} \rightarrow_{D} F \mathcal{I}_{C} » \wedge D$.iso $\psi \wedge \psi \cdot{ }_{D} \iota_{D}=\iota_{1} \cdot D\left(\psi \otimes_{D} \psi\right)$
〈proof〉
lemma $\psi$-eqI:
assumes $« f: \mathcal{I}_{D} \rightarrow_{D} F \mathcal{I}_{C}$ » and D.iso $f$ and $f \cdot_{D} \iota_{D}=\iota_{1} \cdot{ }_{D}\left(f \otimes_{D} f\right)$
shows $f=\psi$
$\langle p r o o f\rangle$
lemma lunit-coherence1:
assumes C.ide a
shows $1_{1}\left[\begin{array}{ll}F & a\end{array}\right] \cdot{ }_{D}\left(\psi \otimes_{D} F a\right)=l_{D}\left[\begin{array}{ll}F & a\end{array}\right]$
〈proof〉
lemma lunit－coherence2：
assumes C．ide a
shows $F \mathrm{l}_{C}[a] \cdot{ }_{D} \varphi\left(\mathcal{I}_{C}, a\right)=\mathrm{l}_{1}[F a]$
〈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 $\mathrm{l}_{D}[F a]=F \mathrm{l}_{C}[a] \cdot{ }_{D} \varphi\left(\mathcal{I}_{C}, a\right) \cdot{ }_{D}\left(\psi \otimes_{D} F a\right)$
〈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 $\lambda f . \varphi(s n d f, f s t f)$ 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 \(C^{\prime}\) : opposite-monoidal-category \(C T_{C} \alpha_{C} \iota_{C}\langle p r o o f\rangle\)
    interpretation \(D^{\prime}\) : opposite-monoidal-category \(D T_{D} \alpha_{D} \iota_{D}\langle p r o o f\rangle\)
    interpretation \(T_{D}{ }^{\prime} o F F\) : composite-functor C.CC.comp D.CC.comp D FF.map \(D^{\prime}\). \(T\langle\) proof \(\rangle\)
    interpretation \(F o T_{C}{ }^{\prime}\) : composite-functor C.CC.comp \(C D C^{\prime} . T\) F \(\langle\) proof \(\rangle\)
    interpretation \(\varphi^{\prime}\) : natural-transformation C.CC.comp \(D T_{D}{ }^{\prime}\) oFF.map Fo \(T_{C}{ }^{\prime}\).map
                                    \(\langle\lambda f . \varphi(s n d f, f s t f)\rangle\)
        \(\langle p r o o f\rangle\)
    interpretation \(\varphi^{\prime}\) : natural-isomorphism C.CC.comp \(D T_{D}{ }^{\prime}\) oFF.map Fo \(T_{C}{ }^{\prime}\).map
                        \(\langle\lambda f . \varphi(\) snd \(f, f s t f)\rangle\)
        〈proof〉
    interpretation \(F^{\prime}:\) monoidal-functor \(C C^{\prime} . T C^{\prime} . \alpha \iota_{C} D D^{\prime} . T D^{\prime} . \alpha \iota_{D} F<\lambda f . \varphi\) (snd \(f\), fst
f) >
    \(\langle p r o o f\rangle\)
```

    lemma induces-monoidal-functor-between-opposites:
    shows monoidal-functor \(C C^{\prime} . T C^{\prime} . \alpha \iota_{C} D D^{\prime} . T D^{\prime} . \alpha \iota_{D} F(\lambda f . \varphi(\) snd \(f, f s t f))\)
        \(\langle p r o o f\rangle\)
    lemma runit-coherence:
    assumes C.ide a
    shows \(\mathrm{r}_{D}[F a]=F \mathrm{r}_{C}[a] \cdot{ }_{D} \varphi\left(a, \mathcal{I}_{C}\right) \cdot{ }_{D}\left(F a \otimes_{D} \psi\right)\)
    \(\langle p r o o f\rangle\)
    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 \(T_{C}::{ }^{\prime} c *{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\alpha_{C}::{ }^{\prime} c *{ }^{\prime} c *^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\iota_{C}::{ }^{\prime} c\)
and \(D::\) 'd comp (infixr \(\cdot{ }_{D} 55\) )
and \(T_{D}::{ }^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
and \(F::{ }^{\prime} c \Rightarrow{ }^{\prime} d+\)
```

```
assumes strictly-preserves-ı: \(F \iota_{C}=\iota_{D}\)
and strictly-preserves- \(T: \llbracket C\).arr \(f ; C\).arr \(g \rrbracket \Longrightarrow F\left(T_{C}(f, g)\right)=T_{D}(F f, F g)\)
and strictly-preserves- \(\alpha\)-ide: \(\llbracket C\).ide \(a ;\) C.ide \(b ; C . i d e ~ c \rrbracket \Longrightarrow\)
                    \(F\left(\alpha_{C}(a, b, c)\right)=\alpha_{D}(F a, F b, F c)\)
begin
notation C．tensor
and C．unity
and C．lunit
and C．runit
and C．assoc
and D．tensor
and D．unity
and D．lunit
and D．runit
and D．assoc
```

（infixr $\otimes_{C} 53$ ）
$\left(\mathcal{I}_{C}\right)$
$\left(l_{C}[-]\right)$
$\left(\mathrm{r}_{C}[-]\right)$
$\left(\mathrm{a}_{C}[-,-,-]\right)$
（infixr $\otimes_{D} 53$ ）
$\left(\mathcal{I}_{D}\right)$
$\left(\mathrm{l}_{D}[-]\right)$
$\left(\mathrm{r}_{D}[-]\right)$
$\left(\mathrm{a}_{D}[-,-,-]\right)$

```
lemma strictly－preserves－tensor：
assumes \(C\) ．arr \(f\) and C．arr \(g\)
shows \(F\left(f \otimes_{C} g\right)=F f \otimes_{D} F g\) \(\langle p r o o f\rangle\)
lemma strictly－preserves－\(\alpha\) ：
assumes \(C . \operatorname{arr} f\) and \(C . \operatorname{arr} g\) and C．arr \(h\)
shows \(F\left(\alpha_{C}(f, g, h)\right)=\alpha_{D}(F f, F g, F h)\)
〈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 \(F \mathrm{l}_{C}[a]=\mathrm{l}_{D}\left[\begin{array}{ll}F & a\end{array}\right]\)
〈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 $\varphi$ ．For a strict monoidal functor，this is an identity transformation．

```
interpretation FF: product-functor C C D D FF \(\langle\) proof \(\rangle\)
interpretation \(F_{o} T_{C}\) : composite-functor C.CC.comp \(C D T_{C} F\langle p r o o f\rangle\)
interpretation \(T_{D} o F F\) : composite-functor C.CC.comp D.CC.comp D FF.map \(T_{D}\langle p r o o f\rangle\)
lemma structure-is-trivial:
shows \(T_{D} o F F\).map \(=\) Fo \(T_{C}\). map
〈proof〉
abbreviation \(\varphi\) where \(\varphi \equiv T_{D}\) oFF.map
lemma structure-is-natural-isomorphism:
shows natural-isomorphism C.CC.comp D \(T_{D}\) oFF.map Fo \(T_{C}\). map \(\varphi\)
    \(\langle\) proof〉
end
A strict monoidal functor is a monoidal functor．
sublocale strict－monoidal－functor \(\subseteq\) monoidal－functor \(C T_{C} \alpha_{C} \iota_{C} D T_{D} \alpha_{D} \iota_{D} F \varphi\)〈proof〉
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 \(B T_{B} \alpha_{B} \iota_{B} D T_{D} \alpha_{D} \iota_{D}(G \circ F)\)
〈proof〉
```

An equivalence of monoidal categories is a monoidal functor whose underlying ordi－ nary 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 CDFG \(\eta \varepsilon+\)
    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 \(T_{C}::{ }^{\prime} c *{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\alpha_{C}::{ }^{\prime} c *^{\prime} c *^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\iota_{C}::{ }^{\prime} c\)
and \(D::\) 'd comp (infixr \(\left.\cdot{ }_{D} 55\right)\)
and \(T_{D}::{ }^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
and \(F::{ }^{\prime} d \Rightarrow{ }^{\prime} c\)
and \(\varphi::{ }^{\prime} d *{ }^{\prime} d \Rightarrow{ }^{\prime} c\)
and \(\iota::{ }^{\prime} c\)
and \(G::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
and \(\eta::{ }^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\varepsilon::{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
end
```


## Chapter 4

# The Free Monoidal Category 

theory FreeMonoidalCategory<br>imports Category3.Subcategory MonoidalFunctor<br>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 («- : - ->
```

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 \(t u \equiv \operatorname{Par} t u \wedge\lfloor t\rfloor=\lfloor u\rfloor\)
```

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 \(A R R\) where \(A R R f \equiv f \neq\{ \} \wedge(\forall t . t \in f \longrightarrow f=\) Collect (equiv \(t))\)
lemma not-ARR-empty:
shows \(\neg A R R\}\)
    \(\langle p r o o f\rangle\)
```

lemma $A R R$-eqI:
assumes $A R R f$ and $A R R g$ and $f \cap g \neq\{ \}$
shows $f=g$
$\langle p r o o f\rangle$

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

```
definition norm ( \(\mid\) - \(-\|\) )
where \(\|t\|=\operatorname{Inv}(\operatorname{Cod} t \downarrow) \cdot\lfloor t\rfloor \cdot \operatorname{Dom} t \downarrow\)
```

If $t$ is a formal arrow, then $t$ is equivalent to its normal form.
lemma equiv-norm-Arr:
assumes Arr $t$
shows equiv $\|t\| t$
$\langle p r o o f\rangle$
Equivalent arrows have identical normal forms.

```
lemma norm-respects-equiv:
assumes equiv \(t u\)
shows \(\|t\|=\|u\|\)
    \(\langle p r o o f\rangle\)
```

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|| <de \lfloort\rfloor
    <proof\rangle
```

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 Cant
shows Can |t|
    <proof>
lemma Par-Arr-norm:
assumes Arr t
shows Par |t|t
    <proof\rangle
lemma Diagonalize-norm [simp]:
assumes Arr t
shows \lfloor|tt|\rfloor= \t\rfloor
    <proof>
lemma unique-norm:
assumes ARR f
shows \exists!t.\forallu.u\inf\longrightarrow|u|=t
<proof>
lemma Dom-norm:
assumes Arr t
shows Dom|t| = Dom t
    <proof\rangle
lemma Cod-norm:
assumes Arr t
shows Cod |t| = Cod t
    <proof\rangle
lemma norm-in-Hom:
assumes Arr t
shows |t|| Hom(Dom t)(Cod t)
    <proof\rangle
```

As all the elements of an equivalence class have the same normal form, we can use the normal form of an arbitrarily chosen element as a canonical representative.

```
definition rep where rep f}\equiv|SOME t.t\inf
lemma rep-in-ARR:
assumes ARRf
shows rep f\inf
    <proof>
lemma Arr-rep-ARR:
assumes ARRf
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 \(\neg\) Arr \(t\)
shows mkarr \(t=\{ \}\)
    〈proof〉
lemma \(A R R\)-mkarr:
assumes \(A r r t\)
shows \(A R R\) (mkarr \(t\) )
    \(\langle\) proof \(\rangle\)
lemma mkarr-memb-ARR:
assumes \(A R R f\) and \(t \in f\)
shows mkarr \(t=f\)
    \(\langle p r o o f\rangle\)
lemma mkarr-rep-ARR [simp]:
assumes \(A R R f\)
shows mkarr (rep f) \(=f\)
    \(\langle p r o o f\rangle\)
lemma Arr-in-mkarr:
assumes Arr \(t\)
shows \(t \in\) mkarr \(t\)
    \(\langle p r o o f\rangle\)
```

Two terms are related by equiv iff they are both formal arrows and have identical normal forms．
lemma equiv－iff－eq－norm：
shows equiv $t u \longleftrightarrow \operatorname{Arr} t \wedge \operatorname{Arr} u \wedge\|t\|=\|u\|$
〈proof〉
lemma norm-norm [simp]:
assumes Arr $t$
shows $\|\|t\|\|\|=\| t \|$
$\langle p r o o f\rangle$
lemma norm-in-ARR:
assumes $A R R f$ and $t \in f$
shows $\|t\| \in f$
〈proof〉
lemma norm-rep-ARR [simp]:
assumes $A R R f$

```
shows |rep f| = rep f
    <proof>
```

lemma norm-memb-eq-rep-ARR:
assumes $A R R f$ and $t \in f$
shows norm $t=$ rep $f$
$\langle p r o o f\rangle$
lemma rep-mkarr:
assumes $\operatorname{Arr} f$
shows rep $(\operatorname{mkarr} f)=\|f\|$
$\langle p r o o f\rangle$

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 Parfg 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 \(D O M\) where \(D O M f \equiv \operatorname{Dom}(r e p f)\)
```

abbreviation $C O D$ where $C O D f \equiv \operatorname{Cod}(r e p f)$
lemma DOM-mkarr:
assumes Arr $t$
shows DOM (mkarr $t)=$ Dom $t$
〈proof〉

```
lemma COD-mkarr:
assumes Arr \(t\)
shows \(C O D(\) mkarr \(t)=\operatorname{Cod} t\)
    \(\langle p r o o f\rangle\)
```

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

```
definition comp (infixr • 55)
    where comp \(f g \equiv(\) if \(A R R f \wedge A R R g \wedge D O M f=C O D g\)
                            then mkarr \(((\operatorname{rep} f) \cdot(\) 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 ( $«-$ : - $\rightarrow->$ )

The empty set serves as the null for the composition．

```
lemma null-char:
shows null \(=\{ \}\)
〈proof〉
lemma \(A R R\)-comp:
assumes \(A R R f\) and \(A R R g\) and \(D O M f=C O D g\)
shows \(A R R(f \cdot g)\)
    〈proof〉
lemma DOM-comp [simp]:
assumes \(A R R f\) and \(A R R g\) and \(D O M f=C O D g\)
shows \(D O M(f \cdot g)=D O M g\)
    〈proof〉
lemma COD-comp [simp]:
assumes \(A R R f\) and \(A R R g\) and \(D O M f=C O D g\)
shows \(C O D(f \cdot g)=C O D f\)
    \(\langle p r o o f\rangle\)
lemma comp-assoc:
assumes \(g \cdot f \neq\) null and \(h \cdot g \neq\) null
shows \(h \cdot(g \cdot f)=(h \cdot g) \cdot f\)
\(\langle p r o o f\rangle\)
lemma Comp-in-comp-ARR:
assumes \(A R R f\) and \(A R R g\) and \(D O M f=C O D g\)
and \(t \in f\) and \(u \in g\)
shows \(t \cdot u \in f \cdot g\)
〈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\equivARR f^(\forallt.t\inf\longrightarrowIde \lfloort\rfloor)\wedgeDOMf=CODf
lemma IDE-implies-ARR:
assumes IDE f
shows ARR f
    \langleproof\rangle
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 ARRf\longleftrightarrow domains f}\not={
<proof>
lemma ARR-iff-has-codomain:
shows ARR f}\longleftrightarrow\mathrm{ codomains }f\not={
<proof>
lemma arr-iff-ARR:
shows arr f \longleftrightarrowARRf
    <proof\rangle
```

The arrows of the category are the equivalence classes of formal arrows．
lemma arr－char：
shows $\operatorname{arr} f \longleftrightarrow f \neq\{ \} \wedge(\forall t . t \in f \longrightarrow f=$ mkarr $t)$
$\langle p r o o f\rangle$
lemma seq-char:
shows seq $g f \longleftrightarrow g \cdot f \neq$ null
〈proof〉
lemma seq－char＇：
shows seq $g f \longleftrightarrow A R R f \wedge A R R g \wedge D O M g=C O D f$
〈proof〉
Finally，we can show that the composition comp determines a category．
interpretation category comp
〈proof〉
lemma mkarr－rep［simp］：
assumes $\operatorname{arr} f$
shows mkarr（rep f）$=f$
$\langle p r o o f\rangle$
lemma arr－mkarr［simp］：
assumes Arr $t$
shows arr（mkarr t）
$\langle p r o o f\rangle$
lemma mkarr－memb：
assumes $t \in f$ and $\operatorname{arr} f$
shows Arr $t$ and mkarr $t=f$
$\langle p r o o f\rangle$
lemma rep－in－arr［simp］：
assumes arr $f$
shows rep $f \in f$
$\langle p r o o f\rangle$

```
lemma Arr-rep [simp]:
assumes arr f
shows Arr (repf)
    <proof\rangle
lemma rep-in-Hom:
assumes arr f
shows rep f\inHom (DOMf)(CODf)
    \langleproof\rangle
lemma norm-memb-eq-rep:
assumes arr f and t\inf
shows |t|= rep f
    \langleproof\rangle
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 $\operatorname{Dom} t=\operatorname{Cod} u$
shows mkarr $t \cdot m k a r r \quad u=m k a r r(t \cdot u)$
〈proof〉
lemma dom－char：
shows $\operatorname{dom} f=($ if arr $f$ then mkarr $(D O M f)$ else null $)$
$\langle p r o o f\rangle$
lemma dom－simp：
assumes arr $f$
shows $\operatorname{dom} f=m k a r r(D O M f)$
$\langle p r o o f\rangle$
lemma cod－char：
shows $\operatorname{cod} f=($ if arr $f$ then mkarr $(C O D f)$ else null $)$
〈proof〉
lemma cod－simp：
assumes arr $f$
shows $\operatorname{cod} f=\operatorname{mkarr}(C O D f)$
$\langle p r o o f\rangle$
lemma Dom－memb：
assumes $\operatorname{arr} f$ and $t \in f$
shows $\operatorname{Dom} t=D O M f$

```
    \langleproof\rangle
lemma Cod-memb:
assumes arr f and t\inf
shows Cod t=CODf
    \langleproof\rangle
lemma dom-mkarr [simp]:
assumes Arr t
shows dom (mkarr t)=mkarr (Dom t)
    \langleproof\rangle
lemma cod-mkarr [simp]:
assumes Arr t
shows cod (mkarr t)=mkarr (Cod t)
    \langleproof\rangle
lemma mkarr-in-hom:
assumes Arr t
shows «mkarr t : mkarr (Dom t) -> mkarr (Cod t)»
    \langleproof\rangle
lemma DOM-in-dom [intro]:
assumes arr f
shows DOMf\in\operatorname{dom}f
    \langleproof\rangle
lemma COD-in-cod [intro]:
assumes arr f
shows COD f\in\operatorname{cod}f
    \langleproof\rangle
lemma DOM-dom:
assumes arr f
shows DOM (domf)=DOMf
    \langleproof\rangle
lemma DOM-cod:
assumes arr f
shows DOM ( cod f)=CODf
    <proof\rangle
lemma memb-equiv:
assumes arr f and t\inf and u\inf
shows Par t u and \lfloort\rfloor=\lflooru\rfloor
<proof\rangle
```

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 $f \cdot g=($ if seq $f g$ then mkarr（rep $f \cdot$ 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 mkarr $t=$ mkarr $u$
shows $t=u$
〈proof〉

```
lemma Comp-in-comp [intro]:
assumes \(\operatorname{arr} f\) and \(g \in \operatorname{hom}(\operatorname{domg})(\operatorname{dom} f)\) and \(t \in f\) and \(u \in g\)
shows \(t \cdot u \in f \cdot g\)
〈proof〉
```

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

```
definition can
where \(\operatorname{can} f \equiv \operatorname{arr} f \wedge(\exists t . t \in f \wedge I d e\lfloor t\rfloor)\)
lemma can-def-alt:
shows \(\operatorname{can} f \longleftrightarrow \operatorname{arr} f \wedge(\forall t . t \in f \longrightarrow\) 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 ide $f \longleftrightarrow \operatorname{can} f \wedge \operatorname{dom} f=\operatorname{cod} f$
〈proof〉
lemma ide－iff－IDE：
shows ide $a \longleftrightarrow I D E$ a
$\langle p r o o f\rangle$
lemma ide－mkarr－Ide：
assumes Ide a
shows ide（mkarr a）
$\langle p r o o f\rangle$
lemma rep－dom：

```
assumes arr f
shows rep (dom f) =|DOM f|
    <proof>
lemma rep-cod:
assumes arr f
shows rep (\operatorname{cod}f)=|CODf|
    <proof>
lemma rep-preserves-seq:
assumes seq gf
shows Seq (rep g) (rep f)
    <proof>
lemma rep-comp:
assumes seq g f
shows rep (g\cdotf)=|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）
\(\langle p r o o f\rangle\)
lemma ide－implies－can：
assumes ide a
shows can a
\(\langle p r o o f\rangle\)
lemma Can－rep－can：
assumes \(\operatorname{can} f\)
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 \(f(\operatorname{mkarr}(\operatorname{Inv}(D O M f \downarrow) \cdot\lfloor r e p f\rfloor \cdot C O D f \downarrow))\)
〈proof〉
lemma inv－mkarr［simp］：
assumes Can \(t\)
```

shows inv（mkarr $t)=\operatorname{mkarr}(\operatorname{Inv} t)$
〈proof〉
lemma iso－can：
assumes $\operatorname{can} f$
shows iso $f$
〈proof〉
The following function produces the unique canonical arrow between two given ob－ jects，if such an arrow exists．

```
definition mkcan
where mkcan a b = mkarr (Inv (COD b\downarrow) •(DOM a\downarrow))
lemma can-mkcan:
assumes ide a and ide b and \lfloorDOM a\rfloor=\lfloorCOD b\rfloor
shows can (mkcan a b) and «mkcan a b:a ->b»
<proof>
lemma dom-mkcan:
assumes ide a and ide b and \lfloorDOMa\rfloor=\lfloorCOD b \
shows dom (mkcan a b) =a
    \langleproof\rangle
lemma cod-mkcan:
assumes ide a and ide b and \lfloorDOMa\rfloor=\lfloorCOD b\rfloor
shows cod (mkcan a b)=b
    <proof\rangle
lemma can-coherence':
assumes can f
shows mkcan (domf) (\operatorname{cod}f)=f
<proof>
lemma Ide-Diagonalize-rep-ide:
assumes ide a
shows Ide \lfloorrep a \rfloor
    <proof\rangle
lemma Diagonalize-DOM:
assumes arr f
shows \lfloorDOM f\rfloor= Dom \lfloorrep f\rfloor
    \langleproof\rangle
lemma Diagonalize-COD:
assumes arr f
shows \lfloorCOD f\rfloor=Cod \lfloorrep f\rfloor
    <proof\rangle
```

lemma Diagonalize-rep-preserves-seq:

```
assumes seq gf
shows Seq \lfloorrep g\rfloor \rep f\rfloor
    <proof>
lemma Dom-Diagonalize-rep:
assumes arr f
shows Dom \lfloorrep f\rfloor=\lfloorrep(domf)\rfloor
    \langleproof\rangle
lemma Cod-Diagonalize-rep:
assumes arrf
shows Cod \lfloorrep f f = \lfloorrep (codf) \rfloor
    <proof>
lemma mkarr-Diagonalize-rep:
assumes arr f and Diag (DOM f) and Diag (COD f)
shows mkarr \lfloorrep f\rfloor=f
<proof>
We define tensor product of arrows via the constructor \((\otimes)\) on terms.
definition tensor}\mp@subsup{F}{FMC}{}\quad(infixr \otimes53
    where f\otimesg\equiv(if arr f ^ arr g then mkarr (rep f\otimes rep g) else null)
lemma arr-tensor [simp]:
assumes arr f and arr g
shows arr (f\otimesg)
    \langleproof\rangle
lemma rep-tensor:
assumes arr f and arr g
shows rep (f\otimesg)=|rep f\otimes rep g|
    <proof>
lemma Par-memb-rep:
assumes arr f and t\inf
shows Part (rep f)
    \langleproof\rangle
lemma Tensor-in-tensor [intro]:
assumes arr f and arr g and t\inf and u\ing
shows t\otimesu\inf\otimesg
\langleproof\rangle
lemma DOM-tensor [simp]:
assumes arr f and arr g
shows DOM (f\otimesg)=DOMf\otimesDOMg
    \langleproof\rangle
lemma COD-tensor [simp]:
```

```
assumes arr f and arr g
shows COD (f\otimesg)=COD f\otimesCODg
    \langleproof\rangle
lemma tensor-in-hom [simp]:
assumes «f:a->b» and «g :c->d»
shows《f}\otimesg:a\otimesc->b\otimesd
\langleproof\rangle
lemma dom-tensor [simp]:
assumes arr f and arr g
shows dom (f\otimesg)=\operatorname{dom}f\otimes\operatorname{dom}g
    \langleproof\rangle
lemma cod-tensor [simp]:
assumes arr f and arr g
shows cod (f\otimesg)=\operatorname{cod}f\otimes\operatorname{cod}g
    \langleproof\rangle
lemma tensor-mkarr [simp]:
assumes Arr t and Arr u
shows mkarr t m mkarr u=mkarr (t\otimesu)
    \langleproof\rangle
lemma tensor-preserves-ide:
assumes ide a and ide b
shows ide (a\otimesb)
\langleproof\rangle
lemma tensor-preserves-can:
assumes can f and can g
shows can (f\otimesg)
    <proof\rangle
lemma comp-preserves-can:
assumes can f and can g and dom f=\operatorname{cod}g
shows can (f | g)
<proof>
The remaining structure required of a monoidal category is also defined syntactically.
```

```
definition unity \(y_{F M C}::\) 'c arr
```

definition unity $y_{F M C}::$ 'c arr
where $\mathcal{I}=m k a r r \mathcal{I}$
where $\mathcal{I}=m k a r r \mathcal{I}$
definition lunit $_{F M C}::$ 'c arr $\Rightarrow{ }^{\prime}$ c arr
definition lunit $_{F M C}::$ 'c arr $\Rightarrow{ }^{\prime}$ c arr
where $\mathrm{I}[a]=$ mkarr $\mathrm{l}[$ rep $a]$
where $\mathrm{I}[a]=$ mkarr $\mathrm{l}[$ rep $a]$
definition runit $_{F M C}::$ 'c arr $\Rightarrow{ }^{\prime} c$ arr
definition runit $_{F M C}::$ 'c arr $\Rightarrow{ }^{\prime} c$ arr
(r[-])
(r[-])
where $\mathrm{r}[a]=$ mkarr $\mathbf{r}[$ rep $a]$

```
where \(\mathrm{r}[a]=\) mkarr \(\mathbf{r}[\) rep \(a]\)
```



```
where a[a,b,c]=mkarr a[rep a, rep b, rep c]
lemma can-lunit:
assumes ide a
shows can 1[a]
    <proof\rangle
lemma lunit-in-hom:
assumes ide a
shows «1[a]:\mathcal{I}\otimesa->a»
<proof>
lemma arr-lunit [simp]:
assumes ide a
shows arr l[a]
    <proof\rangle
lemma dom-lunit [simp]:
assumes ide a
shows dom l[a]=\mathcal{I}\otimesa
    <proof\rangle
lemma cod-lunit [simp]:
assumes ide a
shows cod l[a]=a
    \langleproof\rangle
lemma can-runit:
assumes ide a
shows can r[a]
    \langleproof\rangle
lemma runit-in-hom [simp]:
assumes ide a
shows «r[a]:a\otimes\mathcal{I}->a»
\langleproof\rangle
lemma arr-runit [simp]:
assumes ide a
shows arr r [a]
    \langleproof\rangle
lemma dom-runit [simp]:
assumes ide a
shows dom r }[a]=a\otimes\mathcal{I
    \langleproof\rangle
lemma cod-runit [simp]:
```

```
assumes ide a
shows cod r [a]=a
    <proof\rangle
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《\textrm{a}[a,b,c]:(a\otimesb)\otimesc->a\otimesb\otimesc»
<proof>
lemma arr-assoc [simp]:
assumes ide a and ide b and ide c
shows arr a [a,b,c]
    <proof\rangle
lemma dom-assoc [simp]:
assumes ide a and ide b and ide c
shows dom a[a,b,c] = (a\otimesb)\otimesc
    <proof\rangle
lemma cod-assoc [simp]:
assumes ide a and ide b and ide c
shows cod a [a,b,c] =a\otimesb\otimesc
    <proof\rangle
lemma ide-unity [simp]:
shows ide I
    \langleproof\rangle
lemma Unity-in-unity [simp]:
shows \mathcal{I}\in\mathcal{I}
    \langleproof\rangle
lemma rep-unity [simp]:
shows rep \mathcal{I}=|\mathcal{I}|
    <proof\rangle
lemma Lunit-in-lunit [intro]:
assumes arr f and t\inf
shows l[t] \in l[f]
<proof>
lemma Runit-in-runit [intro]:
assumes arr f and t\inf
shows r [t] \in r [f]
```

```
<proof>
lemma Assoc-in-assoc [intro]:
assumes arr f and arr g and arr h
and t\inf and u\ing and v\inh
shows a [t,u,v] \in a[f,g,h]
<proof\rangle
```

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}\mp@subsup{F}{FMC}{
<proof>
```

lemma is-elementary-monoidal-category:
shows elementary-monoidal-category
comp tensor $F_{F M C}$ unity $_{F M C}$ lunit $_{F M C}$ runit $_{F M C}$ assoc $_{F M C}$
$\langle$ proof $\rangle$
abbreviation $T_{F M C}$ where $T_{F M C} \equiv E M C . T$
abbreviation $\alpha_{F M C}$ where $\alpha_{F M C} \equiv E M C . \alpha$
abbreviation $\iota_{F M C}$ where $\iota_{F M C} \equiv E M C . \iota$
interpretation $M C$ : monoidal-category comp $T_{F M C} \alpha_{F M C} \iota_{F M C}$
$\langle p r o o f\rangle$
lemma induces-monoidal-category:
shows monoidal-category comp $T_{F M C} \alpha_{F M C} \iota_{F M C}$
$\langle p r o o f\rangle$
end
sublocale free-monoidal-category $\subseteq$
elementary-monoidal-category
comp tensor $F M C$ unity ${ }_{F M C}$ lunit $_{F M C}$ runit $_{F M C}$ assoc $_{F M C}$
$\langle p r o o f\rangle$
sublocale free-monoidal-category $\subseteq$ monoidal-category $\operatorname{comp} T_{F M C} \alpha_{F M C} \iota_{F M C}$
〈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]=|\mathbf{l}[\mathrm{ rep a}]|
    <proof>
lemma rep-runit:
assumes ide a
shows rep r [a] =|r[rep a]|
    \langleproof\rangle
lemma rep-assoc:
assumes ide a and ide b and ide c
shows rep a [a,b,c]=|\mathbf{a}[rep a, rep b, rep c]|
    \langleproof\rangle
lemma mkarr-Unity:
shows mkarr \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 \(\mathbf{l}[t]=\mathfrak{l}(\) mkarr \(t)\)
〈proof〉
lemma mkarr－Lunit＇：
assumes Arr \(t\)
shows mkarr \(\mathbf{l}^{-1}[t]=\mathfrak{l}^{\prime}(\) mkarr \(t)\)
〈proof〉
lemma mkarr－Runit：
assumes Arr \(t\)
shows mkarr \(\mathbf{r}[t]=\varrho(\) mkarr \(t)\)
〈proof〉
lemma mkarr－Runit＇：
assumes Arr \(t\)
shows mkarr \(\mathbf{r}^{-1}[t]=\varrho^{\prime}(\) mkarr \(t)\)
〈proof〉
lemma mkarr－Assoc：
assumes Arr \(t\) and Arr \(u\) and Arr \(v\)
shows mkarr \(\mathbf{a}[t, u, v]=\alpha(\) mkarr \(t\), mkarr \(u\), mkarr \(v)\)
〈proof〉
lemma mkarr－Assoc＇：
assumes Arr \(t\) and Arr \(u\) and Arr \(v\)
shows mkarr \(\mathbf{a}^{-1}[t, u, v]=\alpha^{\prime}(\) 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 \(\equiv \lambda f\). if C.arr f then mkarr \(\langle f\rangle\) else null
    lemma inclusion-is-functor:
    shows functor C comp inclusion-of-generators
    \(\langle p r o o f\rangle\)
    end

We now show that, given a functor $V$ from $C$ to a monoidal category $D$, the evaluation map that takes formal arrows of the monoidal language of $C$ to arrows of $D$ induces a strict monoidal functor from $\mathcal{F} C$ to $D$.

```
locale evaluation-functor \(=\)
    \(C\) : category \(C+\)
    D: monoidal-category \(D T_{D} \alpha_{D} \iota_{D}+\)
    evaluation-map \(C D T_{D} \alpha_{D} \iota_{D} V+\)
    \(\mathcal{F} C\) : free-monoidal-category \(C\)
for \(C::\) 'c comp (infixr \({ }^{\circ} C\) 55)
and \(D::\) 'd comp (infixr \(\left.\cdot{ }_{D} 55\right)\)
and \(T_{D}::{ }^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}:: ' d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
and \(V::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
begin
notation eval
definition map
where map \(f \equiv\) 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 $\{\|t\|\}=\{t\}$
$\langle p r o o f\rangle$
interpretation functor $\mathcal{F} C$.comp D map
〈proof〉
lemma is-functor:
shows functor $\mathcal{F} C . c o m p D \operatorname{map}\langle p r o o f\rangle$
interpretation $F F$ : product-functor $\mathcal{F} C$.comp $\mathcal{F} C . c o m p ~ D ~ D ~ m a p ~ m a p ~\langle p r o o f\rangle ~$
interpretation $F o T$ : composite-functor $\mathcal{F} C . C C . c o m p \mathcal{F} C . c o m p ~ D \mathcal{F} C . T_{F M C} \operatorname{map}\langle p r o o f\rangle$ interpretation ToFF: composite-functor FC.CC.comp D.CC.comp D FF.map $T_{D}\langle p r o o f\rangle$

```
interpretation strict-monoidal-functor
                    \mathcal{FC.comp FC.T}\mp@subsup{T}{FMC}{\mathcal{FC.\alpha F}C.\iota}D T TD 的 \iotaD map
    <proof\rangle
    lemma is-strict-monoidal-functor:
    shows strict-monoidal-functor \mathcal{FC.comp \mathcal{FC.T}\mp@subsup{T}{FMC}{}\mathcal{F}C.\alpha\mathcal{F}C.\iota D T TD 的 \iotaD map}
    <proof>
end
sublocale evaluation-functor }\subseteq\mathrm{ strict-monoidal-functor
                            \mathcal{FC.comp FFC.T TFMC FFC.\alpha}
<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 \(C_{0}::{ }^{\prime} c_{0}\) comp
and \(C::\) 'c comp (infixr \(\cdot C\) 55)
and \(T_{C}::{ }^{\prime} c *{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\alpha_{C}::{ }^{\prime} c *{ }^{\prime} c *{ }^{\prime} c \Rightarrow{ }^{\prime} c\)
and \(\iota_{C}:: ' c\)
and \(D::\) 'd comp (infixr \(\cdot{ }^{D}\) 55)
and \(T_{D}:: ' d * ' d \Rightarrow ' d\)
and \(\alpha_{D}:: ' d *^{\prime} d * ' d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::\) ' \(d\)
and \(I:: ' c_{0} \Rightarrow{ }^{\prime} c\)
and \(V::{ }^{\prime} c_{0} \Rightarrow{ }^{\prime} d\)
and \(F::{ }^{\prime} c \Rightarrow{ }^{\prime} d+\)
assumes is-extension: \(\forall f . C_{0}\).arr \(f \longrightarrow F(I f)=V f\)
sublocale evaluation-functor \(\subseteq\)
strict-monoidal-extension \(C \mathcal{F} C . c o m p \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha \mathcal{F} C . \iota D T_{D} \alpha_{D} \iota_{D}\)
    \(\mathcal{F} C\).inclusion-of-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 \(C \mathcal{F} C . c o m p \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha \mathcal{F} C . \iota D T_{D} \alpha_{D} \iota_{D}\)
                            \(\mathcal{F} C\).inclusion-of-generators \(V F\)
for \(C::\) ' \(c\) comp (infixr \(\cdot C\) 55)
and \(D::\) 'd comp (infixr \(\left.\cdot_{D} 55\right)\)
and \(T_{D}::{ }^{\prime} d *{ }^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
and \(V::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
and \(F::{ }^{\prime} c\) free-monoidal-category.arr \(\Rightarrow{ }^{\prime} d\)
begin
```

    lemma strictly-preserves-everything:
    shows \(C\).arr \(f \Longrightarrow F(\mathcal{F} C . m k a r r\langle f\rangle)=V f\)
    and \(F(\mathcal{F} C\).mkarr \(\mathcal{I})=\mathcal{I}_{D}\)
    and \(\llbracket \operatorname{Arr} t ; \operatorname{Arr} u \rrbracket \Longrightarrow F(\mathcal{F C} . m k a r r(t \otimes u))=F(\mathcal{F C} . m k a r r t) \otimes_{D} F(\mathcal{F C} . m k a r r u)\)
    and \(\llbracket\) Arr \(t ;\) Arr \(u ; \operatorname{Dom} t=\operatorname{Cod} u \rrbracket \Longrightarrow\)
        \(F(\mathcal{F C}\). mkarr \((t \cdot u))=F(\mathcal{F C . m k a r r} t) \cdot{ }_{D} F(\mathcal{F C}\). mkarr \(u)\)
    and Arr \(t \Longrightarrow F(\mathcal{F} C . m k a r r \operatorname{l}[t])=D \cdot \mathfrak{l}(F(\mathcal{F} C . m k a r r t))\)
    and \(\operatorname{Arr} t \Longrightarrow F\left(\mathcal{F} C . m k a r r \mathbf{l}^{-1}[t]\right)=D . \mathfrak{l}^{\prime}\). map \((F(\mathcal{F} C . m k a r r t))\)
    and \(\operatorname{Arr} t \Longrightarrow F(\mathcal{F C} . m k a r r \mathbf{r}[t])=D . \varrho(F(\mathcal{F C}\). mkarr \(t))\)
    and \(\operatorname{Arr} t \Longrightarrow F\left(\mathcal{F C} . m k a r r \mathbf{r}^{-1}[t]\right)=D \cdot \varrho^{\prime} \cdot \operatorname{map}(F(\mathcal{F C} . m k a r r t))\)
    and \(\llbracket\) Arr \(t ; A r r u ; A r r v \rrbracket \Longrightarrow\)
        \(F(\mathcal{F} C . m k a r r \mathbf{a}[t, u, v])=\alpha_{D}(F(\mathcal{F} C . m k a r r t), F(\mathcal{F} C . m k a r r u), F(\mathcal{F} C . m k a r r v))\)
    and \(\llbracket\) Arr \(t ; A r r u ; A r r v \rrbracket \Longrightarrow\)
            \(F\left(\mathcal{F C} . m k a r r \mathbf{a}^{-1}[t, u, v]\right)\)
            \(=D \cdot \alpha^{\prime}(F(\mathcal{F} C . m k a r r t), F(\mathcal{F} C . m k a r r u), F(\mathcal{F} C . m k a r r v))\)
    \(\langle p r o o f\rangle\)
    end
sublocale evaluation-functor $\subseteq$ strict-monoidal-extension-to-free-monoidal-category

$$
C D T_{D} \alpha_{D} \iota_{D} V m a p
$$

$\langle p r o o f\rangle$

## context free-monoidal-category <br> 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 $C D T_{D} \alpha_{D} \iota_{D} V$
$\langle p r o o f\rangle$
end

## 4．3 Strict Subcategory

context free－monoidal－category<br>begin

In this section we show that $\mathcal{F} C$ is monoidally equivalent to its full subcategory $\mathcal{F}_{S} C$ whose objects are the equivalence classes of diagonal identity terms，and that this subcategory is the free strict monoidal category generated by $C$ ．

```
interpretation \(\mathcal{F}_{S} C\) : full-subcategory comp \(\langle\lambda f\). ide \(f \wedge \operatorname{Diag}(D O M f)\rangle\)
    〈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 \(\left\lfloor\right.\) rep \(f\) 」 else \(\mathcal{F}_{S} C\).null
```

The arrows of $\mathcal{F}_{S} C$ are those equivalence classes whose canonical representative term has diagonal formal domain and codomain．
lemma strict－arr－char：
shows $\mathcal{F}_{S} C . \operatorname{arr} f \longleftrightarrow \operatorname{arr} f \wedge \operatorname{Diag}(D O M f) \wedge \operatorname{Diag}(C O D f)$
〈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 \(f \longleftrightarrow \operatorname{arr} f \wedge D f=f\)
〈proof〉
interpretation \(D\) : functor comp \(\mathcal{F}_{S} C . \operatorname{comp} D\)
〈proof〉
lemma diagonalize-is-functor:
shows functor comp \(\mathcal{F}_{S} C\).comp \(D\langle\) proof \(\rangle\)
lemma diagonalize-strict-arr:
assumes \(\mathcal{F}_{S} C\).arr \(f\)
shows \(D f=f\)
    \(\langle p r o o f\rangle\)
lemma diagonalize-is-idempotent:
shows \(D\) o \(D=D\)
    \(\langle p r o o f\rangle\)
lemma diagonalize-tensor:
assumes arr \(f\) and arr \(g\)
shows \(D(f \otimes g)=D(D f \otimes D g)\)
```

```
<proof\rangle
```

lemma ide－diagonalize－can：
assumes $\operatorname{can} f$
shows ide（ $D f$ ）
$\langle p r o o f\rangle$
We next show that the diagonalization functor and the inclusion of the full sub－ category $\mathcal{F}_{S} C$ underlie an equivalence of categories．The arrows mkarr（ $D O M a \downarrow$ ）， determined by reductions of canonical representatives，are the components of a natural isomorphism．

```
interpretation \(S\) : full-inclusion-functor comp \(\langle\lambda f\). ide \(f \wedge \operatorname{Diag}(D O M f)\rangle\langle p r o o f\rangle\)
interpretation DoS: composite-functor \(\mathcal{F}_{S} C . \operatorname{comp} \operatorname{comp} \mathcal{F}_{S} C . \operatorname{comp} \mathcal{F}_{S} C . m a p ~ D\)
    \(\langle p r o o f\rangle\)
interpretation SoD: composite-functor comp \(\mathcal{F}_{S} C . \operatorname{comp} \operatorname{comp} D \mathcal{F}_{S} C . m a p\langle p r o o f\rangle\)
interpretation \(\nu\) : transformation-by-components
    comp comp map SoD.map 〈 \(\lambda a\). mkarr (DOM a \(\downarrow\) )〉
\(\langle p r o o f\rangle\)
interpretation \(\nu\) : natural-isomorphism comp comp map SoD.map \(\nu . m a p\)
    〈proof〉
```

The restriction of the diagonalization functor to the subcategory $\mathcal{F}_{S} C$ is the identity.
lemma $D o S-e q-\mathcal{F}_{S} C$ :
shows DoS.map $=\mathcal{F}_{S}$ C.map
$\langle$ proof $\rangle$
interpretation $\mu$ : transformation-by-components
$\mathcal{F}_{S}$ C.comp $\mathcal{F}_{S} C . c o m p$ DoS.map $\mathcal{F}_{S} C . m a p\langle\lambda a . a\rangle$
$\langle p r o o f\rangle$
interpretation $\mu$ ：natural－isomorphism $\mathcal{F}_{S} C$ ．comp $\mathcal{F}_{S} C$ ．comp DoS．map $\mathcal{F}_{S} C$ ．map $\mu$ ．map $\langle p r o o f\rangle$
interpretation equivalence－of－categories $\mathcal{F}_{S} C . \operatorname{comp} \operatorname{comp} D \mathcal{F}_{S} C . m a p \nu . m a p$ ．map $\langle p r o o f\rangle$
We defined the natural isomorphisms $\mu$ and $\nu$ by giving their components（i．e．their values at objects）．However，it is helpful in exporting these facts to have simple charac－ terizations of their values for all arrows．

```
definition \(\mu\)
```

where $\mu \equiv \lambda f$. if $\mathcal{F}_{S} C$.arr $f$ then $f$ else $\mathcal{F}_{S} C$.null
definition $\nu$
where $\nu \equiv \lambda f$. if arr $f$ then mkarr ( $C O D f \downarrow$ ) $f$ else null
lemma $\mu$-char:
shows $\mu$.map $=\mu$
〈proof〉

```
lemma \(\nu\)-char:
shows \(\nu\).map \(=\nu\)
    \(\langle p r o o f\rangle\)
lemma is-equivalent-to-strict-subcategory:
shows equivalence-of-categories \(\mathcal{F}_{S} C . c o m p\) comp \(D \mathcal{F}_{S} C . m a p ~ \nu \mu\)
〈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 \(C \operatorname{comp} \mathcal{F}_{S} C\) ．comp inclusion－of－generators \(D\langle\) proof \(\rangle\)
lemma DoI－eq－I：
shows DoI．map \(=\) inclusion－of－generators
\(\langle p r o o f\rangle\)
end
```

Next，we show that the subcategory $\mathcal{F}_{S} C$ inherits monoidal structure from the am－ bient 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 \(\lambda f\). \(\mathcal{F} C . i d e f \wedge \operatorname{Diag}(\mathcal{F} C . D O M f)\)
    for \(C::{ }^{\prime} c\) comp
begin
    interpretation \(D\) : functor \(\mathcal{F} C . \operatorname{comp} \operatorname{comp} \mathcal{F} C . D\)
        \(\langle p r o o f\rangle\)
    notation comp (infixr \(\cdot S 55\) )
    definition tensor \({ }_{S} \quad\left(\right.\) infixr \(\otimes_{S}\) 53)
    where \(f \otimes_{S} g \equiv \mathcal{F} C . D(\mathcal{F C} . t e n s o r f g)\)
    definition \(\operatorname{assoc}_{S} \quad\left(\mathrm{a}_{S}[-,-,-]\right)\)
    where \(a_{s s o c}^{S}\) a \(b c \equiv a \otimes_{S} b \otimes_{S} c\)
    lemma tensor-char:
    assumes arr \(f\) and arr \(g\)
    shows \(f \otimes_{S} g=\mathcal{F} C . m k a r r(\lfloor\mathcal{F} C . r e p f\rfloor\lfloor\otimes\rfloor\lfloor\mathcal{F} C . r e p g\rfloor)\)
        \(\langle p r o o f\rangle\)
lemma tensor-in-hom [simp]:
assumes \(« f: a \rightarrow b »\) and \(« g: c \rightarrow d »\)
shows \(« f \otimes_{S} g: a \otimes_{S} c \rightarrow b \otimes_{S} d »\)
```

```
    \langleproof\rangle
lemma arr-tensor [simp]:
assumes arr f and arr g
shows arr (f}\mp@subsup{|}{S}{}g
    <proof\rangle
lemma dom-tensor [simp]:
assumes arr f and arr g
shows dom (f 的 g) = dom f *}\mp@subsup{|}{S}{}\operatorname{dom}
    \langleproof\rangle
lemma cod-tensor [simp]:
assumes arr f and arr g
shows cod (f\mp@subsup{\otimes}{S}{}g)=\operatorname{cod}f\mp@subsup{\otimes}{S}{}\operatorname{cod}g
    \langleproof\rangle
lemma tensor-preserves-ide:
assumes ide a and ide b
shows ide (a \otimesS b)
    \langleproof\rangle
lemma tensor-tensor:
assumes arr f and arr g and arr h
```




```
<proof\rangle
lemma tensor-assoc:
assumes arr f and arr g and arr h
shows}(f\mp@subsup{\otimes}{S}{}g)\mp@subsup{\otimes}{S}{}h=f\mp@subsup{\otimes}{S}{}g\mp@subsup{\otimes}{S}{}
    \langleproof\rangle
lemma arr-unity:
shows arr I
    <proof\rangle
lemma tensor-unity-arr:
assumes arr f
shows }\mathcal{I}\mp@subsup{\otimes}{S}{}f=
    \langleproof\rangle
lemma tensor-arr-unity:
assumes arr f
shows f}\mp@subsup{\otimes}{S}{}\mathcal{I}=
    \langleproof\rangle
lemma assoc-char:
assumes ide a and ide b and ide c
```



```
    <proof\rangle
lemma assoc-in-hom:
assumes ide a and ide b and ide c
shows «\mp@subsup{a}{S}{}[a,b,c]:(a*\mp@subsup{\otimes}{S}{}b)\mp@subsup{\otimes}{S}{}c->a\mp@subsup{\otimes}{S}{}b\mp@subsup{\otimes}{S}{}c»
    <proof\rangle
The category }\mp@subsup{\mathcal{F}}{S}{}C\mathrm{ is a monoidal category.
interpretation EMC: elementary-monoidal-category comp tensor }\mp@subsup{\mp@code{S}}{S}{\mathcal{I}}\langle\lambdaa.a\rangle\langle\lambdaa.a\rangleassoc, S
<proof>
lemma is-elementary-monoidal-category:
shows elementary-monoidal-category comp tensor }\mp@subsup{S}{S}{}\mathcal{I}(\lambdaa.a)(\lambdaa.a) assoccs \langleproof
abbreviation T}\mp@subsup{T}{FSMC}{}\mathrm{ where T T TSMC 三EMC.T
abbreviation }\mp@subsup{\alpha}{FSMC}{}\mathrm{ where }\mp@subsup{\alpha}{FSMC}{}\equivEMC.
abbreviation }\mp@subsup{\iota}{FSMC}{}\mathrm{ where }\mp@subsup{\iota}{FSMC}{FEEMC.\iota
lemma is-monoidal-category:
```



```
    \langleproof\rangle
end
sublocale free-strict-monoidal-category }
    elementary-monoidal-category comp tensors I I \lambdaa. a \lambdaa. a assoc}\mp@subsup{S}{S}{
    \langleproof\rangle
```



```
    <proof>
sublocale free-strict-monoidal-category \subseteq
```



```
    <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 $\mathcal{F} C . c o m p$ comp $\mathcal{F} C . D$
$\langle p r o o f\rangle$
interpretation $I$ ：composite－functor $C \mathcal{F} C$ ．comp comp $\mathcal{F} C$ ．inclusion－of－generators $\mathcal{F} C . D$〈proof〉
definition inclusion－of－generators
where inclusion－of－generators $\equiv \mathcal{F}$ C．inclusion－of－generators
lemma inclusion－is－functor：
shows functor $C$ comp inclusion－of－generators
〈proof〉
The diagonalization functor is strict monoidal．
interpretation $D:$ strict－monoidal－functor $\mathcal{F} C . \operatorname{comp} \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha_{F M C} \mathcal{F} C . \iota_{F M C}$
comp $T_{F S M C} \alpha_{F S M C} \iota_{F S M C}$
$\mathcal{F} C . D$
$\langle p r o o f\rangle$
lemma diagonalize－is－strict－monoidal－functor：
shows strict－monoidal－functor $\mathcal{F} C . \operatorname{comp} \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha_{F M C} \mathcal{F} C . \iota_{F M C}$
comp $T_{F S M C} \alpha_{F S M C} \iota_{F S M C}$
$\mathcal{F} C . D$
$\langle p r o o f\rangle$
interpretation $\varphi$ ：natural－isomorphism
$\mathcal{F} C . C C . c o m p ~ c o m p ~ D . T_{D}$ oFF．map D．Fo $T_{C}$. map D．$\varphi$
$\langle p r o o f\rangle$
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 $\mathcal{F} C . c o m p \mathcal{F} C . D \operatorname{map} \mathcal{F} C . \nu \mathcal{F} C . \mu$〈proof〉
interpretation $D$ ：monoidal－functor $\mathcal{F} C . \operatorname{comp} \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha_{F M C} \mathcal{F} C . \iota_{F M C}$
comp $T_{F S M C} \alpha_{F S M C} \iota_{F S M C}$ $\mathcal{F} C . D$ D．$\varphi$
$\langle p r o o f\rangle$
interpretation equivalence－of－monoidal－categories comp $T_{F S M C} \alpha_{F S M C} \iota_{F S M C}$
$\mathcal{F} C . c o m p \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha_{F M C} \mathcal{F} C . \iota_{F M C}$ $\mathcal{F} C . D D . \varphi \mathcal{I}$
map $\mathcal{F} C . \nu \mathcal{F} C . \mu$
$\langle p r o o f\rangle$
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 $T_{F S M C} \alpha_{F S M C} \iota_{F S M C}$
$\mathcal{F} C . \mathrm{comp} \mathcal{F} C . T_{F M C} \mathcal{F} C . \alpha_{F M C} \mathcal{F} C . \iota_{F M C}$
$\mathcal{F} C . D$ D．$\varphi$
map $\mathcal{F} C . \nu \mathcal{F} C . \mu$
$\langle p r o o f\rangle$
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 \(C::{ }^{\prime} c\) comp (infixr \(\cdot C\) 55)
and \(D::\) 'd comp (infixr \(\left.{ }_{D}{ }^{5} 55\right)\)
and \(T_{D}::{ }^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}:: ' d\)
and \(V::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
begin
notation \(\mathcal{F}\) C.in-hom ( \((«-:-\rightarrow-»)\)
notation \(\mathcal{F}_{S}\) C.in-hom \(\left(《-:-\rightarrow_{S}-»\right)\)
definition map
where map \(\equiv \lambda f\). 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 $\langle$ proof $\rangle$
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 $C a n t \Longrightarrow$ D．ide $\{t\}$
$\langle p r o o f\rangle$
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 canon－ ical 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 $(\mathcal{F} C . D f)=E . \operatorname{map} f$
〈proof〉

```
lemma strictly-preserves-tensor:
assumes \(\mathcal{F}_{S} C\).arr \(f\) and \(\mathcal{F}_{S} C\).arr \(g\)
shows map \(\left(\mathcal{F}_{S} C\right.\).tensor \(\left.f g\right)=\operatorname{map} f \otimes_{D}\) map \(g\)
〈proof〉
lemma is-strict-monoidal-functor:
shows strict-monoidal-functor \(\mathcal{F}_{S} C . c o m p \mathcal{F}_{S} C . T_{F S M C} \mathcal{F}_{S} C . \alpha \mathcal{F}_{S} C . \iota D T_{D} \alpha_{D} \iota_{D}\) map
〈proof〉
end
sublocale strict-evaluation-functor \(\subseteq\)
        strict-monoidal-functor \(\mathcal{F}_{S} C . \operatorname{comp} \mathcal{F}_{S} C . T_{F S M C} \mathcal{F}_{S} C . \alpha \mathcal{F}_{S} C . \iota D T_{D} \alpha_{D} \iota_{D}\) map
    \(\langle p r o o f\rangle\)
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 \(C \mathcal{F}_{S} C . \operatorname{comp} \mathcal{F}_{S} C . T_{F S M C} \mathcal{F}_{S} C . \alpha \mathcal{F}_{S} C . \iota D T_{D} \alpha_{D} \iota_{D}\)
    \(\mathcal{F}_{S}\) C.inclusion-of-generators \(V F\)
for \(C::\) 'c comp (infixr \(\cdot{ }_{C}\) 55)
and \(D::\) 'd comp \(\quad\left(\right.\) infixr \(\left.\cdot{ }_{D} 55\right)\)
and \(T_{D}::{ }^{\prime} d *{ }^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\alpha_{D}::{ }^{\prime} d *^{\prime} d *^{\prime} d \Rightarrow{ }^{\prime} d\)
and \(\iota_{D}::{ }^{\prime} d\)
and \(V::{ }^{\prime} c \Rightarrow{ }^{\prime} d\)
and \(F::\) 'c free-monoidal-category.arr \(\Rightarrow{ }^{\prime} d\)
sublocale strict-evaluation-functor \(\subseteq\)
    strict-monoidal-extension \(C \mathcal{F}_{S} C . \operatorname{comp} \mathcal{F}_{S} C . T_{F S M C} \mathcal{F}_{S} C . \alpha \mathcal{F}_{S} C . \iota D T_{D} \alpha_{D} \iota_{D}\)
    \(\mathcal{F}_{S}\) C.inclusion-of-generators \(V\) map
\(\langle p r o o f\rangle\)
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 $C D T_{D} \alpha_{D} \iota_{D} V$
$\langle p r o o f\rangle$
end

## Chapter 5

## Cartesian Monoidal Category

theory CartesianMonoidalCategory<br>imports MonoidalCategory Category3.CartesianCategory<br>begin

### 5.1 Symmetric Monoidal Category

```
locale symmetric-monoidal-category \(=\)
    monoidal-category \(C T \alpha \iota+\)
    \(S\) : symmetry-functor \(C C+\)
    ToS: composite-functor CC.comp CC.comp C S.map \(T+\)
    \(\sigma\) : natural-isomorphism CC.comp C T ToS.map \(\sigma\)
for \(C\) :: 'a comp
                                (infixr • 55)
and \(T::{ }^{\prime} a * ' a \Rightarrow{ }^{\prime} a\)
and \(\alpha::{ }^{\prime} a * ' a *{ }^{\prime} a \Rightarrow{ }^{\prime} a\)
and \(\iota:::^{\prime} a\)
and \(\sigma::{ }^{\prime} a *{ }^{\prime} a \Rightarrow{ }^{\prime} a+\)
assumes sym-inverse: \(\llbracket\) ide \(a ;\) ide \(b \rrbracket \Longrightarrow\) inverse-arrows \((\sigma(a, b))(\sigma(b, a))\)
and unitor-coherence: ide \(a \Longrightarrow \mathrm{l}[a] \cdot \sigma(a, \mathcal{I})=\mathrm{r}[a]\)
and assoc-coherence: \(\llbracket\) ide a; ide b; ide \(c \rrbracket \Longrightarrow\)
    \(\alpha(b, c, a) \cdot \sigma(a, b \otimes c) \cdot \alpha(a, b, c)\)
\(\quad=(b \otimes \sigma(a, c)) \cdot \alpha(b, a, c) \cdot(\sigma(a, b) \otimes c)\)
```

begin
abbreviation sym
where sym $a b \equiv \sigma(a, b)$
end
locale elementary-symmetric-monoidal-category $=$
elementary-monoidal-category $C$ tensor unity lunit runit assoc
for $C$ :: 'a comp
(infixr - 55)
and tensor $::{ }^{\prime} a \Rightarrow^{\prime} a \Rightarrow{ }^{\prime} a$
and unity $::{ }^{\prime} a$
and lunit :: ' $a \Rightarrow{ }^{\prime} a$
(infixr $\otimes 53$ )
(I)
(l[-])

```
and runit \(::\) ' \(a \Rightarrow\) ' \(a \quad\) (r[-])
and assoc \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{a}[-,-,-])\)
and sym \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{~s}[-,-])+\)
assumes sym-in-hom: 【ide \(a\); ide \(b \rrbracket \Longrightarrow 《 \mathrm{~s}[a, b]: a \otimes b \rightarrow b \otimes a »\)
and sym-naturality: 【arr \(f ;\) arr \(g \rrbracket \Longrightarrow \mathrm{~s}[\operatorname{cod} f, \operatorname{cod} g] \cdot(f \otimes g)=(g \otimes f) \cdot \mathrm{s}[\operatorname{dom} f, \operatorname{dom} g]\)
and sym-inverse: 【ide \(a\); ide \(b \rrbracket \Longrightarrow\) inverse-arrows \(\mathrm{s}[a, b] \mathrm{s}[b, a]\)
and unitor-coherence: ide \(a \Longrightarrow \mathrm{l}[a] \cdot \mathrm{s}[a, \mathcal{I}]=\mathrm{r}[a]\)
and assoc-coherence: \(\llbracket\) ide \(a\); ide \(b\); ide \(c \rrbracket \Longrightarrow\)
\[
\begin{aligned}
& \mathrm{a}[b, c, a] \cdot \mathrm{s}[a, b \otimes c] \cdot \mathrm{a}[a, b, c] \\
& \quad=(b \otimes \mathrm{~s}[a, c]) \cdot \mathrm{a}[b, a, c] \cdot(\mathrm{s}[a, b] \otimes c)
\end{aligned}
\]
```

begin
lemma sym－simps［simp］：
assumes ide $a$ and ide $b$
shows $\operatorname{arr} \mathrm{s}[a, b]$
and $\operatorname{dom} \mathrm{s}[a, b]=a \otimes b$
and $\operatorname{cod} \mathrm{s}[a, b]=b \otimes a$
$\langle p r o o f\rangle$
interpretation $C C$ ：product－category $C C\langle$ proof $\rangle$
sublocale $M C$ ：monoidal－category $C T \alpha \iota$
$\langle p r o o f\rangle$
interpretation $S$ ：symmetry－functor $C$ C $\langle$ proof $\rangle$
interpretation ToS：composite－functor CC．comp CC．comp C S．map $T\langle p r o o f\rangle$
definition $\sigma::{ }^{\prime} a *{ }^{\prime} a \Rightarrow{ }^{\prime} a$
where $\sigma f \equiv$ if CC．arr $f$ then $\mathrm{s}[\operatorname{cod}(f s t f)$ ，cod $(\operatorname{snd} f)] \cdot(f s t f \otimes$ snd $f)$ else null
interpretation $\sigma$ ：natural－isomorphism CC．comp $C$ T ToS．map $\sigma$
〈proof〉
interpretation symmetric－monoidal－category $C T \alpha \iota \sigma$
〈proof〉
lemma induces－symmetric－monoidal－category $y_{C M C}$ ：
shows symmetric－monoidal－category $C T \alpha \iota \sigma$
$\langle p r o o f\rangle$
end
context symmetric－monoidal－category
begin
interpretation EMC：elementary－monoidal－category $C$ tensor unity lunit runit assoc $\langle p r o o f\rangle$
lemma induces－elementary－symmetric－monoidal－category ${ }_{C M C}$ ：
shows elementary－symmetric－monoidal－category

```
        C tensor unity lunit runit assoc ( }\lambdaab.\sigma(a,b)
    \langleproof\rangle
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 \(C T \alpha \iota\)
for \(C\) :: 'a comp
(infixr • 55)
and \(T::{ }^{\prime} a *^{\prime} a \Rightarrow{ }^{\prime} a\)
and \(\alpha:: ' a *{ }^{\prime} a *{ }^{\prime} a \Rightarrow^{\prime} a\)
and \(\iota::{ }^{\prime} a+\)
assumes terminal-unity: terminal \(\mathcal{I}\)
and tensor-is-product:
    \(\llbracket i d e ~ a ;\) ide \(b ; « t_{a}: a \rightarrow \mathcal{I} » ; « t_{b}: b \rightarrow \mathcal{I} » \rrbracket \Longrightarrow\)
        has-as-binary-product ab(r[a]•(a@tb))(l[b]•(ta@b))
begin
```

    sublocale category-with-terminal-object
        \(\langle p r o o f\rangle\)
    lemma is-category-with-terminal-object:
    shows category-with-terminal-object \(C\)
        \(\langle p r o o f\rangle\)
    definition the-trm ( \(\mathrm{t}[-]\) )
    where the-trm \(\equiv \lambda f\). THE \(t . « t: \operatorname{dom} f \rightarrow \mathcal{I} »\)
    lemma trm-in-hom [intro]:
    assumes ide a
    shows \(« \mathrm{t}[a]: a \rightarrow \mathcal{I}\) »
        \(\langle p r o o f\rangle\)
    lemma trm-simps [simp]:
    assumes ide a
    shows arr \(\mathrm{t}[a]\) and \(d o m \mathrm{t}[a]=a\) and \(\operatorname{cod} \mathrm{t}[a]=\mathcal{I}\)
        \(\langle p r o o f\rangle\)
    interpretation elementary-category-with-terminal-object \(C \mathcal{I}\) the-trm
    〈proof〉
    lemma extends-to-elementary-category-with-terminal-object \({ }_{C M C}\) :
    ```
shows elementary-category-with-terminal-object C I the-trm
    <proof\rangle
definition pro (\mp@subsup{p}{0}{[-, -])}
where proa b 三l[b] . (t[a]\otimesb)
definition prr ( }\mp@subsup{\mathfrak{p}}{1}{[-, -])
where pr r a b = r[a] . (a\otimes t[b])
sublocale ECC: elementary-category-with-binary-products C pro pr r
<proof>
lemma induces-elementary-category-with-binary-products }\mp@subsup{C}{MC}{}\mathrm{ :
shows elementary-category-with-binary-products C pro pr r
    \langleproof\rangle
lemma is-category-with-binary-products:
shows category-with-binary-products C
    <proof\rangle
sublocale category-with-binary-products C
    <proof\rangle
sublocale ECC: elementary-cartesian-category C proprr I I the-trm <proof\rangle
lemma extends-to-elementary-cartesian-category }\mp@subsup{\}{MC}{}\mathrm{ :
shows elementary-cartesian-category C proprr I I the-trm
    \langleproof\rangle
lemma is-cartesian-category:
shows cartesian-category C
    \langleproof\rangle
sublocale cartesian-category C
    \langleproof\rangle
abbreviation dup (d[-])
where dup \equivECC.dup
abbreviation tuple (\langle-, -\rangle)
where }\langlef,g\rangle\equivECC.tuple f
lemma prod-eq-tensor:
shows ECC.prod = tensor
<proof>
lemma Prod-eq-T:
```

```
shows ECC.Prod =T
<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 $\mathfrak{p}_{0}[\mathcal{I}, a]=1[a]$
$\langle p r o o f\rangle$
lemma runit－eq：
assumes ide a
shows $\mathfrak{p}_{1}[a, \mathcal{I}]=\mathrm{r}[a]$
$\langle p r o o f\rangle$
interpretation $S$ ：symmetry－functor $C C\langle p r o o f\rangle$
interpretation ToS：composite－functor CC．comp CC．comp C S．map $T\langle p r o o f\rangle$
interpretation $\sigma$ ：natural－transformation CC．comp $C$ T ToS．map ECC．$\sigma$〈proof〉
interpretation $\sigma$ ：natural－isomorphism CC．comp C T ToS．map ECC．$\sigma$
$\langle p r o o f\rangle$
sublocale SMC：symmetric－monoidal－category $C T \alpha \iota E C C . \sigma$〈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 \(:: ' a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathbf{i n f i x r} \otimes 53)\)
and unity \(::{ }^{\prime} a\)
    (I)
```

```
and lunit :: ' \(a \Rightarrow\) ' \(a\) ( \([-]\) )
and runit \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{r}[-])\)
and assoc :: ' \(a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{a}[-,-,-])\)
and trm \(::{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{t}[-])\)
and \(d u p::{ }^{\prime} a \Rightarrow{ }^{\prime} a \quad(\mathrm{~d}[-])+\)
assumes trm-in-hom: ide \(a \Longrightarrow « \mathrm{t}[a]: a \rightarrow \mathcal{I}\) »
and trm-unity: \(\mathrm{t}[\mathcal{I}]=\mathcal{I}\)
and trm-naturality: arr \(f \Longrightarrow \mathrm{t}[\operatorname{cod} f] \cdot f=\mathrm{t}[\operatorname{dom} f]\)
and dup-in-hom [intro]: ide \(a \Longrightarrow « \mathrm{~d}[a]: a \rightarrow a \otimes a »\)
and dup-naturality: arr \(f \Longrightarrow \mathrm{~d}[\operatorname{cod} f] \cdot f=(f \otimes f) \cdot \mathrm{d}[\operatorname{dom} f]\)
and prj0-dup: ide \(a \Longrightarrow \mathrm{r}[a] \cdot(a \otimes \mathrm{t}[a]) \cdot \mathrm{d}[a]=a\)
and prj1-dup: ide \(a \Longrightarrow \mathrm{l}[a] \cdot(\mathrm{t}[a] \otimes a) \cdot \mathrm{d}[a]=a\)
and tuple-prj: \(\llbracket\) ide \(a ;\) ide \(b \rrbracket \Longrightarrow(\mathrm{r}[a] \cdot(a \otimes \mathrm{t}[b]) \otimes 1[b] \cdot(\mathrm{t}[a] \otimes b)) \cdot \mathrm{d}[a \otimes b]=a \otimes b\)
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
    \(\langle p r o o f\rangle\)
    lemma induces-elementary-cartesian-monoidal-category \({ }_{C M C}\) :
    shows elementary-cartesian-monoidal-category \(C\) tensor \(\mathcal{I}\) lunit runit assoc the-trm dup
        \(\langle p r o o f\rangle\)
    end
context elementary-cartesian-monoidal-category
begin
lemma trm-simps [simp]:
assumes ide a
shows arr $\mathrm{t}[a]$ and $d o m \mathrm{t}[a]=a$ and $\operatorname{cod} \mathrm{t}[a]=\mathcal{I}$
$\langle p r o o f\rangle$
lemma dup-simps [simp]:
assumes ide a
shows arr $\mathrm{d}[a]$ and $d o m \mathrm{~d}[a]=a$ and $\operatorname{cod} \mathrm{d}[a]=a \otimes a$
〈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 I trm
    <proof\rangle
interpretation MC: monoidal-category CT \alpha \iota
    <proof>
interpretation ECBP: elementary-category-with-binary-products C
                        <\lambdaa b. l[b] ( (t[a]\otimesb)\rangle\langle\lambdaab.r[a]\cdot(a\otimest[b])\rangle
<proof\rangle
lemma induces-elementary-category-with-binary-products ECMC
shows elementary-category-with-binary-products C
            (\lambdaab. l[b]\cdot(t[a]\otimesb))(\lambdaab.r[a]\cdot(a\otimest[b]))
    <proof\rangle
sublocale cartesian-monoidal-category CT \alpha \iota
<proof>
lemma induces-cartesian-monoidal-category ECMC:
shows cartesian-monoidal-category CT \alpha \iota
    \langleproof\rangle
end
locale diagonal-functor =
    C: category C +
    CC: product-category C C
for C :: 'a comp
begin
    abbreviation map
    where map f\equivif 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\rangle
end
context cartesian-monoidal-category
begin
    sublocale \Delta: diagonal-functor C \langleproof\rangle
```

interpretation To $\Delta$ : composite-functor $C$ CC.comp $C \Delta . \operatorname{map} T\langle p r o o f\rangle$
sublocale $\delta$ : natural-transformation $C C$ map $\langle T$ o $\Delta$.map $\rangle d u p$〈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 \langleproof\rangle
interpretation CCC: product-category C CC.comp \langleproof\rangle
interpretation T: binary-functor C C C Prod
    <proof>
interpretation T: binary-endofunctor C Prod \langleproof\rangle
interpretation ToTC: functor CCC.comp C T.ToTC
        \langleproof\rangle
interpretation ToCT: functor CCC.comp C T.ToCT
    <proof>
interpretation \alpha: natural-isomorphism CCC.comp C T.ToTC T.ToCT \alpha
    <proof>
interpretation L: functor C C <\lambdaf. Prod ( }\operatorname{cod}\iota,f)
    <proof>
interpretation L: endofunctor C <\lambdaf. Prod (cod \iota, f)\rangle\langleproof\rangle
interpretation l: transformation-by-components C C
                            <\lambdaf. Prod }(\operatorname{cod}\iota,f)\rangle map <\lambdaa.pr0 (\operatorname{cod}\iota)a
    <proof>
interpretation l: natural-isomorphism C C<\lambdaf. Prod (cod \iota, f)\rangle map l.map
    <proof>
interpretation L: equivalence-functor C C <\lambdaf. Prod ( }\operatorname{cod}\iota,f)
        \langleproof\rangle
interpretation R: functor C C <\lambdaf. Prod (f, \operatorname{cod \iota)\rangle}
    <proof>
interpretation R: endofunctor C <\lambdaf. Prod (f, cod \iota)\rangle\langleproof\rangle
interpretation @: transformation-by-components C C
                        \langle\lambdaf. Prod (f,\operatorname{cod \iota)\rangle map \\lambdaa. \mathfrak{p}}[{,\operatorname{cod}\iota]\rangle
        <proof>
    interpretation @: natural-isomorphism C C<\lambdaf. Prod (f, cod \iota)\rangle map @.map
        <proof>
interpretation R: equivalence-functor C C < |f. Prod ( f, \operatorname{cod \iota)\rangle}
        <proof>
```

```
interpretation MC: monoidal-category C Prod \alpha \iota
    <proof>
```

lemma induces-monoidal-category ${ }_{E C C}$ :
shows monoidal-category $C$ Prod $\alpha \iota$
$\langle p r o o f\rangle$
lemma unity-agreement:
shows MC.unity $=1$
$\langle p r o o f\rangle$
lemma assoc-agreement:
assumes ide $a$ and ide $b$ and ide $c$
shows MC.assoc a $b c=\mathrm{a}[a, b, c]$
$\langle p r o o f\rangle$
lemma assoc'-agreement:
assumes ide $a$ and ide $b$ and ide $c$
shows MC.assoc' a b $c=\mathrm{a}^{-1}[a, b, c]$
$\langle p r o o f\rangle$
lemma runit-char-eqn:
assumes ide a
shows $\mathrm{r}[a] \otimes \mathbf{1}=(a \otimes \iota) \cdot \mathrm{a}[a, \mathbf{1}, \mathbf{1}]$
$\langle p r o o f\rangle$
lemma runit-agreement:
assumes ide a
shows MC.runit $a=\mathrm{r}[a]$
$\langle p r o o f\rangle$
lemma lunit－char－eqn：
assumes ide a
shows $\mathbf{1} \otimes 1[a]=(\iota \otimes a) \cdot \mathrm{a}^{-1}[\mathbf{1}, \mathbf{1}, a]$
〈proof〉
lemma lunit－agreement：
assumes ide a
shows MC．lunit $a=1[a]$
$\langle$ proof $\rangle$
interpretation CMC：cartesian－monoidal－category C Prod $\alpha$ ८〈proof〉
lemma extends－to－cartesian－monoidal－category ${ }_{E C C}$ ：
shows cartesian－monoidal－category $C$ Prod $\alpha \iota$
$\langle p r o o f\rangle$

```
    lemma trm-agreement:
    assumes ide a
    shows CMC.the-trm a = t[a]
        <proof\rangle
    lemma pr-agreement:
    assumes ide a and ide b
```



```
    <proof>
    lemma dup-agreement:
    assumes ide a
    shows CMC.dup a = d[a]
    <proof>
end
```


### 5.5 Cartesian Monoidal Category from Elementary Cartesian Category

```
context elementary-cartesian-category
begin
    interpretation MC: monoidal-category C Prod \alpha \iota
    <proof>
lemma triangle:
assumes ide a and ide b
shows (a\otimesl[b])\cdot\textrm{a}[a,\mathbf{1},b]=\textrm{r}[a]\otimesb
    <proof\rangle
lemma induces-elementary-cartesian-monoidal-category ECC
shows elementary-cartesian-monoidal-category (.) prod 1 lunit runit assoc trm dup
    <proof\rangle
end
context cartesian-category
begin
interpretation ECC: elementary-cartesian-category C
    some-pr0 some-pr1 some-terminal some-terminator
    <proof>
lemma extends-to-cartesian-monoidal-category \({ }_{C C}\) :
shows cartesian-monoidal-category C ECC.Prod ECC. \(\alpha\) ECC.८
```

```
    <proof\rangle
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.

