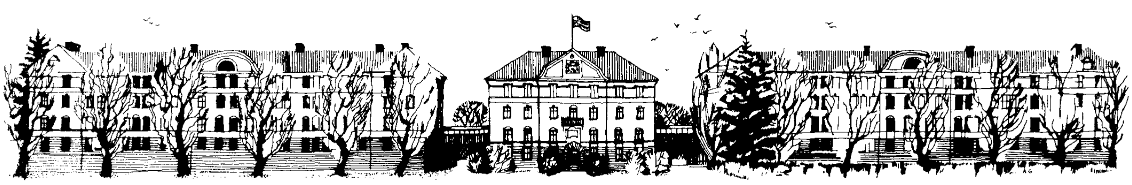
AN E-BICATEGORY OF E-CATEGORIES EXEMPLIFYING A TYPE-THEORETIC APPROACH TO BICATEGORIES

Olov Wilander

U.U.D.M. Report 2005:48
ISSN 1101–3591



Department of Mathematics

Uppsala University

AN E-BICATEGORY OF E-CATEGORIES EXEMPLIFYING A TYPE-THEORETIC APPROACH TO BICATEGORIES

K.O. WILANDER

Abstract. A type-theoretic formalisation of bicategories is introduced, and it is shown that small E-categories, together with their functor categories, form such an E-bicategory. This is carried out using only basic recursive definitions, in the version of predicative type theory with a hierarchy of universes implemented by Agda. This relates to earlier work by Huet and Saïbi, who constructed a large category of small categories in Coq, but with the use of inductive families. The construction presented here may be considered more natural, particularly from the point of view of higher-dimensional category theory.

This paper presents a formalisation of some parts of category theory, including a first step towards higher-dimensional category theory. The formalisation is carried out in Agda, a type-theoretic framework with a hierarchy of universes implemented at Chalmers University of Technology, Gothenburg. Agda and Alfa (the version with a graphical interface) are intended to replace the earlier ALF system (see [2, 3, 11]). Further, not all features of the framework were used; restricting myself to use only basic recursive definitions excluded both Id-types and the `Equal_hom` construction of [7].

One of the driving ideas behind this work is the idea from category theory that we should study things only up to isomorphism. The definition of an E-category takes this one step further in that we have no notion at all of equality of objects, and on arrows only for those between the same two objects (a notion of equality for all arrows of course gives an equality relation on objects by comparing identity arrows[1]– similarly, an equality of functors $\mathbf{1} \to \mathcal{C}$ is the same as an equality on objects of $\mathcal{C}$, so we can not allow equality on functors either). These considerations are also important in the study of weak higher-dimensional categorical structures, so it is fitting that such structures should arise.

For related work, there is a short survey included in [12]. The work in [7] was an important motivation for the current work. It is also interesting to look at [8], which characterizes E-categories as a particular class of bicategories (namely those whose hom-categories are groupoids). In the light of that description, we might suspect that the E-bicategories presented here form a particular class of tricategories, though this idea has not been pursued further.

The formalisation work has required great care to be taken with the tools. In fact, a first attempt foundered: the formulation of E-bicategory grew beyond what the interactive tools could handle while remaining responsive. This was at least partly due to accepting the mechanically obtained solutions for some 'holes', which tended to be rather longer than necessary. The second attempt made extensive use

Date: December 21, 2005.

The author is supported by FMB, the Swedish Graduate School in Mathematics and Computing.

[1]From the construction in [7], we get the Id-type.

of the language's `let` construct, and also divided the formalisation into more and smaller parts.

1. Basic Notions

We start with a very brief introduction to some notions that we will use. They are standard notions, and the formalisations used were pre-existing.

The notations used will, for the sake of readability, differ significantly from those used in the proof script.

1.1. **Setoids.** In Bishop-style constructive mathematics, these play the same rôle as do sets in classical mathematics. A setoid consists of a small type together with a notion of equality on that type. Such a notion of equality is a record type consisting of a binary relation together with proof objects for its reflexivity, symmetry, and transitivity. When talking about equality this will, unless otherwise indicated, mean equality in the appropriate setoid.

We must also consider maps between setoids. These should of course take equal values for equal inputs (in set theory, this is a more-or-less trivial condition, but in this context, it is really saying that it respects the equivalence relation we think of as equality). This notion of map is formalised, again as a record type.

1.2. **E-Categories.** The notion of an E-category is a type-theoretic formulation of categories, originally due to P. Aczel (who introduced it in [1]). An E-category consists of a small type of objects, together with a setoid of arrows for every pair of objects, identity arrows, and a composition map (so, particularly, composition respects equality of arrows), satisfying the usual axioms for a category (see for example [5]). It is worth noticing that we have no notion of equal objects, but only of equal arrows between the same objects.

2. E-Functors

We now start extending the library of formalisations, first defining E-functors. Given two categories $\mathcal{A}$ and $\mathcal{B}$, a functor $F : \mathcal{A} \to \mathcal{B}$ assigns to each $a \in \text{ob}\ \mathcal{A}$ an object Fa of $\mathcal{B}$, and to each $f : a \to b$ in $\mathcal{A}$ an arrow $Ff : Fa \to Fb$ in $\mathcal{B}$, in a way that preserves both identity arrows and composition. The formalised version, an E-functor between E-categories `A` and `B` has an operation `objectfunction`, assigning an object of `B` to each object of `A`, and for each pair `x`,`y` of objects of `A`, an extensional map `arrowfunction x y` from `A.hom x y` to `B.hom (objectfunction x) (objectfunction y)`. Further, there are proof objects for the functoriality conditions, that the image of an identity arrow equals the appropriate identity arrows, and that the image of a composition equals the composite of the images. We will suppress `objectfunction` and `arrowfunction` and just write `Fx` and `Ff` for images of objects and arrows under an E-functor `F` (but this is only for this paper – the proof assistant has no support for this).

E-functors can of course be composed, in exactly the same way as ordinary functors. The construction is simple, composing the operations and maps from the two E-functors given, and the required proof-objects are easily constructed.

There is also an identity E-functor `Id` on any E-category `C`, which of course acts by the identity operation on objects, and identity maps on arrows. Functoriality is immediate.

3. E-Natural Transformations

The next objects of interest are natural transformations. Given categories $\mathcal{A}$ and $\mathcal{B}$, and functors $F, G : \mathcal{A} \to \mathcal{B}$, a natural transformation $\alpha : F \to G$ assigns to each

$x \in \text{ob}\,\mathcal{A}$ an arrow $\alpha_x : Fx \to Gx$ in $\mathcal{B}$ so as to make

$$\begin{tikzcd} Fx \arrow[r, "\alpha_x"] \arrow[d, "Ff"'] & Gx \arrow[d, "Gf"] \\ Fy \arrow[r, "\alpha_y"] & Gy \end{tikzcd}$$

commute for all arrows $f : x \to y$ in $\mathcal{A}$. Similarly, given E-categories `A` and `B`, and E-functors `F` and `G` from `A` to `B`, an E-natural transformation consists of an assignment **`arrows`** of an element of `B.hom Fx Gx` to each object `x` of `A`, and a proof that all squares such as the one above commute.

Two E-natural transformations from F to G are equal if they assign equal arrows to every object. This gives us a setoid of E-natural transformations.

Given E-categories `A` and `B`, E-functors `F`, `G`, and `H` from `A` to `B`, and E-natural transformations `a` from `F` to `G` and `b` from `G` to `H`, we may compose these to obtain an E-natural transformation `b` $\circ$ `a` from `F` to `H`, simply by composing components. The proof of naturality is easy, but tedious, to construct.

There is also a second way of composing natural transformations, known as horizontal composition (that of the previous paragraph being vertical). It can be (usefully) thought of as the effect of functor composition on natural transformations, but perhaps a diagram is the clearest explanation:

$$\begin{tikzcd} C \arrow[r, bend left, "F"] \arrow[r, bend right, "H"'] \arrow[r, phantom, "\Downarrow a"] & D \arrow[r, bend left, "G"] \arrow[r, bend right, "K"'] \arrow[r, phantom, "\Downarrow b"] & E \end{tikzcd} \quad \mapsto \quad \begin{tikzcd} C \arrow[r, bend left, "GF"] \arrow[r, bend right, "KH"'] \arrow[r, phantom, "\Downarrow b*a"] & E \end{tikzcd}$$

In constructing this natural transformation, there is a decision to be made, taking the component at an object `x` of `C` to be either $\mathtt{Ka_x} \circ \mathtt{b_{Fx}}$ or $\mathtt{b_{Hx}} \circ \mathtt{Ga_x}$ (these are of course equal). Having made a choice (picking, in this case, the former), proving naturality is easy.

4. E-Functor Categories

Given any two categories $\mathcal{C}$ and $\mathcal{D}$ there is a category $[\mathcal{C}, \mathcal{D}]$ of functors between them. We can in the same way, given E-categories `C` and `D`, construct an E-category `[C,D]` of E-functors between them. Its objects are the E-functors (note that these indeed form a small type), its hom-setoids are the setoids of E-natural transformations, composition is the first one above, and identities are taken component-wise. The axioms are then immediate from the axioms of `D`.

We are now also in position to formulate and prove, as lemmas, that an E-natural transformation is an iso (in the appropriate E-functor category) if and only if all of its components are isos.

5. Product E-Categories

One important way of constructing new categories is that of, from categories $\mathcal{A}$ and $\mathcal{B}$, forming their product category $\mathcal{A} \times \mathcal{B}$, having as objects pairs of objects from $\text{ob}\,\mathcal{A} \times \text{ob}\,\mathcal{B}$, and whose arrows are pairs of arrows from $\mathcal{A}$ and $\mathcal{B}$. This is a categorical product in **Cat**. We mimic this construction, obtaining from E-categories `A` and `B` a product E-category `A`$\times$`B`. We can not, of course, expect it to be a categorical product, but it comes nonetheless with a pairing, and a corresponding product of functors.

6. Some Particular E-Categories

At this point, we may as well introduce some examples of E-categories, and some associated constructions, particularly since we will need some of them.

The first example is the empty E-category. It has an empty type of objects, and is somewhat trivial.

The second, and useful, example is the unit E-category, which we will denote by `1`. It has a singleton type of objects, and singleton setoids of arrows. The resulting structure obviously satisfies the E-category axioms.

Having constructed the unit category, we now construct, for each E-category `C`, canonical functors (in fact equivalences) `E_rightunitcatelim` : $\mathtt{C} \times \mathtt{1} \to \mathtt{C}$ and `E_leftunitcatelim` : $\mathtt{1} \times \mathtt{C} \to \mathtt{C}$, corresponding to the canonical isomorphisms $\mathcal{A} \times \mathbf{1} \cong \mathcal{A}$ and $\mathbf{1} \times \mathcal{A} \cong \mathcal{A}$, respectively.

7. E-Bicategories

Classically, we may consider a (large) category **Cat** of all small categories. This, however requires us to talk about equality of functors, which is problematic in the current setting. With the use of a more general form of recursive definitions, such a construction was carried out in [7]. We shall refrain from such[2], and instead construct a different structure.

It is well known that **Cat** has more structure than just that of a category. In fact, the structure formed is known as a *2-category*. The slightly weaker notion that we are going to be interested in is that of a *bicategory* ([4, 10], or, shorter, [9]). The definitions are formalised to give the notion of an E-bicategory.

An E-bicategory has a (large) type `obj` of objects (or 0-cells), and for all objects `a` and `b`, there is an E-category `hom a b` (whose objects and arrows are called 1-cells and 2-cells, respectively). Further, there is (for all objects `a`, `b`, and `c`) an E-functor

$$(\mathtt{comp\ a\ b\ c}) : (\mathtt{hom\ b\ c}) \times (\mathtt{hom\ a\ b}) \to \mathtt{hom\ a\ c}\ ,$$

and for all objects `a` an E-functor $(\mathtt{identity\ a}) : \mathtt{1} \to \mathtt{hom\ a\ a}$. Finally, there are families of isomorphisms, replacing the category axioms. Thus, for all objects `a`, `b`, `c`, and `d`, there is an E-natural isomorphism `associativity a b c d`

$$\begin{tikzcd}
((\mathtt{hom\ d\ c}) \times (\mathtt{hom\ b\ c})) \times (\mathtt{hom\ a\ b}) \arrow[r, "\cong"] \arrow[dd, "\mathtt{comp}\times\mathtt{Id}"'] & (\mathtt{hom\ d\ c}) \times ((\mathtt{hom\ b\ c}) \times (\mathtt{hom\ a\ b})) \arrow[d, "\mathtt{Id}\times\mathtt{comp}"] \\
 & (\mathtt{hom\ c\ d}) \times (\mathtt{hom\ a\ c}) \arrow[d, "\mathtt{comp}"] \\
(\mathtt{hom\ b\ d}) \times (\mathtt{hom\ a\ b}) \arrow[r, "\mathtt{comp}"'] \arrow[ur, phantom, "\Nearrow\ \mathtt{associativity}"] & \mathtt{hom\ a\ d}
\end{tikzcd}$$

[2]Using the `idata` construction of Agda, an equivalent construction is easily carried out.

and for all objects `a` and `b` there are E-natural isomorphisms `rightid a b` and `leftid a b`

$$\begin{tikzcd}
\mathtt{1} \times (\mathtt{hom\ a\ b}) \arrow[d, "\mathtt{identity}\times\mathtt{Id}"] \arrow[dd, bend right=90, "\cong"'] & (\mathtt{hom\ a\ b}) \times \mathtt{1} \arrow[d, "\mathtt{Id}\times\mathtt{identity}"'] \arrow[dd, bend left=90, "\cong"] \\
(\mathtt{hom\ a\ a}) \times (\mathtt{hom\ a\ b}) \arrow[d, "\mathtt{comp}"] \arrow[l, phantom, "\overset{\mathtt{leftid}}{\Longleftarrow}"] & (\mathtt{hom\ a\ b}) \times (\mathtt{hom\ b\ b}) \arrow[d, "\mathtt{comp}"'] \arrow[r, phantom, "\overset{\mathtt{rightid}}{\Longrightarrow}"] \\
\mathtt{hom\ a\ b} & \mathtt{hom\ a\ b}
\end{tikzcd}$$

These must then satisfy some coherence axioms (more about which later). The formalisation gives an object at the next level of our hierarchy of universes.

The attentive reader might spot a 'white lie' in the diagrams above: the arrows marked as isomorphisms. Recall that we have no notion of isomorphism for categories (since that would require a notion of equality for functors). The indicated arrows are not canonical isomorphisms, but canonical equivalences of E-categories.

The formalised definition of an E-bicategory becomes very long, and it is reasonably clear that this is rather an unpleasant way of writing down the definition of a bicategory. Already the associativity and unit morphisms are fairly complicated compared to their diagrammatic descriptions, but are as nothing compared to the excessive verbosity of the two coherence axioms. These coherence axioms can also be more succinctly expressed, most commonly by saying that, for all choices of f, g, h, and k from appropriate hom-categories, the following two diagrams (writing a, r, and l for appropriate components of `associativity`, `rightid`, and `leftid`, respectively, and `id` for identity E-natural transformations) must commute:

$$\begin{tikzcd}
& & ((kh)g)f \arrow[dll, "a"'] \arrow[drr, "a*\mathtt{id}"] & & \\
(kh)(gf) \arrow[dr, "a"'] & & & & (k(hg))f \arrow[dl, "a"] \\
& k(h(gf)) & & k((hg)f) \arrow[ll, "\mathtt{id}*a"] &
\end{tikzcd}$$

$$\begin{tikzcd}
(gI)f \arrow[rr, "a"] \arrow[dr, "r*\mathtt{id}"'] & & g(If) \arrow[dl, "\mathtt{id}*l"] \\
& gf &
\end{tikzcd}$$

8. The E-Bicategory of E-Categories

We wish to show that there is an E-bicategory of E-categories, with the functor categories as hom-categories. This is where the largest part of the work happens. Having decided what the first two fields should be, we now need to construct the others, one by one.

First out is the composition E-functor,

$$\mathtt{comp\ A\ B\ C} : [\mathtt{B}, \mathtt{C}] \times [\mathtt{A}, \mathtt{B}] \to [\mathtt{A}, \mathtt{C}] \ ,$$

defined on objects, that is pairs $(\mathtt{F}, \mathtt{G})$ of E-functors $\mathtt{F} : \mathtt{B} \to \mathtt{C}$ and $\mathtt{G} : \mathtt{A} \to \mathtt{B}$, by functor composition. On arrows, it acts by horizontal composition. We must also provide proof of functoriality: that horizontal composition respects equality, and that identities are preserved are both straightforward to prove. The final property to be shown is that composition is preserved. This is exactly the interchange law.

The proof is long, and not particularly enlightening (but then again, a 'standard' proof is not enlightening either).

Next is the collection of identity 1-cells, E-functors

$$\mathtt{identity\ A} : \mathtt{1} \to [\mathtt{A}, \mathtt{A}]\ ,$$

simply picking out the identity E-functor, together with its identity E-natural transformation. The required proof objects are then easily constructed.

We now need to construct the associativity morphisms. These are E-natural transformations between E-functor categories, so have E-natural transformations as components. Conveniently, their components are simply the relevant identity morphisms, and their naturality is easily proved. Proving naturality for the associativity morphism itself is slightly more complicated. The entire construction is then essentially repeated, to construct the inverse required for the proof that the associativity morphism is an E-natural isomorphism.

The constructions of the right and left unit morphisms are similar to that of the preceding morphism. Again, their components are E-natural transformations having the identity morphisms as components, and easy proofs of naturality. The component-wise inverses are easily constructed. The naturality of the identity morphisms is now easily proved, and having shown as a lemma that an E-natural transformation is an iso if and only if all its components are, proofs that the identity morphisms are isos are obtained immediately.

The only things that remain are proof objects for the axioms. These are surprisingly easily constructed, since the relevant arrows all equal identity arrows. This, of course, does not entirely prevent the formalisation from growing long.

Having constructed all the necessary pieces, it is time to put them together. We obtain an E-bicategory `ECat` of E-categories.

9. Adjoint E-Functors

Having obtained a bicategory-like structure, the notion of an adjoint pair is now easily formalised. The definition as given in [6] fits into the type-theoretical formulation unchanged. Two 1-cells $\mathtt{f} : \mathtt{a} \to \mathtt{b}$ and $\mathtt{g} : \mathtt{b} \to \mathtt{a}$ form an adjoint pair if there are 2-cells $\epsilon : \mathtt{fg} \to \mathtt{1_b}$ and $\eta : \mathtt{1_a} :\to \mathtt{gf}$ (where $\mathtt{1\_}$ denotes the 1-cell chosen by `identity`) such that the composite 2-cells

$$\mathtt{f} \xrightarrow{r^{-1}} \mathtt{f} \circ \mathtt{1_a} \xrightarrow{\mathtt{id}*\eta} \mathtt{f} \circ (\mathtt{g} \circ \mathtt{f}) \xrightarrow{a^{-1}} (\mathtt{f} \circ \mathtt{g}) \circ \mathtt{f} \xrightarrow{\epsilon*\mathtt{id}} \mathtt{1_b} \circ \mathtt{f} \xrightarrow{l} \mathtt{f}$$

and

$$\mathtt{g} \xrightarrow{l^{-1}} \mathtt{1_a} \circ \mathtt{g} \xrightarrow{\eta*\mathtt{id}} (\mathtt{g} \circ \mathtt{f}) \circ \mathtt{g} \xrightarrow{a} \mathtt{g} \circ (\mathtt{f} \circ \mathtt{g}) \xrightarrow{\mathtt{id}*\epsilon} \mathtt{g} \circ \mathtt{1_b} \xrightarrow{r} \mathtt{g}$$

are both equal to identity 2-cells. In `ECat` this definition is equivalent to a more ad-hoc formalised definition more closely resembling the usual unit–co-unit definition of adjoint functors.

10. Polymorphism and Monomorphism

There are only two major differences between notations in this article, and in the proof script. The first one is the suppression of `objectfunction` and `arrowfunction` when applying functors, as discussed earlier. The second is that we when using families (such as `comp`), particularly in diagrams, have suppressed the first few arguments, in a style reminiscent of a polymorphic type theory. In some places, this notational sleight of hand has allowed the use of infix operators, rather than cumbersome functions of five or more arguments.

Current development on Agda, the proof assistant used, has introduced a form of hidden arguments for such purposes. Some limited experiments show that while this greatly improves matters, it does not do all one might hope for. Since it is probably only reasonable for a system to find *unique* solutions for hidden variables, there are some areas of particular weakness. For example, where a hidden variable is to be solved by an extensional function, the system can usually find the mapping part uniquely, but since we can not expect there to be a unique proof of extensionality to go with it, this hidden variable can not be solved for. Similar problems exist for E-natural transformations, and probably many other structures.

The usefulness of these hidden variables is probably most obvious when considering equational reasoning (and there is quite a lot of equational reasoning in this formalisation). For example, in a setoid, the proof object for transitivity is a function taking as arguments three objects `a`, `b`, and `c`; a proof of the equality of `a` and `b`; and a proof of the equality of `b` and `c`; producing a proof that `a` equals `c`. We might naïvely hope to be able to suppress the first three arguments. While some thought should convince us that the second argument can not be suppressed, it turns out that we can not generally suppress *any* of them. It is not even possible always to suppress the only argument of the reflexivity proof object! Even though it is possible to provide these hidden arguments where Agda is unable to derive them, the situations where this is necessary are sufficiently common in this formalisation for the gain to be comparatively small.

References

[1] P. Aczel. Galois: A theory development project. Available from http://www.cs.man.ac.uk/~petera/papers.html, June 1995.

[2] Agda homepage. http://www.cs.chalmers.se/~catarina/agda/.

[3] Alfa homepage. http://www.cs.chalmers.se/~hallgren/Alfa/.

[4] J. Bénabou. Introduction to bicategories In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, Berlin, 1967. Lecture Notes in Mathematics, Vol. 47.

[5] F. Borceux. *Handbook of categorical algebra. 1*, volume 50 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. Basic category theory.

[6] J. W. Gray. *Formal category theory: adjointness for 2-categories*. Springer-Verlag, Berlin, 1974. Lecture Notes in Mathematics, Vol. 391.

[7] G. Huet and A. Saïbi. Constructive category theory. In *Proof, language, and interaction*, Found. Comput. Ser., pages 239–275. MIT Press, Cambridge, MA, 2000.

[8] Y. Kinoshita. A bicategorical analysis of E-categories. *Math. Japon.*, 47(1):157–169, 1998.

[9] T. Leinster. Basic bicategories, Oct. 1998, arXiv:math.CT/9810017.

[10] T. Leinster. *Higher operads, higher categories*, volume 298 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 2004.

[11] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's type theory*, volume 7 of *International Series of Monographs on Computer Science*. The Clarendon Press Oxford University Press, New York, 1990. An introduction.

[12] G. O'Keefe. Towards a readable formalisation of category theory. In M. Atkinson, editor, *Proceedings of Computing: The Australasian Theory Symposium (CATS) 2004*, volume 91 of *ENTCS*, pages 212–228. Elsevier, 2004.

[13] K. O. Wilander. Complete formalisation on-line . http://www.math.uu.se/~wilander/bicat/. Also included as an appendix. The formalisation is written for the old stable 'IAgda' release and will not necessarily load into more recent versions.

K.O. Wilander, Department of Mathematics, Uppsala University, P.O. Box 480, SE-751 06 Uppsala

E-mail address: `wilander@math.uu.se`

`BasicTypeTheory.agda` (from E. Palmgren's library)

```
-- {\Large \bf Basic Type Theory}

--

--

Fam (A::Set) :: Type
  = A -> Set

-- {\em Syntactic sugar for the $\Pi$-construction}

Pi (A::Set)(B::Fam A) :: Set
  = (x::A) -> B x

ForAll (A::Set)(B::Fam A) :: Set
  = (x::A) -> B x

-- {\em Syntactic sugar relating to the $\Sigma$-construction}

Sigma (A::Set)(B::Fam A) :: Set
  = sig{_1 :: A;
        _2 :: B _1;}

Exists (A::Set)(B::Fam A) :: Set
  = sig{_1 :: A;
        _2 :: B _1;}

pair (A::Set)(B::Fam A)(a::A)(b::B a) :: Sigma A B
  = struct {
      _1 = a;
      _2 = b;}

pairExists (A::Set)(B::Fam A)(a::A)(b::B a) :: Exists A B
  = struct {
      _1 = a;
      _2 = b;}

split (A::Set)
      (B::Fam A)
      (C::Fam (Sigma A B))
      (c::Sigma A B)
      (d::(x::A) -> (y::B x) -> C (pair A B x y))
  :: C c
  = d c._1 c._2

splitExists (A::Set)
            (B::Fam A)
            (C::Fam (Exists A B))
            (c::Exists A B)
            (d::(x::A) -> (y::B x) -> C (pairExists A B x y))
  :: C c
  = d c._1 c._2

Cart (A::Set)(B::Set) :: Set
  = sig{_1 :: A;
        _2 :: B;}

pairCart (A::Set)(B::Set)(a::A)(b::B) :: Cart A B
  = struct {
      _1 = a;
      _2 = b;}

proj1 (A::Set)(B::Set)(c::Cart A B) :: A
  = c._1

proj2 (A::Set)(B::Set)(c::Cart A B) :: B
  = c._2

and (A::Set)(B::Set) :: Set
  = Cart A B

-- {\em Disjoint binary sums}

Sum (A::Set)(B::Set) :: Set
  = data inl (x::A) | inr (y::B)

or (A::Set)(B::Set) :: Set
  = Sum A B

when (A::Set)
     (B::Set)
     (C::Fam (Sum A B))
     (c::Sum A B)
     (d::(x::A) -> C (inl@_ x))
     (e::(y::B) -> C (inr@_ y))
  :: C c
  = case c of {
      (inl x) -> d x;
      (inr y) -> e y;}

-- {\em If and only if}

iff (A::Set)(B::Set) :: Set
  = Cart (A -> B) (B -> A)

-- {\em The empty set}

empty :: Set
  = data

Absurd :: Set
  = empty

elempty (A::Set)(x::empty) :: A
  = case x of { }

not (A::Set) :: Set
  = A -> Absurd

-- {\em A unit set}

Unit :: Set
  = data elt

True :: Set
  = Unit

-- {\em Booleans}

Bool :: Set
  = data ff | tt

-- {\em Natural numbers}

Nat :: Set
  = data zero | succ (x::Nat)

rec (C::(z::Nat) -> Set)
    (t::Nat)
    (f::C zero@_)
    (g::(x::Nat) -> (y::C x) -> C (succ@_ x))
  :: C t
  = case t of {
      (zero) -> f;
      (succ x) -> g x (rec C x f g);}
```

```
-- {\em A basic dependent type}

L (z::Nat) :: Set
  = case z of {
      (zero) -> empty;
      (succ x) -> Nat;}

-- {\em Deriving first projection from split}

pr1 (A::Set)(B::Fam A)(c::Sigma A B) :: A
  = split A B (\(h::Sigma A B) -> A) c (\(x::A) -> \(y::B x) -> x)

pr2 (A::Set)(B::Fam A)(c::Sigma A B) :: B (pr1 A B c)
  = split
      A
      B
      (\(h::Sigma A B) -> B (pr1 A B h))
      c
      (\(x::A) -> \(y::B x) -> y)
{-# Alfa unfoldgoals off
brief on
hidetypeannots off
wide

nd
hiding on
con "zero" as "0"
con "succ" as "S"
var "Nat" as "N" with symbolfont
var "Prod" infix 7 as "" with symbolfont
var "Sum" infix as "+"
var "Fun" infix rightassoc as "" with symbolfont
var "apply" hide 2
var "pair" hide 2 tuple
var "pr1" hide 2
var "pr2" hide 2
var "rec" hide 1
var "when" hide 3
var "empty" as "" with symbolfont
var "elempty" as "!"
var "Pi" as "P" with symbolfont
var "Sigma" as "S" with symbolfont
var "split" hide 3
var "Cart" infix as "" with symbolfont
con "sigma" as "s" with symbolfont
con "emb" as "e" with symbolfont
var "Power" as "Rw"
var "Member" hide 2
var "Subclass" hide 3 infix as "" with symbolfont
var "iff" infix as "" with symbolfont
var "Can" as "G" with symbolfont
var "ind"
var "fam"
var "R" as "R"
var "and" infix as "" with symbolfont
var "MemberSE" hide 3 infix as "e" with symbolfont
var "MemberE" hide 2 infix as "e" with symbolfont
var "EqualE" hide 2 infix as "=="
var "SubsetsE" hide 2 infix as "" with symbolfont
var "EqualsetsSE" hide 3 infix as "=="
var "SubsetSE" hide 3 infix as "" with symbolfont
var "E"
var "Es"
var "Rs" infix as "<" with symbolfont
var "Ex" infix as "<<" with symbolfont
var "Exs" infix as "<<" with symbolfont
var "proj1" hide 2
var "proj2" hide 2
var "ForAll" hide 1 quantifier domain on as "\"" with symbolfont
var "Exists" hide 1 quantifier domain on as "$" with symbolfont
con "cross" infix as "" with symbolfont
var "or" infix as "" with symbolfont
var "not" as "" with symbolfont
var "Absurd" as "^" with symbolfont
var "pairExists" hide 2 tuple
var "pairCart" hide 2 tuple
 #-}
```

SE.agda (from E. Palmgren's library)

```
--#include "BasicTypeTheory.agda"

-- Sets with equality and substitution properties

--

-- {\em The type of equivalence relations on X}

--

EQ (X::Set) :: Type
  = sig{eq :: X -> X -> Set;
        ref :: (x::X) -> eq x x;
        sym :: (x::X) -> (y::X) -> eq x y -> eq y x;
        tra :: (x::X) -> (y::X) -> (z::X) -> eq x y -> eq y z -> eq x z;}

-- The type of sets with equality

-- "base" is the underlying base set with no special equality

-- assumed.

-- "er" is an equivalence relation on base

--

SE :: Type
  = sig{base :: Set;
        er :: EQ base;}

Equal (A::SE)(x::A.base)(y::A.base) :: Set
  = A.er.eq x y

reflexive (A::SE)(x::A.base) :: Equal A x x
  = A.er.ref x

symmetric (A::SE)(x::A.base)(y::A.base)(pf::Equal A x y)
  :: Equal A y x
  = A.er.sym x y pf

transitive (A::SE)
           (x::A.base)
           (y::A.base)
           (z::A.base)
           (pf1::Equal A x y)
           (pf2::Equal A y z)
  :: Equal A x z
  = A.er.tra x y z pf1 pf2

transitive3 (A::SE)
            (x::A.base)
            (y::A.base)
            (z::A.base)
            (u::A.base)
```

```
            (pf1::Equal A x y)
            (pf2::Equal A y z)
            (pf3::Equal A z u)
  :: Equal A x u
  = A.er.tra x y u pf1 (A.er.tra y z u pf2 pf3)

--

--  A set with equality is substitutive if

--  the substituition rule holds for it.

--  In Agda we have to derive this property

--  even for natural numbers and finite sets.

--

SubstitutiveRel (X::Set)(R::X -> X -> Set) :: Type
  = (C::Fam X) -> (a::X) -> (b::X) -> (p::R a b) -> (q::C a) -> C b

Substitutive (D::SE) :: Type
  = (C::Fam D.base) ->
    (a::D.base) ->
    (b::D.base) ->
    (p::D.er.eq a b) ->
    (q::C a) ->
    C b

subst_and_refl_is_equiv (X::Set)
                        (R::X -> X -> Set)
                        (sub::SubstitutiveRel X R)
                        (refl::(x::X) -> R x x)
  :: EQ X
  = struct {
      eq = R;
      ref = refl;
      sym =
        \(x::X) ->
        \(y::X) ->
        \(h::eq x y) ->
        sub (\(z::X) -> R z x) x y h (refl x);
      tra =
        \(x::X) ->
        \(y::X) ->
        \(z::X) ->
        \(h::eq x y) ->
        \(h'::eq y z) ->
        sub (\(u::X) -> R x u) y z h' h;}

-- Restrict an SE to a subset

restrictSE (A::SE)(P::Fam A.base) :: SE
  = struct {
      base =
        sig{el :: A.base;
            insubset :: P el;};
      er =
        struct {
          eq = \(x::base) -> \(y::base) -> A.er.eq x.el y.el;
          ref = \(x::base) -> A.er.ref x.el;
          sym = \(x::base) -> \(y::base) -> \(h::eq x y) -> A.er.sym x.el y.el h;
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            A.er.tra x.el y.el z.el h h';};}
```



```
--

-- Unit SE and empty SE

--

UNIT :: SE
  = struct {
      base = Unit;
      er =
        struct {
          eq = \(h::base) -> \(h'::base) -> True;
          ref = \(x::base) -> elt@_;
          sym = \(x::base) -> \(y::base) -> \(h::eq x y) -> elt@_;
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            elt@_;};}

substUNIT :: Substitutive UNIT
  = \(C::Fam UNIT.base) ->
    \(a::UNIT.base) ->
    \(b::UNIT.base) ->
    \(p::UNIT.er.eq a b) ->
    \(q::C a) ->
    case a of { (elt) -> case b of { (elt) -> q;};}

EMPTY :: SE
  = struct {
      base = empty;
      er =
        struct {
          eq = \(h::base) -> \(h'::base) -> True;
          ref = \(x::base) -> elt@_;
          sym = \(x::base) -> \(y::base) -> \(h::eq x y) -> elt@_;
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            elt@_;};}

substEMPTY :: Substitutive EMPTY
  = \(C::Fam EMPTY.base) ->
    \(a::EMPTY.base) ->
    \(b::EMPTY.base) ->
    \(p::EMPTY.er.eq a b) ->
    \(q::C a) ->
    elempty (C b) a

--

-- Equal boolean values

--

equalBool (x::Bool)(y::Bool) :: Set
  = case x of {
      (ff) ->
        case y of {
          (ff) -> True;
          (tt) -> Absurd;};
      (tt) ->
        case y of {
```

```
          (ff) -> Absurd;
          (tt) -> True;};}

substBool (C::Fam Bool)(a::Bool)(b::Bool)(p::equalBool a b)(q::C a)
  :: C b
  = case a of {
      (ff) ->
        case b of {
          (ff) -> q;
          (tt) -> elempty (C tt@_) p;};
      (tt) ->
        case b of {
          (ff) -> elempty (C ff@_) p;
          (tt) -> q;};}

BOOL :: SE
  = struct {
      base = Bool;
      er =
        struct {
          eq = equalBool;
          ref =
            \(x::base) ->
            case x of {
              (ff) -> elt@_;
              (tt) -> elt@_;};
          sym =
            \(x::base) ->
            \(y::base) ->
            \(h::eq x y) ->
            case x of {
              (ff) ->
                case y of {
                  (ff) -> h;
                  (tt) -> h;};
              (tt) ->
                case y of {
                  (ff) -> h;
                  (tt) -> h;};};
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            case x of {
              (ff) ->
                case y of {
                  (ff) -> h';
                  (tt) -> elempty (eq ff@_ z) h;};
              (tt) ->
                case y of {
                  (ff) -> elempty (eq tt@_ z) h;
                  (tt) -> h';};};};}

substB :: Substitutive BOOL
  = substBool

-- Cartesian product of set with equality

() (A::SE)(B::SE) :: SE
  = struct {
      base = Cart A.base B.base;
      er =
        struct {
          eq =
            \(w::base) ->
            \(z::base) ->
            and (A.er.eq w._1 z._1) (B.er.eq w._2 z._2);
```

```
          ref =
            \(x::base) ->
            struct {
              _1 = A.er.ref x._1;
              _2 = B.er.ref x._2;};
          sym =
            \(x::base) ->
            \(y::base) ->
            \(h::eq x y) ->
            struct {
              _1 = A.er.sym x._1 y._1 h._1;
              _2 = B.er.sym x._2 y._2 h._2;};
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            struct {
              _1 = A.er.tra x._1 y._1 z._1 h._1 h'._1;
              _2 = B.er.tra x._2 y._2 z._2 h._2 h'._2;};};}

substCART (A::SE)(B::SE)(s::Substitutive A)(t::Substitutive B)
  :: Substitutive (A  B)
  = \(C::Fam (A  B).base) ->
    \(a::(A  B).base) ->
    \(b::(A  B).base) ->
    \(p::(A  B).er.eq a b) ->
    \(q::C a) ->
    let D (x::A.base) :: Set
          = (y::B.base) ->
            (v::B.base) ->
            C
              (struct {
                 _1 = x;
                 _2 = y;}) ->
            B.er.eq y v ->
            C
              (struct {
                 _1 = x;
                 _2 = v;})
    in  let lemma1 :: D a._1
              = \(y::B.base) ->
                \(v::B.base) ->
                \(h::C
                       (struct {
                          _1 = a._1;
                          _2 = y;})) ->
                \(h'::B.er.eq y v) ->
                t
                  (\(h0::B.base) ->
                   C
                     (struct {
                        _1 = a._1;
                        _2 = h0;}))
                  y
                  v
                  h'
                  h
        in  let lemma2 :: A.er.eq a._1 b._1 -> D a._1 -> D b._1
                  = s D a._1 b._1
            in  let lemma3 :: D b._1
                      = lemma2 p._1 lemma1
                in  lemma3
                      a._2
                      b._2
                      (s
                         (\(h::A.base) ->
                          C
```

```
                            (struct {
                               _1 = h;
                               _2 = a._2;}))
                         a._1
                         b._1
                         p._1
                         q)
                      p._2

DisjSum (A::SE)(B::SE) :: SE
  = struct {
      base = Sum A.base B.base;
      er =
        struct {
          eq =
            \(h::base) ->
            \(h'::base) ->
            case h of {
              (inl x) ->
                case h' of {
                  (inl x') -> A.er.eq x x';
                  (inr y) -> Absurd;};
              (inr y) ->
                case h' of {
                  (inl x) -> Absurd;
                  (inr y') -> B.er.eq y y';};};
          ref =
            \(x::base) ->
            case x of {
              (inl x') -> A.er.ref x';
              (inr y) -> B.er.ref y;};
          sym =
            \(x::base) ->
            \(y::base) ->
            \(h::eq x y) ->
            case x of {
              (inl x') ->
                case y of {
                  (inl x0) -> A.er.sym x' x0 h;
                  (inr y') -> h;};
              (inr y') ->
                case y of {
                  (inl x') -> h;
                  (inr y0) -> B.er.sym y' y0 h;};};
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            case x of {
              (inl x') ->
                case y of {
                  (inl x0) ->
                    case z of {
                      (inl x1) -> A.er.tra x' x0 x1 h h';
                      (inr y') -> h';};
                  (inr y') -> elempty (eq (inl@_ x') z) h;};
              (inr y') ->
                case y of {
                  (inl x') -> elempty (eq (inr@_ y') z) h;
                  (inr y0) ->
                    case z of {
                      (inl x') -> h';
                      (inr y1) -> B.er.tra y' y0 y1 h h';};};};};}

substDisjSum (A::SE)(B::SE)(s::Substitutive A)(t::Substitutive B)
  :: Substitutive (DisjSum A B)
  = \(C::Fam (DisjSum A B).base) ->
    \(a::(DisjSum A B).base) ->
    \(b::(DisjSum A B).base) ->
    \(p::(DisjSum A B).er.eq a b) ->
    \(q::C a) ->
    case a of {
      (inl x) ->
        case b of {
          (inl x') -> s (\(h::A.base) -> C (inl@_ h)) x x' p q;
          (inr y) -> elempty (C (inr@_ y)) p;};
      (inr y) ->
        case b of {
          (inl x) -> elempty (C (inl@_ x)) p;
          (inr y') -> t (\(h::B.base) -> C (inr@_ h)) y y' p q;};}

--

-- The set of functions from A to B which respects

-- equality

--

(==>) (A::SE)(B::SE) :: Set
  = sig{op :: A.base -> B.base;
        ext ::
          (x::A.base) -> (y::A.base) -> A.er.eq x y -> B.er.eq (op x) (op y);}

idfunc (A::SE) :: A.base -> A.base
  = \(h::A.base) -> h

idSE (A::SE) :: (A ==> A)
  = struct {
      op = idfunc A;
      ext = \(x::A.base) -> \(y::A.base) -> \(h::A.er.eq x y) -> h;}

oSE (A::SE)(B::SE)(C::SE)(f::(B ==> C))(g::(A ==> B)) :: (A ==> C)
  = struct {
      op = \(x::A.base) -> f.op (g.op x);
      ext =
        \(x::A.base) ->
        \(y::A.base) ->
        \(h::A.er.eq x y) ->
        f.ext (g.op x) (g.op y) (g.ext x y h);}

HomSE (A::SE)(B::SE) :: SE
  = struct {
      base = (A ==> B);
      er =
        struct {
          eq =
            \(f::base) -> \(g::base) -> (x::A.base) -> B.er.eq (f.op x) (g.op x);
          ref = \(f::base) -> \(x::A.base) -> B.er.ref (f.op x);
          sym =
            \(f::base) ->
            \(g::base) ->
            \(p::eq f g) ->
            \(x::A.base) ->
            B.er.sym (f.op x) (g.op x) (p x);
          tra =
            \(f::base) ->
            \(g::base) ->
            \(h::base) ->
            \(p::eq f g) ->
            \(q::eq g h) ->
            \(x::A.base) ->
            B.er.tra (f.op x) (g.op x) (h.op x) (p x) (q x);};}

subst_domain_ext (A::SE)
                 (B::SE)
```

```
                 (p::Substitutive A)
                 (f::A.base -> B.base)
  :: (A ==> B)
  = struct {
      op = f;
      ext =
        \(x::A.base) ->
        \(y::A.base) ->
        \(h::A.er.eq x y) ->
        p (\(z::A.base) -> B.er.eq (op x) (op z)) x y h (B.er.ref (op x));}
{-# Alfa unfoldgoals off
brief on
hidetypeannots off
wide

nd
hiding on
var "ExistsUnique" quantifier domain on as "$!" with symbolfont
var "Fix" hide 2
var "Nodeq" infix as "=="
var "equalNsym" hide 2
var "equalNtra" hide 3
var "AddNcongl" hide 3
var "AddNcongr" hide 3
var "equalN" infix as "==_N"
var "equalBool" infix as "==_B"
var "True_of_State" mixfix as "_ , _ |= _"
var "True_of_Path" mixfix as "_ , _ |= _"
con "imp" infix as "->"
con "and" infix as "" with symbolfont
con "or" infix as "" with symbolfont
con "var" mixfix as "(_)"
var "DisjSum" infix as " " with symbolfont
var "oSE" hide 3 infix 9 as "o"
var "Equal" distfix3b as "==="
var "reflexive" hide 1
var "symmetric" hide 3
var "transitive" hide 4
var "transitive3" hide 5
 #-}
```

E_categories.agda (from E. Palmgren's library)

```
--#include "SE.agda"

E_category :: Type
  = sig{obj :: Set;
        hom :: (a::obj) -> (b::obj) -> SE;
        Hom (a::obj)(b::obj) :: Set
          = (hom a b).base;
        id :: (a::obj) -> Hom a a;
        comp ::
          (a::obj) -> (b::obj) -> (c::obj) -> Hom b c -> Hom a b -> Hom a c;
        left_unit ::
          (a::obj) ->
          (b::obj) ->
          (f::Hom a b) ->
          Equal (hom a b) (comp a b b (id b) f) f;
        right_unit ::
          (a::obj) ->
          (b::obj) ->
          (f::Hom a b) ->
          Equal (hom a b) (comp a a b f (id a)) f;
        assoc ::
          (a::obj) ->
          (b::obj) ->
          (c::obj) ->
          (d::obj) ->
          (f::Hom c d) ->
          (g::Hom b c) ->
          (h::Hom a b) ->
          Equal
            (hom a d)
            (comp a b d (comp b c d f g) h)
            (comp a c d f (comp a b c g h));
        cong ::
          (a::obj) ->
          (b::obj) ->
          (c::obj) ->
          (f::Hom b c) ->
          (f'::Hom b c) ->
          (g::Hom a b) ->
          (g'::Hom a b) ->
          Equal (hom b c) f f' ->
          Equal (hom a b) g g' ->
          Equal (hom a c) (comp a b c f g) (comp a b c f' g');}

Hom (C::E_category)(a::C.obj)(b::C.obj) :: Set
  = (C.hom a b).base

compose (C::E_category)
        (a::C.obj)
        (b::C.obj)
        (c::C.obj)
        (f::Hom C b c)
        (g::Hom C a b)
  :: Hom C a c
  = C.comp a b c f g

congruence (C::E_category)
           (a::C.obj)
           (b::C.obj)
           (c::C.obj)
           (f::(C.hom b c).base)
           (f'::(C.hom b c).base)
           (g::(C.hom a b).base)
           (g'::(C.hom a b).base)
           (p1::Equal (C.hom b c) f f')
           (p2::Equal (C.hom a b) g g')
  :: Equal (C.hom a c) (C.comp a b c f g) (C.comp a b c f' g')
  = C.cong a b c f f' g g' p1 p2

associative (C::E_category)
            (a::C.obj)
            (b::C.obj)
            (c::C.obj)
            (d::C.obj)
            (f::(C.hom c d).base)
            (g::(C.hom b c).base)
            (h::(C.hom a b).base)
  :: Equal
       (C.hom a d)
       (C.comp a b d (C.comp b c d f g) h)
       (C.comp a c d f (C.comp a b c g h))
  = C.assoc a b c d f g h

Triangle (C::E_category)(a::C.obj)(b::C.obj)(c::C.obj) :: Set
  = sig{top :: Hom C a b;
        left :: Hom C a c;
        right :: Hom C b c;}

CommutesTri (C::E_category)
            (a::C.obj)
            (b::C.obj)
            (c::C.obj)
```

```
            (tri::Triangle C a b c)
  :: Set
  = Equal (C.hom a c) tri.left (compose C a b c tri.right tri.top)

InverseArrows (C::E_category)
              (a::C.obj)
              (b::C.obj)
              (f::Hom C a b)
              (g::Hom C b a)
  :: Set
  = and
      (Equal (C.hom b b) (compose C b a b f g) (C.id b))
      (Equal (C.hom a a) (compose C a b a g f) (C.id a))

ComposeTri (C::E_category)
           (a::C.obj)
           (b::C.obj)
           (c::C.obj)
           (d::C.obj)
           (tri1::Triangle C a b d)
           (tri2::Triangle C b c d)
           (c1::CommutesTri C a b d tri1)
           (c2::CommutesTri C b c d tri2)
           (e::Equal (C.hom b d) tri1.right tri2.left)
  :: CommutesTri
       C
       a
       c
       d
       (struct {
          top = compose C a b c tri2.top tri1.top;
          left = tri1.left;
          right = tri2.right;})
  = transitive
      (C.hom a d)
      tri1.left
      (compose C a b d (compose C b c d tri2.right tri2.top) tri1.top)
      (compose C a c d tri2.right (compose C a b c tri2.top tri1.top))
      (transitive
         (C.hom a d)
         tri1.left
         (compose C a b d tri1.right tri1.top)
         (compose C a b d (compose C b c d tri2.right tri2.top) tri1.top)
         c1
         (congruence
            C
            a
            b
            d
            tri1.right
            (compose C b c d tri2.right tri2.top)
            tri1.top
            tri1.top
            (transitive
               (C.hom b d)
               tri1.right
               tri2.left
               (compose C b c d tri2.right tri2.top)
               e
               c2)
            (reflexive (C.hom a b) tri1.top)))
      (C.assoc a b c d tri2.right tri2.top tri1.top)

Mono (C::E_category)(a::C.obj)(b::C.obj)(f::Hom C a b) :: Set
  = (x::C.obj) ->
    (g::Hom C x a) ->
    (h::Hom C x a) ->
    Equal (C.hom x b) (compose C x a b f g) (compose C x a b f h) ->
    Equal (C.hom x a) g h
```



```
Monos_compose (C::E_category)
              (a::C.obj)
              (b::C.obj)
              (c::C.obj)
              (f::Hom C a b)
              (g::Hom C b c)
              (m1::Mono C a b f)
              (m2::Mono C b c g)
  :: Mono C a c (compose C a b c g f)
  = \(x::C.obj) ->
    \(g'::Hom C x a) ->
    \(h'::Hom C x a) ->
    \(p::Equal
           (C.hom x c)
           (compose C x a c (compose C a b c g f) g')
           (compose C x a c (compose C a b c g f) h')) ->
    m1
      x
      g'
      h'
      (m2
         x
         (compose C x a b f g')
         (compose C x a b f h')
         (transitive
            (C.hom x c)
            (compose C x b c g (compose C x a b f g'))
            (compose C x a c (compose C a b c g f) h')
            (compose C x b c g (compose C x a b f h'))
            (transitive
               (C.hom x c)
               (compose C x b c g (compose C x a b f g'))
               (compose C x a c (compose C a b c g f) g')
               (compose C x a c (compose C a b c g f) h')
               (symmetric
                  (C.hom x c)
                  (compose C x a c (compose C a b c g f) g')
                  (compose C x b c g (compose C x a b f g'))
                  (associative C x a b c g f g'))
               p)
            (associative C x a b c g f h')))

Mon (C::E_category)(x::C.obj) :: Set
  = sig{dom :: C.obj;
        arr :: Hom C dom x;
        ismono :: Mono C dom x arr;}

mono_incl (C::E_category)(x::C.obj)(M::Mon C x)(N::Mon C x) :: Set
  = sig{mediator :: Hom C M.dom N.dom;
        commutes ::
          CommutesTri
            C
            M.dom
            N.dom
            x
            (struct {
               top = mediator;
               left = M.arr;
               right = N.arr;});}

mono_incl_ref (C::E_category)(x::C.obj)(M::Mon C x)
  :: mono_incl C x M M
  = struct {
      mediator = C.id M.dom;
      commutes =
        symmetric
          (C.hom M.dom x)
          (compose C M.dom M.dom x M.arr mediator)
```

```
          M.arr
          (C.right_unit M.dom x M.arr);}

mono_incl_tra (C::E_category)
              (x::C.obj)
              (M::Mon C x)
              (N::Mon C x)
              (P::Mon C x)
              (p1::mono_incl C x M N)
              (p2::mono_incl C x N P)
  :: mono_incl C x M P
  = struct {
      mediator = compose C M.dom N.dom P.dom p2.mediator p1.mediator;
      commutes =
        ComposeTri
          C
          M.dom
          N.dom
          P.dom
          x
          (struct {
             top = p1.mediator;
             left = M.arr;
             right = N.arr;})
          (struct {
             top = p2.mediator;
             left = N.arr;
             right = P.arr;})
          p1.commutes
          p2.commutes
          (reflexive (C.hom N.dom x) N.arr);}

mono_equality (C::E_category)(x::C.obj)(M::Mon C x)(N::Mon C x) :: Set
  = and (mono_incl C x M N) (mono_incl C x N M)

incl_mediator_is_mono (C::E_category)
                      (x::C.obj)
                      (M::Mon C x)
                      (N::Mon C x)
                      (p::mono_incl C x M N)
  :: Mono C M.dom N.dom p.mediator
  = \(x'::C.obj) ->
    \(g::Hom C x' M.dom) ->
    \(h::Hom C x' M.dom) ->
    \(h'::Equal
            (C.hom x' N.dom)
            (compose C x' M.dom N.dom p.mediator g)
            (compose C x' M.dom N.dom p.mediator h)) ->
    M.ismono
      x'
      g
      h
      (transitive
         (C.hom x' x)
         (compose C x' M.dom x M.arr g)
         (compose C x' M.dom x (compose C M.dom N.dom x N.arr p.mediator) g)
         (compose C x' M.dom x M.arr h)
         (congruence
            C
            x'
            M.dom
            x
            M.arr
            (compose C M.dom N.dom x N.arr p.mediator)
            g
            g
            p.commutes
            (reflexive (C.hom x' M.dom) g))
         (transitive
            (C.hom x' x)
            (compose C x' M.dom x (compose C M.dom N.dom x N.arr p.mediator) g)
            (compose C x' M.dom x (compose C M.dom N.dom x N.arr p.mediator) h)
            (compose C x' M.dom x M.arr h)
            (transitive
               (C.hom x' x)
               (compose
                  C
                  x'
                  M.dom
                  x
                  (compose C M.dom N.dom x N.arr p.mediator)
                  g)
               (compose
                  C
                  x'
                  N.dom
                  x
                  N.arr
                  (compose C x' M.dom N.dom p.mediator g))
               (compose
                  C
                  x'
                  M.dom
                  x
                  (compose C M.dom N.dom x N.arr p.mediator)
                  h)
               (associative C x' M.dom N.dom x N.arr p.mediator g)
               (transitive
                  (C.hom x' x)
                  (compose
                     C
                     x'
                     N.dom
                     x
                     N.arr
                     (compose C x' M.dom N.dom p.mediator g))
                  (compose
                     C
                     x'
                     N.dom
                     x
                     N.arr
                     (compose C x' M.dom N.dom p.mediator h))
                  (compose
                     C
                     x'
                     M.dom
                     x
                     (compose C M.dom N.dom x N.arr p.mediator)
                     h)
                  (congruence
                     C
                     x'
                     N.dom
                     x
                     N.arr
                     N.arr
                     (compose C x' M.dom N.dom p.mediator g)
                     (compose C x' M.dom N.dom p.mediator h)
                     (reflexive (C.hom N.dom x) N.arr)
                     h')
                  (symmetric
                     (C.hom x' x)
                     (compose
                        C
                        x'
                        M.dom
                        x
```

```
                        (compose C M.dom N.dom x N.arr p.mediator)
                        h)
                     (compose
                        C
                        x’
                        N.dom
                        x
                        N.arr
                        (compose C x’ M.dom N.dom p.mediator h))
                     (associative C x’ M.dom N.dom x N.arr p.mediator h))))
            (symmetric
               (C.hom x’ x)
               (compose C x’ M.dom x M.arr h)
               (compose
                  C
                  x’
                  M.dom
                  x
                  (compose C M.dom N.dom x N.arr p.mediator)
                  h)
               (congruence
                  C
                  x’
                  M.dom
                  x
                  M.arr
                  (compose C M.dom N.dom x N.arr p.mediator)
                  h
                  h
                  p.commutes
                  (reflexive (C.hom x’ M.dom) h)))))

incl_mediator_is_unique (C::E_category)
                        (x::C.obj)
                        (M::Mon C x)
                        (N::Mon C x)
                        (p::mono_incl C x M N)
                        (q::mono_incl C x M N)
  :: Equal (C.hom M.dom N.dom) p.mediator q.mediator
  = N.ismono
      M.dom
      p.mediator
      q.mediator
      (transitive
         (C.hom M.dom x)
         (compose C M.dom N.dom x N.arr p.mediator)
         M.arr
         (compose C M.dom N.dom x N.arr q.mediator)
         (symmetric
            (C.hom M.dom x)
            M.arr
            (compose C M.dom N.dom x N.arr p.mediator)
            p.commutes)
         q.commutes)

Equal_subobjects_are_isomorphic (C::E_category)
                                (x::C.obj)
                                (M::Mon C x)
                                (N::Mon C x)
                                (p::mono_equality C x M N)
  :: InverseArrows C M.dom N.dom p._1.mediator p._2.mediator
  = struct {
      _1 =
        incl_mediator_is_unique
          C
          x
          N
          N
          (mono_incl_tra C x N M N p._2 p._1)
          (mono_incl_ref C x N);
      _2 =
        incl_mediator_is_unique
          C
          x
          M
          M
          (mono_incl_tra C x M N M p._1 p._2)
          (mono_incl_ref C x M);}

Subobj (C::E_category)(x::C.obj) :: SE
  = struct {
      base = Mon C x;
      er =
        struct {
          eq = \(M::base) -> \(N::base) -> mono_equality C x M N;
          ref =
            \(M::base) ->
            struct {
              _1 = mono_incl_ref C x M;
              _2 = mono_incl_ref C x M;};
          sym =
            \(M::base) ->
            \(N::base) ->
            \(p::eq M N) ->
            struct {
              _1 = p._2;
              _2 = p._1;};
          tra =
            \(M::base) ->
            \(N::base) ->
            \(P::base) ->
            \(p1::eq M N) ->
            \(p2::eq N P) ->
            struct {
              _1 = mono_incl_tra C x M N P p1._1 p2._1;
              _2 = mono_incl_tra C x P N M p2._2 p1._2;};};}

Terminal (C::E_category)(t::C.obj) :: Set
  = sig{construction :: (a::C.obj) -> Hom C a t;
        uniqueness ::
          (a::C.obj) ->
          (f::Hom C a t) ->
          (g::Hom C a t) ->
          Equal (C.hom a t) f g;}

Pdiagram (C::E_category)(a::C.obj)(p::C.obj)(b::C.obj) :: Set
  = sig{pr1 :: Hom C p a;
        pr2 :: Hom C p b;}

Product (C::E_category)
        (a::C.obj)
        (p::C.obj)
        (b::C.obj)
        (diag::Pdiagram C a p b)
  :: Set
  = sig{construction ::
          (q::C.obj) -> (diag2::Pdiagram C a q b) -> Hom C q p;
        universal ::
          (q::C.obj) ->
          (diag2::Pdiagram C a q b) ->
          and
            (Equal
               (C.hom q a)
               (compose C q p a diag.pr1 (construction q diag2))
               diag2.pr1)
            (Equal
               (C.hom q b)
               (compose C q p b diag.pr2 (construction q diag2))
```

```
               diag2.pr2);
        unique ::
          (q::C.obj) ->
          (diag2::Pdiagram C a q b) ->
          (f::Hom C q p) ->
          (f'::Hom C q p) ->
          (pf1::and
                  (Equal (C.hom q a) (compose C q p a diag.pr1 f) diag2.pr1)
                  (Equal (C.hom q b) (compose C q p b diag.pr2 f) diag2.pr2)) ->
          (pf2::and
                  (Equal (C.hom q a) (compose C q p a diag.pr1 f') diag2.pr1)
                  (Equal (C.hom q b) (compose C q p b diag.pr2 f') diag2.pr2)) ->
          Equal (C.hom q p) f f';}

Square (C::E_category)(a::C.obj)(b::C.obj)(c::C.obj)(d::C.obj) :: Set
  = sig{left :: Hom C a c;
        right :: Hom C b d;
        upper :: Hom C a b;
        lower :: Hom C c d;}

Commutes (C::E_category)
         (a::C.obj)
         (b::C.obj)
         (c::C.obj)
         (d::C.obj)
         (sq::Square C a b c d)
  :: Set
  = Equal
      (C.hom a d)
      (compose C a c d sq.lower sq.left)
      (compose C a b d sq.right sq.upper)

Commutes2Triangles (C::E_category)
                   (x::C.obj)
                   (a::C.obj)
                   (ab::C.obj)
                   (b::C.obj)
                   (f::Hom C x a)
                   (fg::Hom C x ab)
                   (g::Hom C x b)
                   (p1::Hom C ab a)
                   (p2::Hom C ab b)
  :: Set
  = and
      (Equal (C.hom x a) (compose C x ab a p1 fg) f)
      (Equal (C.hom x b) (compose C x ab b p2 fg) g)

Pullback (C::E_category)
         (a::C.obj)
         (b::C.obj)
         (c::C.obj)
         (d::C.obj)
         (sq::Square C a b c d)
  :: Set
  = sig{commutes :: Commutes C a b c d sq;
        construction ::
          (x::C.obj) ->
          (f::Hom C x b) ->
          (g::Hom C x c) ->
          (pf::Commutes
                 C
                 x
                 b
                 c
                 d
                 (struct {
                    left = g;
                    right = sq.right;
                    upper = f;
                    lower = sq.lower;})) ->
          Hom C x a;
        universal ::
          (x::C.obj) ->
          (f::Hom C x b) ->
          (g::Hom C x c) ->
          (pf::Commutes
                 C
                 x
                 b
                 c
                 d
                 (struct {
                    left = g;
                    right = sq.right;
                    upper = f;
                    lower = sq.lower;})) ->
          Commutes2Triangles
            C
            x
            b
            a
            c
            f
            (construction x f g pf)
            g
            sq.upper
            sq.left;
        unique ::
          (x::C.obj) ->
          (f::Hom C x b) ->
          (g::Hom C x c) ->
          (pf::Commutes
                 C
                 x
                 b
                 c
                 d
                 (struct {
                    left = g;
                    right = sq.right;
                    upper = f;
                    lower = sq.lower;})) ->
          (fg::Hom C x a) ->
          (fg'::Hom C x a) ->
          (cpf::Commutes2Triangles C x b a c f fg g sq.upper sq.left) ->
          (cpf'::Commutes2Triangles C x b a c f fg' g sq.upper sq.left) ->
          Equal (C.hom x a) fg fg';}

pullbacks_preserve_monos (C::E_category)
                         (a::C.obj)
                         (b::C.obj)
                         (c::C.obj)
                         (d::C.obj)
                         (sq::Square C a b c d)
                         (ispb::Pullback C a b c d sq)
                         (ismono::Mono C c d sq.lower)
  :: Mono C a b sq.upper
  = \(x::C.obj) ->
    \(g::Hom C x a) ->
    \(h::Hom C x a) ->
    \(pf::Equal
            (C.hom x b)
            (compose C x a b sq.upper g)
            (compose C x a b sq.upper h)) ->
    let mutual eq1
             :: Equal
                  (C.hom x d)
                  (compose C x b d sq.right (compose C x a b sq.upper g))
```

```
       (compose C x b d sq.right (compose C x a b sq.upper h))
  = congruence
      C
      x
      b
      d
      sq.right
      sq.right
      (compose C x a b sq.upper g)
      (compose C x a b sq.upper h)
      (reflexive (C.hom b d) sq.right)
      pf
eq3
  :: Equal
       (C.hom x c)
       (compose C x a c sq.left g)
       (compose C x a c sq.left h)
  = ismono
      x
      (compose C x a c sq.left g)
      (compose C x a c sq.left h)
      (transitive
         (C.hom x d)
         (compose C x c d sq.lower (compose C x a c sq.left g))
         (compose C x a d (compose C a c d sq.lower sq.left) h)
         (compose C x c d sq.lower (compose C x a c sq.left h))
         (transitive
            (C.hom x d)
            (compose C x c d sq.lower (compose C x a c sq.left g))
            (compose
               C
               x
               a
               d
               (compose C a b d sq.right sq.upper)
               h)
            (compose C x a d (compose C a c d sq.lower sq.left) h)
            (transitive
               (C.hom x d)
               (compose
                  C
                  x
                  c
                  d
                  sq.lower
                  (compose C x a c sq.left g))
               (compose
                  C
                  x
                  b
                  d
                  sq.right
                  (compose C x a b sq.upper h))
               (compose
                  C
                  x
                  a
                  d
                  (compose C a b d sq.right sq.upper)
                  h)
               (transitive
                  (C.hom x d)
                  (compose
                     C
                     x
                     c
                     d
                     sq.lower
                     (compose C x a c sq.left g))
                  (compose
                     C
                     x
                     b
                     d
                     sq.right
                     (compose C x a b sq.upper g))
                  (compose
                     C
                     x
                     b
                     d
                     sq.right
                     (compose C x a b sq.upper h))
                  (transitive
                     (C.hom x d)
                     (compose
                        C
                        x
                        c
                        d
                        sq.lower
                        (compose C x a c sq.left g))
                     (compose
                        C
                        x
                        a
                        d
                        (compose C a c d sq.lower sq.left)
                        g)
                     (compose
                        C
                        x
                        b
                        d
                        sq.right
                        (compose C x a b sq.upper g))
                     (symmetric
                        (C.hom x d)
                        (compose
                           C
                           x
                           a
                           d
                           (compose C a c d sq.lower sq.left)
                           g)
                        (compose
                           C
                           x
                           c
                           d
                           sq.lower
                           (compose C x a c sq.left g))
                        (associative C x a c d sq.lower sq.left g))
                     (transitive
                        (C.hom x d)
                        (compose
                           C
                           x
                           a
                           d
                           (compose C a c d sq.lower sq.left)
                           g)
                        (compose
                           C
                           x
                           a
                           d
                           (compose C a b d sq.right sq.upper)
```

```
                                      g)
                                   (compose
                                      C
                                      x
                                      b
                                      d
                                      sq.right
                                      (compose C x a b sq.upper g))
                                   (congruence
                                      C
                                      x
                                      a
                                      d
                                      (compose C a c d sq.lower sq.left)
                                      (compose C a b d sq.right sq.upper)
                                      g
                                      g
                                      ispb.commutes
                                      (reflexive (C.hom x a) g))
                                   (associative
                                      C
                                      x
                                      a
                                      b
                                      d
                                      sq.right
                                      sq.upper
                                      g)))
                             eq1)
                          (symmetric
                             (C.hom x d)
                             (compose
                                C
                                x
                                a
                                d
                                (compose C a b d sq.right sq.upper)
                                h)
                             (compose
                                C
                                x
                                b
                                d
                                sq.right
                                (compose C x a b sq.upper h))
                             (associative C x a b d sq.right sq.upper h)))
                       (congruence
                          C
                          x
                          a
                          d
                          (compose C a b d sq.right sq.upper)
                          (compose C a c d sq.lower sq.left)
                          h
                          h
                          (symmetric
                             (C.hom a d)
                             (compose C a c d sq.lower sq.left)
                             (compose C a b d sq.right sq.upper)
                             ispb.commutes)
                          (reflexive (C.hom x a) h)))
                    (associative C x a c d sq.lower sq.left h))
in  ispb.unique
      x
      (compose C x a b sq.upper g)
      (compose C x a c sq.left g)
      (transitive
         (C.hom x d)
         (compose C x c d sq.lower (compose C x a c sq.left g))
```

```
             (compose C x a d (compose C a b d sq.right sq.upper) g)
             (compose C x b d sq.right (compose C x a b sq.upper g))
             (transitive
                (C.hom x d)
                (compose C x c d sq.lower (compose C x a c sq.left g))
                (compose C x a d (compose C a c d sq.lower sq.left) g)
                (compose C x a d (compose C a b d sq.right sq.upper) g)
                (symmetric
                   (C.hom x d)
                   (compose C x a d (compose C a c d sq.lower sq.left) g)
                   (compose C x c d sq.lower (compose C x a c sq.left g))
                   (associative C x a c d sq.lower sq.left g))
                (congruence
                   C
                   x
                   a
                   d
                   (compose C a c d sq.lower sq.left)
                   (compose C a b d sq.right sq.upper)
                   g
                   g
                   ispb.commutes
                   (reflexive (C.hom x a) g)))
             (associative C x a b d sq.right sq.upper g))
          g
          h
          (struct {
             _1 = reflexive (C.hom x b) (compose C x a b sq.upper g);
             _2 = reflexive (C.hom x c) (compose C x a c sq.left g);})
          (struct {
             _1 =
               symmetric
                 (C.hom x b)
                 (compose C x a b sq.upper g)
                 (compose C x a b sq.upper h)
                 pf;
             _2 =
               symmetric
                 (C.hom x c)
                 (compose C x a c sq.left g)
                 (compose C x a c sq.left h)
                 eq3;})

has_pullbacks (C::E_category)
  :: (a::C.obj) ->
     (b::C.obj) ->
     (x::C.obj) ->
     (f::Hom C a x) ->
     (g::Hom C b x) ->
     Set
  = \(a::C.obj) ->
    \(b::C.obj) ->
    \(x::C.obj) ->
    \(f::Hom C a x) ->
    \(g::Hom C b x) ->
    sig{c :: C.obj;
        p :: Hom C c a;
        q :: Hom C c b;
        ispb ::
          Pullback
            C
            c
            b
            a
            x
            (struct {
               left = p;
               right = g;
               upper = q;
```

```
              lower = f;});}

has_terminal (C::E_category) :: Set
  = sig{term :: C.obj;
        ist :: Terminal C term;}

Cartesian (C::E_category) :: Set
  = sig{p1 :: has_terminal C;
        p2 ::
          (a::C.obj) ->
          (b::C.obj) ->
          (x::C.obj) ->
          (f::Hom C a x) ->
          (g::Hom C b x) ->
          has_pullbacks C a b x f g;}

has_binary_products (C::E_category) :: (a::C.obj) -> (b::C.obj) -> Set
  = \(a::C.obj) ->
    \(b::C.obj) ->
    sig{axb :: C.obj;
        p1 :: Hom C axb a;
        p2 :: Hom C axb b;
        isproduct ::
          Product
            C
            a
            axb
            b
            (struct {
               pr1 = p1;
               pr2 = p2;});}

Cartesian_has_binary_products (C::E_category)
                              (pc::Cartesian C)
                              (a::C.obj)
                              (b::C.obj)
  :: has_binary_products C a b
  = let bang (x::C.obj) :: Hom C x pc.p1.term
          = pc.p1.ist.construction x
    in  let pb :: has_pullbacks C a b pc.p1.term (bang a) (bang b)
              = pc.p2 a b pc.p1.term (bang a) (bang b)
        in  struct {
              axb = pb.c;
              p1 = pb.p;
              p2 = pb.q;
              isproduct =
                struct {
                  construction =
                    \(q::C.obj) ->
                    \(diag2::Pdiagram C a q b) ->
                    pb.ispb.construction
                      q
                      diag2.pr2
                      diag2.pr1
                      (pc.p1.ist.uniqueness
                         q
                         (compose C q a pc.p1.term (bang a) diag2.pr1)
                         (compose C q b pc.p1.term (bang b) diag2.pr2));
                  universal =
                    \(q::C.obj) ->
                    \(diag2::Pdiagram C a q b) ->
                    let eq2
                          :: Commutes2Triangles
                               C
                               q
                               b
                               pb.c
                               a
                               diag2.pr2
                               (construction q diag2)
                               diag2.pr1
                               pb.q
                               pb.p
                          = pb.ispb.universal
                              q
                              diag2.pr2
                              diag2.pr1
                              (pc.p1.ist.uniqueness
                                 q
                                 (compose C q a pc.p1.term (bang a) diag2.pr1)
                                 (compose C q b pc.p1.term (bang b) diag2.pr2))
                    in  struct {
                          _1 = eq2._2;
                          _2 = eq2._1;};
                  unique =
                    \(q::C.obj) ->
                    \(diag2::Pdiagram C a q b) ->
                    \(f::Hom C q axb) ->
                    \(f'::Hom C q axb) ->
                    \(pf1::and
                             (Equal
                                (C.hom q a)
                                (compose C q axb a p1 f)
                                diag2.pr1)
                             (Equal
                                (C.hom q b)
                                (compose C q axb b p2 f)
                                diag2.pr2)) ->
                    \(pf2::and
                             (Equal
                                (C.hom q a)
                                (compose C q axb a p1 f')
                                diag2.pr1)
                             (Equal
                                (C.hom q b)
                                (compose C q axb b p2 f')
                                diag2.pr2)) ->
                    pb.ispb.unique
                      q
                      diag2.pr2
                      diag2.pr1
                      (pc.p1.ist.uniqueness
                         q
                         (compose C q a pc.p1.term (bang a) diag2.pr1)
                         (compose C q b pc.p1.term (bang b) diag2.pr2))
                      f
                      f'
                      (struct {
                         _1 = pf1._2;
                         _2 = pf1._1;})
                      (struct {
                         _1 = pf2._2;
                         _2 = pf2._1;});};}

subobj_intersection (C::E_category)
                    (pc::Cartesian C)
                    (x::C.obj)
                    (M::Mon C x)
                    (N::Mon C x)
  :: Mon C x
  = let pb :: has_pullbacks C M.dom N.dom x M.arr N.arr
           = pc.p2 M.dom N.dom x M.arr N.arr
    in  struct {
          dom = pb.c;
          arr = compose C dom N.dom x N.arr pb.q;
          ismono =
            Monos_compose
              C
```

```
              dom
              N.dom
              x
              pb.q
              N.arr
              (pullbacks_preserve_monos
                 C
                 dom
                 N.dom
                 M.dom
                 x
                 (struct {
                    left = pb.p;
                    right = N.arr;
                    upper = pb.q;
                    lower = M.arr;})
                 pb.ispb
                 M.ismono)
              N.ismono;}

intersection_lemma1 (C::E_category)
                    (pc::Cartesian C)
                    (x::C.obj)
                    (M::Mon C x)
                    (N::Mon C x)
  :: mono_incl C x (subobj_intersection C pc x M N) M
  = struct {
      mediator = (pc.p2 M.dom N.dom x M.arr N.arr).p;
      commutes =
        symmetric
          (C.hom (subobj_intersection C pc x M N).dom x)
          (compose C (subobj_intersection C pc x M N).dom M.dom x M.arr mediator)
          (subobj_intersection C pc x M N).arr
          (pc.p2 M.dom N.dom x M.arr N.arr).ispb.commutes;}

intersection_lemma2 (C::E_category)
                    (pc::Cartesian C)
                    (x::C.obj)
                    (M::Mon C x)
                    (N::Mon C x)
  :: mono_incl C x (subobj_intersection C pc x M N) N
  = struct {
      mediator = (pc.p2 M.dom N.dom x M.arr N.arr).q;
      commutes =
        reflexive
          (C.hom (subobj_intersection C pc x M N).dom x)
          (compose C (subobj_intersection C pc x M N).dom N.dom x N.arr mediator);}

intersection_lemma3 (C::E_category)
                    (pc::Cartesian C)
                    (x::C.obj)
                    (P::Mon C x)
                    (M::Mon C x)
                    (N::Mon C x)
                    (pf1::mono_incl C x P M)
                    (pf2::mono_incl C x P N)
  :: mono_incl C x P (subobj_intersection C pc x M N)
  = let pb :: has_pullbacks C M.dom N.dom x M.arr N.arr
          = pc.p2 M.dom N.dom x M.arr N.arr
    in  struct {
          mediator =
            pb.ispb.construction
              P.dom
              pf2.mediator
              pf1.mediator
              (transitive
                 (C.hom P.dom x)
                 (compose C P.dom M.dom x M.arr pf1.mediator)
                 P.arr
                 (compose C P.dom N.dom x N.arr pf2.mediator)
                 (symmetric
                    (C.hom P.dom x)
                    P.arr
                    (compose C P.dom M.dom x M.arr pf1.mediator)
                    pf1.commutes)
                 pf2.commutes);
          commutes =
            let eq1
                  :: Equal
                       (C.hom P.dom N.dom)
                       (compose C P.dom pb.c N.dom pb.q mediator)
                       pf2.mediator
                  = (pb.ispb.universal
                       P.dom
                       pf2.mediator
                       pf1.mediator
                       (transitive
                          (C.hom P.dom x)
                          (compose C P.dom M.dom x M.arr pf1.mediator)
                          P.arr
                          (compose C P.dom N.dom x N.arr pf2.mediator)
                          (symmetric
                             (C.hom P.dom x)
                             P.arr
                             (compose C P.dom M.dom x M.arr pf1.mediator)
                             pf1.commutes)
                          pf2.commutes))._1
            in  transitive
                  (C.hom P.dom x)
                  P.arr
                  (compose C P.dom N.dom x N.arr pf2.mediator)
                  (compose
                     C
                     P.dom
                     (subobj_intersection C pc x M N).dom
                     x
                     (subobj_intersection C pc x M N).arr
                     mediator)
                  pf2.commutes
                  (transitive
                     (C.hom P.dom x)
                     (compose C P.dom N.dom x N.arr pf2.mediator)
                     (compose
                        C
                        P.dom
                        N.dom
                        x
                        N.arr
                        (compose
                           C
                           P.dom
                           (subobj_intersection C pc x M N).dom
                           N.dom
                           pb.q
                           mediator))
                     (compose
                        C
                        P.dom
                        (subobj_intersection C pc x M N).dom
                        x
                        (subobj_intersection C pc x M N).arr
                        mediator)
                     (congruence
                        C
                        P.dom
                        N.dom
                        x
                        N.arr
```

```
                        N.arr
                        pf2.mediator
                        (compose C P.dom pb.c N.dom pb.q mediator)
                        (reflexive (C.hom N.dom x) N.arr)
                        (symmetric
                           (C.hom P.dom N.dom)
                           (compose C P.dom pb.c N.dom pb.q mediator)
                           pf2.mediator
                           eq1))
                     (symmetric
                        (C.hom P.dom x)
                        (compose
                           C
                           P.dom
                           (subobj_intersection C pc x M N).dom
                           x
                           (subobj_intersection C pc x M N).arr
                           mediator)
                        (compose
                           C
                           P.dom
                           N.dom
                           x
                           N.arr
                           (compose
                              C
                              P.dom
                              (subobj_intersection C pc x M N).dom
                              N.dom
                              pb.q
                              mediator))
                        (associative
                           C
                           P.dom
                           (subobj_intersection C pc x M N).dom
                           N.dom
                           x
                           N.arr
                           pb.q
                           mediator)));}
{-# Alfa unfoldgoals off
brief on
hidetypeannots off
wide

nd
hiding on
var "compose" hide 4 infix as "o"
var "Hom_ref" hide 3
var "Hom_sym" hide 5
var "Hom_tra" hide 7
var "congruence" hide 8
var "associative" hide 5
 #-}
```

missingbits.agda

```
--#include "E_categories.agda"

Iso (C::E_category)(a::C.obj)(b::C.obj)(f::Hom C a b) :: Set
  = Exists (Hom C b a) (\(h::Hom C b a) -> InverseArrows C a b f h)

{-# Alfa unfoldgoals off
brief on
hidetypeannots off
wide

nd
hiding on
var "Iso" hide 3
 #-}
```

E_functors.agda

```
--#include "E_categories.agda"
--#include "missingbits.agda"

E_functor (A::E_category)(B::E_category) :: Set
  = sig{objectfunction :: A.obj -> B.obj;
        arrowfunction ::
          (a::A.obj) ->
          (b::A.obj) ->
          (A.hom a b ==> (B.hom (objectfunction a) (objectfunction b)));
        ax_preserve_id ::
          (a::A.obj) ->
          Equal
            (B.hom (objectfunction a) (objectfunction a))
            ((arrowfunction a a).op (A.id a))
            (B.id (objectfunction a));
        ax_preserve_composition ::
          (a::A.obj) ->
          (b::A.obj) ->
          (c::A.obj) ->
          (f::Hom A b c) ->
          (g::Hom A a b) ->
          Equal
            (B.hom (objectfunction a) (objectfunction c))
            ((arrowfunction a c).op (compose A a b c f g))
            (compose
               B
               (objectfunction a)
               (objectfunction b)
               (objectfunction c)
               ((arrowfunction b c).op f)
               ((arrowfunction a b).op g));}

Arrowfunction (A::E_category)
              (B::E_category)
              (F::E_functor A B)
              (a::A.obj)
              (b::A.obj)
  :: Hom A a b -> Hom B (F.objectfunction a) (F.objectfunction b)
  = (F.arrowfunction a b).op

Arrowfunctionextensionality (A::E_category)
                            (B::E_category)
                            (F::E_functor A B)
                            (a::A.obj)
                            (b::A.obj)
  :: (c::Hom A a b) ->
     (d::Hom A a b) ->
     (pr::Equal (A.hom a b) c d) ->
     Equal
       (B.hom (F.objectfunction a) (F.objectfunction b))
       (Arrowfunction A B F a b c)
       (Arrowfunction A B F a b d)
  = (F.arrowfunction a b).ext

E_natural_transformation (A::E_category)
                         (B::E_category)
                         (F::E_functor A B)
                         (G::E_functor A B)
```

```
  :: Set
  = sig{arrows ::
          (a::A.obj) -> Hom B (F.objectfunction a) (G.objectfunction a);
        ax_naturality ::
          (a::A.obj) ->
          (b::A.obj) ->
          (f::Hom A a b) ->
          Equal
            (B.hom (F.objectfunction a) (G.objectfunction b))
            (compose
               B
               (F.objectfunction a)
               (F.objectfunction b)
               (G.objectfunction b)
               (arrows b)
               (Arrowfunction A B F a b f))
            (compose
               B
               (F.objectfunction a)
               (G.objectfunction a)
               (G.objectfunction b)
               (Arrowfunction A B G a b f)
               (arrows a));}

E_natural_transformation_SE (A::E_category)
                            (B::E_category)
                            (F::E_functor A B)
                            (G::E_functor A B)
  :: SE
  = struct {
      base = E_natural_transformation A B F G;
      er =
        struct {
          eq =
            \(h::base) ->
            \(h'::base) ->
            ForAll
              A.obj
              (\(h0::A.obj) ->
               Equal
                 (B.hom (F.objectfunction h0) (G.objectfunction h0))
                 (h.arrows h0)
                 (h'.arrows h0));
          ref =
            \(x::base) ->
            \(x'::A.obj) ->
            reflexive
              (B.hom (F.objectfunction x') (G.objectfunction x'))
              (x.arrows x');
          sym =
            \(x::base) ->
            \(y::base) ->
            \(h::eq x y) ->
            \(x'::A.obj) ->
            symmetric
              (B.hom (F.objectfunction x') (G.objectfunction x'))
              (x.arrows x')
              (y.arrows x')
              (h x');
          tra =
            \(x::base) ->
            \(y::base) ->
            \(z::base) ->
            \(h::eq x y) ->
            \(h'::eq y z) ->
            \(x'::A.obj) ->
            transitive
              (B.hom (F.objectfunction x') (G.objectfunction x'))
              (x.arrows x')
              (y.arrows x')
              (z.arrows x')
              (h x')
              (h' x');};}

E_functorcategory (A::E_category)(B::E_category) :: E_category
  = struct {
      obj = E_functor A B;
      hom = E_natural_transformation_SE A B;
      id =
        \(a::obj) ->
        struct {
          arrows = \(a'::A.obj) -> B.id (a.objectfunction a');
          ax_naturality =
            \(a'::A.obj) ->
            \(b::A.obj) ->
            \(f::Hom A a' b) ->
            transitive
              (B.hom (a.objectfunction a') (a.objectfunction b))
              (compose
                 B
                 (a.objectfunction a')
                 (a.objectfunction b)
                 (a.objectfunction b)
                 (arrows b)
                 (Arrowfunction A B a a' b f))
              (Arrowfunction A B a a' b f)
              (compose
                 B
                 (a.objectfunction a')
                 (a.objectfunction a')
                 (a.objectfunction b)
                 (Arrowfunction A B a a' b f)
                 (arrows a'))
              (B.left_unit
                 (a.objectfunction a')
                 (a.objectfunction b)
                 (Arrowfunction A B a a' b f))
              (symmetric
                 (B.hom (a.objectfunction a') (a.objectfunction b))
                 (compose
                    B
                    (a.objectfunction a')
                    (a.objectfunction a')
                    (a.objectfunction b)
                    (Arrowfunction A B a a' b f)
                    (arrows a'))
                 (Arrowfunction A B a a' b f)
                 (B.right_unit
                    (a.objectfunction a')
                    (a.objectfunction b)
                    (Arrowfunction A B a a' b f)));};
      comp =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(h::(hom b c).base) ->
        \(h'::(hom a b).base) ->
        struct {
          arrows =
            \(a'::A.obj) ->
            B.comp
              (a.objectfunction a')
              (b.objectfunction a')
              (c.objectfunction a')
              (h.arrows a')
              (h'.arrows a');
          ax_naturality =
            \(a'::A.obj) ->
```

```
\(b'::A.obj) ->
\(f::Hom A a' b') ->
transitive
  (B.hom (a.objectfunction a') (c.objectfunction b'))
  (compose
     B
     (a.objectfunction a')
     (a.objectfunction b')
     (c.objectfunction b')
     (arrows b')
     (Arrowfunction A B a a' b' f))
  (compose
     B
     (a.objectfunction a')
     (a.objectfunction b')
     (c.objectfunction b')
     (compose
        B
        (a.objectfunction b')
        (b.objectfunction b')
        (c.objectfunction b')
        (h.arrows b')
        (h'.arrows b'))
     (Arrowfunction A B a a' b' f))
  (compose
     B
     (a.objectfunction a')
     (c.objectfunction a')
     (c.objectfunction b')
     (Arrowfunction A B c a' b' f)
     (arrows a'))
  (B.cong
     (a.objectfunction a')
     (a.objectfunction b')
     (c.objectfunction b')
     (arrows b')
     (compose
        B
        (a.objectfunction b')
        (b.objectfunction b')
        (c.objectfunction b')
        (h.arrows b')
        (h'.arrows b'))
     (Arrowfunction A B a a' b' f)
     (Arrowfunction A B a a' b' f)
     (reflexive
        (B.hom (a.objectfunction b') (c.objectfunction b'))
        (arrows b'))
     (reflexive
        (B.hom (a.objectfunction a') (a.objectfunction b'))
        (Arrowfunction A B a a' b' f)))
  (transitive
     (B.hom (a.objectfunction a') (c.objectfunction b'))
     (compose
        B
        (a.objectfunction a')
        (a.objectfunction b')
        (c.objectfunction b')
        (compose
           B
           (a.objectfunction b')
           (b.objectfunction b')
           (c.objectfunction b')
           (h.arrows b')
           (h'.arrows b'))
        (Arrowfunction A B a a' b' f))
     (B.comp
        (a.objectfunction a')
        (b.objectfunction b')
        (c.objectfunction b')
        (h.arrows b')
        (B.comp
           (a.objectfunction a')
           (a.objectfunction b')
           (b.objectfunction b')
           (h'.arrows b')
           (Arrowfunction A B a a' b' f)))
     (compose
        B
        (a.objectfunction a')
        (c.objectfunction a')
        (c.objectfunction b')
        (Arrowfunction A B c a' b' f)
        (arrows a'))
     (B.assoc
        (a.objectfunction a')
        (a.objectfunction b')
        (b.objectfunction b')
        (c.objectfunction b')
        (h.arrows b')
        (h'.arrows b')
        (Arrowfunction A B a a' b' f))
     (transitive
        (B.hom (a.objectfunction a') (c.objectfunction b'))
        (B.comp
           (a.objectfunction a')
           (b.objectfunction b')
           (c.objectfunction b')
           (h.arrows b')
           (B.comp
              (a.objectfunction a')
              (a.objectfunction b')
              (b.objectfunction b')
              (h'.arrows b')
              (Arrowfunction A B a a' b' f)))
        (compose
           B
           (a.objectfunction a')
           (b.objectfunction b')
           (c.objectfunction b')
           (h.arrows b')
           (compose
              B
              (a.objectfunction a')
              (b.objectfunction a')
              (b.objectfunction b')
              (Arrowfunction A B b a' b' f)
              (h'.arrows a')))
        (compose
           B
           (a.objectfunction a')
           (c.objectfunction a')
           (c.objectfunction b')
           (Arrowfunction A B c a' b' f)
           (arrows a'))
        (B.cong
           (a.objectfunction a')
           (b.objectfunction b')
           (c.objectfunction b')
           (h.arrows b')
           (h.arrows b')
           (B.comp
              (a.objectfunction a')
              (a.objectfunction b')
              (b.objectfunction b')
              (h'.arrows b')
              (Arrowfunction A B a a' b' f))
           (compose
```

```
      B
      (a.objectfunction a')
      (b.objectfunction a')
      (b.objectfunction b')
      (Arrowfunction A B b a' b' f)
      (h'.arrows a'))
   (reflexive
      (B.hom (b.objectfunction b') (c.objectfunction b'))
      (h.arrows b'))
   (h'.ax_naturality a' b' f))
(transitive
   (B.hom (a.objectfunction a') (c.objectfunction b'))
   (compose
      B
      (a.objectfunction a')
      (b.objectfunction b')
      (c.objectfunction b')
      (h.arrows b')
      (compose
         B
         (a.objectfunction a')
         (b.objectfunction a')
         (b.objectfunction b')
         (Arrowfunction A B b a' b' f)
         (h'.arrows a')))
   (B.comp
      (a.objectfunction a')
      (b.objectfunction a')
      (c.objectfunction b')
      (B.comp
         (b.objectfunction a')
         (b.objectfunction b')
         (c.objectfunction b')
         (h.arrows b')
         (Arrowfunction A B b a' b' f))
      (h'.arrows a'))
   (compose
      B
      (a.objectfunction a')
      (c.objectfunction a')
      (c.objectfunction b')
      (Arrowfunction A B c a' b' f)
      (arrows a'))
   (symmetric
      (B.hom (a.objectfunction a') (c.objectfunction b'))
      (B.comp
         (a.objectfunction a')
         (b.objectfunction a')
         (c.objectfunction b')
         (B.comp
            (b.objectfunction a')
            (b.objectfunction b')
            (c.objectfunction b')
            (h.arrows b')
            (Arrowfunction A B b a' b' f))
         (h'.arrows a'))
      (compose
         B
         (a.objectfunction a')
         (b.objectfunction b')
         (c.objectfunction b')
         (h.arrows b')
         (compose
            B
            (a.objectfunction a')
            (b.objectfunction a')
            (b.objectfunction b')
            (Arrowfunction A B b a' b' f)
            (h'.arrows a')))
```

```
   (B.assoc
      (a.objectfunction a')
      (b.objectfunction a')
      (b.objectfunction b')
      (c.objectfunction b')
      (h.arrows b')
      (Arrowfunction A B b a' b' f)
      (h'.arrows a')))
(transitive
   (B.hom (a.objectfunction a') (c.objectfunction b'))
   (B.comp
      (a.objectfunction a')
      (b.objectfunction a')
      (c.objectfunction b')
      (B.comp
         (b.objectfunction a')
         (b.objectfunction b')
         (c.objectfunction b')
         (h.arrows b')
         (Arrowfunction A B b a' b' f))
      (h'.arrows a'))
   (compose
      B
      (a.objectfunction a')
      (b.objectfunction a')
      (c.objectfunction b')
      (compose
         B
         (b.objectfunction a')
         (c.objectfunction a')
         (c.objectfunction b')
         (Arrowfunction A B c a' b' f)
         (h.arrows a'))
      (h'.arrows a'))
   (compose
      B
      (a.objectfunction a')
      (c.objectfunction a')
      (c.objectfunction b')
      (Arrowfunction A B c a' b' f)
      (arrows a'))
   (B.cong
      (a.objectfunction a')
      (b.objectfunction a')
      (c.objectfunction b')
      (B.comp
         (b.objectfunction a')
         (b.objectfunction b')
         (c.objectfunction b')
         (h.arrows b')
         (Arrowfunction A B b a' b' f))
      (compose
         B
         (b.objectfunction a')
         (c.objectfunction a')
         (c.objectfunction b')
         (Arrowfunction A B c a' b' f)
         (h.arrows a'))
      (h'.arrows a')
      (h'.arrows a')
      (h.ax_naturality a' b' f)
      (reflexive
         (B.hom
            (a.objectfunction a')
            (b.objectfunction a'))
         (h'.arrows a')))
   (transitive
      (B.hom (a.objectfunction a') (c.objectfunction b'))
      (compose
```

```
                          B
                          (a.objectfunction a')
                          (b.objectfunction a')
                          (c.objectfunction b')
                          (compose
                             B
                             (b.objectfunction a')
                             (c.objectfunction a')
                             (c.objectfunction b')
                             (Arrowfunction A B c a' b' f)
                             (h.arrows a'))
                          (h'.arrows a'))
                       (B.comp
                          (a.objectfunction a')
                          (c.objectfunction a')
                          (c.objectfunction b')
                          (Arrowfunction A B c a' b' f)
                          (B.comp
                             (a.objectfunction a')
                             (b.objectfunction a')
                             (c.objectfunction a')
                             (h.arrows a')
                             (h'.arrows a')))
                       (compose
                          B
                          (a.objectfunction a')
                          (c.objectfunction a')
                          (c.objectfunction b')
                          (Arrowfunction A B c a' b' f)
                          (arrows a'))
                       (B.assoc
                          (a.objectfunction a')
                          (b.objectfunction a')
                          (c.objectfunction a')
                          (c.objectfunction b')
                          (Arrowfunction A B c a' b' f)
                          (h.arrows a')
                          (h'.arrows a'))
                       (B.cong
                          (a.objectfunction a')
                          (c.objectfunction a')
                          (c.objectfunction b')
                          (Arrowfunction A B c a' b' f)
                          (Arrowfunction A B c a' b' f)
                          (B.comp
                             (a.objectfunction a')
                             (b.objectfunction a')
                             (c.objectfunction a')
                             (h.arrows a')
                             (h'.arrows a'))
                          (arrows a')
                          (reflexive
                             (B.hom
                                (c.objectfunction a')
                                (c.objectfunction b'))
                             (Arrowfunction A B c a' b' f))
                          (reflexive
                             (B.hom
                                (a.objectfunction a')
                                (c.objectfunction a'))
                             (arrows a'))))))));};
left_unit =
  \(a::obj) ->
  \(b::obj) ->
  \(f::(hom a b).base) ->
  \(x::A.obj) ->
  transitive
    (B.hom (a.objectfunction x) (b.objectfunction x))
    ((comp a b b (id b) f).arrows x)
    (compose
       B
       (a.objectfunction x)
       (b.objectfunction x)
       (b.objectfunction x)
       ((id b).arrows x)
       (f.arrows x))
    (f.arrows x)
    (reflexive
       (B.hom (a.objectfunction x) (b.objectfunction x))
       ((comp a b b (id b) f).arrows x))
    (B.left_unit (a.objectfunction x) (b.objectfunction x) (f.arrows x));
right_unit =
  \(a::obj) ->
  \(b::obj) ->
  \(f::(hom a b).base) ->
  \(x::A.obj) ->
  transitive
    (B.hom (a.objectfunction x) (b.objectfunction x))
    ((comp a a b f (id a)).arrows x)
    (compose
       B
       (a.objectfunction x)
       (a.objectfunction x)
       (b.objectfunction x)
       (f.arrows x)
       ((id a).arrows x))
    (f.arrows x)
    (reflexive
       (B.hom (a.objectfunction x) (b.objectfunction x))
       ((comp a a b f (id a)).arrows x))
    (B.right_unit (a.objectfunction x) (b.objectfunction x) (f.arrows x));
assoc =
  \(a::obj) ->
  \(b::obj) ->
  \(c::obj) ->
  \(d::obj) ->
  \(f::(hom c d).base) ->
  \(g::(hom b c).base) ->
  \(h::(hom a b).base) ->
  \(x::A.obj) ->
  transitive
    (B.hom (a.objectfunction x) (d.objectfunction x))
    ((comp a b d (comp b c d f g) h).arrows x)
    (compose
       B
       (a.objectfunction x)
       (b.objectfunction x)
       (d.objectfunction x)
       (compose
          B
          (b.objectfunction x)
          (c.objectfunction x)
          (d.objectfunction x)
          (f.arrows x)
          (g.arrows x))
       (h.arrows x))
    ((comp a c d f (comp a b c g h)).arrows x)
    (transitive
       (B.hom (a.objectfunction x) (d.objectfunction x))
       ((comp a b d (comp b c d f g) h).arrows x)
       (compose
          B
          (a.objectfunction x)
          (b.objectfunction x)
          (d.objectfunction x)
          ((comp b c d f g).arrows x)
          (h.arrows x))
       (compose
```

```
                B
                (a.objectfunction x)
                (b.objectfunction x)
                (d.objectfunction x)
                (compose
                   B
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (d.objectfunction x)
                   (f.arrows x)
                   (g.arrows x))
                (h.arrows x))
             (reflexive
                (B.hom (a.objectfunction x) (d.objectfunction x))
                ((comp a b d (comp b c d f g) h).arrows x))
             (B.cong
                (a.objectfunction x)
                (b.objectfunction x)
                (d.objectfunction x)
                ((comp b c d f g).arrows x)
                (compose
                   B
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (d.objectfunction x)
                   (f.arrows x)
                   (g.arrows x))
                (h.arrows x)
                (h.arrows x)
                (reflexive
                   (B.hom (b.objectfunction x) (d.objectfunction x))
                   ((comp b c d f g).arrows x))
                (reflexive
                   (B.hom (a.objectfunction x) (b.objectfunction x))
                   (h.arrows x))))
          (transitive
             (B.hom (a.objectfunction x) (d.objectfunction x))
             (compose
                B
                (a.objectfunction x)
                (b.objectfunction x)
                (d.objectfunction x)
                (compose
                   B
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (d.objectfunction x)
                   (f.arrows x)
                   (g.arrows x))
                (h.arrows x))
             (B.comp
                (a.objectfunction x)
                (c.objectfunction x)
                (d.objectfunction x)
                (f.arrows x)
                (B.comp
                   (a.objectfunction x)
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (g.arrows x)
                   (h.arrows x)))
             ((comp a c d f (comp a b c g h)).arrows x)
             (B.assoc
                (a.objectfunction x)
                (b.objectfunction x)
                (c.objectfunction x)
                (d.objectfunction x)
                (f.arrows x)
                (g.arrows x)
                (h.arrows x))
          (transitive
             (B.hom (a.objectfunction x) (d.objectfunction x))
             (B.comp
                (a.objectfunction x)
                (c.objectfunction x)
                (d.objectfunction x)
                (f.arrows x)
                (B.comp
                   (a.objectfunction x)
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (g.arrows x)
                   (h.arrows x)))
             (compose
                B
                (a.objectfunction x)
                (c.objectfunction x)
                (d.objectfunction x)
                (f.arrows x)
                ((comp a b c g h).arrows x))
             ((comp a c d f (comp a b c g h)).arrows x)
             (B.cong
                (a.objectfunction x)
                (c.objectfunction x)
                (d.objectfunction x)
                (f.arrows x)
                (f.arrows x)
                (B.comp
                   (a.objectfunction x)
                   (b.objectfunction x)
                   (c.objectfunction x)
                   (g.arrows x)
                   (h.arrows x))
                ((comp a b c g h).arrows x)
                (reflexive
                   (B.hom (c.objectfunction x) (d.objectfunction x))
                   (f.arrows x))
                (reflexive
                   (B.hom (a.objectfunction x) (c.objectfunction x))
                   ((comp a b c g h).arrows x)))
             (reflexive
                (B.hom (a.objectfunction x) (d.objectfunction x))
                ((comp a c d f (comp a b c g h)).arrows x))));
cong =
  \(a::obj) ->
  \(b::obj) ->
  \(c::obj) ->
  \(f::(hom b c).base) ->
  \(f'::(hom b c).base) ->
  \(g::(hom a b).base) ->
  \(g'::(hom a b).base) ->
  \(h::Equal (hom b c) f f') ->
  \(h'::Equal (hom a b) g g') ->
  \(x::A.obj) ->
  transitive
    (B.hom (a.objectfunction x) (c.objectfunction x))
    ((comp a b c f g).arrows x)
    (compose
       B
       (a.objectfunction x)
       (b.objectfunction x)
       (c.objectfunction x)
       (f.arrows x)
       (g.arrows x))
    ((comp a b c f' g').arrows x)
    (B.cong
       (a.objectfunction x)
       (b.objectfunction x)
```

```
             (c.objectfunction x)
             (f.arrows x)
             (f.arrows x)
             (g.arrows x)
             (g.arrows x)
             (reflexive
                (B.hom (b.objectfunction x) (c.objectfunction x))
                (f.arrows x))
             (reflexive
                (B.hom (a.objectfunction x) (b.objectfunction x))
                (g.arrows x)))
          (transitive
             (B.hom (a.objectfunction x) (c.objectfunction x))
             (compose
                B
                (a.objectfunction x)
                (b.objectfunction x)
                (c.objectfunction x)
                (f.arrows x)
                (g.arrows x))
             (compose
                B
                (a.objectfunction x)
                (b.objectfunction x)
                (c.objectfunction x)
                (f'.arrows x)
                (g'.arrows x))
             ((comp a b c f' g').arrows x)
             (B.cong
                (a.objectfunction x)
                (b.objectfunction x)
                (c.objectfunction x)
                (f.arrows x)
                (f'.arrows x)
                (g.arrows x)
                (g'.arrows x)
                (h x)
                (h' x))
             (B.cong
                (a.objectfunction x)
                (b.objectfunction x)
                (c.objectfunction x)
                (f'.arrows x)
                (f'.arrows x)
                (g'.arrows x)
                (g'.arrows x)
                (reflexive
                   (B.hom (b.objectfunction x) (c.objectfunction x))
                   (f'.arrows x))
                (reflexive
                   (B.hom (a.objectfunction x) (b.objectfunction x))
                   (g'.arrows x))));}

E_functorcomposition (A::E_category)
                     (B::E_category)
                     (C::E_category)
                     (f::E_functor B C)
                     (g::E_functor A B)
  :: E_functor A C
  = struct {
      objectfunction = \(h::A.obj) -> f.objectfunction (g.objectfunction h);
      arrowfunction =
        \(a::A.obj) ->
        \(b::A.obj) ->
        oSE
          (A.hom a b)
          (B.hom (g.objectfunction a) (g.objectfunction b))
          (C.hom (objectfunction a) (objectfunction b))
          (f.arrowfunction (g.objectfunction a) (g.objectfunction b))
          (g.arrowfunction a b);
      ax_preserve_id =
        \(a::A.obj) ->
        transitive
          (C.hom (objectfunction a) (objectfunction a))
          ((arrowfunction a a).op (A.id a))
          (Arrowfunction
             B
             C
             f
             (g.objectfunction a)
             (g.objectfunction a)
             (B.id (g.objectfunction a)))
          (C.id (objectfunction a))
          (Arrowfunctionextensionality
             B
             C
             f
             (g.objectfunction a)
             (g.objectfunction a)
             ((g.arrowfunction a a).op (A.id a))
             (B.id (g.objectfunction a))
             (g.ax_preserve_id a))
          (f.ax_preserve_id (g.objectfunction a));
      ax_preserve_composition =
        \(a::A.obj) ->
        \(b::A.obj) ->
        \(c::A.obj) ->
        \(f'::Hom A b c) ->
        \(g'::Hom A a b) ->
        transitive
          (C.hom (objectfunction a) (objectfunction c))
          ((arrowfunction a c).op (compose A a b c f' g'))
          (Arrowfunction
             B
             C
             f
             (g.objectfunction a)
             (g.objectfunction c)
             (Arrowfunction A B g a c (compose A a b c f' g')))
          (compose
             C
             (objectfunction a)
             (objectfunction b)
             (objectfunction c)
             ((arrowfunction b c).op f')
             ((arrowfunction a b).op g'))
          (reflexive
             (C.hom (objectfunction a) (objectfunction c))
             ((arrowfunction a c).op (compose A a b c f' g')))
          (transitive
             (C.hom (objectfunction a) (objectfunction c))
             (Arrowfunction
                B
                C
                f
                (g.objectfunction a)
                (g.objectfunction c)
                (Arrowfunction A B g a c (compose A a b c f' g')))
             (Arrowfunction
                B
                C
                f
                (g.objectfunction a)
                (g.objectfunction c)
                (compose
                   B
                   (g.objectfunction a)
                   (g.objectfunction b)
```

```
      (g.objectfunction c)
      (Arrowfunction A B g b c f')
      (Arrowfunction A B g a b g')))
(compose
   C
   (objectfunction a)
   (objectfunction b)
   (objectfunction c)
   ((arrowfunction b c).op f')
   ((arrowfunction a b).op g'))
(Arrowfunctionextensionality
   B
   C
   f
   (g.objectfunction a)
   (g.objectfunction c)
   (Arrowfunction A B g a c (compose A a b c f' g'))
   (compose
      B
      (g.objectfunction a)
      (g.objectfunction b)
      (g.objectfunction c)
      (Arrowfunction A B g b c f')
      (Arrowfunction A B g a b g'))
   (g.ax_preserve_composition a b c f' g'))
(transitive
   (C.hom (objectfunction a) (objectfunction c))
   (Arrowfunction
      B
      C
      f
      (g.objectfunction a)
      (g.objectfunction c)
      (compose
         B
         (g.objectfunction a)
         (g.objectfunction b)
         (g.objectfunction c)
         (Arrowfunction A B g b c f')
         (Arrowfunction A B g a b g')))
   (compose
      C
      (objectfunction a)
      (objectfunction b)
      (objectfunction c)
      (Arrowfunction
         B
         C
         f
         (g.objectfunction b)
         (g.objectfunction c)
         (Arrowfunction A B g b c f'))
      (Arrowfunction
         B
         C
         f
         (g.objectfunction a)
         (g.objectfunction b)
         (Arrowfunction A B g a b g')))
   (compose
      C
      (objectfunction a)
      (objectfunction b)
      (objectfunction c)
      ((arrowfunction b c).op f')
      ((arrowfunction a b).op g'))
   (f.ax_preserve_composition
      (g.objectfunction a)
      (g.objectfunction b)
```

```
                    (g.objectfunction c)
                    (Arrowfunction A B g b c f')
                    (Arrowfunction A B g a b g'))
                 (reflexive
                    (C.hom (objectfunction a) (objectfunction c))
                    (compose
                       C
                       (objectfunction a)
                       (objectfunction b)
                       (objectfunction c)
                       ((arrowfunction b c).op f')
                       ((arrowfunction a b).op g')))));}

E_idfunctor (A::E_category) :: E_functor A A
  = struct {
      objectfunction = \(h::A.obj) -> h;
      arrowfunction = \(a::A.obj) -> \(b::A.obj) -> idSE (A.hom a b);
      ax_preserve_id =
        \(a::A.obj) ->
        reflexive
          (A.hom (objectfunction a) (objectfunction a))
          (A.id (objectfunction a));
      ax_preserve_composition =
        \(a::A.obj) ->
        \(b::A.obj) ->
        \(c::A.obj) ->
        \(f::Hom A b c) ->
        \(g::Hom A a b) ->
        reflexive
          (A.hom (objectfunction a) (objectfunction c))
          (compose
             A
             (objectfunction a)
             (objectfunction b)
             (objectfunction c)
             ((arrowfunction b c).op f)
             ((arrowfunction a b).op g));}

NatTransIsoIfComponentsIso
  (C::E_category)(D::E_category)
  (F::E_functor C D)(G::E_functor C D)
  (a::E_natural_transformation C D F G)
  (p::ForAll C.obj
       (\(h::C.obj)->
        Iso D (F.objectfunction h) (G.objectfunction h) (a.arrows h)))
 :: Iso (E_functorcategory C D) F G a
  = struct
    _1 =
      struct
      arrows        =
        \(a'::C.obj)->
        (p a')._1
      ax_naturality =
        \(a'::C.obj)->
        \(b::C.obj)->
        \(f::(C.hom a' b).base)->
        let Ga'::D.obj = G.objectfunction a'
            Fa'::D.obj = F.objectfunction a'
            Gb::D.obj = G.objectfunction b
            Fb::D.obj = F.objectfunction b
            Ff::(D.hom Fa' Fb).base = (F.arrowfunction a' b).op f
            Gf::(D.hom Ga' Gb).base = (G.arrowfunction a' b).op f
            homset::SE = D.hom Ga' Fb
        in  homset.er.tra
              (D.comp Ga' Gb Fb (arrows b) Gf)
              (D.comp Ga' Ga' Fb
                 (D.comp Ga' Gb Fb (arrows b) Gf)
                 (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a')))
              (D.comp Ga' Fa' Fb Ff (arrows a'))
```

```
(homset.er.tra
   (D.comp Ga' Gb Fb (arrows b) Gf)
   (D.comp Ga' Ga' Fb
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.id Ga'))
   (D.comp Ga' Ga' Fb
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a')))
   (homset.er.sym
      (D.comp Ga' Ga' Fb
         (D.comp Ga' Gb Fb (arrows b) Gf)
         (D.id Ga'))
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.right_unit Ga' Fb (D.comp Ga' Gb Fb (arrows b) Gf)))
   (D.cong Ga' Ga' Fb
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.id Ga')
      (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a'))
      (homset.er.ref (D.comp Ga' Gb Fb (arrows b) Gf))
      ((D.hom Ga' Ga').er.sym
         (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a'))
         (D.id Ga')
         (p a')._2._1)))
(homset.er.tra
   (D.comp Ga' Ga' Fb
      (D.comp Ga' Gb Fb (arrows b) Gf)
      (D.comp Ga'  Fa'  Ga' (a.arrows a') (arrows a')))
   (D.comp Ga' Fa' Fb
      (D.comp Fa' Gb Fb
         (arrows b)
         (D.comp Fa' Ga' Gb Gf (a.arrows a')))
      (arrows a'))
   (D.comp Ga' Fa' Fb Ff (arrows a'))
   (homset.er.tra
      (D.comp Ga' Ga' Fb
         (D.comp Ga' Gb Fb (arrows b) Gf)
         (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a')))
      (D.comp Ga' Fa' Fb
         (D.comp Fa' Ga' Fb
            (D.comp Ga' Gb Fb (arrows b) Gf)
            (a.arrows a'))
         (arrows a'))
      (D.comp Ga' Fa' Fb
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Ga' Gb Gf (a.arrows a')))
         (arrows a'))
      (homset.er.sym
         (D.comp Ga' Fa' Fb
            (D.comp Fa' Ga' Fb
               (D.comp Ga' Gb Fb (arrows b) Gf)
               (a.arrows a'))
            (arrows a'))
         (D.comp Ga' Ga' Fb
            (D.comp Ga' Gb Fb (arrows b) Gf)
            (D.comp Ga' Fa' Ga' (a.arrows a') (arrows a')))
         (D.assoc Ga' Fa' Ga' Fb
            (D.comp Ga' Gb Fb (arrows b) Gf)
            (a.arrows a')
            (arrows a')))
      (D.cong Ga' Fa' Fb
         (D.comp Fa' Ga' Fb
            (D.comp Ga' Gb Fb (arrows b) Gf)
            (a.arrows a'))
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Ga' Gb Gf (a.arrows a')))
         (arrows a')
         (arrows a')
         (D.assoc Fa' Ga' Gb Fb (arrows b) Gf (a.arrows a'))
         ((D.hom Ga' Fa').er.ref (arrows a'))))
   (homset.er.tra
      (D.comp Ga' Fa' Fb
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Ga' Gb Gf (a.arrows a')))
         (arrows a'))
      (D.comp Ga' Fa' Fb
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Fb Gb (a.arrows b) Ff))
         (arrows a'))
      (D.comp Ga' Fa' Fb Ff (arrows a'))
      (D.cong Ga' Fa' Fb
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Ga' Gb Gf (a.arrows a')))
         (D.comp Fa' Gb Fb
            (arrows b)
            (D.comp Fa' Fb Gb (a.arrows b) Ff))
         (arrows a')
         (arrows a')
         (D.cong Fa' Gb Fb
            (arrows b)
            (arrows b)
            (D.comp Fa' Ga' Gb Gf (a.arrows a'))
            (D.comp Fa' Fb Gb (a.arrows b) Ff)
            ((D.hom Gb Fb).er.ref (arrows b))
            ((D.hom Fa' Gb).er.sym
               (D.comp Fa' Fb Gb (a.arrows b) Ff)
               (D.comp Fa' Ga' Gb Gf (a.arrows a'))
               (a.ax_naturality a' b f)))
         ((D.hom Ga' Fa').er.ref (arrows a')))
      (homset.er.tra
         (D.comp Ga' Fa' Fb
            (D.comp Fa' Gb Fb
               (arrows b)
               (D.comp Fa' Fb Gb (a.arrows b) Ff))
            (arrows a'))
         (D.comp Ga' Fa' Fb
            (D.comp Fa' Fb Fb
               (D.comp Fb Gb Fb (arrows b) (a.arrows b))
               Ff)
            (arrows a'))
         (D.comp Ga' Fa' Fb Ff (arrows a'))
         (D.cong Ga' Fa' Fb
            (D.comp Fa' Gb Fb
               (arrows b)
               (D.comp Fa' Fb Gb (a.arrows b) Ff))
            (D.comp Fa' Fb Fb
               (D.comp Fb Gb Fb (arrows b) (a.arrows b))
               Ff)
            (arrows a')
            (arrows a')
            ((D.hom Fa' Fb).er.sym
               (D.comp Fa' Fb Fb
                  (D.comp Fb Gb Fb (arrows b) (a.arrows b))
                  Ff)
               (D.comp Fa' Gb Fb
                  (arrows b)
                  (D.comp Fa' Fb Gb (a.arrows b) Ff))
               (D.assoc Fa' Fb Gb Fb (arrows b) (a.arrows b) Ff))
            ((D.hom Ga' Fa').er.ref (arrows a')))
         (D.cong Ga' Fa' Fb
            (D.comp Fa' Fb Fb
               (D.comp Fb Gb Fb (arrows b) (a.arrows b))
               Ff)
```

```
                         Ff
                         (arrows a')
                         (arrows a')
                         ((D.hom Fa' Fb).er.tra
                            (D.comp Fa' Fb Fb
                               (D.comp Fb Gb Fb (arrows b) (a.arrows b))
                               Ff)
                            (D.comp Fa' Fb Fb (D.id Fb) Ff)
                            Ff
                            (D.cong Fa' Fb Fb
                               (D.comp Fb Gb Fb (arrows b) (a.arrows b))
                               (D.id Fb)
                               Ff
                               Ff
                               (p b)._2._2
                               ((D.hom Fa' Fb).er.ref Ff))
                            (D.left_unit Fa' Fb Ff))
                         ((D.hom Ga' Fa').er.ref (arrows a'))))))
    _2 =
      struct
      _1 = \(x::C.obj)-> (p x)._2._1
      _2 = \(x::C.obj)-> (p x)._2._2

ComponentsOfNatIsoAreIso
  (C::E_category)(D::E_category)
  (F::E_functor C D)(G::E_functor C D)
  (a::E_natural_transformation C D F G)
  (p::Iso (E_functorcategory C D) F G a)
 :: ForAll C.obj
          (\(h::C.obj)->
           Iso D (F.objectfunction h) (G.objectfunction h) (a.arrows h))
  = \(x::C.obj)->
    struct
    _1 = p._1.arrows x
    _2 =
      struct
      _1 = p._2._1 x
      _2 = p._2._2 x

{-# Alfa unfoldgoals off
brief on
hidetypeannots off
tall

nd
hiding on
var "E_functor" mixfix as "[_,_]"
var "Arrowfunction" hide 2
var "Arrowfunctionextensionality" hide 5
var "E_natural_transformation" hide 2 infix as "" with symbolfont
var "E_natural_transformation_SE" hide 2
var "E_functorcategory" mixfix as "[_,_]"
var "E_functorcomposition" hide 3 infix as "o" with symbolfont
var "E_idfunctor" as "1" with symbolfont
 #-}
```


E_productcategory.agda

```
--#include "E_categories.agda"

--#include "E_functors.agda"

E_productcategory (A::E_category)(B::E_category) :: E_category
  = struct {
      obj = Cart A.obj B.obj;
      hom = \(a::obj) -> \(b::obj) -> (A.hom a._1 b._1  (B.hom a._2 b._2));
      id =
        \(a::obj) ->
        struct {
          _1 = A.id a._1;
          _2 = B.id a._2;};
      comp =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(h::(hom b c).base) ->
        \(h'::(hom a b).base) ->
        struct {
          _1 = A.comp a._1 b._1 c._1 h._1 h'._1;
          _2 = B.comp a._2 b._2 c._2 h._2 h'._2;};
      left_unit =
        \(a::obj) ->
        \(b::obj) ->
        \(f::(hom a b).base) ->
        struct {
          _1 = A.left_unit a._1 b._1 f._1;
          _2 = B.left_unit a._2 b._2 f._2;};
      right_unit =
        \(a::obj) ->
        \(b::obj) ->
        \(f::(hom a b).base) ->
        struct {
          _1 = A.right_unit a._1 b._1 f._1;
          _2 = B.right_unit a._2 b._2 f._2;};
      assoc =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(d::obj) ->
        \(f::(hom c d).base) ->
        \(g::(hom b c).base) ->
        \(h::(hom a b).base) ->
        struct {
          _1 = A.assoc a._1 b._1 c._1 d._1 f._1 g._1 h._1;
          _2 = B.assoc a._2 b._2 c._2 d._2 f._2 g._2 h._2;};
      cong =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(f::(hom b c).base) ->
        \(f'::(hom b c).base) ->
        \(g::(hom a b).base) ->
        \(g'::(hom a b).base) ->
        \(h::Equal (hom b c) f f') ->
        \(h'::Equal (hom a b) g g') ->
        struct {
          _1 = A.cong a._1 b._1 c._1 f._1 f'._1 g._1 g'._1 h._1 h'._1;
          _2 = B.cong a._2 b._2 c._2 f._2 f'._2 g._2 g'._2 h._2 h'._2;};}

E_productfunctor (A::E_category)
                 (B::E_category)
                 (C::E_category)
                 (D::E_category)
                 (F::E_functor A C)
                 (G::E_functor B D)
  :: E_functor (E_productcategory A B) (E_productcategory C D)
  = struct {
      objectfunction =
        \(h::(E_productcategory A B).obj) ->
        struct {
        _1 = F.objectfunction h._1;
        _2 = G.objectfunction h._2;};
      arrowfunction =
        \(a::(E_productcategory A B).obj) ->
        \(b::(E_productcategory A B).obj) ->
```

```
        struct {
          op =
            \(h::((E_productcategory A B).hom a b).base) ->
            struct {
            _1 = (F.arrowfunction a._1 b._1).op h._1;
            _2 = (G.arrowfunction a._2 b._2).op h._2;};
          ext =
            \(x::((E_productcategory A B).hom a b).base) ->
            \(y::((E_productcategory A B).hom a b).base) ->
            \(h::((E_productcategory A B).hom a b).er.eq x y) ->
            struct {
              _1 = Arrowfunctionextensionality A C F a._1 b._1 x._1 y._1 h._1;
              _2 = Arrowfunctionextensionality B D G a._2 b._2 x._2 y._2 h._2;};};
      ax_preserve_id =
        \(a::(E_productcategory A B).obj) ->
        struct {
          _1 = F.ax_preserve_id a._1;
          _2 = G.ax_preserve_id a._2;};
      ax_preserve_composition =
        \(a::(E_productcategory A B).obj) ->
        \(b::(E_productcategory A B).obj) ->
        \(c::(E_productcategory A B).obj) ->
        \(f::Hom (E_productcategory A B) b c) ->
        \(g::Hom (E_productcategory A B) a b) ->
        struct {
          _1 = F.ax_preserve_composition a._1 b._1 c._1 f._1 g._1;
          _2 = G.ax_preserve_composition a._2 b._2 c._2 f._2 g._2;};}
{-# Alfa unfoldgoals off
brief on
hidetypeannots off
tall

nd
hiding on
var "E_productcategory" infix as "" with symbolfont
var "E_productfunctor" hide 4 infix as "" with symbolfont
 #-}
```

E_particular_categories.agda

```
--#include "E_categories.agda"

--#include "E_productcategory.agda"

E_emptycategory :: E_category
  = struct {
      obj = empty;
      hom = \(a::obj) -> case a of { };
      id = \(a::obj) -> case a of { };
      comp = \(a::obj) -> case a of { };
      left_unit = \(a::obj) -> case a of { };
      right_unit = \(a::obj) -> case a of { };
      assoc = \(a::obj) -> case a of { };
      cong = \(a::obj) -> case a of { };}

E_unitcategory :: E_category
  = struct {
      obj = Unit;
      hom = \(a::obj) -> \(b::obj) -> UNIT;
      id = \(a::obj) -> elt@_;
      comp =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(h::(hom b c).base) ->
        \(h'::(hom a b).base) ->
        elt@_;
      left_unit =
        \(a::obj) ->
        \(b::obj) ->
        \(f::(hom a b).base) ->
        elt@_;
      right_unit =
        \(a::obj) ->
        \(b::obj) ->
        \(f::(hom a b).base) ->
        elt@_;
      assoc =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(d::obj) ->
        \(f::(hom c d).base) ->
        \(g::(hom b c).base) ->
        \(h::(hom a b).base) ->
        elt@_;
      cong =
        \(a::obj) ->
        \(b::obj) ->
        \(c::obj) ->
        \(f::(hom b c).base) ->
        \(f'::(hom b c).base) ->
        \(g::(hom a b).base) ->
        \(g'::(hom a b).base) ->
        \(h::Equal (hom b c) f f') ->
        \(h'::Equal (hom a b) g g') ->
        elt@_;}

E_rightunitcatelim (C::E_category)
  :: E_functor (E_productcategory C E_unitcategory) C
  = struct {
      objectfunction = proj1 C.obj Unit;
      arrowfunction =
        \(a::(E_productcategory C E_unitcategory).obj) ->
        \(b::(E_productcategory C E_unitcategory).obj) ->
        struct {
          op = proj1 (C.hom a._1 b._1).base Unit;
          ext =
            \(x::((E_productcategory C E_unitcategory).hom a b).base) ->
            \(y::((E_productcategory C E_unitcategory).hom a b).base) ->
            \(h::((E_productcategory C E_unitcategory).hom a b).er.eq x y)->
            h._1;};
      ax_preserve_id =
        \(a::(E_productcategory C E_unitcategory).obj) ->
        reflexive (C.hom a._1 a._1) (C.id a._1);
      ax_preserve_composition =
        \(a::(E_productcategory C E_unitcategory).obj) ->
        \(b::(E_productcategory C E_unitcategory).obj) ->
        \(c::(E_productcategory C E_unitcategory).obj) ->
        \(f::Hom (E_productcategory C E_unitcategory) b c) ->
        \(g::Hom (E_productcategory C E_unitcategory) a b) ->
        reflexive (C.hom a._1 c._1) (C.comp a._1 b._1 c._1 f._1 g._1);}

E_leftunitcatelim (C::E_category)
  :: E_functor (E_productcategory E_unitcategory C) C
  = struct {
      objectfunction = proj2 Unit C.obj;
      arrowfunction =
        \(a::(E_productcategory E_unitcategory C).obj) ->
        \(b::(E_productcategory E_unitcategory C).obj) ->
        struct {
          op = proj2 Unit (C.hom a._2 b._2).base;
          ext =
            \(x::((E_productcategory E_unitcategory C).hom a b).base) ->
            \(y::((E_productcategory E_unitcategory C).hom a b).base) ->
```

```
            \(h::((E_productcategory E_unitcategory C).hom a b).er.eq x y)->
            h._2;};
      ax_preserve_id =
        \(a::(E_productcategory E_unitcategory C).obj) ->
        reflexive (C.hom a._2 a._2) (C.id a._2);
      ax_preserve_composition =
        \(a::(E_productcategory E_unitcategory C).obj) ->
        \(b::(E_productcategory E_unitcategory C).obj) ->
        \(c::(E_productcategory E_unitcategory C).obj) ->
        \(f::Hom (E_productcategory E_unitcategory C) b c) ->
        \(g::Hom (E_productcategory E_unitcategory C) a b) ->
        reflexive (C.hom a._2 c._2) (C.comp a._2 b._2 c._2 f._2 g._2);}
{-# Alfa unfoldgoals off
brief on
hidetypeannots off
tall

nd
hiding off
var "E_unitcategory" as "1"
 #-}
```

E_bicategory.agda

```
--#include "E_categories.agda"
--#include "E_productcategory.agda"
--#include "E_particular_categories.agda"
--#include "missingbits.agda"

prodassocright (A::E_category)(B::E_category)(C::E_category)
 :: E_functor (E_productcategory (E_productcategory A B) C)
              (E_productcategory A (E_productcategory B C))
  = struct
    objectfunction          =
      \(x::(E_productcategory (E_productcategory A B) C).obj)->
      struct { _1 = x._1._1; _2 = (struct { _1 = x._1._2; _2 = x._2;});}
    arrowfunction           =
      \(x::(E_productcategory (E_productcategory A B) C).obj)->
      \(y::(E_productcategory (E_productcategory A B) C).obj)->
      struct
      op  =
        \(f::((E_productcategory (E_productcategory A B) C).hom x y).base)->
        struct { _1 = f._1._1; _2 = (struct { _1 = f._1._2; _2 = f._2;});}
      ext =
        \(a::((E_productcategory (E_productcategory A B) C).hom x y).base)->
        \(b::((E_productcategory (E_productcategory A B) C).hom x y).base)->
        \(p::((E_productcategory (E_productcategory A B) C).hom x y).er.eq a b)->
        struct { _1 = p._1._1; _2 = (struct { _1 = p._1._2; _2 = p._2;});}
    ax_preserve_id          =
      \(x::(E_productcategory (E_productcategory A B) C).obj)->
      ((E_productcategory A (E_productcategory B C)).hom
         (objectfunction x) (objectfunction x)).er.ref
          ((E_productcategory A (E_productcategory B C)).id
             (objectfunction x))
    ax_preserve_composition =
      \(x::(E_productcategory (E_productcategory A B) C).obj)->
      \(y::(E_productcategory (E_productcategory A B) C).obj)->
      \(z::(E_productcategory (E_productcategory A B) C).obj)->
      \(f::Hom (E_productcategory (E_productcategory A B) C) y z)->
      \(g::Hom (E_productcategory (E_productcategory A B) C) x y)->
      ((E_productcategory A (E_productcategory B C)).hom
        (objectfunction x) (objectfunction z)).er.ref
         ((arrowfunction x z).op
           (compose (E_productcategory (E_productcategory A B) C) x y z f g))

E_bicategory :: #2
  = sig {
    obj  :: Type;
    hom  :: (a::obj)-> (b::obj)-> E_category;
    comp :: (a::obj)-> (b::obj)-> (c::obj)->
            E_functor (E_productcategory (hom b c) (hom a b))
                      (hom a c);
    identity :: (a::obj)-> E_functor E_unitcategory (hom a a);
    associativity :: (a::obj)-> (b::obj)-> (c::obj)-> (d::obj)->
            let A::E_category = hom c d
                B::E_category = hom b c
                C::E_category = hom a b
                D::E_category = hom b d
                E::E_category = hom a c
                F::E_category = hom a d
                AB::E_category = E_productcategory A B
                BC::E_category = E_productcategory B C
                DC::E_category = E_productcategory D C
                AE::E_category = E_productcategory A E
                AB_C::E_category = E_productcategory AB C
                A_BC::E_category = E_productcategory A BC
                Ftype::Set = E_functor AB_C F
                R::Ftype
                  = E_functorcomposition AB_C A_BC F
                      (E_functorcomposition A_BC AE F
                        (comp a c d)
                        (E_productfunctor A BC A E
                           (E_idfunctor A) (comp a b c)))
                      (prodassocright A B C)
                L::Ftype
                  = E_functorcomposition AB_C DC F (comp a b d)
                      (E_productfunctor AB C D C
                         (comp b c d) (E_idfunctor C))
                Fcat::E_category = E_functorcategory AB_C F
            in  Sigma (Fcat.hom L R).base (Iso Fcat L R);
    rightid :: (a::obj)-> (b::obj)->
            let hab::E_category = hom a b
                hab1::E_category = E_productcategory hab E_unitcategory
                habhaa::E_category = E_productcategory hab (hom a a)
                ftype::Set = E_functor hab1 hab
                fcat::E_category = E_functorcategory hab1 hab
                R::ftype = E_rightunitcatelim hab
                L::ftype
                  = E_functorcomposition hab1 habhaa hab (comp a a b)
                      (E_productfunctor hab E_unitcategory hab
                         (hom a a) (E_idfunctor hab) (identity a))
            in  Sigma (fcat.hom L R).base (Iso fcat L R);
    leftid :: (a::obj)-> (b::obj)->
            let hab::E_category = hom a b
                p1hab::E_category = E_productcategory E_unitcategory hab
                hbbhab::E_category = E_productcategory (hom b b) hab
                Ftype::Set = E_functor p1hab hab
                Fcat::E_category = E_functorcategory p1hab hab
                R::Ftype = E_leftunitcatelim hab
                L::Ftype
                  = E_functorcomposition p1hab hbbhab hab (comp a b b)
                      (E_productfunctor E_unitcategory hab (hom b b) hab
                         (identity b) (E_idfunctor hab))
            in  Sigma (Fcat.hom L R).base (Iso Fcat L R);
    apentagon :: (a::obj)-> (b::obj)-> (c::obj)-> (d::obj)-> (e::obj)->
            (f::(hom a b).obj)-> (g::(hom b c).obj)-> (h::(hom c d).obj)->
            (k::(hom d e).obj)->
            let homcat::E_category
                  = hom a e
                start::homcat.obj
                  = (comp a b e).objectfunction
                      (struct {
                        _1 = (comp b c e).objectfunction
                               (struct { _1 = (comp c d e).objectfunction
                                                (struct { _1 = k; _2 = h;});
```

```
                      _2 = g;});
        _2 = f;})
pl2::homcat.obj
  = (comp a c e).objectfunction
      (struct { _1 = (comp c d e).objectfunction
                       (struct { _1 = k; _2 = h;});
                _2 = (comp a b c).objectfunction
                       (struct { _1 = g; _2 = f;});})
end::homcat.obj
  = (comp a d e).objectfunction
      (struct {
        _1 = k;
        _2 = (comp a c d).objectfunction
               (struct {
                 _1 = h;
                 _2 = (comp a b c).objectfunction
                        (struct { _1 = g; _2 = f;});});})
pointr2::homcat.obj
  = (comp a b e).objectfunction
     (struct {
      _1 = (comp b d e).objectfunction
            (struct { _1 = k;
                      _2 = (comp b c d).objectfunction
                            (struct { _1 = h; _2 = g;});});
      _2 = f;})
pr3::homcat.obj
  = (comp a d e).objectfunction
      (struct {
        _1 = k;
        _2 = (comp a b d).objectfunction
               (struct { _1 = (comp b c d).objectfunction
                                (struct { _1 = h; _2 = g;});
                         _2 = f;});})
l1::Hom homcat start pl2
  = (associativity a b c e)._1.arrows
      (struct {
        _1 = struct { _1 = (comp c d e).objectfunction
                             (struct { _1 = k; _2 = h;});
                      _2 = g;};
        _2 =f;})
l2::Hom homcat pl2 end
  = (associativity a c d e)._1.arrows
      (struct { _1 = struct { _1 = k; _2 = h;};
                _2 = (comp a b c).objectfunction
                       (struct { _1 = g; _2 = f;});})
r1::Hom homcat start pointr2
  = ((comp a b e).arrowfunction
      (struct {
        _1 = (comp b c e).objectfunction
              (struct { _1 = (comp c d e).objectfunction
                               (struct { _1 = k; _2 = h;});
                        _2 = g;});
        _2 = f;})
      (struct {
        _1 = (comp b d e).objectfunction
              (struct { _1 = k;
                        _2 = (comp b c d).objectfunction
                               (struct { _1 = h; _2 = g;});});
        _2 = f;})).op
      (struct {
        _1 = (associativity b c d e)._1.arrows
              (struct { _1 = struct { _1 = k; _2 = h;};
                        _2 = g;});
        _2 = (hom a b).id f;})
r2::Hom homcat pointr2 pr3
  = (associativity a b d e)._1.arrows
      (struct {
        _1 = struct { _1 = k;
                      _2 = (comp b c d).objectfunction
                                         (struct { _1 = h; _2 = g;});};
            _2 = f;})
        r3::Hom homcat pr3 end
          = ((comp a d e).arrowfunction
              (struct {
                _1 = k;
                _2 = (comp a b d).objectfunction
                       (struct { _1 = (comp b c d).objectfunction
                                        (struct { _1 = h; _2 = g;});
                                 _2 = f;});})
              (struct {
                _1 = k;
                _2 = (comp a c d).objectfunction
                       (struct {
                         _1 = h;
                         _2 = (comp a b c).objectfunction
                                (struct { _1 = g; _2 = f;});});})).op
              (struct {
                _1 = (hom d e).id k;
                _2 = (associativity a b c d)._1.arrows
                       (struct { _1 = struct { _1 = h; _2 = g;};
                                 _2 = f;});})
    in  (homcat.hom start end).er.eq
          (compose homcat start pl2 end l2 l1)
          (compose homcat start pointr2 end
                   (compose homcat pointr2 pr3 end r3 r2) r1);
idtriangle :: (a::obj)-> (b::obj)-> (c::obj)->
    (f::(hom a b).obj)-> (g::(hom b c).obj)->
    let homcat::E_category
          = hom a c
        start::homcat.obj
          = (comp a b c).objectfunction
             (struct {
              _1 = (comp b b c).objectfunction
                     (struct {
                      _1 = g;
                      _2 = (identity b).objectfunction elt@_;});
              _2 = f;})
        mid::homcat.obj
          = (comp a b c).objectfunction
             (struct {
              _1 = g;
              _2 = (comp a b b).objectfunction
                     (struct { _1 = (identity b).objectfunction elt@_;
                               _2 = f;});})
        end::homcat.obj
          = (comp a b c).objectfunction (struct { _1 = g; _2 = f;})
        l::Hom homcat start end
          = ((comp a b c).arrowfunction
              (struct {
               _1 = (comp b b c).objectfunction
                      (struct {
                       _1 = g;
                       _2 = (identity b).objectfunction elt@_;});
               _2 = f;})
              (struct { _1 = g; _2 = f;})).op
              (struct { _1 = (rightid b c)._1.arrows
                                (struct { _1 = g; _2 = elt@_;});
                        _2 = (hom a b).id f;})
        r1::Hom homcat start mid
          = (associativity a b b c)._1.arrows
             (struct {
              _1 = struct { _1 = g;
                            _2 = (identity b).objectfunction elt@_;};
              _2 = f;})
        r2::Hom homcat mid end
          = ((comp a b c).arrowfunction
             (struct {
              _1 = g;
```

```
                      _2 = (comp a b b).objectfunction
                           (struct { _1 = (identity b).objectfunction elt@_;
                                     _2 = f;});})
                     (struct { _1 = g; _2 = f;})).op
                     (struct { _1 = (hom b c).id g;
                               _2 = (leftid a b)._1.arrows
                                     (struct { _1 = elt@_; _2 = f;});})
            in  (homcat.hom start end).er.eq
                 l (compose homcat start mid end r2 r1);
  }
```

ECat.agda

```
--#include "E_bicategory.agda"

HorizontalComposition
  (C::E_category)(D::E_category)(E::E_category)
  (F::E_functor C D)(G::E_functor D E)(H::E_functor C D)(K::E_functor D E)
  (a::E_natural_transformation C D F H)(b::E_natural_transformation D E G K)
 :: E_natural_transformation C E
      (E_functorcomposition C D E G F) (E_functorcomposition C D E K H)
  = struct
    arrows        =
      \(a'::C.obj)->
      E.comp ((E_functorcomposition C D E G F).objectfunction a')
             (K.objectfunction (F.objectfunction a'))
             ((E_functorcomposition C D E K H).objectfunction a')
             ((K.arrowfunction (F.objectfunction a')
                               (H.objectfunction a')).op (a.arrows a'))
             (b.arrows (F.objectfunction a'))
    ax_naturality =
      \(a'::C.obj)->
      \(b'::C.obj)->
      \(f::Hom C a' b')->
      let GF::E_functor C E = E_functorcomposition C D E G F
          KF::E_functor C E = E_functorcomposition C D E K F
          KH::E_functor C E = E_functorcomposition C D E K H
          Fa'::D.obj = F.objectfunction a'
          Fb'::D.obj = F.objectfunction b'
          Ha'::D.obj = H.objectfunction a'
          Hb'::D.obj = H.objectfunction b'
          GFa'::E.obj = GF.objectfunction a'
          KFa'::E.obj = KF.objectfunction a'
          KHa'::E.obj = KH.objectfunction a'
          GFb'::E.obj = GF.objectfunction b'
          KFb'::E.obj = KF.objectfunction b'
          KHb'::E.obj = KH.objectfunction b'
          homset::SE  = E.hom GFa' KHb'
          homset2::SE = E.hom KFa' KHb'
          Ff::(D.hom Fa' Fb').base = (F.arrowfunction a' b').op f
          Hf::(D.hom Ha' Hb').base = (H.arrowfunction a' b').op f
          GFf::(E.hom GFa' GFb').base = (GF.arrowfunction a' b').op f
          KFf::(E.hom KFa' KFb').base = (KF.arrowfunction a' b').op f
          KHf::(E.hom KHa' KHb').base = (KH.arrowfunction a' b').op f
          Kaa'::(E.hom KFa' KHa').base
            = (K.arrowfunction Fa' Ha').op (a.arrows a')
          Kab'::(E.hom KFb' KHb').base
            = (K.arrowfunction Fb' Hb').op (a.arrows b')
      in  homset.er.tra (E.comp GFa' GFb' KHb' (arrows b') GFf)
            (E.comp GFa' KFa' KHb'
               (E.comp KFa' KFb' KHb' Kab' KFf) (b.arrows Fa'))
            (E.comp GFa' KHa' KHb' KHf (arrows a'))
            (homset.er.tra
               (E.comp GFa' GFb' KHb' (arrows b') GFf)
               (E.comp GFa' KFb' KHb'
                       Kab' (E.comp GFa' GFb' KFb' (b.arrows Fb') GFf))
               (E.comp GFa' KFa' KHb'
                       (E.comp KFa' KFb' KHb' Kab' KFf) (b.arrows Fa'))
               (E.assoc GFa' GFb' KFb' KHb' Kab' (b.arrows Fb') GFf)
               (homset.er.tra
                  (E.comp GFa' KFb' KHb'
                          Kab' (E.comp GFa' GFb' KFb' (b.arrows Fb') GFf))
                  (E.comp GFa' KFb' KHb'
                          Kab' (E.comp GFa' KFa' KFb' KFf (b.arrows Fa')))
                  (E.comp GFa' KFa' KHb'
                          (E.comp KFa' KFb' KHb' Kab' KFf) (b.arrows Fa'))
                  (E.cong GFa' KFb' KHb' Kab' Kab'
                          (E.comp GFa' GFb' KFb' (b.arrows Fb') GFf)
                          (E.comp GFa' KFa' KFb' KFf (b.arrows Fa'))
                          ((E.hom KFb' KHb').er.ref Kab')
                          (b.ax_naturality Fa' Fb'
                                           ((F.arrowfunction a' b').op f)))
                  (homset.er.sym
                     (E.comp GFa' KFa' KHb'
                             (E.comp KFa' KFb' KHb' Kab' KFf) (b.arrows Fa'))
                     (E.comp GFa' KFb' KHb'
                             Kab' (E.comp GFa' KFa' KFb' KFf (b.arrows Fa')))
                     (E.assoc GFa' KFa' KFb' KHb' Kab' KFf (b.arrows Fa')))))
            (homset.er.tra
               (E.comp GFa' KFa' KHb'
                       (E.comp KFa' KFb' KHb' Kab' KFf) (b.arrows Fa'))
               (E.comp GFa' KFa' KHb'
                       (E.comp KFa' KHa' KHb' KHf Kaa') (b.arrows Fa'))
               (E.comp GFa' KHa' KHb' KHf (arrows a'))
               (E.cong GFa' KFa' KHb'
                  (E.comp KFa' KFb' KHb' Kab' KFf)
                  (E.comp KFa' KHa' KHb' KHf Kaa')
                  (b.arrows Fa')
                  (b.arrows Fa')
                  (homset2.er.tra
                     (E.comp KFa' KFb' KHb' Kab' KFf)
                     ((K.arrowfunction Fa' Hb').op
                         (D.comp Fa' Fb' Hb' (a.arrows b') Ff))
                     (E.comp KFa' KHa' KHb' KHf Kaa')
                     (homset2.er.sym
                        ((K.arrowfunction Fa' Hb').op
                            (D.comp Fa' Fb' Hb' (a.arrows b') Ff))
                        (E.comp KFa' KFb' KHb' Kab' KFf)
                        (K.ax_preserve_composition Fa' Fb' Hb'
                                                   (a.arrows b') Ff))
                     (homset2.er.tra
                        ((K.arrowfunction Fa' Hb').op
                            (D.comp Fa' Fb' Hb' (a.arrows b') Ff))
                        ((K.arrowfunction Fa' Hb').op
                            (D.comp Fa' Ha' Hb' Hf (a.arrows a')))
                        (E.comp KFa' KHa' KHb' KHf Kaa')
                        ((K.arrowfunction Fa' Hb').ext
                            (D.comp Fa' Fb' Hb' (a.arrows b') Ff)
                            (D.comp Fa' Ha' Hb' Hf (a.arrows a'))
                            (a.ax_naturality a' b' f))
                        (K.ax_preserve_composition Fa' Ha' Hb'
                                                   Hf (a.arrows a'))))
                  ((E.hom GFa' KFa').er.ref (b.arrows Fa')))
               (E.assoc GFa' KFa' KHa' KHb' KHf Kaa' (b.arrows Fa')))

HorizontalCompositionIsExtensional
  (C::E_category)(D::E_category)(E::E_category)
  (F::E_functor C D)(G::E_functor D E)
  (H::E_functor C D)(K::E_functor D E)
  (a::E_natural_transformation C D F H)
  (b::E_natural_transformation D E G K)
  (c::E_natural_transformation C D F H)
  (d::E_natural_transformation D E G K)
  (aeqc::(E_natural_transformation_SE C D F H).er.eq a c)
  (beqd::(E_natural_transformation_SE D E G K).er.eq b d)
```

```
 :: (E_natural_transformation_SE C E
      (E_functorcomposition C D E G F) (E_functorcomposition C D E K H)).er.eq
        (HorizontalComposition C D E F G H K a b)
        (HorizontalComposition C D E F G H K c d)
  = \(x::C.obj)->
    E.cong ((E_functorcomposition C D E G F).objectfunction x)
           (K.objectfunction (F.objectfunction x))
           ((E_functorcomposition C D E K H).objectfunction x)
           ((K.arrowfunction (F.objectfunction x) (H.objectfunction x)).op
              (a.arrows x))
           ((K.arrowfunction (F.objectfunction x) (H.objectfunction x)).op
              (c.arrows x))
           (b.arrows (F.objectfunction x))
           (d.arrows (F.objectfunction x))
           ((K.arrowfunction (F.objectfunction x) (H.objectfunction x)).ext
               (a.arrows x) (c.arrows x) (aeqc x))
           (beqd (F.objectfunction x))

HorizontalCompositionPreservesIdentity
  (C::E_category)(D::E_category)(E::E_category)
  (F::E_functor C D)(G::E_functor D E)
 :: (E_natural_transformation_SE C E
      (E_functorcomposition C D E G F)
      (E_functorcomposition C D E G F)).er.eq
        (HorizontalComposition C D E F G F G
           ((E_functorcategory C D).id F) ((E_functorcategory D E).id G))
        ((E_functorcategory C E).id (E_functorcomposition C D E G F))
  = \(x::C.obj)->
    let Fx::D.obj = F.objectfunction x
        GF::E_functor C E = E_functorcomposition C D E G F
        GFx::E.obj = GF.objectfunction x
        homcat::SE = E.hom GFx GFx
        idF::E_natural_transformation C D F F
          = (E_functorcategory C D).id F
        GidFx::homcat.base
          = (G.arrowfunction Fx Fx).op (idF.arrows x)
        idG::E_natural_transformation D E G G
          = (E_functorcategory D E).id G
        idGF::E_natural_transformation C E GF GF
          = (E_functorcategory C E).id GF
    in  homcat.er.tra
          (E.comp GFx GFx GFx GidFx (idG.arrows Fx))
          GidFx
          (idGF.arrows x)
          (E.right_unit GFx GFx GidFx)
          (G.ax_preserve_id Fx)

MiddleFourExchange (C::E_category)(D::E_category)(E::E_category)
  (F::E_functor C D)(G::E_functor D E)
  (F'::E_functor C D)(G'::E_functor D E)
  (F''::E_functor C D)(G''::E_functor D E)
  (a::E_natural_transformation C D F F')
  (b::E_natural_transformation D E G G')
  (a'::E_natural_transformation C D F' F'')
  (b'::E_natural_transformation D E G' G'')
 :: (E_natural_transformation_SE C E
      (E_functorcomposition C D E G F)
      (E_functorcomposition C D E G'' F'')).er.eq
        (HorizontalComposition C D E F G F'' G''
           ((E_functorcategory C D).comp F F' F'' a' a)
           ((E_functorcategory D E).comp G G' G'' b' b))
        ((E_functorcategory C E).comp
           (E_functorcomposition C D E G F)
           (E_functorcomposition C D E G' F')
           (E_functorcomposition C D E G'' F'')
           (HorizontalComposition C D E F' G' F'' G'' a' b')
           (HorizontalComposition C D E F G F' G' a b))
  = \(x::C.obj)->
    let GF::E_functor C E = E_functorcomposition C D E G F
```

```
        G'F'::E_functor C E = E_functorcomposition C D E G' F'
        G''F''::E_functor C E = E_functorcomposition C D E G'' F''
        Fcat::E_category = E_functorcategory C E
        Fx::D.obj = F.objectfunction x
        F'x::D.obj = F'.objectfunction x
        F''x::D.obj = F''.objectfunction x
        GFx::E.obj = G.objectfunction Fx
        G'Fx::E.obj = G'.objectfunction Fx
        G''Fx::E.obj = G''.objectfunction Fx
        G'F'x::E.obj = G'.objectfunction F'x
        G''F'x::E.obj = G''.objectfunction F'x
        G''F''x::E.obj = G''.objectfunction F''x
        G'ax::(E.hom G'Fx G'F'x).base
          = (G'.arrowfunction Fx F'x).op (a.arrows x)
        G''ax::(E.hom G''Fx G''F'x).base
          = (G''.arrowfunction Fx F'x).op (a.arrows x)
        G''a'x::(E.hom G''F'x G''F''x).base
          = (G''.arrowfunction F'x F''x).op (a'.arrows x)
        bFx::(E.hom GFx G'Fx).base = b.arrows Fx
        b'Fx::(E.hom G'Fx G''Fx).base = b'.arrows Fx
        b'F'x::(E.hom G'F'x G''F'x).base = b'.arrows F'x
        homset::SE = E.hom GFx G''F''x
        bstara::(Fcat.hom GF G'F').base
          = HorizontalComposition C D E F G F' G' a b
        b'stara'::(Fcat.hom G'F' G''F'').base
          = HorizontalComposition C D E F' G' F'' G'' a' b'
        a'oa::E_natural_transformation C D F F''
          = (E_functorcategory C D).comp F F' F'' a' a
        b'ob::E_natural_transformation D E G G''
          = (E_functorcategory D E).comp G G' G'' b' b
        b'obstara'oa::(Fcat.hom GF G''F'').base
          = HorizontalComposition C D E F G F'' G'' a'oa b'ob
        G''a'oax::(E.hom G''Fx G''F''x).base
          = (G''.arrowfunction Fx F''x).op (a'oa.arrows x)
    in  homset.er.tra
          (b'obstara'oa.arrows x)
          (E.comp GFx G'Fx G''F''x
             (E.comp G'Fx G''F'x G''F''x
                     G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax)) bFx)
          ((Fcat.comp GF G'F' G''F'' b'stara' bstara).arrows x)
          (homset.er.tra
            (b'obstara'oa.arrows x)
            (E.comp GFx G'Fx G''F''x
               (E.comp G'Fx G''Fx G''F''x G''a'oax b'Fx) bFx)
            (E.comp GFx G'Fx G''F''x
               (E.comp G'Fx G''F'x G''F''x
                  G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax)) bFx)
            (homset.er.sym
              (E.comp GFx G'Fx G''F''x
                 (E.comp G'Fx G''Fx G''F''x G''a'oax b'Fx) bFx)
              (b'obstara'oa.arrows x)
              (E.assoc GFx G'Fx G''Fx G''F''x G''a'oax b'Fx bFx))
            (E.cong GFx G'Fx G''F''x
               (E.comp G'Fx G''Fx G''F''x G''a'oax b'Fx)
               (E.comp G'Fx G''F'x G''F''x
                  G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax))
               bFx
               bFx
               ((E.hom G'Fx G''F''x).er.tra
                  (E.comp G'Fx G''Fx G''F''x G''a'oax b'Fx)
                  (E.comp G'Fx G''F'x G''F''x
                     G''a'x (E.comp G'Fx G''Fx G''F'x G''ax b'Fx))
                  (E.comp G'Fx G''F'x G''F''x
                     G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax))
                  ((E.hom G'Fx G''F''x).er.tra
                     (E.comp G'Fx G''Fx G''F''x G''a'oax b'Fx)
                     (E.comp G'Fx G''Fx G''F''x
                        (E.comp G''Fx G''F'x G''F''x G''a'x G''ax) b'Fx)
                     (E.comp G'Fx G''F'x G''F''x
```

```
                        G''a'x (E.comp G'Fx G''Fx G''F'x G''ax b'Fx))
                     (E.cong G'Fx G''Fx G''F''x
                        G''a'oax
                        (E.comp G''Fx G''F'x G''F''x G''a'x G''ax)
                        b'Fx
                        b'Fx
                        (G''.ax_preserve_composition Fx F'x F''x
                             (a'.arrows x) (a.arrows x))
                        ((E.hom G'Fx G''Fx).er.ref b'Fx))
                     (E.assoc G'Fx G''Fx G''F'x G''F''x G''a'x G''ax b'Fx))
                  (E.cong G'Fx G''F'x G''F''x
                     G''a'x
                     G''a'x
                     (E.comp G'Fx G''Fx G''F'x G''ax b'Fx)
                     (E.comp G'Fx G'F'x G''F'x b'F'x G'ax)
                     ((E.hom G''F'x G''F''x).er.ref G''a'x)
                     ((E.hom G'Fx G''F'x).er.sym
                         (E.comp G'Fx G'F'x G''F'x b'F'x G'ax)
                         (E.comp G'Fx G''Fx G''F'x G''ax b'Fx)
                         (b'.ax_naturality Fx F'x (a.arrows x)))))
               ((E.hom GFx G'Fx).er.ref bFx)))
          (homset.er.tra
            (E.comp GFx G'Fx G''F''x
               (E.comp G'Fx G''F'x G''F''x
                  G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax)) bFx)
            (E.comp GFx G'Fx G''F''x
               (E.comp G'Fx G'F'x G''F''x (b'stara'.arrows x) G'ax) bFx)
            ((Fcat.comp GF G'F' G''F'' b'stara' bstara).arrows x)
            (E.cong GFx G'Fx G''F''x
               (E.comp G'Fx G''F'x G''F''x
                  G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax))
               (E.comp G'Fx G'F'x G''F''x (b'stara'.arrows x) G'ax)
               bFx
               bFx
               ((E.hom G'Fx G''F''x).er.sym
                   (E.comp G'Fx G'F'x G''F''x (b'stara'.arrows x) G'ax)
                   (E.comp G'Fx G''F'x G''F''x
                      G''a'x (E.comp G'Fx G'F'x G''F'x b'F'x G'ax))
                   (E.assoc G'Fx G'F'x G''F'x G''F''x G''a'x b'F'x G'ax))
               ((E.hom GFx G'Fx).er.ref bFx))
            (E.assoc GFx G'Fx G'F'x G''F''x (b'stara'.arrows x) G'ax bFx))

FunctorCompositionAsFunctor (A::E_category)(B::E_category)(C::E_category)
 :: E_functor
      (E_productcategory (E_functorcategory B C) (E_functorcategory A B))
      (E_functorcategory A C)
  = let FAB::E_category = E_functorcategory A B
        FBC::E_category = E_functorcategory B C
    in  struct
        objectfunction          =
          \(h::(E_productcategory FBC FAB).obj)->
          E_functorcomposition A B C h._1 h._2
        arrowfunction           =
          \(a::(E_productcategory FBC FAB).obj)->
          \(b::(E_productcategory FBC FAB).obj)->
          struct
          op  =
            \(h::((E_productcategory FBC FAB).hom a b).base)->
            HorizontalComposition A B C a._2 a._1 b._2 b._1 h._2 h._1
          ext =
            \(x::((E_productcategory FBC FAB).hom a b).base)->
            \(y::((E_productcategory FBC FAB).hom a b).base)->
            \(h::((E_productcategory FBC FAB).hom a b).er.eq x y)->
            HorizontalCompositionIsExtensional
              A B C a._2 a._1 b._2 b._1 x._2 x._1 y._2 y._1 h._2 h._1
        ax_preserve_id          =
          \(a::(E_productcategory FBC FAB).obj)->
          HorizontalCompositionPreservesIdentity A B C a._2 a._1
        ax_preserve_composition =
```

```
          \(a::(E_productcategory FBC FAB).obj)->
          \(b::(E_productcategory FBC FAB).obj)->
          \(c::(E_productcategory FBC FAB).obj)->
          \(f::((E_productcategory FBC FAB).hom b c).base)->
          \(g::((E_productcategory FBC FAB).hom a b).base)->
          MiddleFourExchange
            A B C a._2 a._1 b._2 b._1 c._2 c._1 g._2 g._1 f._2 f._1

IdFunctorWithId (C::E_category)
 :: E_functor E_unitcategory (E_functorcategory C C)
  = struct
    objectfunction          =
      \(h::E_unitcategory.obj)-> E_idfunctor C
    arrowfunction           =
      \(a::E_unitcategory.obj)->
      \(b::E_unitcategory.obj)->
      struct
      op  =
        \(h::(E_unitcategory.hom elt@_ elt@_).base)->
        (E_functorcategory C C).id (E_idfunctor C)
      ext =
        \(x::(E_unitcategory.hom elt@_ elt@_).base)->
        \(y::(E_unitcategory.hom elt@_ elt@_).base)->
        \(h::(E_unitcategory.hom elt@_ elt@_).er.eq x y)->
        ((E_functorcategory C C).hom
            (E_idfunctor C) (E_idfunctor C)).er.ref (op elt@_)
    ax_preserve_id          =
      \(a::E_unitcategory.obj)->
      (E_natural_transformation_SE C C (E_idfunctor C) (E_idfunctor C)).er.ref
         ((E_functorcategory C C).id (E_idfunctor C))
    ax_preserve_composition =
      \(a::E_unitcategory.obj)->
      \(b::E_unitcategory.obj)->
      \(c::E_unitcategory.obj)->
      \(f::(E_unitcategory.hom b c).base)->
      \(g::(E_unitcategory.hom a b).base)->
      let CC::E_category = E_functorcategory C C
          idC::CC.obj = E_idfunctor C
          ididC::(CC.hom idC idC).base = CC.id idC
      in  (E_natural_transformation_SE C C idC idC).er.sym
             (CC.comp idC idC idC ididC ididC) ididC
             (CC.right_unit idC idC ididC)

FunctorCompAssocNatTrans
  (A::E_category)(B::E_category)(C::E_category)(D::E_category)
  (F::E_functor A B)(G::E_functor B C)(H::E_functor C D)
 :: E_natural_transformation A D
      (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
      (E_functorcomposition A C D H (E_functorcomposition A B C G F))
  = struct
    arrows        =
      \(a::A.obj)->
      D.id ((E_functorcomposition A B D
               (E_functorcomposition B C D H G) F).objectfunction a)
    ax_naturality =
      \(a::A.obj)->
      \(b::A.obj)->
      \(f::Hom A a b)->
      let HG_F::E_functor A D
            = E_functorcomposition A B D (E_functorcomposition B C D H G) F
          H_GF::E_functor A D
            = E_functorcomposition A C D H (E_functorcomposition A B C G F)
          HG_Fa::D.obj = HG_F.objectfunction a
          HG_Fb::D.obj = HG_F.objectfunction b
          H_GFa::D.obj = H_GF.objectfunction a
          H_GFb::D.obj = H_GF.objectfunction b
          HG_Ff::(D.hom HG_Fa HG_Fb).base = (HG_F.arrowfunction a b).op f
          H_GFf::(D.hom H_GFa H_GFb).base = (H_GF.arrowfunction a b).op f
      in  (D.hom HG_Fa H_GFb).er.tra
```

```
            (D.comp HG_Fa HG_Fb H_GFb (arrows b) HG_Ff)
            HG_Ff
            (D.comp HG_Fa H_GFa H_GFb H_GFf (arrows a))
            (D.left_unit HG_Fa H_GFb HG_Ff)
            ((D.hom HG_Fa H_GFb).er.sym
               (D.comp HG_Fa H_GFa H_GFb H_GFf (arrows a))
               HG_Ff
               (D.right_unit HG_Fa H_GFb HG_Ff))

FunctorCompAssocNatTransNatural
  (A::E_category)(B::E_category)(C::E_category)(D::E_category)
  (F::E_functor A B)(G::E_functor B C)(H::E_functor C D)
  (F'::E_functor A B)(G'::E_functor B C)(H'::E_functor C D)
  (a::E_natural_transformation A B F F')
  (b::E_natural_transformation B C G G')
  (c::E_natural_transformation C D H H')
 :: (E_natural_transformation_SE A D
       (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
       (E_functorcomposition A C D
          H' (E_functorcomposition A B C G' F'))).er.eq
      ((E_functorcategory A D).comp
          (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
          (E_functorcomposition A B D (E_functorcomposition B C D H' G') F')
          (E_functorcomposition A C D H' (E_functorcomposition A B C G' F'))
          (FunctorCompAssocNatTrans A B C D F' G' H')
          (HorizontalComposition A B D
             F (E_functorcomposition B C D H G)
             F' (E_functorcomposition B C D H' G')
             a (HorizontalComposition B C D G H G' H' b c)))
      ((E_functorcategory A D).comp
          (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
          (E_functorcomposition A C D H (E_functorcomposition A B C G F))
          (E_functorcomposition A C D H' (E_functorcomposition A B C G' F'))
          (HorizontalComposition A C D
             (E_functorcomposition A B C G F) H
             (E_functorcomposition A B C G' F') H'
             (HorizontalComposition A B C F G F' G' a b) c)
          (FunctorCompAssocNatTrans A B C D F G H))
  = \(x::A.obj)->
    let HG::E_functor B D = E_functorcomposition B C D H G
        H'G'::E_functor B D = E_functorcomposition B C D H' G'
        GF::E_functor A C = E_functorcomposition A B C G F
        G'F'::E_functor A C = E_functorcomposition A B C G' F'
        HG_F::E_functor A D = E_functorcomposition A B D HG F
        H'G'_F'::E_functor A D = E_functorcomposition A B D H'G' F'
        H_GF::E_functor A D = E_functorcomposition A C D H GF
        H'_G'F'::E_functor A D = E_functorcomposition A C D H' G'F'
        Fx::B.obj = F.objectfunction x
        GFx::C.obj = G.objectfunction Fx
        HGFx::D.obj = H.objectfunction GFx
        F'x::B.obj = F'.objectfunction x
        G'F'x::C.obj = G'.objectfunction F'x
        H'G'F'x::D.obj = H'.objectfunction G'F'x
        G'Fx::C.obj = G'.objectfunction Fx
        H'GFx::D.obj = H'.objectfunction GFx
        H'G'Fx::D.obj = H'.objectfunction G'Fx
        ax::(B.hom Fx F'x).base = a.arrows x
        bFx::(C.hom GFx G'Fx).base = b.arrows Fx
        cGFx::(D.hom HGFx H'GFx).base = c.arrows GFx
        G'ax::(C.hom G'Fx G'F'x).base = (G'.arrowfunction Fx F'x).op ax
        H'G'ax::(D.hom H'G'Fx H'G'F'x).base = (H'G'.arrowfunction Fx F'x).op ax
        H'bFx::(D.hom H'GFx H'G'Fx).base = (H'.arrowfunction GFx G'Fx).op bFx
        homset::SE = D.hom HGFx H'G'F'x
        cstarb::E_natural_transformation B D HG H'G'
          = HorizontalComposition B C D G H G' H' b c
        bstara::E_natural_transformation A C GF G'F'
          = HorizontalComposition A B C F G F' G' a b
        bstarax::(C.hom GFx G'F'x).base = bstara.arrows x
        cstarb_stara::E_natural_transformation A D HG_F H'G'_F'
          = HorizontalComposition A B D F HG F' H'G' a cstarb
        cstarb_starax::homset.base = cstarb_stara.arrows x
        H'bstarax::(D.hom H'GFx H'G'F'x).base
          = (H'.arrowfunction GFx G'F'x).op bstarax
        cstar_bstara::E_natural_transformation A D H_GF H'_G'F'
          = HorizontalComposition A C D GF H G'F' H' bstara c
        cstar_bstarax::homset.base
          = cstar_bstara.arrows x
        assocFGH::E_natural_transformation A D HG_F H_GF
          = FunctorCompAssocNatTrans A B C D F G H
        assocF'G'H'::E_natural_transformation A D H'G'_F' H'_G'F'
          = FunctorCompAssocNatTrans A B C D F' G' H'
        Fcat::E_category = E_functorcategory A D
        cGFxo_HbFxoHGax::homset.base
          = D.comp HGFx H'GFx H'G'F'x
              (D.comp H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx) cGFx
    in  homset.er.tra
          ((Fcat.comp HG_F H'G'_F' H'_G'F' assocF'G'H' cstarb_stara).arrows x)
          cstarb_starax
          ((Fcat.comp HG_F H_GF H'_G'F' cstar_bstara assocFGH).arrows x)
          (D.left_unit HGFx H'G'F'x cstarb_starax)
          (homset.er.sym
             ((Fcat.comp HG_F H_GF H'_G'F' cstar_bstara assocFGH).arrows x)
             cstarb_starax
             (homset.er.tra
                ((Fcat.comp HG_F H_GF H'_G'F' cstar_bstara assocFGH).arrows x)
                cstar_bstarax
                cstarb_starax
                (D.right_unit HGFx H'G'F'x cstar_bstarax)
                (homset.er.tra
                   cstar_bstarax
                   cGFxo_HbFxoHGax
                   cstarb_starax
                   (D.cong HGFx H'GFx H'G'F'x
                      H'bstarax (D.comp H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx)
                      cGFx cGFx
                      (H'.ax_preserve_composition GFx G'Fx G'F'x G'ax bFx)
                      ((D.hom HGFx H'GFx).er.ref cGFx))
                   (D.assoc HGFx H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx cGFx))))

RevFunctorCompAssocNatTrans
  (A::E_category)(B::E_category)(C::E_category)(D::E_category)
  (F::E_functor A B)(G::E_functor B C)(H::E_functor C D)
 :: E_natural_transformation A D
      (E_functorcomposition A C D H (E_functorcomposition A B C G F))
      (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
  = struct
    arrows        =
      \(a::A.obj)->
      D.id ((E_functorcomposition A B D
               (E_functorcomposition B C D H G) F).objectfunction a)
    ax_naturality =
      \(a::A.obj)->
      \(b::A.obj)->
      \(f::Hom A a b)->
      let HG_F::E_functor A D
            = E_functorcomposition A B D (E_functorcomposition B C D H G) F
          H_GF::E_functor A D
            = E_functorcomposition A C D H (E_functorcomposition A B C G F)
          HG_Fa::D.obj = HG_F.objectfunction a
          HG_Fb::D.obj = HG_F.objectfunction b
          H_GFa::D.obj = H_GF.objectfunction a
          H_GFb::D.obj = H_GF.objectfunction b
          HG_Ff::(D.hom HG_Fa HG_Fb).base = (HG_F.arrowfunction a b).op f
          H_GFf::(D.hom H_GFa H_GFb).base = (H_GF.arrowfunction a b).op f
      in  (D.hom H_GFa HG_Fb).er.tra
            (D.comp H_GFa H_GFb HG_Fb (arrows b) H_GFf)
            HG_Ff
            (D.comp H_GFa HG_Fa HG_Fb HG_Ff (arrows a))
```

```
          (D.left_unit HG_Fa H_GFb HG_Ff)
          ((D.hom H_GFa HG_Fb).er.sym
             (D.comp H_GFa HG_Fa HG_Fb HG_Ff (arrows a))
             HG_Ff
             (D.right_unit H_GFa HG_Fb HG_Ff))

RevFunctorCompAssocNatTransNatural
  (A::E_category)(B::E_category)(C::E_category)(D::E_category)
  (F::E_functor A B)(G::E_functor B C)(H::E_functor C D)
  (F'::E_functor A B)(G'::E_functor B C)(H'::E_functor C D)
  (a::E_natural_transformation A B F F')
  (b::E_natural_transformation B C G G')
  (c::E_natural_transformation C D H H')
 :: (E_natural_transformation_SE A D
       (E_functorcomposition A C D H (E_functorcomposition A B C G F))
       (E_functorcomposition A B D
          (E_functorcomposition B C D H' G') F')).er.eq
      ((E_functorcategory A D).comp
          (E_functorcomposition A C D H (E_functorcomposition A B C G F))
          (E_functorcomposition A C D H' (E_functorcomposition A B C G' F'))
          (E_functorcomposition A B D (E_functorcomposition B C D H' G') F')
          (RevFunctorCompAssocNatTrans A B C D F' G' H')
          (HorizontalComposition A C D
             (E_functorcomposition A B C G F) H
             (E_functorcomposition A B C G' F') H'
             (HorizontalComposition A B C F G F' G' a b) c))
      ((E_functorcategory A D).comp
          (E_functorcomposition A C D H (E_functorcomposition A B C G F))
          (E_functorcomposition A B D (E_functorcomposition B C D H G) F)
          (E_functorcomposition A B D (E_functorcomposition B C D H' G') F')
          (HorizontalComposition A B D
             F (E_functorcomposition B C D H G)
             F' (E_functorcomposition B C D H' G')
             a (HorizontalComposition B C D G H G' H' b c))
          (RevFunctorCompAssocNatTrans A B C D F G H))
  = \(x::A.obj)->
    let Fcat::E_category = E_functorcategory A D
        GF::E_functor A C = E_functorcomposition A B C G F
        H_GF::E_functor A D = E_functorcomposition A C D H GF
        HG::E_functor B D = E_functorcomposition B C D H G
        HG_F::E_functor A D = E_functorcomposition A B D HG F
        G'F'::E_functor A C = E_functorcomposition A B C G' F'
        H'_G'F'::E_functor A D = E_functorcomposition A C D H' G'F'
        H'G'::E_functor B D = E_functorcomposition B C D H' G'
        H'G'_F'::E_functor A D = E_functorcomposition A B D H'G' F'
        assocFGH::E_natural_transformation A D H_GF HG_F
          = RevFunctorCompAssocNatTrans A B C D F G H
        assocF'G'H'::E_natural_transformation A D H'_G'F' H'G'_F'
          = RevFunctorCompAssocNatTrans A B C D F' G' H'
        Fx::B.obj = F.objectfunction x
        GFx::C.obj = G.objectfunction Fx
        HGFx::D.obj = H.objectfunction GFx
        F'x::B.obj = F'.objectfunction x
        G'F'x::C.obj = G'.objectfunction F'x
        H'G'F'x::D.obj = H'.objectfunction G'F'x
        H'GFx::D.obj = H'.objectfunction GFx
        G'Fx::C.obj = G'.objectfunction Fx
        H'G'Fx::D.obj = H'.objectfunction G'Fx
        homset::SE = D.hom HGFx H'G'F'x
        cstarb::E_natural_transformation B D HG H'G'
          = HorizontalComposition B C D G H G' H' b c
        cstarb_stara::E_natural_transformation A D HG_F H'G'_F'
          = HorizontalComposition A B D F HG F' H'G' a cstarb
        cstarb_starax::homset.base = cstarb_stara.arrows x
        bstara::E_natural_transformation A C GF G'F'
          = HorizontalComposition A B C F G F' G' a b
        cstar_bstara::E_natural_transformation A D H_GF H'_G'F'
          = HorizontalComposition A C D GF H G'F' H' bstara c
        cstar_bstarax::homset.base = cstar_bstara.arrows x
        cGFx::(D.hom HGFx H'GFx).base = c.arrows GFx
        H'bstarax::(D.hom H'GFx H'G'F'x).base
          = (H'.arrowfunction GFx G'F'x).op (bstara.arrows x)
        H'bFx::(D.hom H'GFx H'G'Fx).base
          = (H'.arrowfunction GFx G'Fx).op (b.arrows Fx)
        H'G'ax::(D.hom H'G'Fx H'G'F'x).base
          = (H'G'.arrowfunction Fx F'x).op (a.arrows x)
    in  homset.er.tra
          ((Fcat.comp H_GF H'_G'F' H'G'_F' assocF'G'H' cstar_bstara).arrows x)
          cstar_bstarax
          ((Fcat.comp H_GF HG_F H'G'_F' cstarb_stara assocFGH).arrows x)
          (D.left_unit HGFx H'G'F'x cstar_bstarax)
          (homset.er.tra
             cstar_bstarax
             cstarb_starax
             ((Fcat.comp H_GF HG_F H'G'_F' cstarb_stara assocFGH).arrows x)
             (homset.er.tra
                cstar_bstarax
                (D.comp HGFx H'GFx H'G'F'x
                   (D.comp H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx) cGFx)
                cstarb_starax
                (D.cong HGFx H'GFx H'G'F'x
                   H'bstarax (D.comp H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx)
                   cGFx      cGFx
                   (H'.ax_preserve_composition GFx G'Fx G'F'x
                     ((G'.arrowfunction Fx F'x).op (a.arrows x)) (b.arrows Fx))
                   ((D.hom HGFx H'GFx).er.ref cGFx))
                (D.assoc HGFx H'GFx H'G'Fx H'G'F'x H'G'ax H'bFx cGFx))
             (homset.er.sym
                ((Fcat.comp H_GF HG_F H'G'_F' cstarb_stara assocFGH).arrows x)
                cstarb_starax
                (D.right_unit HGFx H'G'F'x cstarb_starax)))

ElimRightIdfunctorNatTra (C::E_category)(D::E_category)(F::E_functor C D)
 :: E_natural_transformation C D
      (E_functorcomposition C C D F (E_idfunctor C)) F
  = struct
    arrows        =
      \(a::C.obj)-> D.id (F.objectfunction a)
    ax_naturality =
      \(a::C.obj)->
      \(b::C.obj)->
      \(f::(C.hom a b).base)->
      let Fa::D.obj = F.objectfunction a
          Fb::D.obj = F.objectfunction b
          homset::SE = D.hom Fa Fb
          Ff::homset.base = (F.arrowfunction a b).op f
      in  homset.er.tra
            (D.comp Fa Fb Fb (arrows b) Ff) Ff (D.comp Fa Fa Fb Ff (arrows a))
            (D.left_unit Fa Fb Ff)
            (homset.er.sym (D.comp Fa Fa Fb Ff (arrows a)) Ff
               (D.right_unit Fa Fb Ff))

RevElimRightIdfunctorNatTra (C::E_category)(D::E_category)(F::E_functor C D)
 :: E_natural_transformation C D
      F (E_functorcomposition C C D F (E_idfunctor C))
  = struct
    arrows        = (ElimRightIdfunctorNatTra C D F).arrows
    ax_naturality = (ElimRightIdfunctorNatTra C D F).ax_naturality

ElimRightIdfunctor (C::E_category)(D::E_category)
 :: let FCD::E_category = E_functorcategory C D
        FCD1::E_category = E_productcategory FCD E_unitcategory
        FCC::E_category = E_functorcategory C C
        FCDFCC::E_category = E_productcategory FCD FCC
        ftype::Set = E_functor FCD1 FCD
        fcat::E_category = E_functorcategory FCD1 FCD
        R::ftype = E_rightunitcatelim FCD
        L::ftype
```

```
    = E_functorcomposition FCD1 FCDFCC FCD
        (FunctorCompositionAsFunctor C C D)
        (E_productfunctor FCD E_unitcategory FCD FCC
           (E_idfunctor FCD) (IdFunctorWithId C))
  in  Sigma (fcat.hom L R).base (Iso fcat L R)
= let FCD::E_category = E_functorcategory C D
      FCD1::E_category = E_productcategory FCD E_unitcategory
      FCC::E_category = E_functorcategory C C
      FCDFCC::E_category = E_productcategory FCD FCC
      ftype::Set = E_functor FCD1 FCD
      fcat::E_category = E_functorcategory FCD1 FCD
      R::ftype = E_rightunitcatelim FCD
      L::ftype
        = E_functorcomposition FCD1 FCDFCC FCD
            (FunctorCompositionAsFunctor C C D)
            (E_productfunctor FCD E_unitcategory FCD FCC
               (E_idfunctor FCD) (IdFunctorWithId C))
  in  struct
      _1 =
        struct
        arrows        =
          \(a::(E_productcategory (E_functorcategory C D) E_unitcategory).obj)->
          ElimRightIdfunctorNatTra C D a._1
        ax_naturality =
          \(a::FCD1.obj)->
          \(b::FCD1.obj)->
          \(f::(FCD1.hom a b).base)->
          \(x::C.obj)->
          let La::FCD.obj = L.objectfunction a
              Lb::FCD.obj = L.objectfunction b
              Lax::D.obj = La.objectfunction x
              Lbx::D.obj = Lb.objectfunction x
              Lf::(FCD.hom La Lb).base = (L.arrowfunction a b).op f
              Lfx::(D.hom Lax Lbx).base = Lf.arrows x
              homset::SE
                = D.hom ((L.objectfunction a).objectfunction x)
                        ((R.objectfunction b).objectfunction x)
          in  homset.er.tra
                ((FCD.comp La Lb b._1 (arrows b) Lf).arrows x)
                (f._1.arrows x)
                ((FCD.comp La a._1 b._1 f._1 (arrows a)).arrows x)
                (homset.er.tra
                   ((FCD.comp La Lb b._1 (arrows b) Lf).arrows x)
                   Lfx
                   (f._1.arrows x)
                   (D.left_unit Lax (b._1.objectfunction x) Lfx)
                   (homset.er.tra
                      Lfx
                      (D.comp Lax Lbx (b._1.objectfunction x)
                         (D.id (b._1.objectfunction x))
                         (f._1.arrows x))
                      (f._1.arrows x)
                      (D.cong
                         Lax (b._1.objectfunction x) (b._1.objectfunction x)
                         ((b._1.arrowfunction x x).op (C.id x))
                         (D.id (b._1.objectfunction x))
                         (f._1.arrows x)
                         (f._1.arrows x)
                         (b._1.ax_preserve_id x)
                         (homset.er.ref (f._1.arrows x)))
                      (D.left_unit (a._1.objectfunction x)
                         (b._1.objectfunction x)
                         (f._1.arrows x))))
                (homset.er.sym
                   ((FCD.comp La a._1 b._1 f._1 (arrows a)).arrows x)
                   (f._1.arrows x)
                   (D.right_unit (a._1.objectfunction x)
                      (b._1.objectfunction x) (f._1.arrows x)))
      _2   =
```

```
          NatTransIsoIfComponentsIso FCD1 FCD L R _1
            (\(x::FCD1.obj)->
             struct {
             _1 = RevElimRightIdfunctorNatTra C D x._1;
             _2 = struct { _1 = FCD.right_unit x._1 x._1 (FCD.id x._1);
                           _2 = FCD.left_unit x._1 x._1 (FCD.id x._1);};})

ElimLeftIdfunctorNatTra (C::E_category)(D::E_category)(F::E_functor C D)
 :: E_natural_transformation C D
      (E_functorcomposition C D D (E_idfunctor D) F) F
  = struct
    arrows        = (ElimRightIdfunctorNatTra C D F).arrows
    ax_naturality = (ElimRightIdfunctorNatTra C D F).ax_naturality

RevElimLeftIdfunctorNatTra (C::E_category)(D::E_category)(F::E_functor C D)
 :: E_natural_transformation C D
      F (E_functorcomposition C D D (E_idfunctor D) F)
  = struct
    arrows        = (ElimLeftIdfunctorNatTra C D F).arrows
    ax_naturality = (ElimLeftIdfunctorNatTra C D F).ax_naturality

ElimLeftIdfunctor (C::E_category)(D::E_category)
 :: let FCD::E_category = E_functorcategory C D
        P1FCD::E_category = E_productcategory E_unitcategory FCD
        FDD::E_category = E_functorcategory D D
        FDDFCD::E_category = E_productcategory FDD FCD
        ftype::Set = E_functor P1FCD FCD
        fcat::E_category = E_functorcategory P1FCD FCD
        R::ftype = E_leftunitcatelim FCD
        L::ftype
          = E_functorcomposition P1FCD FDDFCD FCD
              (FunctorCompositionAsFunctor C D D)
              (E_productfunctor E_unitcategory FCD FDD FCD
                 (IdFunctorWithId D) (E_idfunctor FCD))
    in  Sigma (fcat.hom L R).base (Iso fcat L R)
  = let FCD::E_category = E_functorcategory C D
        P1FCD::E_category = E_productcategory E_unitcategory FCD
        FDD::E_category = E_functorcategory D D
        FDDFCD::E_category = E_productcategory FDD FCD
        ftype::Set = E_functor P1FCD FCD
        fcat::E_category = E_functorcategory P1FCD FCD
        R::ftype = E_leftunitcatelim FCD
        L::ftype
          = E_functorcomposition P1FCD FDDFCD FCD
              (FunctorCompositionAsFunctor C D D)
              (E_productfunctor E_unitcategory FCD FDD FCD
                 (IdFunctorWithId D) (E_idfunctor FCD))
    in  struct
        _1 =
          struct
          arrows        =
            \(a::P1FCD.obj)-> ElimLeftIdfunctorNatTra C D a._2
          ax_naturality =
            \(a::P1FCD.obj)->
            \(b::P1FCD.obj)->
            \(f::(P1FCD.hom a b).base)->
            \(x::C.obj)->
            let La::FCD.obj = L.objectfunction a
                Lb::FCD.obj = L.objectfunction b
                homset::SE
                  = D.hom (La.objectfunction x) (b._2.objectfunction x)
                Lf::(FCD.hom La Lb).base = (L.arrowfunction a b).op f
            in  homset.er.tra
                  ((FCD.comp La Lb b._2 (arrows b) Lf).arrows x)
                  (f._2.arrows x)
                  ((FCD.comp La a._2 b._2 f._2 (arrows a)).arrows x)
                  (homset.er.tra
                     ((FCD.comp La Lb b._2 (arrows b) Lf).arrows x)
                     (Lf.arrows x)
```

```
                    (f._2.arrows x)
                    (D.left_unit (La.objectfunction x)
                                 (b._2.objectfunction x)
                                 (Lf.arrows x))
                    (D.right_unit (La.objectfunction x)
                                  (b._2.objectfunction x)
                                  (f._2.arrows x)))
                 ((D.hom (La.objectfunction x)
                         (b._2.objectfunction x)).er.sym
                    ((FCD.comp La a._2 b._2 f._2 (arrows a)).arrows x)
                    (f._2.arrows x)
                    (D.right_unit (a._2.objectfunction x)
                                  (b._2.objectfunction x)
                                  (f._2.arrows x)))
        _2   =
          NatTransIsoIfComponentsIso P1FCD FCD L R _1
            (\(x::P1FCD.obj)->
             struct {
             _1 = RevElimLeftIdfunctorNatTra C D x._2;
             _2 = struct {
                  _1 = FCD.right_unit x._2 x._2 (FCD.id x._2);
                  _2 = FCD.left_unit x._2 x._2 (FCD.id x._2);};})

CompAssocNatTrans (A::E_category)(B::E_category)(C::E_category)(D::E_category)
 :: let FAB::E_category = E_functorcategory A B
        FBC::E_category = E_functorcategory B C
        FCD::E_category = E_functorcategory C D
        FAC::E_category = E_functorcategory A C
        FBD::E_category = E_functorcategory B D
        FAD::E_category = E_functorcategory A D
        FBCFAB::E_category = E_productcategory FBC FAB
        FCDFBC::E_category = E_productcategory FCD FBC
        FBDFAB::E_category = E_productcategory FBD FAB
        FCDFAC::E_category = E_productcategory FCD FAC
        FCDFBC_FAB::E_category = E_productcategory FCDFBC FAB
        FCD_FBCFAB::E_category = E_productcategory FCD FBCFAB
        R::E_functor FCDFBC_FAB FAD
          = E_functorcomposition FCDFBC_FAB FCD_FBCFAB FAD
              (E_functorcomposition FCD_FBCFAB FCDFAC FAD
                 (FunctorCompositionAsFunctor A C D)
                 (E_productfunctor FCD FBCFAB FCD FAC
                    (E_idfunctor FCD) (FunctorCompositionAsFunctor A B C)))
              (prodassocright FCD FBC FAB)
        L::E_functor FCDFBC_FAB FAD
          = E_functorcomposition FCDFBC_FAB FBDFAB FAD
              (FunctorCompositionAsFunctor A B D)
              (E_productfunctor FCDFBC FAB FBD FAB
                 (FunctorCompositionAsFunctor B C D) (E_idfunctor FAB))
        Fcat::E_category = E_functorcategory FCDFBC_FAB FAD
    in  Sigma (Fcat.hom L R).base (Iso Fcat L R)
  = let FAB::E_category = E_functorcategory A B
        FBC::E_category = E_functorcategory B C
        FCD::E_category = E_functorcategory C D
        FAC::E_category = E_functorcategory A C
        FAD::E_category = E_functorcategory A D
        FBD::E_category = E_functorcategory B D
        FCDFBC::E_category = E_productcategory FCD FBC
        FBCFAB::E_category = E_productcategory FBC FAB
        FBDFAB::E_category = E_productcategory FBD FAB
        FCDFAC::E_category = E_productcategory FCD FAC
        FCDFBC_FAB::E_category = E_productcategory FCDFBC FAB
        FCD_FBCFAB::E_category = E_productcategory FCD FBCFAB
        R::E_functor FCDFBC_FAB FAD
          = E_functorcomposition FCDFBC_FAB FCD_FBCFAB FAD
              (E_functorcomposition FCD_FBCFAB FCDFAC FAD
                 (FunctorCompositionAsFunctor A C D)
                 (E_productfunctor FCD FBCFAB FCD FAC
                    (E_idfunctor FCD) (FunctorCompositionAsFunctor A B C)))
              (prodassocright FCD FBC FAB)
        L::E_functor FCDFBC_FAB FAD
          = E_functorcomposition FCDFBC_FAB FBDFAB FAD
              (FunctorCompositionAsFunctor A B D)
              (E_productfunctor FCDFBC FAB FBD FAB
                 (FunctorCompositionAsFunctor B C D) (E_idfunctor FAB))
    in  struct
        _1 =
          struct
          arrows        =
            \(a::FCDFBC_FAB.obj)->
            FunctorCompAssocNatTrans A B C D a._2 a._1._2 a._1._1
          ax_naturality =
            \(a::FCDFBC_FAB.obj)->
            \(b::FCDFBC_FAB.obj)->
            \(f::(FCDFBC_FAB.hom a b).base)->
            FunctorCompAssocNatTransNatural A B C D
              a._2 a._1._2 a._1._1 b._2 b._1._2 b._1._1 f._2 f._1._2 f._1._1
        _2   =
          struct
          _1 =
            struct
            arrows        =
              \(a::FCDFBC_FAB.obj)->
              RevFunctorCompAssocNatTrans A B C D a._2 a._1._2 a._1._1
            ax_naturality =
              \(a::FCDFBC_FAB.obj)->
              \(b::FCDFBC_FAB.obj)->
              \(f::(FCDFBC_FAB.hom a b).base)->
              RevFunctorCompAssocNatTransNatural A B C D
                a._2 a._1._2 a._1._1 b._2 b._1._2 b._1._1 f._2 f._1._2 f._1._1
          _2 =
            struct
            _1 =
              \(x::FCDFBC_FAB.obj)->
              \(x'::A.obj)->
              D.right_unit
                ((R.objectfunction x).objectfunction x')
                ((R.objectfunction x).objectfunction x')
                (D.id ((R.objectfunction x).objectfunction x'))
            _2 =
              \(x::FCDFBC_FAB.obj)->
              \(x'::A.obj)->
              D.left_unit
                ((L.objectfunction x).objectfunction x')
                ((L.objectfunction x).objectfunction x')
                (D.id ((L.objectfunction x).objectfunction x'))

AssocPentagon
  (A::E_category)(B::E_category)(C::E_category)(D::E_category)(E::E_category)
  (f::E_functor A B)(g::E_functor B C)(h::E_functor C D)(k::E_functor D E)
 :: let homcat::E_category = E_functorcategory A E
        ftype::Set = E_functor A E
        kh::E_functor C E = E_functorcomposition C D E k h
        gf::E_functor A C = E_functorcomposition A B C g f
        hg::E_functor B D = E_functorcomposition B C D h g
        hg_f::E_functor A D = E_functorcomposition A B D hg f
        h_gf::E_functor A D = E_functorcomposition A C D h gf
        kh_g::E_functor B E = E_functorcomposition B C E kh g
        k_hg::E_functor B E = E_functorcomposition B D E k hg
        kh_g__f::ftype = E_functorcomposition A B E kh_g f
        kh_gf::ftype = E_functorcomposition A C E kh gf
        k__h_gf::ftype = E_functorcomposition A D E k h_gf
        k_hg__f::ftype = E_functorcomposition A B E k_hg f
        k__hg_f::ftype = E_functorcomposition A D E k hg_f
        l1::(homcat.hom kh_g__f kh_gf).base
          = FunctorCompAssocNatTrans A B C E f g kh
        l2::(homcat.hom kh_gf k__h_gf).base
          = FunctorCompAssocNatTrans A C D E gf h k
        r1::(homcat.hom kh_g__f k_hg__f).base
```

```
         = HorizontalComposition A B E f kh_g f k_hg
             ((E_functorcategory A B).id f)
             (FunctorCompAssocNatTrans B C D E g h k)
       r2::(homcat.hom k_hg__f k__hg_f).base
         = FunctorCompAssocNatTrans A B D E f hg k
       r3::(homcat.hom k__hg_f k__h_gf).base
         = HorizontalComposition A D E hg_f k h_gf k
             (FunctorCompAssocNatTrans A B C D f g h)
             ((E_functorcategory D E).id k)
   in  (E_natural_transformation_SE A E kh_g__f k__h_gf).er.eq
          (homcat.comp kh_g__f kh_gf k__h_gf l2 l1)
          (homcat.comp kh_g__f k_hg__f k__h_gf
             (homcat.comp k_hg__f k__hg_f k__h_gf r3 r2) r1)
 = \(x::A.obj)->
   let hg::E_functor B D = E_functorcomposition B C D h g
       k_hg::E_functor B E = E_functorcomposition B D E k hg
       fx::B.obj = f.objectfunction x
       gfx::C.obj = g.objectfunction fx
       hgfx::D.obj = h.objectfunction gfx
       khgfx::E.obj = k.objectfunction hgfx
       homset::SE = E.hom khgfx khgfx
   in  homset.er.tra
         (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
         (E.comp khgfx khgfx khgfx
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx)))
         (E.comp khgfx khgfx khgfx
            (E.comp khgfx khgfx khgfx
               (E.comp khgfx khgfx khgfx
                  ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                  (E.id khgfx))
               (E.id khgfx))
            (E.comp khgfx khgfx khgfx
               ((k_hg.arrowfunction fx fx).op (B.id fx))
               (E.id khgfx)))
         (E.cong khgfx khgfx khgfx
            (E.id khgfx)
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
            (E.id khgfx)
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
            (homset.er.sym
               (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
               (E.id khgfx)
               (E.right_unit khgfx khgfx (E.id khgfx)))
            (homset.er.sym
               (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
               (E.id khgfx)
               (E.right_unit khgfx khgfx (E.id khgfx))))
         (E.cong khgfx khgfx khgfx
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
            (E.comp khgfx khgfx khgfx
               (E.comp khgfx khgfx khgfx
                  ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                  (E.id khgfx))
               (E.id khgfx))
            (E.comp khgfx khgfx khgfx (E.id khgfx) (E.id khgfx))
            (E.comp khgfx khgfx khgfx
               ((k_hg.arrowfunction fx fx).op (B.id fx))
               (E.id khgfx))
            (E.cong khgfx khgfx khgfx
               (E.id khgfx)
               (E.comp khgfx khgfx khgfx
                  ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                  (E.id khgfx))
               (E.id khgfx)
               (E.id khgfx)
               (homset.er.sym
                  (E.comp khgfx khgfx khgfx
                     ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                     (E.id khgfx))
                  (E.id khgfx)
                  (homset.er.tra
                     (E.comp khgfx khgfx khgfx
                        ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                        (E.id khgfx))
                     ((k.arrowfunction hgfx hgfx).op (D.id hgfx))
                     (E.id khgfx)
                     (E.right_unit khgfx khgfx
                        ((k.arrowfunction hgfx hgfx).op (D.id hgfx)))
                     (k.ax_preserve_id hgfx)))
               (homset.er.ref (E.id khgfx)))
            (E.cong khgfx khgfx khgfx
               (E.id khgfx)
               ((k_hg.arrowfunction fx fx).op (B.id fx))
               (E.id khgfx)
               (E.id khgfx)
               (homset.er.sym
                  ((k_hg.arrowfunction fx fx).op (B.id fx))
                  (E.id khgfx)
                  (k_hg.ax_preserve_id fx))
               (homset.er.ref (E.id khgfx))))

IdentityTriangle (C::E_category)(D::E_category)(E::E_category)
  (f::E_functor C D)(g::E_functor D E)
 :: let homcat::E_category = E_functorcategory C E
        ftype::Set = E_functor C E
        I::E_functor D D = E_idfunctor D
        gI::E_functor D E = E_functorcomposition D D E g I
        If::E_functor C D = E_functorcomposition C D D I f
        gI_f::ftype = E_functorcomposition C D E gI f
        g_If::ftype = E_functorcomposition C D E g If
        gf::ftype = E_functorcomposition C D E g f
        l::E_natural_transformation C E gI_f gf
          = HorizontalComposition C D E f gI f g
              ((E_functorcategory C D).id f) (ElimRightIdfunctorNatTra D E g)
        r1::E_natural_transformation C E gI_f g_If
          = FunctorCompAssocNatTrans C D D E f I g
        r2::E_natural_transformation C E g_If gf
          = HorizontalComposition C D E If g f g
              (ElimLeftIdfunctorNatTra C D f) ((E_functorcategory D E).id g)
    in  (E_natural_transformation_SE C E gI_f gf).er.eq
           l ((E_functorcategory C E).comp gI_f g_If gf r2 r1)
  = \(x::C.obj)->
    let I   ::E_functor D D = E_idfunctor D
        gI  ::E_functor D E = E_functorcomposition D D E g I
        If  ::E_functor C D = E_functorcomposition C D D I f
        gI_f::E_functor C E = E_functorcomposition C D E gI f
        g_If::E_functor C E = E_functorcomposition C D E g If
        gf  ::E_functor C E = E_functorcomposition C D E g f
        gfx ::E.obj         = gf.objectfunction x
    in  (E.hom gfx gfx).er.sym
           (E.comp gfx gfx gfx
              ((HorizontalComposition C D E If g f g
                  (ElimLeftIdfunctorNatTra C D f)
                  ((E_functorcategory D E).id g)).arrows x)
              (E.id gfx))
           ((HorizontalComposition C D E f gI f g
               ((E_functorcategory C D).id f)
               (ElimRightIdfunctorNatTra D E g)).arrows x)
           (E.right_unit gfx gfx
              ((HorizontalComposition C D E If g f g
                  (ElimLeftIdfunctorNatTra C D f)
                  ((E_functorcategory D E).id g)).arrows x))

ECat :: E_bicategory
  = struct
    obj          = E_category
    hom          = E_functorcategory
```

```
    comp          = FunctorCompositionAsFunctor
    identity      = IdFunctorWithId
    associativity = CompAssocNatTrans
    rightid       = ElimRightIdfunctor
    leftid        = ElimLeftIdfunctor
    apentagon     = AssocPentagon
    idtriangle    = IdentityTriangle

{-# Alfa unfoldgoals off
brief on
hidetypeannots off
tall

nd
hiding on
 #-}
```

E_adjunction.agda

```
--#include "ECat.agda"

E_adjunction_adhoc (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
 :: Set
  = sig {
      -- unit == eta
      unit::E_natural_transformation C C
             (E_idfunctor C) (E_functorcomposition C D C G F);
      -- counit == epsilon
      counit::E_natural_transformation D D
               (E_functorcomposition D C D F G) (E_idfunctor D);
      unittrianglelaw
        -- 1_F = epsilon_F o F eta
        :: (E_natural_transformation_SE C D F F).er.eq
             ((E_functorcategory C D).id F)
             ((E_functorcategory C D).comp F
               (E_functorcomposition C D D (E_functorcomposition D C D F G) F)
               F
               (struct
                arrows       =
                 \(a::C.obj)-> counit.arrows (F.objectfunction a)
                ax_naturality =
                 \(a::C.obj)-> \(b::C.obj)-> \(f::Hom C a b)->
                 counit.ax_naturality
                   (F.objectfunction a) (F.objectfunction b)
                   ((F.arrowfunction a b).op f)
               )
               (struct
                arrows       =
                 \(a::C.obj)->
                 (F.arrowfunction
                     a (G.objectfunction (F.objectfunction a))).op
                   (unit.arrows a)
                ax_naturality =
                 \(a::C.obj)-> \(b::C.obj)-> \(f::Hom C a b)->
                 (D.hom
                   (F.objectfunction a)
                   ((E_functorcomposition C D D
                       (E_functorcomposition D C D F G)
                      F).objectfunction b)).er.tra
                     (compose D (F.objectfunction a) (F.objectfunction b)
                       ((E_functorcomposition C D D
                          (E_functorcomposition D C D F G)
                         F).objectfunction b)
                       (arrows b)
                       (Arrowfunction C D F a b f))
                     ((F.arrowfunction
```

```
   a
   (G.objectfunction (F.objectfunction b))).op
  (C.comp
   a
   ((E_functorcomposition C D C G F).objectfunction a)
   (G.objectfunction (F.objectfunction b))
   (((E_functorcomposition C D C
        G F).arrowfunction a b).op f)
   (unit.arrows a)))
(compose
  D (F.objectfunction a)
  ((E_functorcomposition C D D
      (E_functorcomposition D C D F G)
      F).objectfunction a)
  ((E_functorcomposition C D D
      (E_functorcomposition D C D F G)
      F).objectfunction b)
  (Arrowfunction C D
     (E_functorcomposition C D D
       (E_functorcomposition D C D F G) F)
     a b f)
  (arrows a))
((D.hom
   (F.objectfunction a)
   ((E_functorcomposition C D D
       (E_functorcomposition D C D F G)
       F).objectfunction b)).er.tra
    (compose D
       (F.objectfunction a)
       (F.objectfunction b)
       ((E_functorcomposition C D D
          (E_functorcomposition D C D F G)
          F).objectfunction b)
       (arrows b)
       (Arrowfunction C D F a b f))
    ((F.arrowfunction
        a
        (G.objectfunction (F.objectfunction b))).op
      (compose
        C
        ((E_idfunctor C).objectfunction a)
        ((E_idfunctor C).objectfunction b)
        ((E_functorcomposition C D C
             G F).objectfunction b)
        (unit.arrows b)
        (Arrowfunction C C (E_idfunctor C) a b f)))
    ((F.arrowfunction
          a (G.objectfunction (F.objectfunction b))).op
       (C.comp
          a
          ((E_functorcomposition C D C
              G F).objectfunction a)
          (G.objectfunction (F.objectfunction b))
          (((E_functorcomposition C D C
              G F).arrowfunction a b).op f)
          (unit.arrows a)))
    ((D.hom
        (F.objectfunction a)
        ((E_functorcomposition C D D
            (E_functorcomposition D C D
               F G) F).objectfunction b)).er.sym
          ((F.arrowfunction
               a
               (G.objectfunction
                  (F.objectfunction b))).op
             (compose
               C
               ((E_idfunctor C).objectfunction a)
               ((E_idfunctor C).objectfunction b)
```

```
                                  ((E_functorcomposition C D C
                                      G F).objectfunction b)
                                  (unit.arrows b)
                                  (Arrowfunction C C
                                     (E_idfunctor C) a b f)))
                            (compose
                               D
                               (F.objectfunction a)
                               (F.objectfunction b)
                               ((E_functorcomposition C D D
                                   (E_functorcomposition D C D
                                      F G) F).objectfunction b)
                               (arrows b)
                               (Arrowfunction C D F a b f))
                            (F.ax_preserve_composition
                               a b
                               (G.objectfunction (F.objectfunction b))
                               (unit.arrows b) f))
                     ((F.arrowfunction
                         a
                         (G.objectfunction (F.objectfunction b))).ext
                       (compose
                          C
                          ((E_idfunctor C).objectfunction a)
                          ((E_idfunctor C).objectfunction b)
                          ((E_functorcomposition C D C
                              G F).objectfunction b)
                          (unit.arrows b)
                          (Arrowfunction C C (E_idfunctor C) a b f))
                       (C.comp
                          a
                          ((E_functorcomposition C D C
                              G F).objectfunction a)
                          (G.objectfunction (F.objectfunction b))
                          (((E_functorcomposition C D C
                               G F).arrowfunction a b).op f)
                          (unit.arrows a))
                       (unit.ax_naturality a b f)))
                (F.ax_preserve_composition
                   a
                   ((E_functorcomposition C D C G F).objectfunction a)
                   (G.objectfunction (F.objectfunction b))
                   (((E_functorcomposition C D C
                        G F).arrowfunction a b).op f)
                   (unit.arrows a))
          ));
counittrianglelaw
  -- 1_G = G epsilon o eta_G
  :: (E_natural_transformation_SE D C G G).er.eq
        ((E_functorcategory D C).id G)
        ((E_functorcategory D C).comp
            G
            (E_functorcomposition D D C
               G (E_functorcomposition D C D F G))
            G
            (struct
             arrows        =
              \(a::D.obj)->
              (G.arrowfunction
                 ((E_functorcomposition D C D F G).objectfunction a)
                 a).op (counit.arrows a)
             ax_naturality =
              \(a::D.obj)->
              \(b::D.obj)->
              \(f::Hom D a b)->
              let FG::E_functor D D  = E_functorcomposition D C D F G
                  GFG::E_functor D C = E_functorcomposition D D C G FG
                  FGa::D.obj = FG.objectfunction a
                  FGb::D.obj = FG.objectfunction b
                  FGf::(D.hom FGa FGb).base = (FG.arrowfunction a b).op f
                  GFGa::C.obj        = GFG.objectfunction a
                  GFGb::C.obj        = GFG.objectfunction b
                  GFGf::(C.hom GFGa GFGb).base
                    = (GFG.arrowfunction a b).op f
                  Ga::C.obj          = G.objectfunction a
                  Gb::C.obj          = G.objectfunction b
                  Gf::(C.hom Ga Gb).base
                    = (G.arrowfunction a b).op f
                  ea::(D.hom FGa a).base = counit.arrows a
                  eb::(D.hom FGb b).base = counit.arrows b
                  Gea::(C.hom GFGa Ga).base
                    = (G.arrowfunction FGa a).op ea
                  Geb::(C.hom GFGb Gb).base
                    = (G.arrowfunction FGb b).op eb
                  goal::(C.hom GFGa Gb).er.eq
                          (C.comp GFGa GFGb Gb Geb GFGf)
                          (C.comp GFGa Ga Gb Gf Gea)
                    = (C.hom GFGa Gb).er.tra
                         (C.comp GFGa GFGb Gb Geb GFGf)
                         ((G.arrowfunction FGa b).op
                             (D.comp FGa FGb b eb FGf))
                         (C.comp GFGa Ga Gb Gf Gea)
                         ((C.hom GFGa Gb).er.sym
                             ((G.arrowfunction FGa b).op
                                 (D.comp FGa FGb b eb FGf))
                             (C.comp GFGa GFGb Gb Geb GFGf)
                             (G.ax_preserve_composition
                                FGa FGb b eb FGf))
                         ((C.hom GFGa Gb).er.tra
                             ((G.arrowfunction FGa b).op
                                 (D.comp FGa FGb b eb FGf))
                             ((G.arrowfunction FGa b).op
                                  (D.comp FGa a b f ea))
                             (C.comp GFGa Ga Gb Gf Gea)
                             ((G.arrowfunction FGa b).ext
                                 (D.comp FGa FGb b eb FGf)
                                 (D.comp FGa a b f ea)
                                 (counit.ax_naturality a b f))
                             (G.ax_preserve_composition FGa a b f ea))
              in  goal
            )
            (struct
             arrows        =
              \(a::D.obj)-> unit.arrows (G.objectfunction a)
             ax_naturality =
              \(a::D.obj)->
              \(b::D.obj)->
              \(f::Hom D a b)->
              unit.ax_naturality (G.objectfunction a)
                                 (G.objectfunction b)
                                 ((G.arrowfunction a b).op f)
            ));
    }

E_adjunction (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
 :: Set
  = let FG::E_functor D D = E_functorcomposition D C D F G
        GF::E_functor C C = E_functorcomposition C D C G F
        G_FG::E_functor D C = E_functorcomposition D D C G FG
        GF_G::E_functor D C = E_functorcomposition D C C GF G
        F_GF::E_functor C D = E_functorcomposition C C D F GF
        FG_F::E_functor C D = E_functorcomposition C D D FG F
        goal::Set
         = sig {
             -- unit == eta
             unit::E_natural_transformation C C
                     (E_idfunctor C) GF;
             -- counit == epsilon
```

```
counit::E_natural_transformation D D
         FG (E_idfunctor D);
unittrianglelaw
  -- 1_F = epsilon_F o F eta
  :: (E_natural_transformation_SE C D F F).er.eq
        ((E_functorcategory C D).id F)
        ((E_functorcategory C D).comp
            F
            FG_F
            F
            ((E_functorcategory C D).comp
                FG_F
                (E_functorcomposition C D D
                   (E_idfunctor D) F)
                F
                (ElimLeftIdfunctorNatTra C D F)
                (HorizontalComposition C D D
                   F FG F (E_idfunctor D)
                   ((E_functorcategory C D).id F)
                   counit))
            ((E_functorcategory C D).comp
                F F_GF FG_F
                ((ECat.associativity C D C D)._2._1.arrows
                   (struct {
                    _1 = struct { _1 = F; _2 = G;};
                    _2 = F;}))
                ((E_functorcategory C D).comp
                    F
                    (E_functorcomposition C C D
                       F (E_idfunctor C))
                    F_GF
                    (HorizontalComposition C C D
                       (E_idfunctor C) F GF F
                       unit ((E_functorcategory C D).id F))
                    (RevElimRightIdfunctorNatTra C D F))));
counittrianglelaw
  -- 1_G = G epsilon o eta_G
  :: (E_natural_transformation_SE D C G G).er.eq
        ((E_functorcategory D C).id G)
        ((E_functorcategory D C).comp
            G
            G_FG
            G
            ((E_functorcategory D C).comp
                G_FG
                (E_functorcomposition D D C
                   G (E_idfunctor D))
                G
                (ElimRightIdfunctorNatTra D C G)
                (HorizontalComposition D D C
                   FG G (E_idfunctor D) G
                   counit
                   ((E_functorcategory D C).id G)))
            ((E_functorcategory D C).comp
                G
                GF_G
                G_FG
                ((ECat.associativity D C D C)._1.arrows
                    (struct {
                     _1 = struct { _1 = G; _2 = F;};
                     _2 = G;}))
                ((E_functorcategory D C).comp
                    G
                    (E_functorcomposition D C C
                       (E_idfunctor C) G)
                    GF_G
                    (HorizontalComposition D C C
                       G (E_idfunctor C) G GF
                       ((E_functorcategory D C).id G)
                       unit)
                    (RevElimLeftIdfunctorNatTra D C G))));
          }
    in  goal

E_bicatadjunction (B::E_bicategory)(C,D::B.obj)
       (F::(B.hom C D).obj)(G::(B.hom D C).obj)
 :: Set
  = sig {
      -- some local names first
      FG::(B.hom D D).obj
        = (B.comp D  C  D).objectfunction (struct { _1 = F; _2 = G;});
      GF::(B.hom C C).obj
        = (B.comp C D C).objectfunction (struct { _1 = G; _2 = F;});
      G_FG::(B.hom D C).obj
        = (B.comp D D C).objectfunction (struct { _1 = G; _2 = FG;});
      GF_G::(B.hom D C).obj
        = (B.comp D C C).objectfunction (struct { _1 = GF; _2 = G;});
      F_GF::(B.hom C D).obj
        = (B.comp C C D).objectfunction (struct { _1 = F; _2 = GF;});
      FG_F::(B.hom C D).obj
        = (B.comp C D D).objectfunction (struct { _1 = FG; _2 = F;});
      idC::(B.hom C C).obj = (B.identity C).objectfunction elt@_;
      idD::(B.hom D D).obj = (B.identity D).objectfunction elt@_;
      idF::((B.hom C D).hom F F).base = (B.hom C D).id F;
      idG::((B.hom D C).hom G G).base = (B.hom D C).id G;
      -- now the real things:
      unit::((B.hom C C).hom idC GF).base;
      counit::((B.hom D D).hom FG idD).base;
      unittrianglelaw::((B.hom C D).hom F F).er.eq
        idF
        ((B.hom C D).comp
           F FG_F F
           ((B.hom C D).comp
              FG_F
              ((B.comp C D D).objectfunction (struct { _1 = idD; _2 = F;}))
              F
              ((B.leftid C D)._1.arrows (struct { _1 = elt@_; _2 = F;}))
              (((B.comp C D D).arrowfunction
                     (struct { _1 = FG; _2 = F;})
                     (struct { _1 = idD;_2 = F;})).op
                  (struct { _1 = counit; _2 = idF;})))
           ((B.hom C D).comp
              F F_GF FG_F
              ((B.associativity C D C D)._2._1.arrows
                 (struct { _1 = struct { _1 = F; _2 = G;}; _2 = F;}))
              ((B.hom C D).comp
                 F
                 ((B.comp C C D).objectfunction (struct { _1 = F; _2 = idC;}))
                 F_GF
                 (((B.comp C C D).arrowfunction
                        (struct { _1 = F; _2 = idC;})
                        (struct { _1 = F; _2 = GF;})).op
                     (struct { _1 = idF; _2 = unit;}))
                 ((B.rightid C D)._2._1.arrows
                    (struct { _1 = F; _2 = elt@_;})))));
      counittrianglelaw::((B.hom D C).hom G G).er.eq
        idG
        ((B.hom D C).comp G G_FG G
           ((B.hom D C).comp
              G_FG
              ((B.comp D D C).objectfunction (struct { _1 = G; _2 = idD;}))
              G
              ((B.rightid D C)._1.arrows (struct { _1 = G; _2 = elt@_;}))
              (((B.comp D D C).arrowfunction
                     (struct { _1 = G; _2 = FG;})
                     (struct { _1 = G; _2 = idD;})).op
                  (struct { _1 = idG; _2 = counit;})))
           ((B.hom D C).comp
```

```
              G
              GF_G
              G_FG
              ((B.associativity D C D C)._1.arrows
                 (struct { _1 = struct { _1 = G; _2 = F;}; _2 = G;}))
              ((B.hom D C).comp
                 G
                 ((B.comp D C C).objectfunction (struct { _1 = idC; _2 = G;}))
                 GF_G
                 (((B.comp D C C).arrowfunction
                        (struct { _1 = idC; _2 = G;})
                        (struct { _1 = GF; _2 = G;})).op
                     (struct { _1 = unit; _2 = idG;}))
                 ((B.leftid D C)._2._1.arrows
                    (struct { _1 = elt@_; _2 = G;})))));
    }

-- Now, we'd like these to be equivalent

-- These proofs could be done more economically.

adhoc_to_adj (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
             (adj::E_adjunction_adhoc C D F G)
 :: E_adjunction C D F G
  = struct
    unit              =
      adj.unit
    counit            =
      adj.counit
    unittrianglelaw   =
      \(x::C.obj)->
      let Fx::D.obj = F.objectfunction x
          idFx::(D.hom Fx Fx).base = D.id Fx
          GFx::C.obj = G.objectfunction Fx
          FGFx::D.obj = F.objectfunction GFx
          idFGFx::(D.hom FGFx FGFx).base = D.id FGFx
      in  (D.hom Fx Fx).er.tra
            idFx
            (D.comp Fx FGFx Fx
              (counit.arrows Fx)
              ((F.arrowfunction x GFx).op (unit.arrows x)))
            (D.comp Fx FGFx Fx
              (D.comp FGFx Fx Fx
                idFx (D.comp FGFx Fx Fx idFx (counit.arrows Fx)))
              (D.comp Fx FGFx FGFx
                idFGFx
                (D.comp Fx Fx FGFx
                  (D.comp Fx Fx FGFx
                    ((F.arrowfunction x GFx).op (unit.arrows x))
                    idFx)
                  idFx)))
            (adj.unittrianglelaw x)
            (D.cong Fx FGFx Fx
              (counit.arrows Fx)
              (D.comp FGFx Fx Fx
                idFx (D.comp FGFx Fx Fx idFx (counit.arrows Fx)))
              ((F.arrowfunction x GFx).op (unit.arrows x))
              (D.comp Fx FGFx FGFx
                idFGFx
                (D.comp Fx Fx FGFx
                  (D.comp Fx Fx FGFx
                    ((F.arrowfunction x GFx).op (unit.arrows x)) idFx)
                  idFx))
              ((D.hom FGFx Fx).er.sym
                 (D.comp FGFx Fx Fx
                   idFx
                   (D.comp FGFx Fx Fx
                     idFx (counit.arrows Fx)))
                 (counit.arrows Fx)
                 ((D.hom FGFx Fx).er.tra
                    (D.comp FGFx Fx Fx
                      idFx
                      (D.comp FGFx Fx Fx idFx (counit.arrows Fx)))
                    (D.comp FGFx Fx Fx idFx (counit.arrows Fx))
                    (counit.arrows Fx)
                    (D.left_unit FGFx Fx
                      (D.comp FGFx Fx Fx idFx (counit.arrows Fx)))
                    (D.left_unit FGFx Fx (counit.arrows Fx))))
              ((D.hom Fx FGFx).er.sym
                 (D.comp Fx FGFx FGFx
                   idFGFx
                   (D.comp Fx Fx FGFx
                     (D.comp Fx Fx FGFx
                       ((F.arrowfunction x GFx).op (unit.arrows x))
                       idFx)
                     idFx))
                 ((F.arrowfunction x GFx).op (unit.arrows x))
                 ((D.hom Fx FGFx).er.tra
                    (D.comp Fx FGFx FGFx
                      idFGFx
                      (D.comp Fx Fx FGFx
                        (D.comp Fx Fx FGFx
                          ((F.arrowfunction x GFx).op (unit.arrows x)) idFx)
                        idFx))
                    (D.comp Fx Fx FGFx
                      ((F.arrowfunction x GFx).op (unit.arrows x)) idFx)
                    ((F.arrowfunction x GFx).op (unit.arrows x))
                    ((D.hom Fx FGFx).er.tra
                       (D.comp Fx FGFx FGFx
                         idFGFx
                         (D.comp Fx Fx FGFx
                           (D.comp Fx Fx FGFx
                             ((F.arrowfunction x GFx).op (unit.arrows x)) idFx)
                           idFx))
                       (D.comp Fx Fx FGFx
                         (D.comp Fx Fx FGFx
                           ((F.arrowfunction x GFx).op (unit.arrows x))
                           idFx)
                         idFx)
                       (D.comp Fx Fx FGFx
                         ((F.arrowfunction x GFx).op (unit.arrows x))
                         idFx)
                       (D.left_unit Fx FGFx
                         (D.comp Fx Fx FGFx
                           (D.comp Fx Fx FGFx
                             ((F.arrowfunction x GFx).op (unit.arrows x)) idFx)
                           idFx))
                       (D.right_unit Fx FGFx
                         (D.comp Fx Fx FGFx
                           ((F.arrowfunction x GFx).op (unit.arrows x))
                           idFx)))
                    (D.right_unit Fx FGFx
                      ((F.arrowfunction x GFx).op (unit.arrows x))))))
    counittrianglelaw =
      \(x::D.obj)->
      let Gx::C.obj = G.objectfunction x
          FGx::D.obj = F.objectfunction Gx
          GFGx::C.obj = G.objectfunction FGx
          idGx::(C.hom Gx Gx).base = C.id Gx
          idGFGx::(C.hom GFGx GFGx).base = C.id GFGx
          GF::E_functor C C = E_functorcomposition C D C G F
      in  ((C.hom Gx Gx).er.tra
             idGx
             (C.comp Gx GFGx Gx
               ((G.arrowfunction FGx x).op (adj.counit.arrows x))
               (adj.unit.arrows Gx))
             (C.comp Gx GFGx Gx
               (C.comp GFGx Gx Gx
```

```
    idGx
    (C.comp GFGx GFGx Gx
      ((G.arrowfunction FGx x).op (counit.arrows x))
      idGFGx))
  (C.comp Gx GFGx GFGx
    idGFGx
    (C.comp Gx Gx GFGx
      (C.comp Gx GFGx GFGx
        ((GF.arrowfunction Gx Gx).op idGx)
        (unit.arrows Gx))
      idGx)))
(adj.counittrianglelaw x)
((C.hom Gx Gx).er.sym
   (C.comp Gx GFGx Gx
     (C.comp GFGx Gx Gx
       idGx
       (C.comp GFGx GFGx Gx
         ((G.arrowfunction FGx x).op (counit.arrows x))
         idGFGx))
     (C.comp Gx GFGx GFGx
       idGFGx
       (C.comp Gx Gx GFGx
         (C.comp Gx GFGx GFGx
           ((GF.arrowfunction Gx Gx).op idGx)
           (unit.arrows Gx))
         idGx)))
   (C.comp Gx GFGx Gx
     ((G.arrowfunction FGx x).op (adj.counit.arrows x))
     (adj.unit.arrows Gx))
   (C.cong Gx GFGx Gx
     (C.comp GFGx Gx Gx
       idGx
       (C.comp GFGx GFGx Gx
         ((G.arrowfunction FGx x).op (counit.arrows x))
         idGFGx))
     ((G.arrowfunction FGx x).op (adj.counit.arrows x))
     (C.comp Gx GFGx GFGx
       idGFGx
       (C.comp Gx Gx GFGx
         (C.comp Gx GFGx GFGx
           ((GF.arrowfunction Gx Gx).op idGx)
           (unit.arrows Gx))
         idGx))
     (adj.unit.arrows Gx)
     ((C.hom GFGx Gx).er.tra
        (C.comp GFGx Gx Gx
          idGx
          (C.comp GFGx GFGx Gx
            ((G.arrowfunction FGx x).op (counit.arrows x))
            idGFGx))
        (C.comp GFGx GFGx Gx
          ((G.arrowfunction FGx x).op (counit.arrows x))
          idGFGx)
        ((G.arrowfunction FGx x).op (counit.arrows x))
        (C.left_unit GFGx Gx
          (C.comp GFGx GFGx Gx
            ((G.arrowfunction FGx x).op (counit.arrows x))
            idGFGx))
        (C.right_unit GFGx Gx
          ((G.arrowfunction FGx x).op (counit.arrows x))))
     ((C.hom Gx GFGx).er.tra
        (C.comp Gx GFGx GFGx
          idGFGx
          (C.comp Gx Gx GFGx
            (C.comp Gx GFGx GFGx
              ((GF.arrowfunction Gx Gx).op idGx)
              (unit.arrows Gx))
            idGx))
        (C.comp Gx
```

```
        GFGx
        GFGx
        idGFGx
        (unit.arrows Gx))
(adj.unit.arrows Gx)
((C.hom Gx GFGx).er.tra
   (C.comp Gx GFGx GFGx
     idGFGx
     (C.comp Gx Gx GFGx
       (C.comp Gx GFGx GFGx
         ((GF.arrowfunction Gx Gx).op idGx)
         (unit.arrows Gx))
       idGx))
   (C.comp Gx GFGx GFGx
     ((GF.arrowfunction Gx Gx).op idGx)
     (unit.arrows Gx))
   (C.comp Gx GFGx GFGx
     idGFGx (unit.arrows Gx))
   ((C.hom Gx GFGx).er.tra
      (C.comp Gx GFGx GFGx
        idGFGx
        (C.comp Gx Gx GFGx
          (C.comp Gx GFGx GFGx
            ((GF.arrowfunction Gx Gx).op idGx)
            (unit.arrows Gx))
          idGx))
      (C.comp Gx Gx GFGx
        (C.comp Gx GFGx GFGx
          ((GF.arrowfunction Gx Gx).op idGx)
          (unit.arrows Gx))
        idGx)
      (C.comp Gx GFGx GFGx
        ((GF.arrowfunction Gx Gx).op idGx)
        (unit.arrows Gx))
      (C.left_unit Gx GFGx
        (C.comp Gx Gx GFGx
          (C.comp Gx GFGx GFGx
            ((GF.arrowfunction Gx Gx).op idGx)
            (unit.arrows Gx))
          idGx))
      (C.right_unit Gx GFGx
        (C.comp Gx GFGx GFGx
          ((GF.arrowfunction Gx Gx).op idGx)
          (unit.arrows Gx))))
   (C.cong Gx GFGx GFGx
     ((GF.arrowfunction Gx Gx).op idGx)
     idGFGx
     (unit.arrows Gx)
     (unit.arrows Gx)
     (GF.ax_preserve_id Gx)
     ((C.hom Gx GFGx).er.ref (unit.arrows Gx))))
(C.left_unit Gx GFGx
  (unit.arrows Gx))))))

adj_to_adhoc (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
             (adj::E_adjunction C D F G)
 :: E_adjunction_adhoc C D F G
  = struct
    unit             =
      adj.unit
    counit           =
      adj.counit
    unittrianglelaw  =
      \(x::C.obj)->
      let Fx::D.obj = F.objectfunction x
          idFx::(D.hom Fx Fx).base = D.id Fx
          GFx::C.obj = G.objectfunction Fx
          FGFx::D.obj = F.objectfunction GFx
          idGFGx::(D.hom FGFx FGFx).base = D.id FGFx
```

```
in  (D.hom Fx Fx).er.tra
      idFx
      (D.comp Fx FGFx Fx
        (D.comp FGFx Fx Fx
          idFx
          (D.comp FGFx Fx Fx
            idFx (counit.arrows Fx)))
        (D.comp Fx FGFx FGFx
          idFGFx
          (D.comp Fx Fx FGFx
            (D.comp Fx Fx FGFx
              ((F.arrowfunction x GFx).op (unit.arrows x))
              idFx)
            idFx)))
      (D.comp Fx FGFx Fx
        (counit.arrows Fx)
        ((F.arrowfunction x GFx).op (unit.arrows x)))
      (adj.unittrianglelaw x)
      (D.cong Fx FGFx Fx
        (D.comp FGFx Fx Fx
          idFx
          (D.comp FGFx Fx Fx
            idFx (counit.arrows Fx)))
        (counit.arrows Fx)
        (D.comp Fx FGFx FGFx
           idFGFx
           (D.comp Fx Fx FGFx
             (D.comp Fx Fx FGFx
               ((F.arrowfunction x GFx).op (unit.arrows x))
               idFx)
             idFx))
        ((F.arrowfunction x GFx).op (unit.arrows x))
        ((D.hom FGFx Fx).er.tra
           (D.comp FGFx Fx Fx
             idFx
             (D.comp FGFx Fx Fx
               idFx (counit.arrows Fx)))
           (D.comp FGFx Fx Fx
             idFx (counit.arrows Fx))
           (counit.arrows Fx)
           (D.left_unit FGFx Fx
             (D.comp FGFx Fx Fx
               idFx (counit.arrows Fx)))
           (D.left_unit FGFx Fx (counit.arrows Fx)))
        ((D.hom Fx FGFx).er.tra
           (D.comp Fx FGFx FGFx
             idFGFx
             (D.comp Fx Fx FGFx
               (D.comp Fx Fx FGFx
                 ((F.arrowfunction x GFx).op (unit.arrows x))
                 idFx)
               idFx))
           (D.comp Fx Fx FGFx
             ((F.arrowfunction x GFx).op (unit.arrows x))
             idFx)
           ((F.arrowfunction x GFx).op (unit.arrows x))
           ((D.hom Fx FGFx).er.tra
              (D.comp Fx FGFx FGFx
                idFGFx
                (D.comp Fx Fx FGFx
                  (D.comp Fx Fx FGFx
                    ((F.arrowfunction x GFx).op (unit.arrows x))
                    idFx)
                  idFx))
              (D.comp Fx Fx FGFx
                (D.comp Fx Fx FGFx
                    ((F.arrowfunction x GFx).op (unit.arrows x))
                  idFx)
                idFx)
```

```
                (D.comp Fx Fx FGFx
                  ((F.arrowfunction x GFx).op (unit.arrows x))
                  idFx)
                (D.left_unit Fx FGFx
                  (D.comp Fx Fx FGFx
                    (D.comp Fx Fx FGFx
                      ((F.arrowfunction x GFx).op (unit.arrows x))
                      idFx)
                    idFx))
                (D.right_unit Fx
                              FGFx
                              (D.comp Fx
                                      Fx
                                      FGFx
                                      ((F.arrowfunction x GFx).op (unit.arrows x))
                                      idFx)))
             (D.right_unit Fx FGFx
               ((F.arrowfunction x GFx).op (unit.arrows x)))))
counittrianglelaw =
  \(x::D.obj)->
  let Gx::C.obj = G.objectfunction x
      homset::SE = C.hom Gx Gx
      idGx::homset.base = C.id Gx
      GF::E_functor C C = E_functorcomposition C D C G F
      FGx::D.obj = F.objectfunction Gx
      GFGx::C.obj = G.objectfunction FGx
      idGFGx::(C.hom GFGx GFGx).base = C.id GFGx
  in  homset.er.tra
        idGx
        (C.comp Gx GFGx Gx
          (C.comp GFGx Gx Gx
            idGx
            (C.comp GFGx GFGx Gx
              ((G.arrowfunction FGx x).op (adj.counit.arrows x))
              idGFGx))
          (C.comp Gx GFGx GFGx
            idGFGx
            (C.comp Gx Gx GFGx
              (C.comp Gx GFGx GFGx
                ((GF.arrowfunction Gx Gx).op idGx)
                (adj.unit.arrows Gx))
              idGx)))
        (C.comp Gx GFGx Gx
          ((G.arrowfunction FGx x).op (counit.arrows x))
          (unit.arrows (G.objectfunction x)))
        (adj.counittrianglelaw x)
        (C.cong Gx GFGx Gx
          (C.comp GFGx Gx Gx
            idGx
            (C.comp GFGx GFGx Gx
              ((G.arrowfunction FGx x).op (adj.counit.arrows x))
              idGFGx))
          ((G.arrowfunction FGx x).op (counit.arrows x))
          (C.comp Gx GFGx GFGx
            idGFGx
            (C.comp Gx Gx GFGx
              (C.comp Gx GFGx GFGx
                ((GF.arrowfunction Gx Gx).op idGx)
                (adj.unit.arrows Gx))
              idGx))
          (unit.arrows Gx)
          ((C.hom GFGx Gx).er.tra
            (C.comp GFGx Gx Gx
              idGx
              (C.comp GFGx GFGx Gx
                ((G.arrowfunction FGx x).op (adj.counit.arrows x))
                idGFGx))
            (C.comp GFGx GFGx Gx
               ((G.arrowfunction FGx x).op (adj.counit.arrows x))
```

```
                   idGFGx)
                ((G.arrowfunction FGx x).op (counit.arrows x))
                (C.left_unit GFGx Gx
                  (C.comp GFGx GFGx Gx
                    ((G.arrowfunction FGx x).op (adj.counit.arrows x))
                    idGFGx))
                (C.right_unit GFGx Gx
                  ((G.arrowfunction FGx x).op (adj.counit.arrows x))))
              ((C.hom Gx GFGx).er.tra
                 (C.comp Gx GFGx GFGx
                   idGFGx
                   (C.comp Gx Gx GFGx
                     (C.comp Gx GFGx GFGx
                       ((GF.arrowfunction Gx Gx).op idGx)
                       (adj.unit.arrows Gx))
                     idGx))
                   (C.comp Gx GFGx GFGx
                    idGFGx (adj.unit.arrows Gx))
                 (unit.arrows Gx)
                 (C.cong Gx GFGx GFGx
                   idGFGx
                   idGFGx
                   (C.comp Gx Gx GFGx
                     (C.comp Gx GFGx GFGx
                       ((GF.arrowfunction Gx Gx).op idGx)
                       (adj.unit.arrows Gx))
                     idGx)
                   (adj.unit.arrows Gx)
                   ((C.hom GFGx GFGx).er.ref idGFGx)
                   ((C.hom Gx GFGx).er.tra
                     (C.comp Gx Gx GFGx
                       (C.comp Gx GFGx GFGx
                         ((GF.arrowfunction Gx Gx).op idGx)
                         (adj.unit.arrows Gx))
                       idGx)
                     (C.comp Gx GFGx GFGx
                       idGFGx (adj.unit.arrows Gx))
                     (adj.unit.arrows Gx)
                     ((C.hom Gx GFGx).er.tra
                       (C.comp Gx Gx GFGx
                         (C.comp Gx GFGx GFGx
                           ((GF.arrowfunction Gx Gx).op idGx)
                           (adj.unit.arrows Gx))
                         idGx)
                       (C.comp Gx GFGx GFGx
                         ((GF.arrowfunction Gx Gx).op idGx)
                         (adj.unit.arrows Gx))
                       (C.comp Gx GFGx GFGx
                         idGFGx (adj.unit.arrows Gx))
                       (C.right_unit Gx GFGx
                          (C.comp Gx GFGx GFGx
                            ((GF.arrowfunction Gx Gx).op idGx)
                            (adj.unit.arrows Gx)))
                       (C.cong Gx GFGx GFGx
                         ((GF.arrowfunction Gx Gx).op idGx)
                         idGFGx
                         (adj.unit.arrows Gx)
                         (adj.unit.arrows Gx)
                         (GF.ax_preserve_id Gx)
                         ((C.hom Gx GFGx).er.ref (adj.unit.arrows Gx))))
                     (C.left_unit Gx GFGx (adj.unit.arrows Gx))))
                 (C.left_unit Gx GFGx (unit.arrows Gx))))

adj_to_bicat (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
             (adj::E_adjunction C D F G)
 :: E_bicatadjunction ECat C D F G
  = struct
    unit              = adj.unit
    counit            = adj.counit
    unittrianglelaw   = adj.unittrianglelaw
    counittrianglelaw = adj.counittrianglelaw

bicat_to_adj (C,D::E_category)(F::E_functor C D)(G::E_functor D C)
             (adj::E_bicatadjunction ECat C D F G)
 :: E_adjunction C D F G
  = struct
    unit              = adj.unit
    counit            = adj.counit
    unittrianglelaw   = adj.unittrianglelaw
    counittrianglelaw = adj.counittrianglelaw
```