# Chapter 4

# Categorical organisation of the ornament–refinement framework

## 4.1 Formalisation of categories

In this section we formalise some basic category-theoretic terms in Agda, establishing vocabulary for Sections 4.2 and 4.3.

### 4.1.1 Definitions of categories and functors

We will define a category to be a set of objects and sets of morphisms indexed by source and target, together with the usual laws. Special attention must be paid to equality on morphisms, though, which is usually coarser than definitional equality — for example, in the category of sets and (total) functions, it is necessary to identify functions up to extensional equality, so uniqueness of morphisms in universal properties would make sense. One simple way to achieve this in Agda's intensional setting is to use *setoids* Barthe et al. [2003] — i.e., sets with an explicitly specified equivalence relation — to represent sets of morphisms. Subsequently, all operations on morphisms should respect the equivalence.

In Agda, the type of setoids can be defined as a record which contains a carrier set, an equivalence relation on the set, and the three laws for the equivalence relation:[1]

```
record Setoid {c d : Level} : Set (suc (c ⊔ d)) where
  field
    Carrier : Set c
    _≈_     : Carrier → Carrier → Set d
    refl′   : ∀ {x} → x ≈ x
    sym     : ∀ {x y} → x ≈ y → y ≈ x
    trans   : ∀ {x y z} → x ≈ y → y ≈ z → x ≈ z
```

For example, we can define a setoid of functions that uses extensional equality:

```
FunSetoid : Set → Set → Setoid
FunSetoid A B = record { Carrier = A → B
                       ; _≈_     = _≐_
                       ; proofs − of − laws}
```

where $\_ \doteq \_$ is defined by $f \doteq g = \forall\, x \rightarrow f\, x \equiv g\, x$. Proofs of the three laws are omitted from the paper.

Similarly, we can define the type of categories as a record containing a set of objects, a collection of setoids of morphisms indexed by source and target (so morphisms with the same source and target — and only such morphisms — can be compared for equality), the composition operator on morphisms, the identity morphisms, and the laws of categories. The definition is shown in Figure 4.1. Two notations are introduced to improve readability: $X ==> Y$ is defined to be the carrier set of the setoid of morphisms from $X$ to $Y$, and $f \approx g$ is defined to be the equivalence between the morphisms $f$ and $g$ as specified by the setoid to which $f$ and $g$ belong. The last two laws *cong − l* and *cong − r* require composition of morphisms to respect the equivalence on

[1]The definition of setoids uses Agda's universe polymorphism, so the definition can be instantiated at suitable levels of the Set hierarchy as needed. We will give the first few universe-polymorphic definitions with full detail about the levels, but will later switch to writing just 'Set _' to suppress the noise.

```
record Category {l m n : Level} : Set (suc (l ⊔ m ⊔ n)) where
  field
    Object    : Set l
    Morphism  : Object → Object → Setoid {m} {n}
  _ ==> _ : Object → Object → Set m
  X ==> Y  =  Setoid.Carrier (Morphism X Y)
  _ ≈ _ : ∀ {X Y} → X ==> Y → X ==> Y → Set n
  _ ≈ _ {X} {Y}  =  Setoid._ ≈ _ (Morphism X Y)
  field
    _Δ_  : ∀ {X Y Z} → Y ==> Z → X ==> Y → X ==> Z
    id   : ∀ {X} → X ==> X
    id − l   : ∀ {X Y} (f : X ==> Y) → id Δ f ≈ f
    id − r   : ∀ {X Y} (f : X ==> Y) → f Δ id ≈ f
    assoc    : ∀ {X Y Z W}
               (f : Z ==> W) (g : Y ==> Z) (h : X ==> Y) →
               (f Δ g) Δ h ≈ f Δ (g Δ h)
    cong − l : ∀ {X Y Z} {f g : Y ==> Z} (h : X ==> Y) →
               f ≈ g → f Δ h ≈ g Δ h
    cong − r : ∀ {X Y Z} (h : Y ==> Z) {f g : X ==> Y} →
               f ≈ g → h Δ f ≈ h Δ g
```

**Figure 4.1** Definition of categories.

morphisms; they are given in this form to work better with the equational reasoning combinators commonly used in Agda (see, for example, the AoPA library Mu et al. [2009]). Now we can define the category *Fun* of sets and (total) functions as

$$
\begin{aligned}
&\mathit{Fun} : \mathit{Category}\\
&\mathit{Fun} = \textbf{record}\ \{\ \mathit{Object} = \mathsf{Set}\\
&\qquad\qquad\qquad ;\ \mathit{Morphism} = \mathit{FunSetoid}\\
&\qquad\qquad\qquad ;\ \_\Delta\_ = \_\circ\_\\
&\qquad\qquad\qquad ;\ \mathit{id} = \lambda\, x \mapsto x\\
&\qquad\qquad\qquad ;\ \mathit{proofs}-\mathit{of}-\mathit{laws}\}
\end{aligned}
$$

Another important category that we will make use of is *Fam*, the category of families of sets and families of functions, which is useful for talking about componentwise structures. An object in *Fam* has type $\Sigma\langle I : \mathsf{Set}\rangle\ I \rightarrow \mathsf{Set}$, i.e., it is a set $I$ and a family of sets indexed by $I$; a morphism from $(J\,,Y)$ to $(I\,,X)$ is a function $e : J \rightarrow I$ and a family of functions from $Y\ j$ to $X\ (e\ j)$ for each $j : J$.

$$
\begin{aligned}
&\mathit{Fam} : \mathit{Category}\\
&\mathit{Fam} = \textbf{record}\\
&\quad \{\ \mathit{Object} = \Sigma\langle I : \mathsf{Set}\rangle\ I \rightarrow \mathsf{Set}\\
&\quad ;\ \mathit{Morphism} =\\
&\qquad \lambda\,(J\,,Y)\ (I\,,X) \mapsto \textbf{record}\\
&\qquad\quad \{\ \mathit{Carrier} = \Sigma\langle e : J \rightarrow I\rangle\ Y \rightrightarrows (X \circ e)\\
&\qquad\quad ;\ \_\approx\_ = \lambda\,(e\,,u)\ (e'\,,u') \mapsto\\
&\qquad\qquad\qquad (e \doteq e') \times (\forall\ \{j\} \rightarrow u\ \{j\}\ \mathit{JMEq}'\ u'\ \{j\})\\
&\qquad\quad ;\ \mathit{proofs}-\mathit{of}-\mathit{laws}\}\\
&\quad ;\ \_\Delta\_ = \lambda\,(e\,,u)\ (f\,,v) \mapsto (e \circ f)\,, (\lambda\,\{k\} \mapsto u\ \{f\ k\} \circ v\ \{k\})\\
&\quad ;\ \mathit{id} = (\lambda\, x \mapsto x)\,, (\lambda\,\{i\}\ x \mapsto x)\\
&\quad ;\ \mathit{proofs}-\mathit{of}-\mathit{laws}\}
\end{aligned}
$$

Note that the equivalence on morphisms is defined to be componentwise extensional equality, which is formulated with the help of McBride's "John Major" heterogeneous equality *_JMEq_* McBride [1999] — the equivalence *_JMEq′_* is defined by $g\ \mathit{JMEq}'\ h = \forall\ x \rightarrow g\ x\ \mathit{JMEq}\ h\ x$. (Given $y : Y\ j$ for some $j : J$,

```
record Functor
  {l m n l′ m′ n′ : Level}
  (C : Category {l} {m} {n}) (D : Category {l′} {m′} {n′}) :
  Set (l ⊔ m ⊔ n ⊔ l′ ⊔ m′ ⊔ n′) where
  field
    object      : CObject → DObject
    morphism  : ∀ {X Y} → X = C => Y → object X = D => object Y
    ≈ −respecting :
      ∀ {X Y} {f g : X = C => Y} →
      f ≈ C g → morphism f ≈ D morphism g
    id − preserving :
      ∀ {X} → morphism (idC {X}) ≈ D idD {object X}
    comp − preserving :
      ∀ {X Y Z} (f : Y = C => Z) (g : X = C => Y) →
      morphism (f ΔC g) ≈ D (morphism f ΔD morphism g)
```

**Figure 4.2** Definition of functors.

the types of *u* {*j*} *y* and *u′* {*j*} *y* are not definitionally equal but only provably equal, so it is necessary to employ heterogeneous equality.)

We will also need functors, whose definition is shown in Figure 4.2: a functor consists of two mappings, one on objects and the other on morphisms, where the morphism part respects equivalence on morphisms and preserves identity and composition. For example, we have two forgetful functors from *Fam* to *Fun*, one summing components together

```
FamF : Functor Fam Fun
FamF = record { object     = λ (I , X) ↦ Σ I X
              ; morphism = λ (e , u ) ↦ e ∗∗ u
              ; proofs − of − laws}
```

and the other extracting the index part.

```
FamI : Functor Fam Fun
```

$$\begin{aligned} \mathit{FamI} \;=\; \textbf{record}\; \{\; & \mathit{object} &&= \lambda\,(I\,,X) \mapsto I \\ ;\; & \mathit{morphism} &&= \lambda\,(e\,,u\,) \mapsto e \\ ;\; & \mathit{proofs-of-laws}\} \end{aligned}$$

The functor laws should be proved for both functors alongside their object and morphism maps. In particular, we need to prove that the morphism part respects equivalence: for *FamF* this means we need to prove, for all $e\;:\;J \rightarrow I$, $u\;:\;Y \rightrightarrows (X \circ e), f\;:\;J \rightarrow I$, and $v\;:\;Y \rightrightarrows (X \circ f)$, that

$$(e\;\doteq\;f) \times (\forall\;\{j\} \rightarrow u\;\{j\}\;\mathit{JMEq}'\;v\;\{j\}) \rightarrow (e\;{*}{*}\;u\;\doteq\;f\;{*}{*}\;v)$$

and for *FamI* we need to prove

$$(e\;\doteq\;f) \times (\forall\;\{j\} \rightarrow u\;\{j\}\;\mathit{JMEq}'\;v\;\{j\}) \rightarrow (e\;\doteq\;f)$$

both of which can be easily discharged.

### 4.1.2 Definition of pullbacks

We will define a pullback to be a product in the suitable slice category, where a product is defined to be a terminal object in the suitable span category. Below we give definitions of all these terms in logical order. Let $C\;:\;\mathit{Category}$ in what follows.

- A *slice category* based on $C$ is parametrised by an object $B$; its objects are those morphisms in $C$ with target $B$ and its morphisms are mediating morphisms giving rise to commutative triangles — diagrammatically,

  objects $$\begin{tikzcd} T \arrow[d, "s"'] \\ B \end{tikzcd}$$ and morphisms $$\begin{tikzcd} T \arrow[rr, "m"] \arrow[dr, "s"'] & & T' \arrow[dl, "s'"] \\ & B & \end{tikzcd}$$

  The slice objects and morphisms are defined in Agda as two records; they are shown in the upper half of Figure 4.3 along with the definition of slice categories. Note that the equivalence on slice morphisms is defined as only the equivalence on the mediating morphisms, essentially achieving proof-irrelevance.

- *Span categories* are similar: parametrised by two objects $L$ and $R$, a span category has

$$\text{objects}\quad L \xleftarrow{l} M \xrightarrow{r} R \qquad \text{and} \quad \text{morphisms}\quad \begin{tikzcd} & M \arrow[dl, "l"'] \arrow[dr, "r"] \arrow[dd, "m"'] & \\ L & & R \\ & M' \arrow[ul, "l'"] \arrow[ur, "r'"'] & \end{tikzcd}$$

  The Agda definitions are shown in the lower half of Figure 4.3, and are similar to those for slice categories.

- An object $X$ in $C$ is *terminal* if it satisfies the universal property that for every object $Y$ there is a unique morphism from $Y$ to $X$:

  $\mathit{Terminal}\ C\ :\ \mathit{Object} \to \mathsf{Set}\ \_$
  $\mathit{Terminal}\ C\ X\ =$
  $\quad (Y\ :\ \mathit{Object}) \to \Sigma\langle f : Y ==> X\rangle\ \mathit{Unique}\ (\mathit{Morphism}\ Y\ X)\, f$

  where uniqueness is defined relative to a setoid:

  $\mathit{Unique}\ :\ (S\ :\ \mathit{Setoid}) \to \mathit{Carrier\_S} \to \mathsf{Set}\ \_$
  $\mathit{Unique}\ S\ x\ =\ (y\ :\ \mathit{Carrier\_S}) \to x \approx S\ y$

- A *product* of two objects $X$ and $Y$ in $C$ is then a $\mathit{Span}\ C\ X\ Y$ that is terminal in $\mathit{SpanCategory}\ C\ X\ Y$:

  $\mathit{Product}\ C\ X\ Y\ :\ \mathit{Span}\ C\ X\ Y \to \mathsf{Set}\ \_$
  $\mathit{Product}\ C\ X\ Y\ =\ \mathit{Terminal}\ (\mathit{SpanCategory}\ C\ X\ Y)$

- A *pullback* of two slices $f, g\ :\ \mathit{Slice}\ C\ X$ is a product of $f$ and $g$ in $\mathit{SliceCategory}\ C\ X$: Define the type of *squares* based on $f$ and $g$ as

  $\mathsf{Square}\ C\, f\ g\ :\ \mathsf{Set}\ \_$
  $\mathsf{Square}\ C\, f\ g\ =\ \mathit{Span}\ (\mathit{SliceCategory}\ C\ X)\, f\ g$

  or diagrammatically,

$$\begin{tikzcd} & W \arrow[dl, "l"'] \arrow[dr, "r"] \arrow[dd] & \\ Y \arrow[dr, "f"'] & & Z \arrow[dl, "g"] \\ & X & \end{tikzcd} \quad \text{which is the same as} \quad \begin{tikzcd} Y \arrow[d, "f"] & W \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] & Z \arrow[d, "g"'] \\ X \arrow[r, phantom, "="] & X \arrow[r, phantom, "="] & X \end{tikzcd}$$

In a square $q$, we will refer to the object $\mathit{Slice.T}\ (\mathit{Span.M}\ q)$, i.e., the node $W$ in the diagrams above, as the *vertex* of $q$. A pullback of $f$ and $g$ is then a square based on $f$ and $g$ that satisfies

```
Pullback C f g : Square C f g → Set _
Pullback C f g = Product (SliceCategory C X) f g
```

Equivalently, if we define the *square category* based on $f$ and $g$ as

```
SquareCategory C f g : Category
SquareCategory C f g =
  SpanCategory (SliceCategory C X) f g
```

then a pullback of $f$ and $g$ is a terminal object in the square category based on $f$ and $g$ — indeed, $\mathit{Product}\ (\mathit{SliceCategory}\ C\ X)\ f\ g$ is definitionally equal to $\mathit{Terminal}\ (\mathit{SquareCategory}\ C\ f\ g)$.

The most important category-theoretic fact that we will use in this paper is that the vertices of any two pullbacks of the same slices are isomorphic. Define the type of isomorphisms between two objects $X$ and $Y$ in $C$ as

```
record Iso C X Y : Set _ where
  field
    to    : X ==> Y
    from  : Y ==> X
    from − to − inverse : from Δ to ≈ id
    to − from − inverse : to Δ from ≈ id
```

(The isomorphism relation _ ≅ _ we used in **??** is formally defined as Iso *Fun*.) Then we can formulate the following lemma: If $p$, $q$ : Square $C\ f\ g$ are both pullbacks, then we have an isomorphism

$$\mathsf{Iso}\ C\ (\mathit{Slice.T}\ (\mathit{Span.M}\ p))\ (\mathit{Slice.T}\ (\mathit{Span.M}\ q))$$

## 4.2 Categorical organisation of the ornament–refinement framework

We proceed to organise the ornament–refinement framework under several concrete categories and functors, aiming to clarify the overall structure of the framework, and then derive useful pullback properties from parallel composition of ornaments.

### 4.2.1 The category of type families and refinement families

We will see that the category *FRef* of type families and refinement families has a very close relationship to the category *Fam*. An object in *FRef* is an indexed family of sets as in *Fam*, and a morphism from $(J\ ,\ Y)$ to $(I\ ,\ X)$ consists of a function $e\ :\ J \rightarrow I$ on the indices and a refinement family of type FRefinement $e\ X\ Y$. As for the equivalence on morphisms, it suffices to use extensional equality on the index functions and componentwise equivalence on refinement families, where the equivalence on refinements is defined to be extensional equality on their forgetful functions (extracted by Refinement.*forget*). In Agda:

```
FRef : Category
FRef = record
  { Object    = Σ⟨ I : Set ⟩ I → Set
  ; Morphism  =
      λ (J , Y) (I , X) ↦ record
        { Carrier = Σ⟨ e : J → I ⟩ FRefinement e X Y
        ; _≈_     =
            λ (e , rs) (e′ , rs′) ↦
              (e ≐ e′) ×
              (∀ j → Refinement.forget (rs (ok j)) JMEq′
                         Refinement.forget (rs′ (ok j)))
        ; proofs − of − laws}
```

```
    ; proofs − of − laws}
```

Two facts support our choice of refinement equivalence: (i) under this definition, if two refinements are equivalent, then their promotion predicates are pointwise isomorphic, i.e., we have

```
forget − iso :
   {X Y : Set} (r s : Refinement X Y) →
   (Refinement.forget r ≐ Refinement.forget s) →
   ∀ x → Refinement.P r x ≅ Refinement.P s x
```

and (ii) we get a forgetful functor *FRefF* : *Functor FRef Fam* which is identity on objects and componentwise Refinement.*forget* on morphisms, the latter respecting equivalence automatically.

```
FRefF : Functor FRef Fam
FRefF = record
   { object     = λ (I , X) ↦ I , X
   ; morphism =
        λ (e , rs) ↦ e , (λ {j} ↦ Refinement.forget (rs (ok j)))
   ; proofs − of − laws}
```

Note that a refinement family from $X : I \to \mathsf{Set}$ to $Y : J \to \mathsf{Set}$ is deliberately cast as a morphism in the opposite direction from $(J\ ,\ Y)$ to $(I\ ,\ X)$, so *FRefF* remains a familiar covariant functor rather than a contravariant one. Think of this as suggesting the direction of the forgetful functions of refinements.

The above discussion suggests that the essential ingredient of a refinement is just its forgetful function. Indeed, from any function we can construct a *canonical refinement*:

```
canonRef : {X Y : Set} → (Y → X) → Refinement X Y
canonRef {X} {Y} f = record
   { P = λ x ↦ Σ⟨y : Y⟩ f y ≡ x
   ; i = record { to    = (split (f) (split ((λ y ↦ y)) (λ y ↦ refl)))
                ; from = outl ∘ outr
                ; proofs − of − laws}}
```

(The operator $\mathit{split}-\mathit{op}$ is defined by $(\mathit{split}\ (g)\ (h)) \ =\ \lambda\, x \ \mapsto\ (g\ x\ ,\ h\ x)$.) The canonical promotion predicate is very simplistic: to promote some $x\ :\ X$ to type $Y$, we are required to supply a complete $y\ :\ Y$ such that $x$ can be recovered from $y$ (rather than only the necessary information that augments $x$ to an element of $Y$). Any refinement $r\ :\ \mathsf{Refinement}\ X\ Y$ is equivalent to $\mathit{canonRef}\ (\mathsf{Refinement}.\mathit{forget}\ r)$, so by $\mathit{forget}-\mathit{iso}$ we have

$$\mathsf{Refinement}.P\ r\ x\ \cong\ \Sigma\langle\, y : Y\,\rangle\ \mathsf{Refinement}.\mathit{forget}\ r\ y\ \equiv\ x$$

for all $x\ :\ X$. That is, a promotion predicate is always pointwise isomorphic to the canonical promotion predicate. Thus all the refinement mechanism provides is a convenient way of expressing intensional (representational) optimisations of the canonical promotion predicate — extensionally, *FRef* is no more powerful than *Fam*. This is reflected in the existence of a functor *FRefC* : *Functor Fam FRef*, whose object part is identity and whose morphism part is componentwise *canonRef*:

```
FRefC : Functor Fam FRef
FRefC = record
  { object     = λ (I , X) ↦ I , X
  ; morphism   = λ (e , u ) ↦ e , (λ (ok j) ↦ canonRef (u {j}))
  ; proofs − of − laws}
```

*FRefC* is strictly inverse to *FRefF*, forming an isomorphism (not merely an equivalence) of categories between *FRef* and *Fam*.

### 4.2.2 The category of descriptions and ornaments

The category ORN has objects of type $\Sigma\langle\, I : \mathsf{Set}\,\rangle\ \mathsf{Desc}\ I$, i.e., descriptions paired with index sets, and morphisms from $(J\ ,\ E)$ to $(I\ ,\ D)$ of type $\Sigma\langle\, e : J \to I\,\rangle\ \mathsf{Orn}\ e\ D\ E$, i.e., ornaments paired with index erasure functions. We also need to devise an equivalence on ornaments

```
OrnEq :
  {I J : Set} {e e′ : J → I} {D : Desc I} {E : Desc J} →
  Orn e D E → Orn e′ D E → Set
```

such that it implies extensional equality of $e$ and $e'$ and that of ornamental forgetful functions:

$$\begin{array}{l}
\mathit{OrnEq}-\mathit{forget} \;:\\
\quad \{I\,J \;:\; \mathsf{Set}\}\;\{e\,e' \;:\; J \rightarrow I\}\;\{D \;:\; \mathsf{Desc}\,I\}\;\{E \;:\; \mathsf{Desc}\,J\}\\
\quad (O \;:\; \mathsf{Orn}\;e\;D\;E)\;(P \;:\; \mathsf{Orn}\;e'\;D\;E) \rightarrow \mathit{OrnEq}\;O\;P \rightarrow\\
\quad (e \;\doteq\; e') \times (\forall\,\{j\} \rightarrow \mathit{forget}\;O\;\{j\}\;\mathit{JMEq}'\,\mathit{forget}\;P\;\{j\})
\end{array}$$

We omit the detail of *OrnEq* from the paper (which depends on the detail of the universe of ornaments). Morphism composition is sequential composition, and there is a family of *identity ornaments*

$$\mathit{idOrn} \;:\; \{I \;:\; \mathsf{Set}\}\;\{D \;:\; \mathsf{Desc}\,I\} \rightarrow \mathsf{Orn}\;(\lambda\,i \mapsto i)\;D\;D$$

such that *idOrn* $\{I\}$ $\{D\}$ simply expresses that $D$ is identical to itself. Unsurprisingly, the identity ornaments serve as identity of sequential composition. To summarise:

$$\begin{array}{l}
\mathbb{O}\textsc{rn} \;:\; \mathit{Category}\\
\mathbb{O}\textsc{rn} \;=\; \textbf{record}\\
\quad \{\;\mathit{Object} \qquad = \;\Sigma\langle\, I : \mathsf{Set}\,\rangle\;\; \mathsf{Desc}\,I\\
\quad ;\;\mathit{Morphism} \;=\\
\qquad \lambda\,(J\,,E)\;(I\,,D) \;\mapsto\; \textbf{record}\\
\qquad\quad \{\;\mathit{Carrier} \;=\; \Sigma\langle\, e : J \rightarrow I\,\rangle\;\; \mathsf{Orn}\;e\;D\;E\\
\qquad\quad ;\;\_\approx\_ \quad = \;\lambda\,(e\,,O)\;(e'\,,O') \;\mapsto\; \mathit{OrnEq}\;O\;O'\\
\qquad\quad ;\;\mathit{proofs}-\mathit{of}-\mathit{laws}\}\\
\quad ;\;\_\Delta\_ \;=\; \lambda\,(e\,,O)\;(f\,,P) \;\mapsto\; (e \circ f)\,,(O \odot P)\\
\quad ;\;\mathit{id} \quad\; = \;(\lambda\,i \;\mapsto\; i)\,,\mathit{idOrn}\\
\quad ;\;\mathit{proofs}-\mathit{of}-\mathit{laws}\}
\end{array}$$

A functor *Ind* : *Functor* $\mathbb{O}$RN *Fam* can then be constructed, which gives the ordinary semantics of descriptions and ornaments: the object part of *Ind* decodes a description $(I\;,\;D)$ to its least fixed point $(I\;,\;\mu\;D)$, and the morphism part translates an ornament $(e\;,\;O)$ to the forgetful function $(e\;,\mathit{forget}\;O)$, the latter respecting equivalence by virtue of $\mathit{OrnEq}-\mathit{forget}$.

$$\begin{array}{l}
\mathit{Ind} \;:\; \mathit{Functor}\;\mathbb{O}\textsc{rn}\;\mathit{Fam}\\
\mathit{Ind} \;=\; \textbf{record}\;\{\;\mathit{object} \qquad = \;\lambda\,(I\,,D) \;\mapsto\; I\,,\mu\,D
\end{array}$$

$$; \mathit{morphism} = \lambda\,(e\,,O) \mapsto e\,,\mathit{forget}\ O$$
$$; \mathit{proofs}-\mathit{of}-\mathit{laws}\}$$

### 4.2.3 Pullback properties for parallel composition

We are now ready to state the pullback properties for parallel composition of ornaments. With suitable choices of encoding for the universes, we could attempt to establish that, for any two ornaments $O : \mathsf{Orn}\ e\ D\ E$ and $P : \mathsf{Orn}\ f\ D\ F$ where $D : \mathsf{Desc}\ I$, $E : \mathsf{Desc}\ J$, and $F : \mathsf{Desc}\ K$, the following square in $\mathbb{O}\text{RN}$ is a pullback:

$$\begin{tikzcd}
e \bowtie f\,,\lfloor O \otimes P \rfloor \arrow[r, "\mathsf{outr}\,,\,\mathit{diffOrn\text{-}r}\ O\ P"] \arrow[d, "\mathsf{outl}\,,\,\mathit{diffOrn\text{-}l}\ O\ P"'] \arrow[dr, "\mathit{pull},\lceil O \otimes P \rceil" description] & K\,,F \arrow[d, "f\,,P"] \\
J\,,E \arrow[r, "e\,,O"'] & I\,,D
\end{tikzcd}$$

This square is encoded in Agda as

$$\begin{aligned}
&\mathit{pc}-\mathit{square} : \mathsf{Square}\ \mathbb{O}\text{RN}\ (\mathit{slice}\ (J\,,E)\ (e\,,O))\ (\mathit{slice}\ (K\,,F)\ (f\,,P))\\
&\mathit{pc}-\mathit{square} = \mathit{span}\ (\mathit{slice}\ (e \bowtie f\,,\lfloor O \otimes P \rfloor)\ (\mathit{pull},\lceil O \otimes P \rceil))\\
&\qquad\qquad\qquad (\mathit{sliceMorphism}\ (\mathsf{outl}\ ,\mathit{diffOrn\text{-}l}\ O\ P)\ (\mathit{hole}\ ()\ (1)))\\
&\qquad\qquad\qquad (\mathit{sliceMorphism}\ (\mathsf{outr}\ ,\mathit{diffOrn\text{-}r}\ O\ P)\ (\mathit{hole}\ ()\ (2)))
\end{aligned}$$

where goal 1 has type $\mathit{OrnEq}\ (O \odot \mathit{diffOrn\text{-}l}\ O\ P)\ \lceil O \otimes P \rceil$ and goal 2 has type $\mathit{OrnEq}\ (P \odot \mathit{diffOrn\text{-}r}\ O\ P)\ \lceil O \otimes P \rceil$, both of which can be discharged.[2] The pullback property of $\mathit{pc}-\mathit{square}$, i.e., $\mathit{Pullback}\ \mathbb{O}\text{RN}\ \_\ \_\ \mathit{pc}-\mathit{square}$, is not too useful by itself, though: $\mathbb{O}\text{RN}$ is quite a restricted category, so a universal property established in $\mathbb{O}\text{RN}$ has limited applicability. Instead, we are more interested in the pullback property of the image of the above square under *Ind* in *Fam*, which is stated in the follow theorem. If $O : \mathsf{Orn}\ e\ D\ E$ and

[2]Since the structure of Agda terms like *pc − square* can be reconstructed from commutative diagrams and the categorical definitions, in the rest of the paper we will present only the commutative diagrams and omit the underlying Agda terms.

$P$ : Orn $f$ $D$ $F$ where $D$ : Desc $I$, $E$ : Desc $J$, and $F$ : Desc $K$, then the following square in *Fam* is a pullback.

$$\begin{tikzcd}
e \bowtie f \,,\, \mu \lfloor O \otimes P \rfloor \arrow[r, "\mathsf{outr} \,,\, \mathit{forget}\ (\mathit{diffOrn\text{-}r}\ O\ P)"] \arrow[d, "\mathsf{outl} \,,\, \mathit{forget}\ (\mathit{diffOrn\text{-}l}\ O\ P)"'] \arrow[dr, "\mathit{pull}, \mathit{forget}\ \lceil O \otimes P \rceil" description] \arrow[dr, phantom, "\lrcorner", very near start] & K \,,\, \mu F \arrow[d, "f \,,\, \mathit{forget}\ P"] \\
J \,,\, \mu E \arrow[r, "e \,,\, \mathit{forget}\ O"'] & I \,,\, \mu D
\end{tikzcd}$$

The proof of the universal property boils down to, very roughly speaking, construction of an inverse to

$$(\mathit{split}\ (\mathit{forget}\ (\mathit{diffOrn\text{-}l}\ O\ P))\ (\mathit{forget}\ (\mathit{diffOrn\text{-}r}\ O\ P)))$$

which involves tricky manipulation of equality proofs but is achievable. After the pullback property is established in *Fam*, since *FamF* is pullback-preserving, we also get a pullback square in *Fun*. If $O$ : Orn $e$ $D$ $E$ and $P$ : Orn $f$ $D$ $F$ where $D$ : Desc $I$, $E$ : Desc $J$, and $F$ : Desc $K$, then the following square in *Fun* is a pullback.

$$\begin{tikzcd}
\Sigma\ (e \bowtie f)\ (\mu \lfloor O \otimes P \rfloor) \arrow[r, "\mathsf{outr} \ast\ast \mathit{forget}\ (\mathit{diffOrn\text{-}r}\ O\ P)"] \arrow[d, "\mathsf{outl} \ast\ast \mathit{forget}\ (\mathit{diffOrn\text{-}l}\ O\ P)"'] \arrow[dr, "\mathit{pull} \ast\ast \mathit{forget}\ \lceil O \otimes P \rceil" description] \arrow[dr, phantom, "\lrcorner", very near start] & \Sigma\ K\ (\mu F) \arrow[d, "f \ast\ast \mathit{forget}\ P"] \\
\Sigma\ J\ (\mu E) \arrow[r, "e \ast\ast \mathit{forget}\ O"'] & \Sigma\ I\ (\mu D)
\end{tikzcd}$$

To translate ORN to *FRef*, i.e., datatype declarations to refinements, a naive way is to use the composite functor

$$\mathbb{O}\textsc{rn} \xrightarrow{\mathit{Ind}} \mathit{Fam} \xrightarrow{\mathit{FRefC}} \mathit{FRef}$$

The resulting refinements would then use canonical promotion predicates. However, the whole point of incorporating ORN in the framework is that we can construct an alternative functor *RSem* directly from ORN to *FRef*. The functor *RSem* is extensionally equal to the above composite functor, but intensionally very different. Its object part still takes the least fixed point of a description, but its morphism part is the refinement semantics of ornaments given in **??**, whose promotion predicates have a more efficient representation.

```
RSem : Functor ORN FRef
RSem = record
  { object    = λ (I , D) ↦ I , μ D
  ; morphism  =
      λ (e , O) ↦ e , (λ (ok j) ↦ record { P  =  OptP O (ok j)
                                          ; i  =  (hole () (3))})
  ; proofs − of − laws}
```

We will give goal 3, i.e., the ornamental promotion isomorphisms, a new construction in the next section.

## 4.3 Reconstruction of the ornamental promotion and modularity isomorphisms

The morphism part of the functor *RSem* : *Functor* ORN *FRef* translates ornaments into refinements that use the optimised predicates, which are defined via parallel composition, so the pullback properties for parallel composition hold for the optimised predicates. The natural step to take, then, is to construct the ornamental promotion isomorphisms using the pullback properties — this we do in the proof of **??** below. Even more closely related are the modularity isomorphisms, which are about parallel composition and optimised predicates. They, too, can be constructed from the pullback properties for parallel composition, which is done in the proof of **??**.

We restate the ornamental promotion isomorphisms as the following theorem. For any ornament $O$ : $\mathsf{Orn}\ e\ D\ E$ where $D$ : $\mathsf{Desc}\ I$ and $E$ : $\mathsf{Desc}\ J$, we have

$$\mu\ E\ j\ \cong\ \Sigma\langle\, x : \mu\ D\ (e\ j)\,\rangle\ \mathsf{OptP}\ O\ (\mathsf{ok}\ j)\ x$$

for all $j$ : $J$. Since the optimised predicates $\mathsf{OptP}\ O$ are defined by parallel composition of $O$ and the singleton ornament $S = \mathit{singOrn}\ D$, the conclusion of the theorem expand to

$$\mu\ E\ j \ \cong\ \Sigma\langle\, x : \mu\ D\ (e\ j)\,\rangle\ \ \mu\ \lfloor O \otimes \lceil S \rceil \rfloor\ (\mathsf{ok}\ j\ ,\ \mathsf{ok}\ (e\ j\ ,\ x)) \tag{4.1}$$

How do we derive these isomorphisms from the pullback properties for parallel composition? It turns out that the pullback property in *Fun* (**??**) can help.

First, observe that we have the following pullback square:

$$\begin{tikzcd}
\Sigma\ J\ (\mu\ E) \arrow[r, "(split\ (e\ **\ forget\ O)\ (singleton\ \circ\ forget\ O\ \circ\ \mathsf{outr}))"] \arrow[d, "id"'] \arrow[dr, phantom, very near start, "\lrcorner"] \arrow[dr, "e\ **\ forget\ O"] & \Sigma\ (\Sigma\ I\ (\mu\ D))\ (\mu\ \lfloor S \rfloor) \arrow[d, "\mathsf{outl}\ **\ forget\ \lceil S \rceil"] \\
\Sigma\ J\ (\mu\ E) \arrow[r, "e\ **\ forget\ O"'] & \Sigma\ I\ (\mu\ D)
\end{tikzcd} \tag{4.2}$$

If we view pullbacks as products of slices, since a singleton ornament does not add information to a datatype, the vertical slice on the right-hand side

$$s \ =\ slice\ (\Sigma\ (\Sigma\ I\ (\mu\ D))\ (\mu\ \lfloor S \rfloor))\ (\mathsf{outl}\ **\ forget\ \lceil S \rceil)$$

behaves like a "multiplicative unit": any (compatible) slice $s'$ alone gives rise to a product of $s$ and $s'$. As a consequence, we have the bottom-left type $\Sigma\ J\ (\mu\ E)$ as the vertex of the pullback. This pullback square is based on the same slices as the one in **??** with $P$ substituted by $\lceil S \rceil$, so by **??** we obtain an isomorphism

$$\Sigma\ J\ (\mu\ E)\ \cong\ \Sigma\ (e \bowtie \mathsf{outl})\ (\mu\ \lfloor O \otimes \lceil S \rceil \rfloor) \tag{4.3}$$

To get from (4.3) to (4.1), we need to look more closely into the construction of (4.3). The right-to-left direction of (4.3) is obtained by applying the universal property of (4.2) to the square in **??** (with $P$ substituted by $\lceil S \rceil$), so it is the unique mediating morphism $m$ that makes the following diagram commute:

$$\begin{tikzcd}
& \Sigma\ (e \bowtie \mathsf{outl})\ (\mu\ \lfloor O \otimes \lceil S \rceil \rfloor) \arrow[dl, "\mathsf{outl}\ **\ forget\ (diffOrn\text{-}l\ O\ P)"'] \arrow[dr, "\mathsf{outr}\ **\ forget\ (diffOrn\text{-}r\ O\ P)"] \arrow[dd, dashed, "m"'] & \\
\Sigma\ J\ (\mu\ E) & & \Sigma\ (\Sigma\ I\ (\mu\ D))\ (\mu\ \lfloor S \rfloor) \\
& \Sigma\ J\ (\mu\ E) \arrow[ul, "id"] \arrow[ur, "\langle e\ **\ forget\ O,\ singleton\ \circ\ forget\ O\ \circ\ \mathsf{outr}\rangle"'] &
\end{tikzcd}$$

From the left commuting triangle, we see that, extensionally, the morphism $m$ is just $\mathsf{outl}\ **\ forget\ (diffOrn\text{-}l\ O\ P)$. This leads us to the following general lemma: If there is an isomorphism

$$\Sigma\, K\, X \;\cong\; \Sigma\, L\, Y$$

whose right-to-left direction is extensionally equal to some $f \;**\; g$, then we have

$$X\,k \;\cong\; \Sigma\langle\, l : f^{\,\text{-1}}\, k\,\rangle\;\; Y\,(\mathit{und}\; l)$$

for all $k \;:\; K$. For a fixed $k \;:\; K$, an element of the form $(k\;,\,x) \;:\; \Sigma\; K\; X$ must correspond, under the isomorphism, to some element $(l\;,\,y) \;:\; \Sigma\; L\; Y$ such that $f\; l \;\equiv\; k$, so the set $X\; k$ corresponds to exactly the sum of the sets $Y\; l$ such that $f\; l \;\equiv\; k$. Specialising **??** for (4.3), we get

$$\mu\; E\; j \;\cong\; \Sigma\langle\, jix : \mathsf{outl}^{\,\text{-1}}\; j\,\rangle\;\; \mu\; \lfloor O \otimes \lceil S \rceil \rfloor\; (\mathit{und}\; jix) \tag{4.4}$$

for all $j \;:\; J$. Finally, observe that a canonical element of type $\mathsf{outl}^{\,\text{-1}}\; j$ must be of the form $\mathsf{ok}\;(\mathsf{ok}\; j\;,\,\mathsf{ok}\;(e\; j\;,\,x))$ for some $x \;:\; \mu\; D\;(e\; j)$, so we perform a change of variables for the summation, turning the right-hand side of (4.4) into

$$\Sigma\langle\, x : \mu\; D\;(e\; j)\,\rangle\;\; \mu\; \lfloor O \otimes \lceil S \rceil \rfloor\; (\mathsf{ok}\; j\;,\,\mathsf{ok}\;(e\; j\;,\,x))$$

and arriving at (4.1).

There is a twist, however, due to Agda's intensionality: It is possible to formalise the above lemma and the change of variables individually and chain them together, but the resulting isomorphisms would have a very complicated definition due to suspended type casts. If we use them to construct the refinement family in the morphism part of *RSem*, it would be rather difficult to prove that the morphism part of *RSem* respects equivalence. We are thus forced to fuse all the above reasoning into one step to get a clean definition when we actually carry out this construction in Agda, but the idea is still essentially the same.

The other important family of isomorphisms we should consider is the modularity isomorphisms. Suppose that there are descriptions $D \;:\; \mathsf{Desc}\; I$, $E \;:\; \mathsf{Desc}\; J$ and $F \;:\; \mathsf{Desc}\; K$, and ornaments $O \;:\; \mathsf{Orn}\; e\; D\; E$, and $P \;:\; \mathsf{Orn}\; f\; D\; F$. Then we have

$$\mathsf{OptP}\; \lceil O \otimes P \rceil\; (\mathsf{ok}\;(j\;,\,k))\; x \;\cong\; \mathsf{OptP}\; O\; j\; x \times \mathsf{OptP}\; P\; k\; x$$

for all $i : I$, $j : e^{-1}\ i$, $k : f^{-1}\ i$, and $x : \mu\ D\ i$. The conclusion of the theorem expands to

$$\begin{aligned} &\mu\ \lfloor \lceil O \otimes P \rceil \otimes \lceil S \rceil \rfloor\ (\mathsf{ok}\ (j\ , k)\ , \mathsf{ok}\ (i\ , x)) \\ &\quad \cong\ \mu\ \lfloor O \otimes \lceil S \rceil \rfloor\ (j\ , \mathsf{ok}\ (i\ , x)) \times \mu\ \lfloor P \otimes \lceil S \rceil \rfloor\ (k\ , \mathsf{ok}\ (i\ , x)) \end{aligned} \tag{4.5}$$

where again $S = \mathit{singOrn}\ D$. A quick observation is that they are componentwise isomorphisms between the two families of sets

$$M\ =\ \mu\ \lfloor \lceil O \otimes P \rceil \otimes \lceil S \rceil \rfloor$$

and

$$\begin{aligned} N\ =\ &\lambda\ (\mathsf{ok}\ (j\ , k)\ , \mathsf{ok}\ (i\ , x)) \mapsto \\ &\quad \mu\ \lfloor O \otimes \lceil S \rceil \rfloor\ (j\ , \mathsf{ok}\ (i\ , x)) \times \mu\ \lfloor P \otimes \lceil S \rceil \rfloor\ (k\ , \mathsf{ok}\ (i\ , x)) \end{aligned}$$

both indexed by $\mathit{pull} \bowtie \mathsf{outl}$ where $\mathit{pull}$ has type $e \bowtie f \to I$ and $\mathsf{outl}$ has type $\Sigma\ I\ X \to I$. This is just an isomorphism in *Fam* between $(\mathit{pull} \bowtie \mathit{proj}_1, M)$ and $(\mathit{pull} \bowtie \mathit{proj}_1, N)$ whose index part (i.e., the isomorphism obtained under the functor *FamI*) is identity. Thus we seek to prove that both $(\mathit{pull} \bowtie \mathit{proj}_1, M)$ and $(\mathit{pull} \bowtie \mathit{proj}_1, N)$ are vertices of pullbacks based on the same slices.

Let us look at $(\mathit{pull} \bowtie \mathit{proj}_1, N)$ first. For fixed $i$, $j$, $k$, and $x$, the set $N\ (\mathsf{ok}\ (j\ , k)\ , \mathsf{ok}\ (i\ , x))$ along with the cartesian projections is a product, which trivially extends to a pullback since there is a forgetful function from each of the two component sets to the *singleton* set $\mu\ \lfloor S \rfloor\ (i\ , x)$, as shown in the following diagram:

$$\begin{tikzcd} N\ (\mathsf{ok}\ (j\ , k)\ , \mathsf{ok}\ (i\ , x)) \arrow[r, "\mathsf{outr}"] \arrow[d, "\mathsf{outl}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mu\ \lfloor P \otimes \lceil S \rceil \rfloor\ (k\ , \mathsf{ok}\ (i\ , x)) \arrow[d, "\mathit{forget}\ (\mathit{diffOrn\text{-}r}\ P\ \lceil S \rceil)"] \\ \mu\ \lfloor O \otimes \lceil S \rceil \rfloor\ (j\ , \mathsf{ok}\ (i\ , x)) & \mu\ \lfloor S \rfloor\ (i\ , x) \arrow[l, "\mathit{forget}\ (\mathit{diffOrn\text{-}r}\ O\ \lceil S \rceil)"] \end{tikzcd}$$

Note that this pullback square is possible because of the common $x$ in the indices of the two component sets — otherwise they cannot project to the same singleton set. Collecting all such pullback squares together, we get the following pullback square in *Fam*:

$$\begin{tikzcd}
\mathit{pull} \bowtie \mathsf{outl} \mathbin{,} N \arrow[r, "\mathsf{outr}"] \arrow[d, "\_ \mathbin{,} \mathsf{outl}"'] \arrow[dr, phantom, "\lrcorner", very near start] & f \bowtie \mathsf{outl} \mathbin{,} \mu \lfloor P \otimes \lceil S \rceil \rfloor \arrow[d, "\mathsf{outr} \mathbin{,} \mathit{forget}\ (\mathit{diffOrn\text{-}r}\ P\ \lceil S \rceil)"] \\
e \bowtie \mathsf{outl} \mathbin{,} \mu \lfloor O \otimes \lceil S \rceil \rfloor \arrow[r, "\mathsf{outr} \mathbin{,} \mathit{forget}\ (\mathit{diffOrn\text{-}r}\ O\ \lceil S \rceil)"'] & \Sigma\ I\ (\mu\ D) \mathbin{,} \mu \lfloor S \rfloor
\end{tikzcd} \tag{4.6}$$

Next we prove that $(\mathit{pull} \bowtie \mathsf{outl} \mathbin{,} M)$ is also the vertex of a pullback based on the same slices as (4.6). This second pullback arises as a consequence of the following lemma. In any category, consider the objects $X$, $Y$, their product $X \Leftarrow X \boxtimes Y \Rightarrow Y$, and products of each of the three objects $X$, $Y$, and $X \boxtimes Y$ with an object $Z$. (All the projections are shown as solid arrows in the diagram below). Then $(X \boxtimes Y) \boxtimes Z$ is the vertex of a pullback of the two projections $X \boxtimes Z \Rightarrow Z$

$$\begin{tikzcd}
X & & X \boxtimes Y \arrow[ll] \arrow[rr] & & Y \\
& & (X \boxtimes Y) \boxtimes Z \arrow[u] \arrow[dl, dashed] \arrow[dr, dashed] \arrow[dd] \arrow[dd, phantom, "\curlyvee", very near start] & & \\
& X \boxtimes Z \arrow[uul] \arrow[dr] & & Y \boxtimes Z \arrow[uur] \arrow[dl] & \\
& & Z & &
\end{tikzcd}$$

and $Y \boxtimes Z \Rightarrow Z$. We again intend to view a pullback as a product of slices, and instantiate **??** in *SliceCategory Fam* $(I \mathbin{,} \mu\ D)$, substituting all the objects by slices consisting of relevant ornamental forgetful functions in (4.5). The substitutions are as follows:

$$\begin{array}{rcl}
X & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ O) \\
Y & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ P) \\
X \boxtimes Y & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ \lceil O \otimes P \rceil) \\
Z & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ \lceil S \rceil) \\
X \boxtimes Z & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ \lceil O \otimes \lceil S \rceil \rceil) \\
Y \boxtimes Z & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ \lceil P \otimes \lceil S \rceil \rceil) \\
(X \boxtimes Y) \boxtimes Z & \mapsto & \mathit{slice}\ \_\ (\_ \mathbin{,} \mathit{forget}\ \lceil \lceil O \otimes P \rceil \otimes \lceil S \rceil \rceil)
\end{array}$$

where $X \boxtimes Y$, $X \boxtimes Z$, $Y \boxtimes Z$, and $(X \boxtimes Y) \boxtimes Z$ indeed give rise to products in *SliceCategory Fam* $(I \mathbin{,} \mu\ D)$, i.e., pullbacks in *Fam*, by instantiating **??**. What we get out of this instantiation of the lemma is a pullback in *SliceCategory Fam* $(I \mathbin{,} \mu\ D)$ rather than *Fam*. This is easy to fix, since there is a forgetful functor from any

*SliceCategory C B* to *C* whose object part is *Slice.T*, and it is pullback-preserving. We thus get a pullback in *Fam* which is based on the same slices as (4.6) and has vertex $(pull \bowtie \mathsf{outl}\ , M)$.

Having the two pullbacks, by **??** we get an isomorphism in *Fam* between $(pull \bowtie \mathsf{outl}\ , M)$ and $(pull \bowtie \mathsf{outl}\ , N)$, whose index part can be shown to be identity, so there are componentwise isomorphisms between $M$ and $N$ in *Fun*, arriving at (4.5).

```
record Slice C B : Set _ where
  constructor slice
  field
    T  : Object
    s  : T ==> B
record SliceMorphism C B (s t : Slice C B) : Set _ where
  constructor sliceMorphism
  field
    m : Slice.T s ==> Slice.T t
    triangle : Slice.s t Δ m ≈ Slice.s s
SliceCategory C B : Category
SliceCategory C B =
  record
    { Object      = Slice C B
    ; Morphism    =
        λ s t ↦ record
                  { Carrier = SliceMorphism C B s t
                  ; _≈_ = λ f g ↦ SliceMorphism.m f ≈
                                  SliceMorphism.m g
                  ; proofs − of − laws}
    ; proofs − of − laws}
record Span C L R : Set _ where
  constructor span
  field
    M : Object
    l  : M ==> L
    r  : M ==> R
record SpanMorphism C L R (s t : Span C L R) : Set _ where
  constructor spanMorphism
  field
    m : Span.M s ==> Span.M t
    triangle − l : Span.l t Δ m ≈ Span.l s
    triangle − r : Span.r t Δ m ≈ Span.r s
SpanCategory C L R : Category
SpanCategory C L R =
  record
    { Object      = Span C L R
    ; Morphism    =
        λ s t ↦ record
                  { Carrier = SpanMorphism C L R s t
```

$$\begin{tikzcd}
\textsc{Orn} \arrow[r, bend right, "RSem"'] \arrow[rr, bend left, "Ind"] & FRef \arrow[r, bend left, "FRefF"] & Fam \arrow[l, bend left, "FRefC"] \arrow[r, bend left, "FamF"] \arrow[r, bend right, "FamI"'] & {}
\end{tikzcd}$$

| | | | |
|---|---|---|---|
| *Object* | $\Sigma\langle I : \mathsf{Set}\rangle\ \mathsf{Desc}\ I$ | $\Sigma\langle I : \mathsf{Set}\rangle\ I \to \mathsf{Set}$ | $\Sigma\langle I : \mathsf{Set}\rangle\ I \to \mathsf{Set}$ |
| $\_ \texttt{==>} \_$ | $\lambda\,(J\,,E)\ (I\,,D) \mapsto \Sigma\langle e : J \to I\rangle\ \mathsf{Orn}\ e\ D\ E$ | $\lambda\,(J\,,Y)\ (I\,,X) \mapsto \Sigma\langle e : J \to I\rangle\ \mathsf{FRefinement}\ e\ X\ Y$ | $\lambda\,(J\,,Y)\ (I\,,X) \mapsto \Sigma\langle e : J \to I\rangle\ Y \rightrightarrows$ |

**Figure 4.4** Categories (whose sets of objects and morphisms are listed below) and functors for the ornament–refinement framework.

# Bibliography

Gilles BARTHE, Venanzio CAPRETTA, and Olivier PONS [2003]. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293. doi:10.1017/S0956796802004501. ↰ page 1

Conor McBRIDE [1999]. *Dependently Typed Functional Programs and their Proofs*. Ph.D. thesis, University of Edinburgh. ↰ page 4

Shin-Cheng MU, Hsiang-Shang KO, and Patrik JANSSON [2009]. Algebra of Programming in Agda: Dependent types for relational program derivation. *Journal of Functional Programming*, 19(5):545–579. doi:10.1017/S0956796809007345. ↰ page 4

# Todo list