Category Theoretic Structure of Setoids

Yoshiki Kinoshita[a,b,1,*], John Power[c,1]

[a] *National Institute of Advanced Industrial Science and Technology, 3-11-46, Nakoji, Amagasaki, Hyogo 661-0974, Japan*
[b] *Current address: Department of Information Sciences, Kanagawa University, Tsuchiya 2946, Hiratsuka-shi, Kanagawa, 259-1293, JAPAN*
[c] *Department of Computer Science, University of Bath, Bath BA2 7AY, United Kingdom*

Abstract

A setoid is a set together with a constructive representation of an equivalence relation on it. Here, we give category theoretic support to the notion. We first define a category Setoid and prove it is cartesian closed with coproducts. We then enrich it in the cartesian closed category Equiv of sets and classical equivalence relations, extend the above results, and prove that Setoid as an Equiv-enriched category has a relaxed form of equalisers. We then recall the definition of $\mathcal{E}$-category, generalising that of Equiv-enriched category, and show that Setoid as an $\mathcal{E}$-category has a relaxed form of coequalisers. In doing all this, we carefully compare our category theoretic constructs with Agda code for type-theoretic constructs on setoids.

Keywords: setoid, proof assistant, proof irrelevance, Cartesian closed category, coproduct, Equiv-category, Equiv-inserter, $\mathcal{E}$-category, $\mathcal{E}$-coinserter

1. Introduction

The notion of setoid, albeit with different nomenclature, was introduced by Bishop in his development of constructive mathematics [1]. The key difference between it and sets is that one does not have equality of elements of a setoid, the closest approximant to equality being given by a constructive

*Corresponding author
Email addresses: yoshiki-ml@aist.go.jp (Yoshiki Kinoshita), a.j.power@bath.ac.uk (John Power)

[1] This work was done as a part of Dependable Engineering for Open Systems (DEOS) project funded by CREST, Japan Science and Technology Agency (JST).

representation of an equivalence relation, that is, a family of sets indexed by elements of the setoid. The elements of the family can be regarded as *proof objects* of the relation: the relation is considered to hold if and only if the corresponding set in the family is inhabited. Over recent years, Bishop's idea has been taken up in the field of theorem proving using proof assistants including Agda, Coq and Isabelle[2, 3, 4]. Here, we give analysis of the structure of setoids in terms of category theory based on naïve set theory.

The *ordinary* category of setoids and their morphisms is Cartesian closed, but it seems there is no equalisers and coequalisers; even if they do exist, it would be something strange that cannot be used in a straightforward way. So, we consider *enrichment* over Equiv. The Equiv-category of setoids does have Equiv-inserters, which are weaker notion of equalisers, and cotensors, but it still seems to lack coequalisers and any of its weaker form. We then study the *$\mathcal{E}$-category* of setoids. The $\mathcal{E}$-category **Setoid** does not only have Cartesian closed structure, $\mathcal{E}$-inserters and cotensors, but also $\mathcal{E}$-coinserters and tensors. These are enough to say that there always exist a *weak notion of* limit and colimit of arbitrary (small) diagram in the $\mathcal{E}$-category of setoids. In fact, we give an Agda code which claims the existence in the Appendix.

We adopt the usual semantic practice of modelling a type by a set and modelling a term in context by a function. The definition of setoid inherently involves a type Set, so we shall assume we have a model of set theory and, with mild overloading of notation, use Set to denote the set of small sets, equivalently a model of sets.

Having adopted those conventions, a setoid A, in classical set-theoretic terms, consists of:

- a set $|A|$
- a family $\approx_A$ of sets indexed by $|A| \times |A|$ (We write $a_0 \approx_A a_1$ for the set indexed by (a_0, a_1).)
- for each $a \in |A|$, an element $\mathsf{refl}_A(a)$ of $a \approx_A a$
- for each pair (a_0, a_1) of elements of $|A|$, a function $\mathsf{sym}_A(a_0, a_1)$: $(a_0 \approx a_1) \longrightarrow (a_1 \approx a_0)$
- for each triple (a_0, a_1, a_2) of elements of $|A|$, a function $\mathsf{trans}_A(a_0, a_1, a_2)$: $(a_1 \approx a_2) \times (a_0 \approx a_1) \longrightarrow (a_0 \approx a_2)$

There is some choice about a natural notion of map between setoids, but one natural option, which we shall make, is that a morphism $f\colon A \longrightarrow B$ consists of:

- a function $\mathsf{fun}_f\colon |A| \longrightarrow |B|$ together with
- for each pair (a_0, a_1) of elements of $|A|$, a function $\mathsf{resp}_f\colon (a_0 \approx a_1) \longrightarrow (\mathsf{fun}_f(a_0) \approx \mathsf{fun}_f(a_1))$.

These definitions can be described by the following Agda code.

```
record Setoid : Set₁ where
  field
    carrier : Set
    _≈_ : carrier → carrier → Set
    refl : {x : carrier} → x ≈ x
    sym : {x y : carrier} → x ≈ y → y ≈ x
    trans : {x y z : carrier} → y ≈ z → x ≈ y → x ≈ z

record _⇝_ (A B : Setoid) : Set where
  open Setoid ; _≅_ = _≈_ A ; _≃_ = _≈_ B
  field
    fun : carrier A → carrier B
    resp : {a₀ a₁ : carrier A} → a₀ ≅ a₁ → fun a₀ ≃ fun a₁
```

The most striking fact about the definition of setoids is the absence of coherence axioms. In particular, the data for reflexivity, symmetry and transitivity are exactly data appropriate for the definition of a groupoid: if one added natural coherence axioms to the definition of setoid, one would in fact have the definition of a groupoid. A central idea in the definition of setoid is *not* to insist upon equality between proof objects. The result is that setoids behave quite differently to groupoids or categories.

The behaviour of setoids would be simpler if the sets $a_0 \approx a_1$ were degenerated into singletons or instances of the empty set. That would correspond to the study of the category Equiv of equivalence relations.

The implications of the lack of coherence axioms are profound. For instance, a morphism of setoids, in contrast to a functor, need not preserve the data for reflexivity or transitivity: it follows from the definition of functor that functors preserve n-fold composition for any natural number n, whereas,

in the absence of a coherence axiom for transitivity, that would not hold if one imposed the usual functoriality condition on a morphism of setoids. And although we will consider equivalences between morphisms of setoids, (cf. natural isomorphisms between functors), it does not make sense to impose a naturality condition on them as, again in the absence of a coherence axiom for transitivity, a composite of such natural transformations would not be natural.

Setoids and morphisms between them generate a category Setoid. The lack of a requirement that the reflexivity, symmetry and transitivity data is preserved by a morphism of setoids impacts on the structure of the category Setoid. If such axioms were imposed on morphisms, the category Setoid would be locally finitely presentable, hence complete and cocomplete. But in fact Setoid seems not to have equalisers, although it does have products and is Cartesian closed.

We will duly study the structure of the category Setoid in this paper, in particular proving that it has products and coproducts and is Cartesian closed: the latter is quite complex. But in theorem proving practice, this category is not of interest per se: constructively, one cannot assert that parallel morphisms $f, g\colon A \longrightarrow B$ are equal; one can only assert that for each a in $|A|$, the set $f(a) \approx_B g(a)$ is inhabited, i.e., is non-empty. We extend Setoid to provide semantics to express the fact of two morphisms of setoids being equivalent, but not necessarily equal.

In order to provide such structure, we extend Setoid with the canonical structure of an Equiv-enriched category, Equiv being Cartesian closed. We induce an Equiv-enrichment of Setoid from the canonical Equiv-enrichment of Equiv. Cartesian closedness and coproducts extend from Setoid as an ordinary category to **Setoid** as an Equiv-enriched category. We further prove that **Setoid** as an Equiv-enriched category has a relaxed form of equaliser that we call an Equiv-inserter, cf. [5].

We make one further step. Equiv-categories have underlying ordinary categories, thus have strict associativity of morphisms. And the structures one considers on Equiv-categories, such as Equiv-products and Equiv-closedness reflect that strictness. But constructively, setoids do not have such strictness. We take advantage of that to prove the existence of further constructions on setoids, such as a relaxed notion of coequaliser that we call an $\mathcal{E}$-coinserter. Equiv-products are *a fortiori* $\mathcal{E}$-products, etc.

The central idea here is that every Equiv-category is an $\mathcal{E}$-category, as we shall discuss. In fact, we show that an Equiv-category is precisely a

strict $\mathcal{E}$-category, Equiv-categories being to $\mathcal{E}$-categories as 2-categories are to bicategories. It is **Setoid** as an $\mathcal{E}$-category about which the type-theoretic theorems and proofs about setoids hold.

This paper is organised as follows. In Section 2, the construction of the ordinary category Equiv of equivalence relations is introduced and its completeness, cocompleteness and Cartesian closedness are proved. This Cartesian closed category is important as a category over which Setoid will be enriched. In Section 3, we introduce another ordinary category Setoid of setoids and show that it is Cartesian closed and has coproducts. In Section 4, we introduce the notion of Equiv-enriched categories with an elementary description; Equiv is enriched to Equiv-enriched category **Equiv**. We extend the ordinary category Setoid to the Equiv-enriched category **Setoid** and study its structure in Section 5. The notion of $\mathcal{E}$-category is introduced in Section 6 and the structure of the specific $\mathcal{E}$-category **Setoid** is studied in Section 7. We conclude in Section 8. In Appendix A, we attach Agda code for setoids and the constructions in the $\mathcal{E}$-category **Setoid** studied in this paper.

Related work

The notion of setoids as presented above is folklore in theorem proving and its use can be traced back at least to Peter Aczel's unpublished report [6]. Čubrić, Dybjer and Scott [7] introduced $\mathcal{P}$-categories, which can be obtained by replacing equivalence relations in $\mathcal{E}$-categories by partial equivalence relations. The first author studied $\mathcal{E}$-categories in connection with bicategories in [8], where the $\mathcal{E}$-equivalence of $\mathcal{E}$-categories and strict $\mathcal{E}$-categories is essentially but only implicitly described. Wilander defined a notion of $\mathcal{E}$-bicategory and studied an $\mathcal{E}$-bicategory of $\mathcal{E}$-categories, in particular Setoids [9, 10, 11, 12]. However, his definitions are given in terms of constructive type theory, whereas ours are given in terms of naïve set theory. We then use the set theoretic notions to study setoids described in constructive type theory.

There has been considerable work on constructive mathematics from a category theoretic perspective, in particular using techniques derived from topos theory and related to taking the exact completion of Set [13, 14, 15, 16, 17, 18].

2. The category Equiv of equivalence relations

A set with an equivalence relation is directly and naturally modelled by the following category Equiv.

Construction 1. The category Equiv consists of the following data.

- Objects are pairs $A = (|A|, \equiv_A)$ of a small set $|A|$ and an equivalence relation $\equiv_A$ on it.
- Morphisms from A to B are functions from $|A|$ to $|B|$ that respect the equivalence relation, i.e., functions $f\colon |A| \longrightarrow |B|$ such that $f(a_0) \equiv_B f(a_1)$ whenever $a_0 \equiv_A a_1$.
- Composition is given by composition of functions.
- Identities are identity functions.

These data satisfy the conditions for a category.

We shall call an object of Equiv an *equivalence relation*.

Theorem 2. Equiv *is complete and cocomplete.*

Proof. A product diagram in Equiv for a family $A = (A_i \mid i \in I)$ is given pointwise by

$$(P \xrightarrow{\pi_i} A_i \mid i \in I),$$

where $|P| \xrightarrow{\pi_i} |A_i|$ is the product diagram in Set and $\equiv_P$ is given by $p \equiv_P p'$ if and only if $\pi_i(p) \equiv_{A_i} \pi_i(p')$ for all $i \in I$. Each π_i evidently respects the equivalence relations, so is a morphism in Equiv. The universal property holds because it is a product cone in Set.

An equaliser for a parallel pair of morphisms $f, g\colon A \longrightarrow B$ in Equiv is given by

$$E \xrightarrow{\quad e \quad} A \underset{g}{\overset{f}{\rightrightarrows}} B$$

where $|E| \xrightarrow{e} |A| \underset{g}{\overset{f}{\rightrightarrows}} |B|$ is the equaliser diagram in Set and $\equiv_E$ is defined by $x \equiv_E y$ if and only if $e(x) \equiv_A e(y)$. This clearly makes e a morphism in Equiv, and the universal property holds because the above is an equaliser diagram in Set.

A coproduct of $(A_i \mid i \in I)$ is also given pointwise. Let $|C|$ be the disjoint union of sets:

$$|C| \stackrel{\text{def}}{=} \coprod \{\, |A_i| \mid i \in I \,\}$$

and define $\equiv_C$ by

$$\equiv_C \stackrel{\text{def}}{=} \coprod \{\, \equiv_{A_i} \mid i \in I \,\}$$

Clearly $\equiv_C$ is an equivalence relation on C and each injection $\iota_i\colon |A_i| \longmapsto |C|$ respects the equivalence relation, so is a morphism in Equiv. It is routine to verify that $(\iota_i\colon A_i \longrightarrow C \mid i \in I)$ is a coproduct diagram in Equiv.

Finally, a coequaliser of a pair of parallel morphisms $f, g\colon A \longrightarrow B$ is given as follows. Define the set $|\mathrm{Coeq}|$ so that $|A| \underset{g}{\overset{f}{\rightrightarrows}} |B| \overset{\mathrm{coeq}}{\to} |\mathrm{Coeq}|$ is the coequaliser in Set.

A binary relation $\equiv_{\mathrm{Coeq}}$ is defined by the transitive closure of the image of $\equiv_B$ under coeq. It is an equivalence relation: reflexivity and symmetry follow from surjectivity of coeq. Therefore $\mathrm{Coeq} = (|\mathrm{Coeq}|, \equiv_{\mathrm{Coeq}}$ is an object and coeq is a morphism of Equiv. It is routine to verify the universal property. $\square$

Theorem 3. Equiv *is Cartesian closed.*

Proof. Given two objects B and C of Equiv, the exponential $[B, C]$ is given by the set of functions $f\colon |B| \longrightarrow |C|$ such that $b_0 \equiv_B b_1$ implies $f(b_0) \equiv_C f(b_1)$, and with equivalence relation given by $f \equiv_{[B,C]} g$ if for all $b \in B$, one has $f(b) \equiv_C g(b)$. The counit $\varepsilon \in \mathsf{Equiv}([B, C] \times B, C)$ sends (f, b) to $f(b)$; to see it is a morphism of Equiv, assume $f \equiv_{[B,C]} f'$ and $b \equiv_B b'$. Then $\varepsilon(f, b) = f(b) \equiv f'(b) \equiv f'(b') = \varepsilon(f', b')$, as required.

To show the universal property, let $A, B, C \in \mathrm{ob}(\mathsf{Equiv})$ and $f \in \mathsf{Equiv}(A \times B, C)$. The currying operator in Set maps f to $\bar{f}\colon |A| \ni a \longmapsto (\lambda b.\, f(a, b)) \in (|B| \to |C|)$, where $(|B| \to |C|)$ is the set of functions from $|B|$ to $|C|$. We first wish to show $\bar{f}(a) \in |[B, C]|$, for all $a \in |A|$. But if $b \equiv b'$, then $\bar{f}(a)(b) = f(a, b) \equiv_C f(a, b') = \bar{f}(a)(b')$ because f respects the equivalence relation. It is obvious that $\bar{f}$ is the unique function that makes the following triangle commute.

$$\begin{array}{ccc} & & A \times B \\ & \overset{\bar{f} \times \mathrm{id}_B}{\swarrow} & \big\downarrow f \\ [B, C] \times B & \xrightarrow[\varepsilon]{} & C \end{array}$$

It remains to show $\bar{f} \in \mathsf{Equiv}(A, [B, C])$, that is, it is a morphism in Equiv. But if $a \equiv_A a'$, $\bar{f}(a) = \lambda b.\, f(a, b) \equiv_{[B,C]} \lambda b.\, f(a', b) = \bar{f}(a')$ as $f(a, b) \equiv_{[B,C]} f(a', b)$ for all $b \in |B|$. □

3. The category Setoid of setoids

A setoid is a constructive representation of an equivalence relation. We can study them in classical terms as follows.

Construction 4. The following data form a category, which we call Setoid.

- Objects are setoids as defined in the Introduction.
- For setoids A, B, $\mathsf{Setoid}(A, B)$ is the set of all morphisms from A to B, also as defined in the Introduction.
- For setoids A, B and C and morphisms $f\colon A \longrightarrow B$ and $g\colon B \longrightarrow C$, the morphism $g \circ f\colon A \longrightarrow C$ is defined to be the pair of the composite $\mathsf{fun}_g \circ \mathsf{fun}_f\colon |A| \longrightarrow |C|$ and, for each $a_0, a_1 \in |A|$, the composite $\mathsf{resp}_g \circ \mathsf{resp}_f$.
- For a setoid A, id_A is the pair of the identity function on $|A|$ and, for each $a_0, a_1 \in |A|$, the identity function on $a_0 \approx_A a_1$.

Setoid is equivalent to a full subcategory of the functor category from the pair of parallel arrows to Set. It is not reflective as equalisers in Setoid are not given pointwise. We will not develop that approach to Setoid in this paper.

The equality between elements of homsets is referred to in these conditions, but such equalities are not available in proof assistants based on constructive type theory, such as Agda. Despite that, the structure of the category Setoid is studied in the rest of this section.

Theorem 5. Setoid *has all products and coproducts.*

Proof. We first construct binary products in Setoid, but the product of an arbitrary number of object is constructed similarly. Let A, B be setoids. We define the setoid $A \times B$ by

- $|A \times B| = |A| \times |B|$
- $(a, b) \approx_{A \times B} (a', b') = (a \approx_A a') \times (b \approx_B b')$

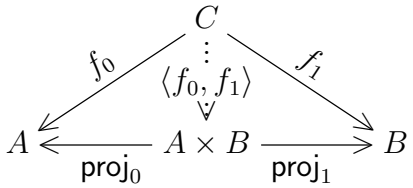
- $\mathsf{refl}_{A\times B}((a,b)) = (\mathsf{refl}_A(a), \mathsf{refl}_B(b))$
- $\mathsf{sym}_{A\times B}\Big((a,b),(a',b')\Big) = \mathsf{sym}_A(a,a') \times \mathsf{sym}_B(b,b')$
  which is a function from $(a \approx_A a') \times (b \approx_B b')$ to $(a' \approx_A a) \times (b' \approx_B b)$.
- $\mathsf{trans}_{A\times B}\Big((a,b),(a',b'),(a'',b'')\Big) = \mathsf{trans}_A(a,a',a'') \times \mathsf{trans}_B(b,b',b'')$,
  a function from $\Big((a' \approx_A a'') \times (a \approx_A a')\Big) \times \Big((b' \approx_B b'') \times (b \approx_B b')\Big)$ to $(a \approx_A a'') \times (b \approx_B b'')$.

The projection $\mathsf{proj}_0 \in \mathsf{Setoid}(A \times B, A)$ is defined as follows.

- $\mathsf{fun}_{\mathsf{proj}_0} \stackrel{\text{def}}{=} \pi_0$, where π_0 is the projection function from $|A| \times |B|$ to $|A|$.
- $\mathsf{resp}_{\mathsf{proj}_0} \stackrel{\text{def}}{=} \pi_0$, where π_0 is the projection function from $(a \approx_A a') \times (b \approx_B b')$ to $(a \approx_A a')$.

The other projection $\mathsf{proj}_1 \colon A \times B \longrightarrow B$ is defined symmetrically.

To show the universal property, let $A \xleftarrow{f_0} C \xrightarrow{f_1} B$ be a cone on A and B. Then we can define $\langle f_0, f_1\rangle \colon C \longrightarrow A \times B$ by $\mathsf{fun}_{\langle f_0,f_1\rangle}(c) = (f_0(c), f_1(c))$ and $\mathsf{resp}_{\langle f_0,f_1\rangle} = (\mathsf{resp}_{f_0}, \mathsf{resp}_{f_1})$.



The above diagram commutes because $(\mathsf{fun}_{\mathsf{proj}_i} \circ \mathsf{fun}_{\langle f_0,f_1\rangle})(c) = \mathsf{fun}_{\mathsf{proj}_i}(f_0(c), f_1(c)) = f_i(c)$ for $i = 0, 1$, and $\mathsf{resp}_{\mathsf{proj}_i} \circ \mathsf{resp}_{\langle f_0,f_1\rangle} = \mathsf{resp}_{\mathsf{proj}_i}$ similarly. It is obvious that $\langle f_0, f_1\rangle$ is the only such.

A coproduct of A_0 and A_1 is also given pointwise. We define the setoid $A_0 + A_1$ as follows.

- $|A_0 + A_1| \stackrel{\text{def}}{=} |A_0| + |A_1|$.
  Let $\iota_i \colon |A_i| \longrightarrow |A_0 + A_1|$ be the injection morphism.
- $\approx_{A_0+A_1} = \approx_{A_0} + \approx_{A_1}$
- $\mathsf{refl}_{A_0+A_1}(i,a)) \stackrel{\text{def}}{=} (i, \mathsf{refl}_{A_i}(a))$.

- $\mathsf{sym}_{A_0+A_1}((i,a),(j,a'))((m,p)) \stackrel{\text{def}}{=} (i, \mathsf{sym}_{A_i}(a,a',p))$.
  (m,p) is an element of $(i,a) \approx_{A_m} (j,a')$, so i, j and m must all be equal. Then $\mathsf{sym}_{A_0+A_1}((i,a),(i,a'))((i,p))$ is the pair of i and $\mathsf{sym}_{A_i}(a,a')(p)$.

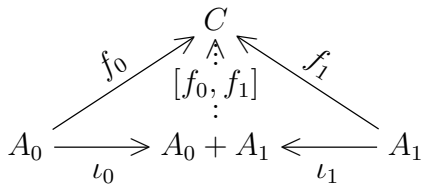
- $\mathsf{trans}_{A_0+A_1}((i,a),(j,a'),(k,a''))((m,p),(n,q)) = (i, \mathsf{trans}_{A_i}(a,a',a'')(p,q)$.
  (m,p) is an element of $(j,a') \approx_{A_m} (k,a'')$, so j, k and m must be equal; similarly, $(n,q) \in ((i,a) \approx_{a_n} (j,a'))$, so $i = j = n$. Therefore $i = j = k = m = n$. The element $\mathsf{trans}_{A_0+A_1}((i,a),(i,a'),(i,a''))((i,p),(i,q))$ is the pair of i and $\mathsf{trans}_{A_i}(a,a,a'')(p,q)$.

The injection $\iota_i \in \mathsf{Setoid}(A_i, A_0 + A_1)$, $(i = 0, 1)$ is defined as follows.

- $\mathsf{fun}_{\iota_i}\colon |A_i| \longrightarrow |A_0 + A_1|$ is defined by $\mathsf{fun}_{\iota_i}(a) \stackrel{\text{def}}{=} (i,a)$.

- For $a, a' \in A_i$, $\mathsf{resp}_{\iota_i}\colon a \approx_{A_i} a' \longrightarrow \mathsf{fun}_{A_i}(a) \approx_{A_0+A_1} \mathsf{fun}_{A_i}(a')$ is defined by $\mathsf{resp}_{\iota_i}(p) = (i,p)$

Then the bottom line in the diagram below is a coproduct diagram in Setoid.



To show it, let C be an arbitrary object and $f_0 \in \mathsf{Setoid}(A_0, C)$, $f_1 \in \mathsf{Setoid}(A_1, C)$ be arbitrary morphisms in Setoid. Define $[f_0, f_1] \in \mathsf{Setoid}(A_0 + A_1, C)$ as follows.

- $\mathsf{fun}_{[f_0,f_1]}((i,a)) \stackrel{\text{def}}{=} \mathsf{fun}_{f_i}(a)$

- For elements a_i and a_i' of $|A_i|$ and p of $a_i \approx_{A_i} a_i'$, $\mathsf{resp}_{[f_0,f_1]}((i,p)) \stackrel{\text{def}}{=} \mathsf{resp}_{f_i}(p)$.

Both triangles commute: the left triangle commutes because $(\mathsf{fun}_{[f_0,f_1]} \circ \mathsf{fun}_{\iota_0})(a) = \mathsf{fun}_{[f_0,f_1]})((0,a)) = \mathsf{fun}_{f_0}(a)$ for all a in $|A_0|$, and $(\mathsf{resp}_{[f_0,f_1]} \circ \mathsf{resp}_{\iota_0})(p) = \mathsf{resp}_{f_0}(p)$ and the right triangle commutes similarly.

To see the uniqueness, let $h \in \mathsf{Setoid}(A_0+A_1, C)$ be a morphism such that $h \circ \iota_i = f_i$ for $i = 0$ and 1, and we shall show $h = [f_0, f_1]$. The uniqueness of the fun part is obvious because it is a coproduct diagram in Set. To show the uniqueness of the resp part, let x and x' be elements of $|A_0 + A_1|$ and p be an

element of $x \approx_{A_0+A_1} x'$. Because $x \approx_{A_0+A_1} x'$ is inhabitant, x, x' and p are of the form (i, a), (i, a') and (i, q) for the same i, $a, a' \in |A_i|$ and $q \in a \approx_{A_i}$. So, $\mathsf{resp}_h(p) = \mathsf{resp}_h((i, q)) = \mathsf{resp}_h(\iota_i(q)) = \mathsf{resp}_{f_i}(q) = \mathsf{resp}_{[f_0,f_1]}(\iota_i(q)) = \mathsf{resp}_{[f_0,f_1]}(p)$, but p is arbitrary, so $\mathsf{resp}_h = \mathsf{resp}_{[f_0,f_1]}$, as required. □

Given setoids A and B, an element of the exponential $[A, B]$ is *not* given by a morphism of setoids from A to B. Rather, it is given by a function $f\colon |A| \longrightarrow |B|$ for which, for all a_0 and a_1 in $|A|$, there *exists* a function φ from $a_0 \approx_A a_1$ to $f(a_0) \approx_B f(a_1)$. So, rather than being a morphism of setoids, an element of the exponential is a morphism of the induced objects of Equiv that is introduced in Section 2.

Theorem 6. Setoid *is Cartesian closed.*

Proof. Leaving the reflexivity, symmetry and transitivity data implicit, given setoids A, B and C, a morphism f of setoids from $A \times B$ to C consists of a function $\mathsf{fun}_f\colon |A| \times |B| \longrightarrow |C|$ together with, for all $a, a' \in |A|$ and all $b, b' \in |B|$, a function

$$\mathsf{resp}_f\colon (a \approx_A a') \times (b \approx_B b') \longrightarrow f(a, b) \approx_C f(a', b')$$

These data can be re-expressed as functions $\mathsf{fun}_h\colon |A| \longrightarrow (|B| \to |C|)$ and

$$\mathsf{resp}_h\colon (a \approx_A a') \longrightarrow ((b \approx_B b') \to (h(a)(b) \approx_C h(a')(b')))$$

So data for a *potential* exponential $[B, C]$ is given by the set $(|B| \to |C|)$ of all functions $k\colon |B| \longrightarrow |C|$, with $k \approx_{[B,C]} k'$ given by the product over all $b, b' \in |B|$ of $((b \approx_B b') \to (k(b) \approx_C k'(b')))$.

However, this data does not satisfy the reflexivity axiom in the definition of setoid: for a setoid, for each $k \in [B, C]$, the set $k \approx_{[B,C]} k$ must be non-empty. That need not be true for an arbitrary function $k\colon |B| \longrightarrow |C|$, but it is true for any function k that underlies a morphism of setoids from B to C.

In fact, given a morphism of setoids f from $A \times B$ to C, for any $a \in |A|$, the function $\mathsf{fun}_f(a, -)\colon |B| \longrightarrow |C|$ does underlie a morphism of setoids.

So an exponential $[B, C]$ does exist: an element is a function $k\colon |B| \longrightarrow |C|$ for which, for every $b, b' \in |B|$, there exists a function from $b \approx_B b'$ to $k(b) \approx_C k(b')$, with $\approx_{[B,C]}$ defined as above.

It is routine to verify that this gives data for a setoid; and its universal property holds by construction. □

From the perspective of setoids as a type-theoretic construct, the closed structure of the category Setoid is remarkable: the elements of the exponential necessarily involve an existence condition because of the requirement of existence of reflexivity, but the family is inherently constructive.

Moreover, $k \approx_{[B,C]} k'$ is given in a specific way: it assigns to each pair b, b' of elements of $|B|$, a function from $b \approx_B b'$ to $k(b) \approx_C k'(b')$, rather than assigning, to each single element $b \in |B|$, a function from $b \approx_B b$ to $k(b) \approx_C k'(b)$. These are not interchangeable: the replacement of the first by the second does not describe the closed structure of the category Setoid.

However, up to equivalence, they do agree. So, when we consider Setoid as an $\mathcal{E}$-category in Section 6, its closed structure as an ordinary category is also $\mathcal{E}$-closed structure, but $\mathcal{E}$-closed structure is unique only up to equivalence, whereas ordinary closed structure is unique up to isomorphism. So in **Setoid** as an $\mathcal{E}$-category, the closed structure as we have defined it and the alternative above both act as $\mathcal{E}$-closed structures.

A morphism in Setoid from 1 to an exponential $[A, B]$ consists of a function from the set 1 to the carrier of the exponential, i.e., a function $\mathsf{fun}_f : |A| \longrightarrow |B|$ for which there *exist* functions from $a \approx_A a'$ to $\mathsf{fun}_f(a) \approx_B \mathsf{fun}_f(a')$, together with a function from 1 to the set of families of functions, of the form resp_f: $a \approx_A a' \longrightarrow \mathsf{fun}_f(a) \approx_B \mathsf{fun}_f(a')$. The existence clause is therefore redundant.

4. Equiv-enriched categories

Categories can be *enriched* over Equiv, as the latter is Cartesian closed. The definition of Equiv-enriched category, or simply Equiv-category, is only an instance of $\mathcal{V}$-enriched category for symmetric monoidal $\mathcal{V}$, but we explicitly state it for the purpose of self-containedness.

Definition 7. An Equiv-enriched category C consists of the following data:

- a set $\mathrm{ob}(\mathsf{C})$, elements of which are called objects of C.
- for each $C, C' \in \mathrm{ob}(\mathsf{C})$, an object $\mathsf{C}(C, C')$ of Equiv.
- for each $C, C', C'' \in \mathrm{ob}(\mathsf{C})$, a function $\Box$: $|\mathsf{C}(C', C'') \times \mathsf{C}(C, C')| \longrightarrow |\mathsf{C}(C, C'')|$ such that $(g \Box f) \equiv_{\mathsf{C}(C,C'')} (g' \Box f')$ whenever $g \equiv_{\mathsf{C}(C',C'')} g'$ and $f \equiv_{\mathsf{C}(C,C')} f'$.
- for each $C \in \mathrm{ob}(\mathsf{C})$, an object $\mathrm{id}_C \in |\mathsf{C}(C, C)|$

These data are subject to the following three conditions.

- For $C, C', C'', C''' \in \mathrm{ob}(\mathsf{C})$ and $f \in |\mathsf{C}(C, C')|$, $g \in |\mathsf{C}(C', C'')|$, $h \in |\mathsf{C}(C'', C''')|$,
$$h \Box (g \Box f) = (h \Box g) \Box f.$$
- For $C, C' \in \mathrm{ob}(\mathsf{C})$ and $f \in |\mathsf{C}(C, C')|$,
$$f \Box \mathrm{id}_C = f.$$
- For $C, C' \in \mathrm{ob}(\mathsf{C})$ and $f \in |\mathsf{C}(C, C')|$,
$$\mathrm{id}_{C'} \Box f = f.$$

So, Equiv-categories have hom-equivalence-classes rather than homsets. Another way of looking at it is that an equivalence relation is defined for parallel pairs of morphisms (elements of the carrier of homobjects).

Given an Equiv-category $\mathbf{C}$, its *underlying category* $\mathbf{C}_\circ$ is defined as follows. Its objects are the same as the objects of $\mathbf{C}$. The homset $\mathbf{C}_\circ(x, y)$ is the carrier set $|\mathbf{C}(x, y)|$ of the homobject for these objects. The composition and identity of the underlying category are the same as those of $\mathbf{C}$.

Let C be a category. If an Equiv-category is defined so that its underlying category coincides with C, we say C is *enriched* to it. In order to enrich a category we have only to give an equivalence relation on each homset such that the composition respects those equivalence relations.

Construction 8. The category Equiv is enriched to the Equiv-enriched category **Equiv**. The equivalence relation on each homset is defined so that $f \equiv g$ if and only $f(a) \equiv g(a)$ for all a.

To see the composition preserves these equivalence relations, let $f \equiv g$ and $f' \equiv g'$, with domains and codomains agreeing. Then for all a, $f'f(a) \equiv f'g(a) \equiv g'g(a)$, the first of these holding because $f \equiv g$ and because f' is a morphism of equivalence relations, with the latter holding because $f' \equiv g'$.

The general theory of enriched categories, or more specifically that of 2-categories, determines definitions of Equiv-functor, Equiv-natural transformation, Equiv-enriched adjoint, Equiv-products, Equiv-coproducts, Equiv-cotensors, and Equiv-closedness [5, 19]. We also adopt the notion of inserter from the theory of 2-categories [5]. We can express these definitions in elementary terms of Equiv-categories.

For example, given Equiv-enriched categories $\mathbf{C}$ and $\mathbf{D}$, an Equiv-*functor* from $\mathbf{C}$ to $\mathbf{D}$ is an ordinary functor $H\colon \mathbf{C}_\circ \longrightarrow \mathbf{D}_\circ$ such that if $f \equiv_{\mathbf{C}(c,c')} f'$, then $H(f) \equiv_{\mathbf{D}(H(c),H(c'))} H(f')$. Just as $\mathbf{C}$ has an ordinary underlying category $\mathbf{C}_\circ$, an Equiv-functor H has an underlying ordinary functor $H_\circ$: it has exactly the same data as H.

An Equiv-*enriched natural transformation* from H to K is exactly an ordinary natural transformation from $H_\circ$ to $K_\circ$.

Equiv-categories, Equiv-functors and Equiv-natural transformations form a 2-category, as is the case for enriched categories in general, and that determines a definition of Equiv-enriched adjoint. In elementary terms, an Equiv-functor $H\colon \mathbf{C} \longrightarrow \mathbf{D}$ has an Equiv-enriched left adjoint if the ordinary functor $H_\circ\colon \mathbf{C}_\circ \longrightarrow \mathbf{D}_\circ$ has an ordinary left adjoint L subject to the additional condition that the bijections

$$\mathbf{C}_\circ(L(d), c) \cong \mathbf{D}_\circ(d, H(c))$$

respect the equivalence relations on $\mathbf{C}(L(d), c)$ and $\mathbf{D}(d, H(c))$.

Given objects a and b of an Equiv-enriched category $\mathbf{C}$, an Equiv-*product* of a and b is a product $a \times b$ in $\mathbf{C}_\circ$ subject to one additional property: if $f \equiv_{\mathbf{C}(c,a)} f'$ and $g \equiv_{\mathbf{C}(c,b)} g'$, then $(f, g) \equiv_{\mathbf{C}(c,a\times b)} (f', g')$.

Equiv-*coproduct* is the dual notion to Equiv-product.

The notion of cotensor appears generally in enriched category theory, but not often in ordinary category theory, so we spell out its definition in detail here.

Definition 9. Given an equivalence relation X and an object a of an Equiv-category $\mathbf{C}$, an Equiv-*cotensor* a^X of a by X is an object with the universal property that, for any object b of $\mathbf{C}$, there is a natural bijection between the set of morphisms of equivalence relations

$$X \longrightarrow \mathbf{C}(b, a)$$

and the set of morphisms in $\mathbf{C}$

$$b \longrightarrow a^X$$

with the bijection respecting equivalent morphisms.

Definition 10. An Equiv-*inserter* of morphisms $f, g\colon a \longrightarrow b$ in an Equiv-category $\mathbf{C}$ is an object e together with a morphism $\iota\colon E \longrightarrow A$ such that

$f \circ \iota$ is equivalent to $g \circ \iota$, universally so, i.e., for any object c and morphism $\gamma\colon c \longrightarrow a$ such that $f \circ \gamma \equiv_{C(c,b)} g \circ \gamma$, there is a unique morphism $\bar{\gamma}\colon c \longrightarrow e$ such that $\iota \circ \bar{\gamma} = \gamma$. Moreover, $\gamma \equiv_{C(c,a)} \gamma'$ implies $\bar{\gamma} \equiv_{C(c,e)} \bar{\gamma'}$.

An Equiv-category $\mathbf{C}$ with finite products is called Equiv-*closed* if for every object a of $\mathbf{C}$, the Equiv-functor $(- \times a)\colon \mathbf{C} \longrightarrow \mathbf{C}$ has an Equiv-enriched right adjoint.

5. The Equiv-category Setoid of setoids

Although setoids and equivalence relations are different, the former are often considered to be a representation of the latter in type theoretic practice. The construction of the following reflection explains it.

Proposition 11. *There is an evident inclusion* $J\colon \mathsf{Equiv} \longrightarrow \mathsf{Setoid}$*, and it has a left adjoint* F *that sends a setoid* A *to an equivalence relation* $(|A|, \equiv_A)$ *where* $a \equiv_A a'$ *if and only if* $(\exists p)\, p \in (a \approx_A a')$.

$$F \dashv J\colon \mathsf{Equiv} \hookrightarrow \mathsf{Setoid}$$

The left adjoint F corresponds to "degenerating" the proofs of equivalence. We use F to give the category Setoid a canonical Equiv-enrichment as follows.

Definition 12. The following data defines an Equiv-category **Setoid**.

- The set of objects is the set of setoids.
- For setoids A, B, $\mathbf{Setoid}(A, B)$ is the equivalence relation on the set of setoid morphisms from A to B, where two morphisms f and g are equivalent if and only if $F(f)$ and $F(g)$ are equivalent in $\mathsf{Equiv}(F(f), F(g))$.
- For all setoids A, B and C, the setoid morphism $\Box_{ABC}\colon \mathbf{Setoid}(B, C) \times \mathbf{Setoid}(A, B) \longrightarrow \mathbf{Setoid}(A, C)$ is defined by $f \Box g = f \circ g$, where $\circ$ is the composition in the category Setoid. Observe that if $f \equiv f'$ and $g \equiv g'$, then $f \Box g \equiv f' \Box g'$.

The ordinary category Setoid has products and coproducts; they enrich to Equiv, i.e., the same constructions satisfy the properties required to be Equiv-products and Equiv-coproducts.

The closed structure of Setoid as an ordinary category extends to closed structure of **Setoid** as an Equiv-category: for any setoid A, the ordinary functor $(-\times A)$: Setoid $\longrightarrow$ Setoid extends to an Equiv-functor, i.e., it respects equivalences between morphisms in Setoid, and its ordinary right adjoint satisfies the property required to be an Equiv-enriched right adjoint.

5.1. Inserters in the Equiv*-category* **Setoid**

Equalisers seem not to exist in Setoid as an ordinary category: if they do, they certainly are not given pointwise. So that *a fortiori* is also true of **Setoid** as an Equiv-enriched category. Similarly, coequalisers do not seem to exist in **Setoid** as an ordinary category, so, *a fortiori*, seem not to exist in **Setoid** as an Equiv-category. Nevertheless, **Setoid** does have Equiv-inserters (Definition 10).

Theorem 13. *All parallel pairs of morphisms in the* Equiv*-category* **Setoid** *have* Equiv*-inserters.*

Proof. Given f, g: $A \longrightarrow B$ in Setoid, let E be the set of elements a of $|A|$ for which $f(a) \approx_B g(a)$ is inhabited, and define $\approx_E$ by restriction of $\approx_A$. Define e: $E \longrightarrow A$ by inclusion.

It is routine to verify that this satisfies the axsa for an Equiv-inserter. □

Equiv-inserters are remarkably non-constructive, and in that specific sense, they differ from iso-inserters in the theory of 2-categories. Their non-constructiveness means that the construction of E in Theorem 13 does not directly correspond to Agda code. For the latter, one wants not just an element a of $|A|$ for which $f(a) \approx_B g(a)$ is inhabited, but rather an element a together with an element of $f(a) \approx_B g(a)$, but such an object is not the Equiv-inserter, and it seems not to be a limit in the Equiv-category **Setoid**.

However, as we shall see, the definition of Equiv-inserter extends naturally to a definition of $\mathcal{E}$-inserter, for which one weakens the commutativity condition $e \circ \bar{x} = x$ to the condition that $e \circ \bar{x}$ is equivalent to x. Doing so means that $\mathcal{E}$-inserters are only defined by to equivalence, rather than up to isomorphism, upon which the natural Agda code does yield an $\mathcal{E}$-inserter, one that is equivalent to the canonical choice determined by the Equiv-inserter, which is, *a fortiori*, an $\mathcal{E}$-inserter.

The situation for coinserters is quite different. The Equiv-category **Setoid** seems not to have Equiv-coinserters, where the notion of coinserter is dual to that of inserter. But it does have $\mathcal{E}$-coinserters and these agree with Agda code.

5.2. Cotensors in the Equiv*-category* **Setoid**

Theorem 14. *The* Equiv*-category* **Setoid** *has cotensors, given as follows. An element of* $|B^X|$ *is a function* $h\colon |X| \longrightarrow |B|$ *such that if* $x \equiv x'$*, then* $h(x) \approx_B h(x')$ *is nonempty (inhabited). An element of* $h \approx_{B^X} h'$ *is given by an assignment, to each element* x *of* $|X|$*, of an element of* $h(x) \approx_B h(x')$*.*

Proof. A morphism of equivalence relations from X to $\mathbf{Setoid}(A, B)$ consists of a function $f\colon |X| \longrightarrow |\mathbf{Setoid}(A, B)|$ such that if $x \equiv_X x'$, $f(x) \equiv_{\mathbf{Setoid}(A,B)} f(x')$, that is, such that for all $a \in |A|$, $f(x)(a) \approx_B f(x')(a)$ is inhabitant.

To give such data is equivalent to giving a function $f\colon X \longrightarrow [A, B]$ together with, for all $x \in |X|$, and for all $a, a' \in |A|$, a function $f_\approx : (a \approx_A a') \longrightarrow (f(x)(a) \approx_B f(x)(a'))$.

Reorganising this data, and adding the condition on f as a morphism of equivalence relations, this is equivalent to giving a function $g\colon A \longrightarrow [X, B]$ together with, for each $a, a' \in |A|$, and for each $x \in |X|$, a function $g_\approx\colon (a \approx_A a') \longrightarrow g(a)(x) \approx_B g(a')(x)$ such that, for all $a \in |A|$, if $x \equiv x'$, then $g(a)(x) \approx_B g(a)(x')$.

Putting $g(a) = h$, this agrees with the construction in the statement of the proposition. The construction in the statement can routinely be checked to possess the requisite reflexivity, symmetry and transitivity data. □

Cotensors bear close resemblance to the closed structure of Setoid. If one considers the equivalence relation X as a setoid, one in which each set in the family has either one element or none, the set B^X is exactly the set given by the closed structure of Setoid.

However, the associated families of sets $h \approx_{[X,B]} h'$ are subtly different. For cotensors, we can express an element of $h \approx_{[X,B]} h'$ as the assignment, to each element x of $|X|$, of a function from $x \approx x$ to $h(x) \approx_B h'(x)$. In contrast, for the closed structure, we required, for each pair (x, x') of elements of X, a function from $x \approx_X x'$ to $h(x) \approx_B h'(x')$.

Thus the closed structure and the cotensors, although closely related to each other, are not isomorphic. However, they are equivalent, i.e., there are morphisms in Setoid $r\colon B^X \longrightarrow [X, B]$ and $s\colon [X, B] \longrightarrow B^X$ for which the composite $r \circ s$ is equivalent to the identity morphism on $[X, B]$ and similarly for $s \circ r$: there are evident choices of such morphisms, the behaviour on carriers being the identity functions.

This means that, although the closed structure and cotensors of Setoid do not agree as Equiv- structures, they do agree as $\mathcal{E}$-structures, i.e., **Setoid**

is $\mathcal{E}$-closed and has $\mathcal{E}$-cotensors, and for any equivalence relation X seen as a setoid, the two $\mathcal{E}$-structures agree.

In stark contrast to this, it seems unlikely that **Setoid** as an Equiv-category has tensors, although it does have $\mathcal{E}$-tensors.

6. $\mathcal{E}$-categories

$\mathcal{E}$-categories naturally arise when categories are treated in constructive settings [20]. $\mathcal{E}$-categories are closely related to $\mathcal{P}$-categories, which also arise when categories are treated constructively. The definition of $\mathcal{P}$-category is obtained by replacing equivalence relations by partial equivalence relations in the definition of $\mathcal{E}$-category. $\mathcal{P}$-categories are studied in [7], but the authors did not always distinguish $\mathcal{P}$-categories from categories enriched over the Cartesian closed category of partial equivalence relations.

6.1. Basic definitions

Definition 15. An *$\mathcal{E}$-category* $\mathbf{C}$ consists of the following data.

- a set $\mathrm{ob}(\mathbf{C})$
- for each $x, y \in \mathrm{ob}(\mathbf{C})$ an object $\mathbf{C}(x, y)$ of Equiv.
- for each $x, y, z \in \mathrm{ob}(\mathbf{C})$, a morphism $\circ\colon \mathbf{C}(y, z) \times \mathbf{C}(x, y) \longrightarrow \mathbf{C}(x, z)$ in Equiv
- for each $x \in \mathrm{ob}(\mathbf{C})$, an element id_x of $|\mathbf{C}(x, x)|$

These data are subject to the following conditions.

- for each $w, x, y, z \in \mathrm{ob}(\mathbf{C})$, $f \in \mathbf{C}(y, z)$, $g \in \mathbf{C}(x, y)$ and $h \in \mathbf{C}(w, x)$, $(f \circ g) \circ h \equiv_{\mathbf{C}(w,z)} f \circ (g \circ h)$
- for each $x, y \in \mathrm{ob}(\mathbf{C})$ and $f \in \mathbf{C}(x, y)$, $\mathrm{id}_y \circ f \equiv_{\mathbf{C}(x,y)} f$ and $f \circ \mathrm{id}_x \equiv_{\mathbf{C}(x,y)} f$.

Example 16. Every Equiv-category is an $\mathcal{E}$-category. Such an $\mathcal{E}$-category is called *strict* in [7]. **Setoid** is a strict $\mathcal{E}$-category, for instance.

The reason for considering $\mathcal{E}$-categories in addition to Equiv-categories is that the axioms for them are written only in terms of equivalence relations with equality of morphisms not appearing at all. Avoiding equality has a practical advantage in proof assistants based on constructive type theory. We emphasise that equality of morphisms is used in the axioms of categories and Equiv-categories and that is an obstacle to deal with them in those proof assistants.

$\mathcal{E}$-categories are special kinds of *bicategories* [21] where the homcategories are equivalence relations, because an equivalence relation is a groupoid each of whose homsets has at most one element. Because of this degeneracy, we do not need some coherence axioms for $\mathcal{E}$-categories that are necessary for bicategories. There is an analysis of $\mathcal{E}$-categories based on this observation [8].

$\mathcal{E}$-functors are a special case of pseudo-functors between bicategories.

Definition 17. Let $\mathbf{A}$ and $\mathbf{B}$ be $\mathcal{E}$-categories. An *$\mathcal{E}$-functor* F from $\mathbf{A}$ to $\mathbf{B}$ consists of the following data.

- A function $F_0 \colon \mathrm{ob}(\mathbf{A}) \longrightarrow \mathrm{ob}(\mathbf{B})$.
- For each $x, y \in \mathrm{ob}(\mathbf{A})$, a function $F_1(x, y) \colon \mathbf{A}(x, y) \longrightarrow \mathbf{B}(F_0(x), F_0(y))$.

We often overload F_0 and F_1 and write F for them. These data are subject to the following conditions.

- For each $x, y, z \in \mathrm{ob}(\mathbf{A})$, $f_0 \in |\mathbf{A}(y, z)|$ and $f_1 \in |\mathbf{A}(x, y)|$, $F(f_0 \circ f_1) \equiv F(f_0) \circ F(f_1)$.
- For each $x \in \mathrm{ob}(\mathbf{A})$, $F(\mathrm{id}_x) \equiv \mathrm{id}_{F(x)}$.

Likewise, $\mathcal{E}$-natural transformations are defined as follows.

Definition 18. Let $\mathbf{A}$ and $\mathbf{B}$ be $\mathcal{E}$-categories and $F, G \colon \mathbf{A} \longrightarrow \mathbf{B}$ be $\mathcal{E}$-functors from $\mathbf{A}$ to $\mathbf{B}$. An *$\mathcal{E}$-natural transformation* is a function which maps $x \in \mathrm{ob}(\mathbf{A})$ to an element of $\mathbf{B}(F(x), G(x))$ subject to $G(f) \circ \alpha_x \equiv \alpha_y \circ F(f)$, for each $f \in \mathbf{A}(x, y)$.

The vertical and horizontal compositions of $\mathcal{E}$-natural transformations, as well as identity $\mathcal{E}$-natural transformations are defined as expected. Modification in $\mathcal{E}$-context becomes an equivalence between natural transformations.

Definition 19. Let $\mathbf{A}$ and $\mathbf{B}$ be $\mathcal{E}$-categories, $F, G \colon \mathbf{A} \longrightarrow \mathbf{B}$ be $\mathcal{E}$-functors. $\mathcal{E}$-natural transformations $\alpha, \beta \colon F \longrightarrow G \colon \mathbf{A} \longrightarrow \mathbf{B}$ are *isomorphic* if and only if $\alpha_x \equiv \beta_x$ for all $x \in \mathrm{ob}(\mathbf{A})$.

Definition 20. Let $\mathbf{A}$ and $\mathbf{B}$ be $\mathcal{E}$-categories. The $\mathcal{E}$-functor $\mathcal{E}$-category[2], written $[\mathbf{A}, \mathbf{B}]$, consists of the following data.

- The set $\mathrm{ob}([\mathbf{A}, \mathbf{B}])$ of objects is the set of $\mathcal{E}$-functors from $\mathbf{A}$ to $\mathbf{B}$.
- For $F, G \in \mathrm{ob}([\mathbf{A}, \mathbf{B}])$, $[\mathbf{A}, \mathbf{B}](F, G)$ consists of the set of $\mathcal{E}$-natural transformations from F to G and the equivalence of $\mathcal{E}$-natural transformations, as defined in Definition 19.
- The composition of morphisms is the vertical composition of $\mathcal{E}$-natural transformations.
- id is the identity $\mathcal{E}$-natural transformation.

If $\mathbf{B}$ is strict, then $[\mathbf{A}, \mathbf{B}]$ is strict.

Following the practice for bicategories, we say two $\mathcal{E}$-categories $\mathbf{A}$ and $\mathbf{B}$ are *$\mathcal{E}$-equivalent* if there are $\mathcal{E}$-functors $H\colon \mathbf{A} \longrightarrow \mathbf{B}$ and $K\colon \mathbf{B} \longrightarrow \mathbf{A}$ such that $K \circ H$ is equivalent to the identity $\mathcal{E}$-functor on $\mathbf{A}$ and $H \circ K$ is equivalent to the identity $\mathcal{E}$-functor on $\mathbf{B}$.

The following result is implicit in [8].

Theorem 21. *Every $\mathcal{E}$-category is $\mathcal{E}$-equivalent to an* Equiv*-category.*

Proof. There is a Yoneda embedding of any small $\mathcal{E}$-category $\mathbf{A}$ into the $\mathcal{E}$-functor $\mathcal{E}$-category $[\mathbf{A}, \mathbf{Equiv}]$. The latter is a strict $\mathcal{E}$-category as $\mathbf{Equiv}$ is a strict $\mathcal{E}$-category. The Yoneda embedding is an equivalence on homs, so $\mathbf{A}$ is $\mathcal{E}$-equivalent to a full sub-$\mathcal{E}$-category of $[\mathbf{A}, \mathbf{Equiv}]$, thus to a strict $\mathcal{E}$-category. □

6.2. Structures on $\mathcal{E}$-categories

An object of Equiv, i.e., a set X together with an equivalence relation $\equiv_X$ on it, may be seen as a category: X is the set of objects and $X(x_0, x_1)$ is 1 if $x_0 \equiv_X x_1$ and is otherwise 0. This construction extends to a functor $J\colon \mathsf{Equiv} \longrightarrow \mathsf{Cat}$, which is fully faithful and has a left adjoint, thus exhibiting Equiv as a full reflective subcategory of Cat, as discussed in Proposition 11.

The functor J induces an inclusion of Equiv-Cat into the category of Cat-categories, i.e., into the category of 2-categories. So every Equiv-category can be seen as a 2-category.

[2]This term is introduced as the "$\mathcal{E}$-version" of the notion of functor category. It does not mean any particular $\mathcal{E}$-functor that is called "$\mathcal{E}$-category".

Every $\mathcal{E}$-category can likewise be seen as a bicategory, as was central to [8]: the key idea of a bicategory as opposed to a 2-category is that composition of morphisms is transitive and has a unit only up to coherent isomorphism [7, 8]. The relationship between Equiv-categories and $\mathcal{E}$-categories is given by the relationship between 2-categories and bicategories.

There is an extensive literature about bicategorical limits, colimits and Cartesian closedness, including [22, 5]. So restriction from bicategories to $\mathcal{E}$-categories immediately yields a theory of limits, colimits and closedness for $\mathcal{E}$-categories.

We shall not spell out the bicategorical definitions as they involve coherence axioms, i.e., axioms that say which composites of two-dimensional isomorphisms are equal to each other, as those issues do not arise here. We simply remark that the limits, colimits, and closedness constructions we consider are all given by restriction of the well-established bicategorical constructs. The key constructs there are those of biproduct, biequaliser (equivalently bi-iso-inserter), and bicotensor and biclosedness. We refer to the restricted notions as $\mathcal{E}$-products, $\mathcal{E}$-inserters, $\mathcal{E}$-cotensors, their $\mathcal{E}$-duals, and $\mathcal{E}$-closedness.

We give the definition of a binary $\mathcal{E}$-product explicitly; an $\mathcal{E}$-product of any number $n \geq 0$ of objects is similar.

Definition 22. An *$\mathcal{E}$-product* of objects X and Y in an $\mathcal{E}$-category consists of an object $X \times Y$ and morphisms $\pi_X \colon X \times Y \longrightarrow X$ and $\pi_Y \colon X \times Y \longrightarrow Y$ such that for any object A and morphisms $f \colon A \longrightarrow X$ and $g \colon A \longrightarrow Y$, there is a morphism $h \colon A \longrightarrow X \times Y$ such that $\pi_X h \equiv f$ and $\pi_Y h \equiv g$. Moreover, if $f \equiv f'$ and $g \equiv g'$, then $h \equiv h'$ for any such h and h'.

Definition 23. An *$\mathcal{E}$-inserter* of morphisms $f, g \colon X \longrightarrow Y$ in an $\mathcal{E}$-category consists of an object $\mathrm{Iso}(f, g)$ and a morphism $i \colon Iso(f, g) \longrightarrow X$ such that $f \circ i \equiv g \circ i$, and such that for any object Z and morphism $z \colon Z \longrightarrow X$ for which $f \circ z \equiv g \circ z$, there is a morphism $\bar{z} \colon Z \longrightarrow Iso(f, g)$ for which $i\bar{z} \equiv z$; and if $z \equiv z'$, then $\bar{z} \equiv \bar{z'}$ for any such $\bar{z}$ and $\bar{z'}$.

The notion of $\mathcal{E}$-cotensor is quite subtle. For an $\mathcal{E}$-cotensor, one relaxes the isomorphism in the definition of Equiv-cotensor to being an equivalence in **Equiv**. This is precisely analogous to the difference between Equiv-products and $\mathcal{E}$-products or between Equiv-inserters and $\mathcal{E}$-inserters.

Definition 24. An *$\mathcal{E}$-cotensor* of an equivalence relation X with an object A of an $\mathcal{E}$-category $\mathbf{C}$ consists of an object A^X such that for every object

B, there is an equivalence $\mathsf{Equiv}(X, \mathbf{C}(B, A)) \simeq \mathbf{C}(B, A^X)$, natural in B, between the objects $\mathsf{Equiv}(X, \mathbf{C}(B, A))$ and $\mathbf{C}(B, A^X)$ of Equiv.

These definitions are weaker than those of Equiv-products, Equiv-inserters and Equiv-cotensors. The Equiv-category **Setoid** has Equiv-products, Equiv-inserters, and Equiv-cotensors; so, *a fortiori*, it has $\mathcal{E}$-products, $\mathcal{E}$-inserters and $\mathcal{E}$-cotensors. However, the defining property of the latter only determines them up to equivalence within an $\mathcal{E}$-category. So any setoid that is equivalent to an Equiv-product in **Setoid** is itself an $\mathcal{E}$-product, although not necessarily an Equiv-product; similarly for inserters and cotensors.

The dual notions of $\mathcal{E}$-product, $\mathcal{E}$-inserter and $\mathcal{E}$-cotensor are called $\mathcal{E}$-*coproduct*, $\mathcal{E}$-*coinserter* and $\mathcal{E}$-*tensor*, following the bicategorical tradition. **Setoid** has Equiv-coproducts, hence $\mathcal{E}$-coproducts, but it seems not to have Equiv-coinserters or Equiv-tensors in general, but it does have $\mathcal{E}$-coinserters and $\mathcal{E}$-tensors.

Definition 25. An $\mathcal{E}$-category $\mathbf{C}$ with finite $\mathcal{E}$-products is $\mathcal{E}$-*closed* if for every object X of $\mathbf{C}$, the $\mathcal{E}$-functor $(- \times X)\colon \mathbf{C} \longrightarrow \mathbf{C}$ has a right $\mathcal{E}$-adjoint.

Again, this is a weakening of the notion of Equiv-closedness. So, as **Setoid** is Equiv-closed, it is necessarily $\mathcal{E}$-closed. Just as for limits, $\mathcal{E}$-closed structure is only determined up to equivalence, so any setoid that is equivalent to an exponential $[A, B]$ is itself an $\mathcal{E}$-exponential, but might not be an Equiv-exponential.

7. The $\mathcal{E}$-category Setoid

Setoids and their morphisms form an $\mathcal{E}$-category **Setoid**, as already discussed in Example 16. In fact, **Setoid** is $\mathcal{E}$-equivalent to **Equiv**: the inclusion $J\colon \mathbf{Equiv} \longrightarrow \mathbf{Setoid}$ is an equivalence on homs, and every setoid A is equivalent in the $\mathcal{E}$-category **Setoid** to $J(F(A))$, i.e., there are morphisms $r\colon A \longrightarrow J(F(A))$ and $s\colon J(F(A)) \longrightarrow A$ such that $s \circ r \equiv \mathrm{id}_A$ and $r \circ s \equiv \mathrm{id}_{J(F(A))}$.

We have already seen that **Setoid** has Equiv-products, Equiv-inserters, Equiv-cotensors, and Equiv-coproducts and is closed as an Equiv-category. So, *a fortiori*, it has all that structure as an $\mathcal{E}$-category too.

The only structure that we have not been able to address in the simpler context of **Setoid** as an Equiv-category, probably because it does not exist, is that of Equiv-coinserters and Equiv-tensors. **Setoid** does have $\mathcal{E}$-coinserters and $\mathcal{E}$-tensors, as we shall now describe.

Theorem 26. *The $\mathcal{E}$-category* **Setoid** *has $\mathcal{E}$-coinserters.*

Proof. Given morphisms of setoids $f, g\colon A \longrightarrow B$, let $\mathrm{Coins}(f, g)$ have carrier $|B|$, with $b_0 \approx_{\mathrm{Coins}(f,g)} b_1$ determined by the transitive closure of the union of the sets given by $\approx_B$ with, for each $a \in |A|$, a singleton set for each of $f(a) \approx g(a)$ and $g(a) \approx f(a)$.

This induces a setoid structure, with reflexivity axiom given by that for B, transitivity by construction, and symmetry by a combination of construction and that for B. Moreover, the inclusion inc generated by the identity morphism id_B has $\mathrm{inc} \circ f \equiv \mathrm{inc} \circ g$. The universal property holds by construction. □

Theorem 27. *The $\mathcal{E}$-category* **Setoid** *has $\mathcal{E}$-tensors.*

Proof. Recall we defined J to be the inclusion of Equiv in Setoid (Proposition 11). Given an object X of Equiv and a setoid A, the product $J(X) \times A$ acts as an $\mathcal{E}$-tensor as

- for any setoid B, the object $\mathbf{Setoid}(J(X) \times A, B)$ of Equiv is isomorphic to $\mathbf{Setoid}(J(X), [A, B])$
- the setoid $[A, B]$ is equivalent (but not isomorphic) to $J(\mathbf{Setoid}(A, B))$, and
- for any object Y of Equiv, $\mathbf{Setoid}(J(X), J(Y))$ is equivalent to $\mathbf{Equiv}(X, Y)$.

□

8. Conclusion

The idea of setoid came from the need for explicit provision of an equivalence relation on each set in constructive mathematics. But then we have the set of proofs of equivalence, and the next question is how to treat equivalence between proofs of equivalence. There are two extreme ways to do this: one is to take the degenerate equivalence relation and the other is to take the discrete one. The former leads to the category Equiv, and the latter leads to our category Setoid.

There is a further issue regarding equivalence: how to treat proofs of equivalence between functions. Enrichment of categories Equiv and Setoid over the Cartesian closed category Equiv, as we discuss in Sections 4 and

5, leads to degeneration of proofs. One could consider enrichment over the Cartesian closed category Setoid as well, but its correspondence with practice in theorem proving is not yet clear.

Enrichment over Equiv apparently is related to the notion of "proof irrelevance" discussed in connection with proof assistants, but the exact correspondence is left for further study.

References

[1] E. Bishop, Foundations of Constructive Analysis, McGraw-Hill, 1967.

[2] Agda, 2012. URL: `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

[3] Coq, 2012. URL: `http://coq.inria.fr`.

[4] Isabelle, 2012. URL: `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`.

[5] G. Kelly, Elementary observations on 2-categorical limits, Bulletin of the Australian Mathematical Society 39 (1989) 301–317.

[6] P. Aczel, Galois: A theory development project, 1993. URL: `http://www.cs.man.ac.uk/ petera/papers.html`, report for the 1993 Turin meeting on the Representation of Mathematics in Logical Frameworks.

[7] D. Čubrić, P. Dybjer, P. J. Scott, Normalization and the Yoneda Embedding, Mathematical Structures in Computer Science 8 (1998) 153–192.

[8] Y. Kinoshita, A bicategorical analysis of E-categories, Mathematica Japonica 47 (1998) 157–169.

[9] K. O. Wilander, An E-bicategory of E-categories: exemplifying a type-theoretic approach to bicategories, Technical Report 2005:48, Uppsala University, Department of Mathematics, 2005.

[10] K. O. Wilander, Setoids and universes, Mathematical Structures in Computer Science 20 (2010) 563–576.

[11] K. O. Wilander, On Constructive Sets and Partial Structures, Ph.D. thesis, Uppsala University, Department of Mathematics, 2011.

[12] K. O. Wilander, Constructing a small category of setoids, Mathematical Structures in Computer Science 22 (2012) 103–121.

[13] A. Carboni, R. C. Magno, The free exact category on a left exact one, Jounal of Australian Mathematical Society 33(A) (1982) 295–301.

[14] A. Carboni, Some free constructions in realizability and proof theory, Journal of Pure and Applied Algebra 103 (1995) 117–148.

[15] A. Carboni, G. Rosolini, Locally cartesian closed exact completions, Journal of Pure and Applied Algebra 154 (2000) 103–116.

[16] M. E. Maietti, G. Rosolini, Quotient completion for the foundation of constructive mathematics, 2012. URL: `http://arxiv.org/abs/1212.1012`, cornell University arXiv.org.

[17] M. E. Maietti, G. Rosolini, Elementary quotient completion, 2012. URL: `http://arxiv.org/abs/1212.0162`, cornell University arXiv.org.

[18] M. E. Maietti, G. Rosolini, Unifying exact completions, 2012. URL: `http://arxiv.org/abs/1212.0966`, cornell University arXiv.org.

[19] G. M. Kelly, Basic concepts of enriched category theory, volume 10 of *Reprints in Theory and Aplications of Categories*, The Editors of Theory and Applications of Categories, 2005. Originally published as [23].

[20] P. Aczel, Galois: A theory development project, 1995. Report for the 1993 Turin meeting on the Representation of Mathematics in Logical Frameworks.

[21] J. Bénabou, Introduction to bicategories, in: Reports of the Midwest Category Seminar, volume 47 of *Lecture Notes in Mathematics*, Springer Berlin Heidelberg, 1967, pp. 1–77. URL: `http://dx.doi.org/10.1007/BFb0074299`. doi:10.1007/BFb0074299.

[22] R. Street, Fibrations in bicategories, Cahiers de topologie et géometrie différentielle catégoriques 21 (1980) 111–160.

[23] G. M. Kelly, Basic concepts of enriched category theory, volume 64 of *London Mathematical Society Lecture Note Series*, Cambridge University Press, 1982.

Appendix A. Agda code for setoids and constructions on them

The following Agda code describes setoids and related constructions in the $\mathcal{E}$-category **Setoid**. Most of the code should be self-explanatory except for the names of the constructions. Those operators suggesting products, coproducts and closures are really for $\mathcal{E}$-products, $\mathcal{E}$-coproducts and $\mathcal{E}$-closures. Moreover, `Eq` is for $\mathcal{E}$-inserters and `Coeq` is for $\mathcal{E}$-coinserters.

```
module ConstructionsInSetoid where
infixr 1 _⊎_
infixr 2 _∧_
infix 4 _!_≈_
infixr 9 _∘_
infixl 20 _′_ _″_
data ∅ : Set where
data _≡_ {A : Set} (a : A) : A → Set where
  refl≡ : a ≡ a
data _⊎_ (A B : Set) : Set where
  ι=₀ : A → A ⊎ B
  ι=₁ : B → A ⊎ B
record _∧_ (A : Set) (B : Set) : Set where
  constructor ∧-intro
  field
    ∧-elim₁ : A ; ∧-elim₂ : B
open _∧_ public
record Σ (A : Set) (B : A → Set) : Set where
  constructor _,_
  field
    π₀ : A  ;  π₁ : B π₀
open Σ public
syntax Σ A (λ x → B) = Σ[ x ∈ A ] B
_×_ : ∀ (A : Set) (B : Set) → Set
A × B = Σ[ x ∈ A ] B
record Setoid : Set₁ where
  infix 4 _≈_
  field
    carrier : Set
    _≈_ : carrier → carrier → Set
    refl : {x : carrier} → x ≈ x
    sym : {x y : carrier} → y ≈ x → x ≈ y
    trans : {x y z : carrier} → y ≈ z → x ≈ y → x ≈ z
|_| : Setoid → Set
| A | = Setoid.carrier A
_!_≈_ : (A : Setoid) → (a₀ a₁ : | A |) → Set
A ! a₀ ≈ a₁ = Setoid._≈_ A a₀ a₁
```

```
record _⇝_ (A B : Setoid) : Set where
  field
    fun : | A | → | B |
    resp : {a₀ a₁ : | A |} →
      A ! a₀ ≈ a₁ → B ! fun a₀ ≈ fun a₁
_′_ : {A B : Setoid} →
  A ⇝ B → | A | → | B |
f ′ a = _⇝_.fun f a
_″_ : ∀ {A B : Setoid} {a₀ a₁ : | A |}
  (f : A ⇝ B) (a₀≈a₁ : A ! a₀ ≈ a₁) → B ! f ′ a₀ ≈ f ′ a₁
f ″ a₀≈a₁ = _⇝_.resp f a₀≈a₁
[_⇝_] : Setoid → Setoid → Setoid
[ A ⇝ B ] = let open Setoid B in
  record
  { carrier = A ⇝ B
  ; _≈_ = λ f g → (a : | A |) → f ′ a ≈ g ′ a
  ; refl = λ a → refl
  ; sym = λ g≈f a → sym (g≈f a)
  ; trans = λ g≈h f≈g a → trans (g≈h a) (f≈g a)
  }
′-Lemma : ∀ {A B : Setoid} {f₀ f₁ : A ⇝ B} {a₀ a₁ : | A |} →
  [ A ⇝ B ] ! f₀ ≈ f₁ → A ! a₀ ≈ a₁ → B ! f₀ ′ a₀ ≈ f₁ ′ a₁
′-Lemma {A} {B} {f₀} {f₁} {a₀} {a₁} f₀≈f₁ a₀≈a₁
  = Setoid.trans B (f₀≈f₁ a₁) (f₀ ″ a₀≈a₁)
id : (A : Setoid) → A ⇝ A
id A = record { fun = λ a → a ; resp = λ a₀≈a₁ → a₀≈a₁ }
_∘_ : {A B C : Setoid} → B ⇝ C → A ⇝ B → A ⇝ C
g ∘ f = record
  { fun = λ a → g ′ (f ′ a)
  ; resp = λ a₀≈a₁ → g ″ (f ″ a₀≈a₁)
  }
left-id : ∀ {A B : Setoid} (f : A ⇝ B) →
  [ A ⇝ B ] ! id B ∘ f ≈ f
left-id {B = B} f a = Setoid.refl B
right-id : ∀ {A B : Setoid} (f : A ⇝ B) →
  [ A ⇝ B ] ! f ∘ id A ≈ f
right-id {B = B} f a = Setoid.refl B
assoc-∘ :  ∀ {A B C D : Setoid}
  (f : C ⇝ D) (g : B ⇝ C) (h : A ⇝ B) →
  [ A ⇝ D ] ! (f ∘ g) ∘ h ≈ (f ∘ (g ∘ h))
assoc-∘ {D = D} f g h = λ a → Setoid.refl D
module EQ (A : Setoid) where
  infixl 3 _≈_by_
  infixl 3 _≈_yb_
  infix 4 ∵_
```

```
  open Setoid A
  ∵_ : (x : carrier) → x ≈ x
  ∵_ x = refl
  _≈_by_ : ∀ {x y : carrier} →
    x ≈ y → (z : carrier) → y ≈ z → x ≈ z
  _≈_by_ x≈y _ y≈z = trans y≈z x≈y
  _≈_yb_ : ∀ {x y : carrier} →
    x ≈ y → (z : carrier) → z ≈ y → x ≈ z
  _≈_yb_ x≈y _ z≈y = trans (sym z≈y) x≈y
[_×_] : Setoid → Setoid → Setoid
[ A × B ] = record
  { carrier = | A | × | B |
  ; _≈_ = λ P Q →
    A ! π₀ P ≈ π₀ Q ∧ B ! π₁ P ≈ π₁ Q
  ; refl = ∧-intro (Setoid.refl A) (Setoid.refl B)
  ; sym = λ p →
    ∧-intro (Setoid.sym A (∧-elim₁ p))
      (Setoid.sym B (∧-elim₂ p))
  ; trans = λ p q →
    ∧-intro (Setoid.trans A (∧-elim₁ p) (∧-elim₁ q))
      (Setoid.trans B (∧-elim₂ p) (∧-elim₂ q))
  }
proj₀ : ∀ {A B : Setoid} → [ A × B ] ⇝ A
proj₀ = record { fun = π₀ ; resp = ∧-elim₁ }
proj₁ : ∀ {A B : Setoid} → [ A × B ] ⇝ B
proj₁ = record { fun = π₁ ; resp = ∧-elim₂ }
⟨_,_⟩ : ∀ {A B C : Setoid} (f : C ⇝ A) (g : C ⇝ B) →
  C ⇝ [ A × B ]
⟨_,_⟩ f g = record
 { fun = λ c → (f ′ c , g ′ c)
 ; resp = λ a₀≈a₁ →
   ∧-intro (f ″ a₀≈a₁) (g ″ a₀≈a₁)
 }
UnivProd∃ : ∀ {A B C : Setoid}
  (f : C ⇝ A) (g : C ⇝ B)  →
  [ C ⇝ A ] ! f ≈ proj₀ {A} {B} ∘ ⟨ f , g ⟩ ∧
  [ C ⇝ B ] ! g ≈ proj₁ {A} {B} ∘ ⟨ f , g ⟩
UnivProd∃ {A} {B} {C} f g =
  ∧-intro (λ c → Setoid.refl A) (λ c → Setoid.refl B)
UnivProd! : ∀ {A B C : Setoid}
  (f : C ⇝ A) (g : C ⇝ B) (h : C ⇝ [ A × B ]) →
  [ C ⇝ A ] ! f ≈ proj₀ {A} {B} ∘ h →
  [ C ⇝ B ] ! g ≈ proj₁ {A} {B} ∘ h →
  [ C ⇝ [ A × B ] ] ! ⟨ f , g ⟩ ≈ h
UnivProd! f g h f≈proj₀∘h g≈proj₁∘h c =
```

```
  ∧-intro (f≈proj₀∘h c) (g≈proj₁∘h c)
× : {I : Set} → (I → Setoid) → Setoid
× {I} B = record
 { carrier = (i : I) → ∣ B i ∣
 ; _≈_ = λ (b₀ b₁ : (i : I) → ∣ B i ∣) →
    (i : I) → B i ! b₀ i ≈ b₁ i
 ; refl = λ i → Setoid.refl (B i)
 ; sym = λ b₁≈b₀ i → Setoid.sym (B i) (b₁≈b₀ i)
 ; trans = λ b₁≈b₂ b₀≈b₁ i →
     Setoid.trans (B i) (b₁≈b₂ i) (b₀≈b₁ i)
 }
proj : ∀ {I : Set} {B : I → Setoid} (i : I) → × B ⇝ B i
proj i = record
 { fun = λ b → b i  ;  resp = λ b₀≈b₁ → b₀≈b₁ i }
tuple : ∀ {I : Set} {B : I → Setoid} {C : Setoid}
  (f : (i : I) → C ⇝ B i) → C ⇝ × B
tuple f = record
 { fun = λ c i → (f i) ′ c
 ; resp = λ a₀≈a₁ i → f i ″ a₀≈a₁
 }
Univ×∃ : ∀ {I : Set} {B : I → Setoid} {C : Setoid}
  (i : I) → (f : (j : I) → C ⇝ B j) →
  [ C ⇝ B i ] ! f i ≈ proj {_} {B} i ∘ tuple f
Univ×∃ {I} {B} {C} i _ _ = Setoid.refl (B i)
Univ×! : ∀ {I : Set} {B : I → Setoid} {C : Setoid}
  (f : (i : I) → C ⇝ B i) (h : C ⇝ × B) →
  ((i : I) → [ C ⇝ B i ] ! f i ≈ proj {_} {B} i ∘ h) →
  [ C ⇝ × B ] ! tuple f ≈ h
Univ×! f h fic≈hci c i = fic≈hci i c
Eq : {A B : Setoid} → (f g : A ⇝ B) → Setoid
Eq {A} {B} f g =
  record
  { carrier = Σ[ a ∈ ∣ A ∣ ]  B ! f ′ a ≈ g ′ a
  ; _≈_ = λ a₀ a₁ → A ! π₀ a₀ ≈ π₀ a₁
  ; refl = Setoid.refl A
  ; sym = Setoid.sym A
  ; trans = Setoid.trans A
  }
eq : {A B : Setoid} → (f g : A ⇝ B) → Eq f g ⇝ A
eq f g =
  record{ fun = π₀  ;  resp = λ a₀≈a₁ → a₀≈a₁ }
EqCone : {A B : Setoid} → (f g : A ⇝ B) →
  [ Eq f g ⇝ B ] ! f ∘ eq f g ≈ g ∘ eq f g
EqCone {A} {B} f g = π₁
```

```
eqMediate : {A B C : Setoid} (f g : A ⇝ B) (h : C ⇝ A)
  (hCone : [ C ⇝ B ] ! f ∘ h ≈ g ∘ h) →
  C ⇝ Eq f g
eqMediate = λ _ _ h hCone → record
 { fun = λ c → (h ′ c , hCone c)
 ; resp = λ a₀≈a₁ → h ″ a₀≈a₁
 }
EqUiv∃ : ∀ {A B C : Setoid} (f g : A ⇝ B) (h : C ⇝ A)
  (hCone : [ C ⇝ B ] ! f ∘ h ≈ g ∘ h) →
  [ C ⇝ A ] ! eq f g ∘ eqMediate f g h hCone ≈ h
EqUiv∃ {A = A} _ _ _ _ _ = Setoid.refl A
EqUiv! : ∀ {A B C : Setoid} (f g : A ⇝ B) (h : C ⇝ A)
  (hCone : [ C ⇝ B ] ! f ∘ h ≈ g ∘ h)
  (k : C ⇝ Eq f g)
  (eq[f][g]∘k≈h : [ C ⇝ A ] ! eq f g ∘ k ≈ h) →
  [ C ⇝ Eq f g ] ! k ≈ eqMediate f g h hCone
EqUiv! _ _ _ _ _ eq[f][g]∘k≈h c = eq[f][g]∘k≈h c
[_⊎_] : Setoid → Setoid → Setoid
[ A ⊎ B ] = record
  { carrier = | A | ⊎ | B |
  ; _≈_ = Equ
  ; refl = λ {x} → Refl x
  ; sym = λ {x} {y} y≈x → Sym x y y≈x
  ; trans = λ {x} {y} {z} y≈z x≈y → Trans x y z y≈z x≈y
  }
  where
  Equ : {A B : Setoid} → | A | ⊎ | B | → | A | ⊎ | B | → Set
  Equ {A} {_} (ι =₀ a₀) (ι =₀ a₁) = A ! a₀ ≈ a₁
  Equ {_} {B} (ι =₁ b₀) (ι =₁ b₁) = B ! b₀ ≈ b₁
  Equ _ _ = ∅
  Refl : (x : | A | ⊎ | B |) → Equ {A} {B} x x
  Refl (ι =₀ _) = Setoid.refl A
  Refl (ι =₁ _) = Setoid.refl B
  Sym : (x y : | A | ⊎ | B |) → Equ y x → Equ x y
  Sym (ι =₀ _) (ι =₀ _) a₁≈a₀ = Setoid.sym A a₁≈a₀
  Sym (ι =₁ _) (ι =₁ _) b₁≈b₀ = Setoid.sym B b₁≈b₀
  Sym (ι =₀ _) (ι =₁ _) ()
  Sym (ι =₁ _) (ι =₀ _) ()
  Trans : (x y z : | A | ⊎ | B |) →
    Equ y z → Equ x y → Equ x z
  Trans (ι =₀ a₀) (ι =₀ a₁) (ι =₀ a₂) a₁≈a₂ a₀≈a₁ =
    Setoid.trans A a₁≈a₂ a₀≈a₁
  Trans (ι =₁ b₀) (ι =₁ b₁) (ι =₁ b₂) b₁≈b₂ b₀≈b₁ =
    Setoid.trans B b₁≈b₂ b₀≈b₁
  Trans (ι =₀ _) (ι =₀ _) (ι =₁ _) () _
```

```
  Trans (ι =₀ _) (ι =₁ _) (ι =₀ _) () _
  Trans (ι =₀ _) (ι =₁ _) (ι =₁ _) _ ()
  Trans (ι =₁ _) (ι =₀ _) (ι =₀ _) _ ()
  Trans (ι =₁ _) (ι =₀ _) (ι =₁ _) _ ()
  Trans (ι =₁ _) (ι =₁ _) (ι =₀ _) () _
in₁ : ∀ {A B : Setoid} → A ⇝ [ A ⊎ B ]
in₁ = record
      { fun = ι =₀ ; resp = λ a₀≈a₁ → a₀≈a₁ }
in₂ : ∀ {A B : Setoid} → B ⇝ [ A ⊎ B ]
in₂ = record
      { fun = ι =₁ ; resp = λ a₀≈a₁ → a₀≈a₁ }
[_,_] : ∀ {A B C : Setoid} (f : A ⇝ C) (g : B ⇝ C) →
  [ A ⊎ B ] ⇝ C
[_,_] {A} {B} {C} f g = record
 { fun = Fun
 ; resp = λ {x₀} {x₁} x₀≈x₁ → Resp {x₀} {x₁} x₀≈x₁
 }
 where
 Fun : | A | ⊎ | B | → | C |
 Fun (ι =₀ a) = f ′ a
 Fun (ι =₁ b) = g ′ b
 Resp : ∀ {x₀ x₁ : | A | ⊎ | B |} →
   [ A ⊎ B ] ! x₀ ≈ x₁ → C ! Fun x₀ ≈ Fun x₁
 Resp {ι =₀ a₀} {ι =₀ a₁} x₀≈x₁ = f ″ x₀≈x₁
 Resp {ι =₀ a₀} {ι =₁ b₁} ()
 Resp {ι =₁ b₀} {ι =₀ a₁} ()
 Resp {ι =₁ b₀} {ι =₁ b₁} x₀≈x₁ = g ″ x₀≈x₁
UnivSum∃ :
  ∀ {A B C : Setoid} (f : A ⇝ C) (g : B ⇝ C)  →
  [ A ⇝ C ] ! f ≈ [ f , g ] ∘ in₁ ∧
  [ B ⇝ C ] ! g ≈ [ f , g ] ∘ in₂
UnivSum∃ {A} {B} {C} f g =
  ∧-intro (λ a → Setoid.refl C {f ′ a})
    (λ b → Setoid.refl C {g ′ b})
UnivSum! : ∀ {A B C : Setoid}
  {f : A ⇝ C} {g : B ⇝ C} {h : [ A ⊎ B ] ⇝ C} →
  [ A ⇝ C ] ! f ≈ h ∘ in₁ →
  [ B ⇝ C ] ! g ≈ h ∘ in₂ →
  [ [ A ⊎ B ] ⇝ C ] ! [ f , g ] ≈ h
UnivSum! f≈h∘in₁ _ (ι =₀ a) = f≈h∘in₁ a
UnivSum! _ g≈h∘in₂ (ι =₁ b) = g≈h∘in₂ b
subst : ∀ {I : Set} {F : I → Set} {i j : I} →
  i ≡ j → F i → F j
subst {I} {F} {i} .{i} refl≡ x = x
data ST* {X : Set} (R : X → X → Set) (x : X) :
```

```
  X → Set where
  ι = : {y : X} → R x y → ST* R x y
  symP : {y : X} → ST* R y x → ST* R x y
  transP : {y z : X} →
    ST* R y z → ST* R x y → ST* R x z
⨄ : {I : Set} → (I → Setoid) → Setoid
⨄ {I} A = record
  { carrier = Σ[ i ∈ I ] | A i |
  ; _≈_ = Equ I A
  ; refl = λ {a} → Refl I A a
  ; sym = λ {a₀} {a₁} c → Sym I A a₀ a₁ c
  ; trans = λ {a₀} {a₁} {a₂} c₀ c₁ →
      Trans I A a₀ a₁ a₂ c₀ c₁
  } where
  Equ : ∀ (I : Set) (A : I → Setoid)
    (a₀ a₁ : Σ[ i ∈ I ] | A i |) → Set
  Equ I A = ST* (equ I A) where
    equ : ∀ (I : Set) (A : I → Setoid)
      (a₀ a₁ : Σ[ i ∈ I ] | A i |) → Set
    equ I A a₀ a₁ =
      Σ[ c ∈ π₀ a₀ ≡ π₀ a₁ ]
      A (π₀ a₁) ! subst {F = λ i → | A i |} c (π₁ a₀) ≈ π₁ a₁
  Refl : ∀ (I : Set) (A : I → Setoid)
    (a : Σ[ i ∈ I ] | A i |) → Equ I A a a
  Refl I A a = ι = (refl≡ , Setoid.refl (A (π₀ a)))
  Sym : ∀ (I : Set) (A : I → Setoid)
    (a₀ a₁ : Σ[ i ∈ I ] | A i |) →
    (Equ I A a₁ a₀) → (Equ I A a₀ a₁)
  Sym I A a₀ a₁ c = symP c
  Trans : ∀ (I : Set) (A : I → Setoid)
    (a₀ a₁ a₂ : Σ[ i ∈ I ] | A i |) →
    (Equ I A a₁ a₂) → (Equ I A a₀ a₁) → (Equ I A a₀ a₂)
  Trans I A a₀ a₁ a₂ c₀ c₁ = transP c₀ c₁
inj : ∀ {I : Set} {A : I → Setoid} (i : I) → A i ⇝ ⨄ A
inj i = record
  { fun = λ a → (i , a)
  ; resp = λ a₀≈a₁ → ι = (refl≡ , a₀≈a₁)
  }
sum : ∀ {I : Set} {A : I → Setoid} {C : Setoid}
  (F : (i : I) → A i ⇝ C) → ⨄ A ⇝ C
sum {I} {A} {C} F  =
  record { fun = λ a → F (π₀ a) ′ π₁ a  ;  resp = Resp }
  where
  Resp :
    ∀ {a₀ a₁ : Σ[ i ∈ I ] | A i |} →
```

```
    ⨄ A ! a₀ ≈ a₁ →
    C ! F (π₀ a₀) ′ (π₁ a₀) ≈ F (π₀ a₁) ′ (π₁ a₁)
  Resp {_} {a₁} (ι = c) =
    Setoid.trans C
      (F (π₀ a₁) ″ π₁ c) (Lem1 (Lem0 (π₀ c)))
    where
    Lem0 : ∀ {i j : I} {a : | A i |}
      (c : i ≡ j) →
      F i ′ a ≡ F j ′ (subst {F = λ i → | A i |} c a)
    Lem0 {i} .{i} {_} refl≡ = refl≡
    Lem1 : ∀ {c₀ c₁ : | C |} →
      c₀ ≡ c₁ → C ! c₀ ≈ c₁
    Lem1 {c} .{c} refl≡ = Setoid.refl C
  Resp (symP c) = Setoid.sym C (Resp c)
  Resp (transP c₀ c₁) =
    Setoid.trans C (Resp c₀) (Resp c₁)
Univ⨄∃ : ∀ {I : Set} {A : I → Setoid} {C : Setoid}
  (F : (i : I) → A i ⇝ C) (i : I) →
  [ A i ⇝ C ] ! F i ≈ sum F ∘ inj {A = A} i
Univ⨄∃ {C = C} F i aᵢ = Setoid.refl C
Univ⨄! : ∀ {I : Set} {A : I → Setoid} {C : Setoid}
  {F : (i : I) → A i ⇝ C} {h : (⨄ A) ⇝ C} →
  ((i : I) → [ A i ⇝ C ] ! F i ≈ h ∘ inj {A = A} i) →
  [ ⨄ A ⇝ C ] ! sum F ≈ h
Univ⨄! h-prop a = h-prop (π₀ a) (π₁ a)
data [_!_]* (X : Setoid) (R : | X | → | X | → Set) :
  | X | → | X | → Set where
  oid* : {x₀ x₁ : | X |} → X ! x₀ ≈ x₁ → [ X ! R ]* x₀ x₁
  i* : {x₀ x₁ : | X |} → R x₀ x₁ → [ X ! R ]* x₀ x₁
  refl* : {x : | X |} → [ X ! R ]* x x
  sym* : {x₀ x₁ : | X |} → [ X ! R ]* x₀ x₁ → [ X ! R ]* x₁ x₀
  trans* : {x₀ x₁ x₂ : | X |} →
    [ X ! R ]* x₁ x₂ → [ X ! R ]* x₀ x₁ → [ X ! R ]* x₀ x₂
Coeq : {A B : Setoid} → (f g : A ⇝ B) → Setoid
Coeq {A} {B} f g =
 let
   _∼_ =
    [ B !
     (λ b b' → Σ[ a ∈ | A | ] B ! b ≈ f ′ a ∧ B ! b' ≈ g ′ a) ]*
 in
 record { carrier = | B |  ;  _≈_ = _∼_
        ; refl = refl*  ;  sym = sym*  ;  trans = trans*}
coeq : {A B : Setoid} → (f g : A ⇝ B) → B ⇝ Coeq f g
coeq {A} {B} f g = record { fun = λ b → b ; resp = oid* }
coeqCone : {A B : Setoid} → (f g : A ⇝ B) →
```

```
  [ A ⇝ Coeq f g ] ! coeq f g ∘ f ≈ coeq f g ∘ g
coeqCone {B = B} f g a =
  i* (a , ∧-intro (Setoid.refl B {f ′ a}) (Setoid.refl B {g ′ a}))
coeqMediate : {A B C : Setoid} (f g : A ⇝ B) (h : B ⇝ C)
  (hCone : [ A ⇝ C ] ! h ∘ f ≈ h ∘ g) → Coeq f g ⇝ C
coeqMediate {A} {B} {C} f g h hCone =
 record { fun = λ x → h ′ x ; resp = Resp } where
   _∼_ =
     [ B !
       (λ b b' →
         Σ[ a ∈ | A | ] B ! b ≈ f ′ a ∧ B ! b' ≈ g ′ a) ]*
   Resp : ∀ {b₀ b₁ : | B |} → b₀ ∼ b₁ → C ! h ′ b₀ ≈ h ′ b₁
   Resp (oid* b₀≈b₁) = h ″ b₀≈b₁
   Resp {b₀} {b₁} (i* Rb₀b₁) =
    let open Setoid in let open EQ C in
    ∵ h ′ b₀
    ≈ h ′ (f ′ π₀ Rb₀b₁) by h ″ ∧-elim₁ (π₁ Rb₀b₁)
    ≈ h ′ (g ′ π₀ Rb₀b₁) by hCone (π₀ Rb₀b₁)
    ≈ h ′ b₁ yb h ″ ∧-elim₂ (π₁ Rb₀b₁)
   Resp refl* = Setoid.refl C
   Resp (sym* b₁∼b₀) = Setoid.sym C (Resp b₁∼b₀)
   Resp (trans* b₁∼b₂ b₀∼b₁) =
     Setoid.trans C (Resp b₁∼b₂) (Resp b₀∼b₁)
CoeqUniv∃ : ∀ {A B C : Setoid} (f g : A ⇝ B) (h : B ⇝ C)
  (hCone : [ A ⇝ C ] ! h ∘ f ≈ h ∘ g) →
  [ B ⇝ C ] ! coeqMediate f g h hCone ∘ coeq f g ≈ h
CoeqUniv∃ {C = C} f g h hCone b = Setoid.refl C
CoeqUniv! : ∀ {A B C : Setoid}
  (f g : A ⇝ B) (h : B ⇝ C)
  (hCone : [ A ⇝ C ] ! h ∘ f ≈ h ∘ g) (k : Coeq f g ⇝ C)
  (eq[f][g]∘k≈h : [ B ⇝ C ] ! k ∘ coeq f g ≈ h) →
  [ Coeq f g ⇝ C ] ! k ≈ coeqMediate f g h hCone
CoeqUniv! f g h hCone k k∘coeq[f][g]≈h b = k∘coeq[f][g]≈h b
Curry : ∀ {B A C : Setoid} →
  [ A × B ] ⇝ C → A ⇝ ([ B ⇝ C ])
Curry {B} {A} {C} F = record
  { fun = λ a → record
    { fun = λ b → F ′ (a , b)
    ; resp = λ b₀≈b₁ →
        F ″ ∧-intro (Setoid.refl A) b₀≈b₁
    }
  ; resp = λ a₀≈a₁ b →
      F ″ ∧-intro a₀≈a₁ (Setoid.refl B)
  }
Uncurry : ∀ {B A C : Setoid} →
```

```
  A ⇝ ([ B ⇝ C ]) → [ A × B ] ⇝ C
Uncurry {B} {A} {C} F = record
  { fun = λ p → F ′ π₀ p ′ π₁ p
  ; resp = λ {p₀} {p₁} a₀≈a₁∧b₀≈b₁ →
    ′-Lemma {B} {C} {_⇝_.fun F (π₀ p₀)}
      {_⇝_.fun F (π₀ p₁)} {π₁ p₀} {π₁ p₁}
      (F ″ ∧-elim₁ a₀≈a₁∧b₀≈b₁) (∧-elim₂ a₀≈a₁∧b₀≈b₁)
  }
CCC∃ : ∀ {B A C : Setoid} (f : [ A × B ] ⇝ C) →
       let open Setoid ([ [ A × B ] ⇝ C ]) in
         f ≈ Uncurry (Curry {B = B} {A = A} f)
CCC∃ {B} {A} {C} f p =
  f ″ ∧-intro (Setoid.refl A) (Setoid.refl B)
CCC! : ∀ {B A C : Setoid} (g : A ⇝ [ B ⇝ C ]) →
  [ A ⇝ [ B ⇝ C ] ] ! g ≈ Curry (Uncurry g)
CCC! {B} {A} {C} g a b = Setoid.refl C
eval : ∀ {B} {C} → [ [ B ⇝ C ] × B ] ⇝ C
eval {B} {C} = record
  { fun = λ p → π₀ p ′ π₁ p
  ; resp = λ {p₀} {p₁} f₀≈f₁∧b₀≈b₁ →
    let open EQ C in
    ∵ π₀ p₀ ′ π₁ p₀
    ≈ π₀ p₁ ′ π₁ p₁
      by ′-Lemma {f₀ = π₀ p₀} {f₁ = π₀ p₁}
        (∧-elim₁ f₀≈f₁∧b₀≈b₁) (∧-elim₂ f₀≈f₁∧b₀≈b₁)
  }
_arr×id_ : ∀ {A} {D} →
  (A ⇝ D) → (B : Setoid) → [ A × B ] ⇝ [ D × B ]
f arr×id B = record
  { fun = λ x → f ′ π₀ x , π₁ x
  ; resp = λ {x₀} {x₁} x₀≈x₁ →
    ∧-intro (f ″ ∧-elim₁ x₀≈x₁) (∧-elim₂ x₀≈x₁)
  }
funct-id : ∀ {B A : Setoid} →
  [ [ A × B ] ⇝ [ A × B ] ] ! id A arr×id B ≈ id [ A × B ]
funct-id {B = B} {A = A} a =
  ∧-intro (Setoid.refl A) (Setoid.refl B)
funct-∘ : ∀ {A B C D : Setoid} (f : B ⇝ C) (g : A ⇝ B) →
  [ [ A × D ] ⇝ [ C × D ] ] !
  (f ∘ g) arr×id D ≈ (f arr×id D) ∘ (g arr×id D)
funct-∘ {C = C} {D = D} f g = λ a →
  ∧-intro (Setoid.refl C) (Setoid.refl D)
Exponent∃ : ∀ {A B C : Setoid} (f : [ A × B ] ⇝ C) →
  [ [ A × B ] ⇝ C ] ! f ≈ eval ∘ Curry {B} {A} f arr×id B
Exponent∃ {C = C} f a = Setoid.refl C
```

```
Exponent! : ∀ {A B C : Setoid}
  (f : [ A × B ] ⇝ C) (h : A ⇝ [ B ⇝ C ]) →
  [ [ A × B ] ⇝ C ] ! eval ∘ (h arr×id B) ≈ f →
  [ A ⇝ [ B ⇝ C ] ] ! h ≈ Curry f
Exponent! _ _ hCond a b = hCond (a , b)
```