

The IFF Type Namespace

Robert E. Kent

December 18, 2007

Contents

1	The Core Component	2
1.1	Introduction	2
1.2	Type Sets	9
1.3	Type Functions	21
1.3.1	Function Morphisms	35
1.4	Type Spans	40
1.4.1	Type Endospans	55
1.4.2	Span Morphisms	57
1.5	Type Predicates	65
1.6	Type Relations	73
1.6.1	Type Endorelations	85
1.7	Type Diagrams	88
1.7.1	Category Theory	88
1.7.2	Introduction	88
1.7.3	Terminology	90
1.7.4	Type Set Pairs	91
1.7.5	Type Set Triples	95
1.7.6	Type Parallel Pairs	99
1.7.7	Type Opspans	105
1.8	Type Limits	112
1.8.1	Introduction	112
1.8.2	Type Binary Products	115
1.8.3	Type Binary Powers	123
1.8.4	Type Ternary Products	127
1.8.5	Type Ternary Powers	134
1.8.6	Type Equalizers	138
1.8.7	Type Pullbacks	146

1.8 Type Limits

Namespace Prefix

Technical: `type.lim`

Recommended: `type.lim`

In this section we axiomatize finite limits. We do not axiomatize general finite limits for two reasons: (1) an axiomatization is not possible, since the tiny IFF-IFF namespace axiomatization of a graph of abstract sets and functions does not have the terms required to express the concepts of general finite limits¹⁴; and (2) since the terminology for general finite limits are not used in the lower levels, in selecting which finite limit terminology to specify, we utilize the principle of *conceptual warrant*.¹⁵ Hence, in this section, we axiomatize the particular finite limits needed in the levels below the `type` level in the metashell: the terminal set, binary products and powers, ternary products and powers, equalizers and pullbacks.¹⁶

1.8.1 Introduction

The nested namespace for finite limits essentially defines four adjunctions (Figure 15) between the constant functor and “the” limit functor¹⁷: binary products on pairs of sets and their functions, ternary products on triples of sets and their functions, equalizers on parallel pairs (of functions) and their morphisms, and pullbacks on opspans and their morphisms. The terminology of this namespace,

¹⁴This seems to contradict the categorical intuition to find the “correct generality” mentioned by Mac Lane [2], which is a sub-principle of *categorical design*. But such intuition should probably be confined to the natural part of the IFF. The metashell is largely governed by set-theoretic concerns. In more detail, the following two implicit criteria have been consistently followed in the IFF design: use the simple syntax specified by the IFF grammar; unroll and bootstrap the IFF, starting from the very topmost IFF-IFF namespace in the metashell, continuing through the IFF-TYPE namespace and next the IFF-META namespace in the metashell, and then on through the generic ontologies in the natural part.

¹⁵Conceptual warrant is an extension of the librarianship notion of literary warrant, which means evidence for or token of authorization. The terms appearing in any meta-ontology exert authority, and hence should require some warrant for their existence.

¹⁶It is also important to establish the connections between the particular limits specified, where each can be defined in terms of some of the others: (1) any set pair is an opspan with terminal opvertex, hence binary product cones are special pullback cones and binary products and their projections are special pullbacks and their projections; (2) any opspan has an underlying set pair and an associated parallel pair, any pullback cone has an underlying binary product cone and an associated parallel pair cone, and the pullback is the equalizer of the parallel pair; likewise, the pullback projection is the composition of the canon of the parallel pair with the binary product projection of the set pair.

¹⁷The definite article is in quotes, since the limit functor is unique only up to isomorphism. Theoretically, this is well understood and immediately dismissed. But practically, this issue is important, since it affects the axiomatization. The use of a strict category-theoretic axiomatization, such as the adjunctive-style axiomatization for limits and exponents that is favored in the IFF specification, does not give a concrete limit, and hence can be regarded as an under-specified axiomatization. Additional axiomatization is needed for a full concrete specification. In the IFF, this additional axiomatization is added, not at the point where the limit is initially axiomatized, such as in this namespace, but at a lower metalevel, where the limit is used.

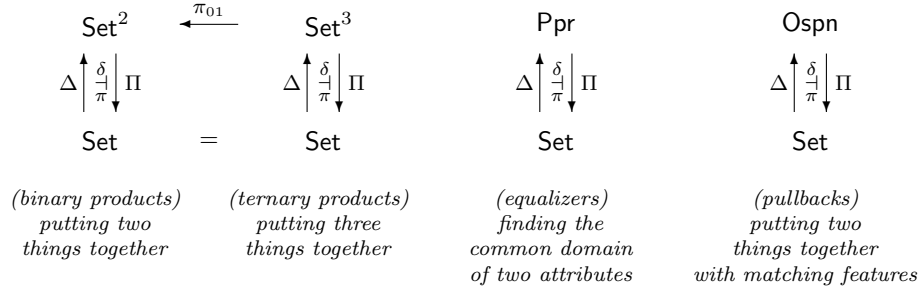


Figure 15: Constant-Limit Adjunctions

which is listed in Table 12, consists of 131 terms and 131 concepts (with no synonyms). There are six types of limit according to diagram shape: binary and ternary products and their power variants, equalizers and pullbacks.

There are two basic types of things, cone and mediator, with six variations according to type of limit (diagram shape): collections of cones for binary products and powers, ternary products and powers, equalizers and pullbacks; and collections of mediators for binary products, ternary products, equalizers and pullbacks. All ten are distinct — these sets are pairwise disjoint. The components of cones and mediators are also introduced here. All cone types are pairs (set, morphism), since they are (isomorphic to) objects in a comma category. For all cone types there is a vertex set component, and for all cone types there is a morphism component: a set pair morphism component (function pair) for binary product cones, a set triple morphism component (function triple) for ternary product cones, a parallel pair morphism component for equalizer cones, and an opspan component for pullback cones. Defining the isomorphism and making the cone a true object of a comma category — a triple (set, morphism, diagram) — is an underlying diagram component, which is the target of the morphism component: a set pair component for binary product cones, a set triple component for ternary product cones, a set component for binary and ternary power cones, a parallel pair component for equalizer cones, and an opspan component for pullback cones. For all cone types there are various function components derived from the morphism component. Similar components are introduced for mediators. Each type has a special limiting cone (limit, projection), consisting of an abstract limit set and a projection morphism. The limit set is unique only up to isomorphism: an abstract binary and ternary product set for both set pairs and set triples, an abstract binary and ternary power set for sets, an abstract equalizer set for parallel pairs, and an abstract pullback set for opsans. For all cone types there is an abstract mediating function.

	IFF Set	IFF Function
prd2.obj	cone	set function-pair function0 function1 cone-set-pair opposite
	mediator	mediator-set function set-pair
		delta-cone delta delta-function
		projection-mediator projection product projection-function-pair projection0 projection1 tau-cone
		packing pairing unpacking components tau
prd2.mor		product
pwr2.obj	cone	set function-pair function0 function1 cone-set
		power projection-function-pair projection0 projection1 pairing
pwr2.mor		power
.....		
prd3.obj	cone	set function-triple function0 function1 function2 cone-set-triple
	mediator	mediator-set function set-triple
		delta-cone delta delta-function
		projection-mediator projection product projection-function-triple projection0 projection1 projection2
		packing tripling unpacking components
prd3.mor		product
pwr3.obj	cone	set function-triple function0 function1 function2 cone-set
		power projection-function-triple projection0 projection1 projection2 tripling
		power
equ.obj	cone	set parallel-pair-morphism function0 cone-parallel-pair
	mediator	mediator-set function parallel-pair
		delta-cone delta delta-function
		projection-mediator projection equalizer projection-parallel-pair-morphism projection-function-pair projection0
		packing parting unpacking
equ.mor		equalizer
.....		
pbk.obj	cone	set opspan-morphism function0 function1 cone-opspan opposite
	mediator	mediator-set function opspan
		delta-cone delta delta-function
		projection-mediator projection pullback projection-opspan-morphism projection-function-pair projection0 projection1 tau-cone injection-cone injection
		packing pairing unpacking components tau
pbk.mor		pullback

Technical and Recommended Prefix : type.lim

Table 12: The Finite Limit Type Namespace

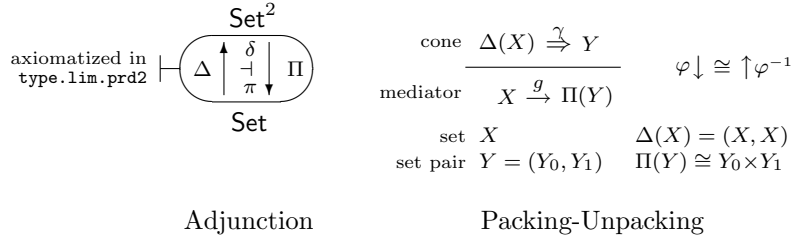


Figure 16: Binary Product

1.8.2 Type Binary Products

`type.lim.prd2`

The essential properties of the binary product construction (see Figure 14..) are illustrated in Figure 16. The horizontal bar designates a natural bijection between the cone above the bar and the mediator below the bar. Let X be a set with constant set pair $\Delta(X) = (X, X)$, and let $Y = (Y_0, Y_1)$ be a set pair with binary product set $\Pi(Y)$ ¹⁸. A natural packing (pairing) process (φ) assigns a mediator $X \rightarrow \Pi(Y)$ below the bar to every cone $\Delta(X) \Rightarrow Y$ above the bar. A natural unpacking (components) process (φ^{-1}) assigns a cone $\Delta(X) \Rightarrow Y$ above the bar to every mediator $X \rightarrow \Pi(Y)$ below the bar. These two processes are inverse to each other. The packing process is realized by application of the product (Π) operation and then composition on the left with the delta function (a component of the unit δ)

$$g = X \xrightarrow{\delta_X} \Pi(\Delta(X)) \xrightarrow{\Pi(\gamma)} \Pi(Y).$$

The unpacking process is realized by application of the constant (Δ) operation and then composition on the right with the projection function pair (a component of the counit π)

$$\gamma = \Delta(X) \xrightarrow{\Delta(g)} \Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y.$$

Objects

`type.lim.prd2.obj`

¹⁸The canonical (Cartesian) product has product set $Y_0 \times Y_1 = \{(b_0, b_1) \mid b_0 \in Y_0, b_1 \in Y_1\}$ and product projection functions $\pi_0 : Y_0 \times Y_1 \rightarrow Y_0 : (b_0, b_1) \mapsto b_0$ and $\pi_1 : Y_0 \times Y_1 \rightarrow Y_1 : (b_0, b_1) \mapsto b_1$. The product type set $\Pi(Y)$ is abstract, and is not necessarily equal, but only isomorphic, to the canonical (Cartesian) product $\Pi(Y) \cong Y_0 \times Y_1$. Hence, the product set is axiomatically underspecified. When the canonical product is needed at some point in a lower or more peripheral level, then additional axiomatization will be given there.

Cones. A binary product *cone* is an object in the comma category

$$\mathbf{Set} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \mathbf{Set}^2$$

over the functorial ospan $\Delta : \mathbf{Set} \rightarrow \mathbf{Set}^2 \leftarrow \mathbf{Set}^2 : 1_{\mathbf{Set}^2}$. More specifically, a cone is a triple (X, Y, γ) consisting of a vertex set X , a set pair Y , and a function pair $\gamma = (g_0, g_1) : \Delta(X) \rightarrow Y$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{cone}} &= \{(X, Y, \gamma) \mid X \in \text{set}, Y \in \text{set}^2, \gamma \in \text{ftn}^2, \partial_0(\gamma) = \Delta(X), \partial_1(\gamma) = Y\}, \text{ or} \\ \text{cone} &= \{(X, \gamma) \mid X \in \text{set}, \gamma \in \text{ftn}^2, \partial_0(\gamma) = \Delta(X)\} \subseteq \text{set} \times \text{ftn}^2. \end{aligned}$$

The map $\widehat{\text{cone}} \rightarrow \text{cone}$ is just projection; the map $\text{cone} \rightarrow \widehat{\text{cone}}$ is defined by $(X, \gamma) \mapsto (X, \partial_1(\gamma), \gamma)$. Cones are packed in the natural isomorphism axiomatization of the constant-product adjunction (expressing universality of the binary product operation). We define the parts of a cone. The cones that are axiomatized here are concrete. They can be referenced as follows.

```
(type.set:set ?X)
(type.dgm.pr.mor:function-pair ?g01)
(= (type.dgm.pr.mor:source ?g01) (type.dgm.pr.obj:constant ?X))

(type.ftn:function ?g)
(= ?g (type.lim.prd2.obj:pairing [?X ?g01]))
```

Here is the axiomatization.

```
(iff:set cone)
(forall ((cone ?c)
  (exists ((type.set:set ?X) (type.dgm.pr.mor:function-pair ?g)
    (= ?c [?X ?g])))
(forall ((type.set:set ?X) (type.dgm.pr.mor:function-pair ?g)
  (<=> (cone [?X ?g])
    (= (type.dgm.pr.mor:source ?g) (type.dgm.pr.obj:constant ?X))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((type.set:set ?X) (type.dgm.pr.mor:function-pair ?g) (cone [?X ?g]))
  (= (set [?X ?g]) ?X))

(iff:function function-pair)
(= (iff:source function-pair) cone)
(= (iff:target function-pair) type.dgm.pr.mor:function-pair)
(forall ((type.set:set ?X) (type.dgm.pr.mor:function-pair ?g) (cone [?X ?g]))
  (= (function-pair [?X ?g]) ?g))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c)
  (= (function0 ?c) (type.dgm.pr.mor:function0 (function-pair ?c))))

(iff:function function1)
(= (iff:source function1) cone)
```

```

(= (iff:target function1) type.ftn:function)
(forall ((cone ?c))
  (= (function1 ?c) (type.dgm.pr.mor:function1 (function-pair ?c))))

(forall ((cone ?c)
  (= (type.dgm.pr.obj:source (function-pair ?c))
    (type.dgm.pr.mor:constant (set ?c))))

(iff:function cone-set-pair)
(= (iff:source cone-set-pair) cone)
(= (iff:target cone-set-pair) type.dgm.pr.obj:set-pair)
(forall ((cone ?g))
  (= (cone-set-pair ?g) (type.dgm.pr.mor:target (function-pair ?g))))

```

For any cone $\Delta(X) \xrightarrow{\gamma} Y$ over set pair Y , there is an opposite cone $\Delta(X) \xrightarrow{\gamma^{\text{op}}} Y^{\text{op}}$ over the opposite set pair Y^{op} .

```

(iff.ftn:function opposite)
(= (iff.ftn:source opposite) cone)
(= (iff.ftn:target opposite) cone)
(forall ((cone ?c))
  (and (= (set (opposite ?c)) (set ?c))
    (= (function-pair (opposite ?c)) (type.dgm.pr.mor:opposite (function-pair ?c)))
    (= (cone-set-pair (opposite ?c)) (type.dgm.pr.obj:opposite (cone-set-pair ?c)))))

```

Mediators. A binary product *mediator* is an object in the comma category

$$\text{Set} \xrightarrow{\pi_0} (1 \downarrow \Pi) \xrightarrow{\pi_1} \text{Set}^2$$

over the functorial ospan $1_{\text{Set}} : \text{Set} \rightarrow \text{Set} \leftarrow \text{Set}^2 : \Pi$. More specifically, a mediator is a triple (X, Y, g) consisting of a set X , a set pair Y , and a function $g : X \rightarrow \Pi(Y)$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{mdtr}} &= \{(X, Y, g) \mid X \in \text{set}, Y \in \text{set}^2, g \in \text{ftn}, \partial_0(g) = X, \partial_1(g) = \Pi(Y)\}, \text{ or} \\ \text{mdtr} &= \{(g, Y) \mid g \in \text{ftn}, Y \in \text{set}^2, \partial_1(g) = \Pi(Y)\} \subseteq \text{ftn} \times \text{set}^2. \end{aligned}$$

The map $\widehat{\text{mdtr}} \rightarrow \text{mdtr}$ is just projection; the map $\text{mdtr} \rightarrow \widehat{\text{mdtr}}$ is defined by $(g, Y) \mapsto (\partial_0(g), Y, g)$. Mediators are unpacked in the natural isomorphism axiomatization of the constant-product adjunction (expressing universality of the binary product operation). We define the parts of a mediator. Although the products used to define mediators are abstract, the mediators themselves are concrete in the sense that they can be referenced as follows.

```

(type.ftn:function ?g)
(type.dgm.pr.obj:set-pair ?Y)
(= (type.ftn:target ?g) (type.lim.prd2.obj:product ?Y))

(type.dgm.pr.mor:function-pair ?g01)
(= ?g01 (type.lim.prd2.obj:components [?g ?Y]))

```

Here is the axiomatization.

$\frac{\Delta(X) \xrightarrow{1} \Delta(X)}{X \xrightarrow{\delta_X} \Pi(\Delta(X))}$	$\frac{\Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y}{\Pi(Y) \xrightarrow{1} \Pi(Y)}$
$\begin{array}{l} \text{set } X \\ \text{set pair } Y = (Y_0, Y_1) \end{array}$	$\begin{array}{l} \Delta(X) = (X, X) \\ \Pi(Y) = Y_0 \times Y_1 \end{array}$
Delta Mediator	Projection Cone

Figure 17: Binary Product Unit and Counit

```

(iff:set mediator)
(forall ((mediator ?m))
  (exists ((type.ftn:function ?g) (type.dgm.pr.obj:set-pair ?Y))
    (= ?m [?g ?Y])))
(forall ((type.ftn:function ?g) (type.dgm.pr.obj:set-pair ?Y))
  (<=> (mediator [?g ?Y])
    (= (type.ftn:target ?g) (product ?Y))))

(iff:function function)
(= (iff.ftn:source function) mediator)
(= (iff.ftn:target function) type.ftn:function)
(forall ((type.ftn:function ?g) (type.dgm.pr.obj:set-pair ?Y) (mediator [?g ?Y]))
  (= (function [?g ?Y]) ?g))

(iff:function set-pair)
(= (iff:source set-pair) mediator)
(= (iff:target set-pair) type.dgm.pr.obj:set-pair)
(forall ((type.ftn:function ?g) (type.dgm.pr.obj:set-pair ?Y) (mediator [?g ?Y]))
  (= (set-pair [?g ?Y]) ?Y))

(forall ((mediator ?m))
  (= (type.ftn:target (function ?m)) (product (set-pair ?m))))

(iff:function mediator-set)
(= (iff:source mediator-set) mediator)
(= (iff:target mediator-set) type.set:set)
(forall ((mediator ?m))
  (= (mediator-set ?m) (type.ftn:source (function ?m))))

```

Delta Unit. Given any type set X , the delta map constructs the type mediator (Figure 17) with function $\delta_X : X \rightarrow \Pi(\Delta(X)) \cong X \times X$, which is the packing of the cone whose function pair $1 : \Delta(X) \rightarrow \Delta(X)$ is the identity on the constant set pair over X (or the constant function pair over the identity function on X) – the delta mediator is the packing of two copies of the identity function. This delta function is a bijection. For any set X , the delta function δ_X is the X^{th} -component of the delta natural transformation $\delta : \mathbf{1}_{\mathbf{Set}} \Rightarrow \Delta \circ \Pi$, the unit of the constant-product adjunction $\Delta \dashv \Pi$. Naturality means that for any function $f : X \rightarrow Y$ we have the commuting diagram $\delta_X \cdot \Pi(\Delta(f)) = f \cdot \delta_Y$, where $\Pi(\Delta(f)) \cong f^2 = f \times f$. Delta naturality is a special case of packing naturality.

```

(iff:function delta-cone)

```

```

(= (iff:source delta-cone) type.set:set)
(= (iff:target delta-cone) cone)
(forall ((type.set:set ?X))
  (and (= (set (delta-cone ?X)) ?X)
    (= (function-pair (delta-cone ?X))
      (type.dgm.pr.mor:identity (type.dgm.pr.obj:constant ?X))))))

(iff:function delta)
(= (iff:source delta) type.set:set)
(= (iff:target delta) mediator)
(forall ((type.set:set ?X))
  (= (delta ?X) (packing (delta-cone ?X))))

(iff.ftn:function delta-function)
(= (iff:source delta-function) type.set:set)
(= (iff:target delta-function) type.ftn:function)
(forall ((type.set:set ?X))
  (and (= (type.ftn:source (delta-function ?X)) ?X)
    (= (type.ftn:target (delta-function ?X))
      (type.lim.prd2.obj:product (type.dgm.pr.obj:constant ?X))))
  (= (delta-function ?X) (function (delta ?X))))

(forall ((type.ftn:function ?f))
  (= (type.ftn:composition
    [(delta-function (type.ftn:source ?f))
     (type.lim.prd2.mor:product (type.dgm.pr.mor:constant ?f))])
    (type.ftn:composition [?f (delta-function (type.ftn:target ?f))])))

```

Projection Counit. Given any type set pair $Y = (Y_0, Y_1)$, the projection map constructs the type cone (Figure 17) with function pair $\pi_Y = (\pi_{Y_0}, \pi_{Y_1}) : \Delta(\Pi(Y)) \rightarrow Y$, which is the unpacking of the mediator whose function $1 : \Pi(Y) \rightarrow \Pi(Y)$ is the identity function for the binary product of the given type set pair – the projection cone is the unpacking of the product identity. This projection function is a bijection. For any set pair $Y = (Y_0, Y_1)$, the projection function pair π_Y is the Y^{th} -component of the projection natural transformation $\pi : \Pi \circ \Delta \Rightarrow 1_{\mathbf{Set}^2}$, the counit of the constant-product adjunction $\Delta \dashv \Pi$. Naturality means that for any function pair $f = (f_0, f_1) : X = (X_0, X_1) \rightarrow (Y_0, Y_1) = Y$ we have the commuting diagram $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$, where $\Pi(f) \cong f_0 \times f_1$. In the morphism namespace, the definition of the product of function pairs follows this. Projection naturality is a special case of unpacking

naturality. ¹⁹

```

(iff:function projection-mediator)
(= (iff:source projection-mediator) type.dgm.pr.obj:set-pair)
(= (iff:target projection-mediator) mediator)
(forall ((type.dgm.pr.obj:set-pair ?Y))
  (and (= (function (projection-mediator ?Y)) (type.ftn:identity (product ?Y)))
        (= (set-pair (projection-mediator ?Y)) ?Y)))

(iff:function projection)
(= (iff:source projection) type.dgm.pr.obj:set-pair)
(= (iff:target projection) cone)
(forall ((type.dgm.pr.obj:set-pair ?Y))
  (= (projection ?Y) (unpacking (projection-mediator ?Y))))

(iff.ftn:function product)
(= (iff:source product) type.dgm.pr.obj:set-pair)
(= (iff:target product) type.set:set)
(forall ((type.dgm.pr.obj:set-pair ?Y))
  (= (product ?Y) (set (projection ?Y))))

(iff.ftn:function projection-function-pair)
(= (iff:source projection-function-pair) type.dgm.pr.obj:set-pair)
(= (iff:target projection-function-pair) type.dgm.pr.mor:function-pair)
(forall ((type.dgm.pr.obj:set-pair ?Y))
  (and (= (type.dgm.pr.mor:source (projection-function-pair ?Y))
          (type.dgm.pr.obj:constant (product ?Y)))
        (= (type.dgm.pr.mor:target (projection-function-pair ?Y)) ?Y)
        (= (projection-function-pair ?Y) (function-pair (projection ?Y)))))

(iff.ftn:function projection0)
(= (iff.ftn:source projection0) type.dgm.pr.obj:set-pair)
(= (iff.ftn:target projection0) type.ftn:function)
(forall ((type.dgm.pr.obj:set-pair ?Y))
  (and (= (type.ftn:source (projection0 ?Y)) (product ?Y))
        (= (type.ftn:target (projection0 ?Y)) (type.dgm.pr.obj:set0 ?Y))
        (= (projection0 ?Y) (type.dgm.pr.mor:function0 (projection-function-pair ?Y)))))

(iff.ftn:function projection1)
(= (iff.ftn:source projection1) type.dgm.pr.obj:set-pair)
(= (iff.ftn:target projection1) type.ftn:function)
(forall ((type.dgm.pr.obj:set-pair ?Y))

```

¹⁹Here we define *product*, *projection* and *mediating* maps

$$\begin{aligned}
\Pi &: \mathbf{set}^2 \rightarrow \mathbf{set} \\
\pi &: \mathbf{set}^2 \rightarrow \mathbf{ftn}^2 \\
\langle - \rangle &: \mathbf{cone} \rightarrow \mathbf{ftn}
\end{aligned}$$

For any type set pair $Y = (Y_0, Y_1)$, the product type set $\Pi(Y) \cong Y_0 \times Y_1$ is a (not necessarily the Cartesian) binary product in the category **Set** of type sets and type functions: this and the binary product projection type function pair $\pi : \Delta(\Pi(Y)) \rightarrow Y$ form a cone that is universal over the given type set pair. For any set pair Y , the structure $(\Pi(Y), \pi)$, consisting of binary product set and projection function pair, is a universal arrow from the constant functor $\Delta : \mathbf{Set} \rightarrow \mathbf{Set}^2$ to the set pair Y . In more detail, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a limiting cone such that to every cone $x : \Delta(X) \rightarrow Y$ there is a unique *mediating* function $\langle x \rangle : X \rightarrow \Pi(Y)$ satisfying $\Delta(\langle x \rangle) \cdot \pi = x$. More concisely, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a terminal object in the slice category $(\Delta \downarrow Y)$.

```

    (and (= (type.ftn:source (projection1 ?Y)) (product ?Y))
         (= (type.ftn:target (projection1 ?Y)) (type.dgm.pr.obj:set1 ?Y))
         (= (projection1 ?Y) (type.dgm.pr.mor:function1 (projection-function-pair ?Y))))

  (forall ((type.dgm.pr.mor:function-pair ?f))
    (= (type.dgm.pr.mor:composition
       [(projection-function-pair (type.dgm.pr.mor:source ?f)) ?f]
       (type.dgm.pr.mor:composition
        [(type.dgm.pr.mor:constant (type.lim.prd2.mor:product ?f))
         (projection-function-pair (type.dgm.pr.mor:target ?f))])))
  
```

The pullback of the opspan of a set pair is the binary product set, and the pullback projection of this opspan is the binary product projection.

```

  (forall ((type.dgm.pr.obj:set-pair ?Y))
    (and (= (product ?Y)
            (type.lim.pbk.obj:pullback (type.dgm.pr.obj:opspan ?Y)))
         (= (projection-function-pair ?Y)
            (type.lim.pbk.obj:projection-function-pair (type.dgm.pr.obj:opspan ?Y))))
  
```

For any set pair $X = (X_0, X_1)$, there is a *tau* or *twist* cone $(\pi_0 : X_0 \leftarrow X_1 \times X_0 \rightarrow X_1 : \pi_1)$ whose function pair $\Delta(\Pi(X^{\text{op}})) \xrightarrow{\pi_{X^{\text{op}}}} X$ is the opposite of the projection function pair of X^{op} .

```

  (iff.ftn:function tau-cone)
  (= (iff.ftn:source tau-cone) type.dgm.pr.obj:set-pair)
  (= (iff.ftn:target tau-cone) cone)
  (forall ((type.dgm.pr.obj:set-pair ?X))
    (and (= (set (tau-cone ?X)) (product (type.dgm.ospn.obj:opposite ?X)))
         (= (function-pair (tau-cone ?X))
            (type.dgm.pr.mor:opposite (projection-function-pair (type.dgm.pr.obj:opposite ?X))))
         (= (cone-set-pair (tau-cone ?X)) ?X)))
  
```

Packing and Unpacking. Any cone can be packed into a mediator (Figure 16). For any cone γ with set X and function pair $\gamma = (g_0, g_1) : \Delta(X) \Rightarrow Y$, there is a mediator $g = \langle \gamma \rangle = \langle g_0, g_1 \rangle : X \rightarrow \Pi(Y)$, which pairs the images of the original two functions. The packing (pairing) process is realized by application of the product (Π) operator to the function pair (g_0, g_1) and then composition on the left with the delta function δ_X (a component of the unit δ):

$$g = \delta_X \cdot \Pi(\gamma) = \delta_X \cdot g_0 \times g_1 : X \rightarrow \Pi(\Delta(X)) \rightarrow \Pi(Y).$$

Any mediator can be unpacked into a cone. For any mediator g with set pair $Y = (Y_0, Y_1)$ and function $g : X \rightarrow \Pi(Y) = Y_0 \times Y_1$, there is a cone $\gamma = g_{01} = (g_0, g_1) : \Delta(X) \rightarrow Y$, which separates the mediator into two component functions. The unpacking (components) process is realized by application of the constant (Δ) operator to the function g and then composition on the right with the function pair π_Y (a component of the counit π):

$$\gamma = \Delta(g) \cdot \pi_Y = (g \cdot \pi_{Y_0}, g \cdot \pi_{Y_1}) : \Delta(X) \Rightarrow \Delta(\Pi(Y)) \Rightarrow Y.$$

These two processes are inverse to each other: $\langle \gamma \rangle_{01} = \gamma$ for each cone γ and $\langle g_{01} \rangle = g$ for each mediator g . This bijection is natural in the components for cones and mediators. This means that the diagram in Figure 18 is commutative for any function $f : X' \rightarrow X$ and any function pair $(h_0, h_1) : (Y_0, Y_1) \rightarrow (Y'_0, Y'_1)$.

```

(iff.ftn:function packing)
(= (iff.ftn:source packing) cone)
(= (iff.ftn:target packing) mediator)
(forall ((cone ?c))
  (and (= (function (packing ?c))
        (type.ftn:composition
          [(delta-function (set ?c))
           (type.lim.prd2.mor:product (function-pair ?c))]))
        (= (set-pair (packing ?c)) (cone-set-pair ?c))))

(iff:function pairing)
(= (iff:source pairing) cone)
(= (iff:target pairing) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (pairing ?c)) (set ?c))
        (= (type.ftn:target (pairing ?c))
            (type.lim.prd2.obj:product (cone-set-pair ?c)))
        (= (pairing ?c) (function (packing ?c)))))

(iff.ftn:function unpacking)
(= (iff.ftn:source unpacking) mediator)
(= (iff.ftn:target unpacking) cone)
(forall ((mediator ?m))
  (and (= (set (unpacking ?m)) (mediator-set ?m))
        (= (function-pair (unpacking ?m))
            (type.dgm.pr.mor:composition
              [(type.dgm.pr.mor:constant (function ?m))
               (projection-function-pair (set-pair ?m))])))

(iff:function components)
(= (iff:source components) mediator)
(= (iff:target components) type.dgm.pr.mor:function-pair)
(forall ((mediator ?m))
  (and (= (type.dgm.pr.mor:source (components ?m))
          (type.dgm.pr.obj:constant (mediator-set ?m)))
        (= (type.dgm.pr.mor:target (components ?m)) (set-pair ?m))
        (= (components ?m) (function-pair (unpacking ?m)))))

(forall ((cone ?c))
  (= (unpacking (packing ?c)) ?c))
(forall ((mediator ?m))
  (= (packing (unpacking ?m)) ?m))

(forall ((cone ?c) (cone ?d) (function ?f))
  (= (target ?f) (set ?c))
  (= (source ?f) (set ?d))
  (= (function-pair ?d)
      (type.dgm.pr.mor:composition
        [(type.dgm.pr.mor:constant ?f) (function-pair ?c)])))
(= (pairing ?d)
    (type.ftn:composition [?f (pairing ?c)])))

(forall ((cone ?c) (cone ?d) (type.dgm.pr.mor:function-pair ?h))
  (= (type.dgm.pr.mor:source ?h) (cone-set-pair ?c))
  (= (type.dgm.pr.mor:target ?h) (cone-set-pair ?d))
  (= (function-pair ?d)
      (type.dgm.pr.mor:composition [(function-pair ?c) ?h])))

```

$$\begin{array}{ccc}
X & Y = (Y_0, Y_1) & \begin{array}{ccc} (g_0, g_1) & \varphi_{X,Y} & g \\ \text{Set}^2[\Delta(X), Y] & \xrightarrow{\quad} & \text{Set}[X, \Pi(Y)] \\ \Delta(f) \cdot (-) \cdot h \downarrow & \varphi_{X',Y'} & \downarrow f \cdot (-) \cdot \Pi(h) \\ \text{Set}^2[\Delta(X'), Y'] & \xrightarrow{\quad} & \text{Set}[X', \Pi(Y')] \\ (f \cdot g_0 \cdot h_0, f \cdot g_1 \cdot h_1) & & f \cdot g \cdot \Pi(h) \end{array} \\
f \uparrow & \downarrow h = (h_0, h_1) & \\
X' & Y' = (Y'_0, Y'_1) &
\end{array}$$

Figure 18: Binary Product Naturality

```
(= (pairing ?d)
  (type.ftn:composition [(pairing ?c) (type.lim.prd2.mor:product ?h)]))
```

For any set pair $X = (X_0, X_1)$, there is a *tau* or *twist* bijection $\tau_X : \Pi(X^{\text{op}}) \xrightarrow{\cong} \Pi(X)$ from the product of the opposite set pair $X^{\text{op}} = (X_1, X_0)$ to the product of X . This is the product pairing of the tau cone $(\pi_0 : X_0 \leftarrow \Pi(X^{\text{op}}) \rightarrow X_1 : \pi_1)$ over the set pair X .

```
(iff.ftn:function tau)
(= (iff.ftn:source tau) type.dgm.pr.obj:set-pair)
(= (iff.ftn:target tau) type.ftn:function)
(forall ((type.dgm.pr.obj:set-pair ?X))
  (and (= (type.ftn:source (tau ?X)) (type.lim.prd2.obj:product (type.dgm.pr.obj:opposite ?X)))
    (= (type.ftn:target (tau ?X)) (type.lim.prd2.obj:product ?X))
    (= (tau ?X) (pairing (tau-cone ?X))))
  (type.ftn:bijection (tau ?X))))
```

Morphisms.

```
type.lim.prd2.mor
```

Products. The binary product operation is extended to morphisms. Given any function pair $f : X \rightarrow Y$, there is a binary product function $\Pi(f) : \Pi(X) \rightarrow \Pi(Y)$ defined using projections: $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$.

```
(iff:function product)
(= (iff:source product) type.dgm.pr.mor:function-pair)
(= (iff:target product) type.ftn:function)
(forall ((type.dgm.pr.mor:function-pair ?f)
  (type.dgm.pr.obj:set-pair ?X) (type.dgm.pr.obj:set-pair ?Y)
  (= (type.dgm.pr.mor:source ?f) ?X) (= (type.dgm.pr.mor:target ?f) ?Y))
  (and (= (type.ftn:source (product ?f)) (type.lim.prd2.obj:product ?X))
    (= (type.ftn:target (product ?f)) (type.lim.prd2.obj:product ?Y))
    (= (type.dgm.pr.mor:composition
      [(type.lim.prd2.obj:projection-function-pair ?X) ?f])
      (type.dgm.pr.mor:composition
        [(type.dgm.pr.mor:delta (product ?f))
         (type.lim.prd2.obj:projection-function-pair ?Y)]))))))
```

1.8.3 Type Binary Powers

```
type.lim.pwr2
```

The type binary power monad $\langle \Delta \circ \Pi, \delta, \Delta \pi \Pi \rangle : \text{Set} \rightarrow \text{Set}$ is sometimes useful.

Objects.

type.lim.pwr2.obj

Cones. A type binary power *cone* is a binary product cone $\Delta(X) \xrightarrow{\cong} \Delta(Y)$, whose underlying set pair is the constant for some type set Y .

```
(iff:set cone)
(forall ((cone ?c) (type.lim.prd2.obj:cone ?c))
(forall ((type.lim.prd2.obj:cone ?c)
  (<=> (cone ?c)
    (exists ((type.set:set ?Y)
      (= (type.lim.prd2.obj:cone-set-pair ?c) (type.dgm.pr.obj:constant ?Y))))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((cone ?c)
  (= (set ?c) (type.lim.prd2.obj:set ?c)))

(iff:function function-pair)
(= (iff:source function-pair) cone)
(= (iff:target function-pair) type.ftn:function)
(forall ((cone ?c)
  (= (function-pair ?c) (type.lim.prd2.obj:function-pair ?c)))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c)
  (= (function0 ?c) (type.dgm.pr.mor:function0 (function-pair ?c))))

(iff:function function1)
(= (iff:source function1) cone)
(= (iff:target function1) type.ftn:function)
(forall ((cone ?c)
  (= (function1 ?c) (type.dgm.pr.mor:function1 (function-pair ?c))))

(iff:function cone-set)
(= (iff:source cone-set) cone)
(= (iff:target cone-set) type.set:set)
(forall ((cone ?c)
  (= (type.dgm.pr.obj:constant (cone-set ?c)) (type.lim.prd2.obj:cone-set-pair ?c))))
```

Monad. For any type set Y , the binary *power* type set Y^2 and the binary power *projection* type function pair $\pi_Y : \Delta(Y^2) \rightarrow \Delta(Y)$ are defined in terms of the binary product set and binary product projection function pair of the type set pair $\Delta(Y)$. For any binary power cone $\Delta(X) \xrightarrow{\cong} \Delta(Y)$, the binary power *pairing* type function $\langle \gamma \rangle = (g_0, g_1) : X \rightarrow Y^2 = \Pi(\Delta(Y))$ is defined in terms of the pairing of the cone as binary product cone. The power set is abstract.

```
(iff:function power)
(= (iff:source power) type.set:set)
(= (iff:target power) type.set:set)
(forall ((type.set:set ?Y)
  (= (power ?Y) (type.lim.prd2.obj:product (type.dgm.pr.obj:constant ?Y))))
```

```

(iff:function projection-function-pair)
(= (iff:source projection-function-pair) type.set:set)
(= (iff:target projection-function-pair) type.dgm.pr.mor:function-pair)
(forall ((type.set:set ?Y))
  (and (= (type.dgm.pr.mor:source (projection-function-pair ?Y))
        (type.dgm.pr.obj:constant (power ?Y)))
        (= (type.dgm.pr.mor:target (projection-function-pair ?Y))
          (type.dgm.pr.obj:constant ?Y))
        (= (projection-function-pair ?Y)
          (type.lim.prd2.obj:projection-function-pair (type.dgm.pr.obj:constant ?Y))))))

(iff:ftn:function projection0)
(= (iff:ftn:source projection0) type.set:set)
(= (iff:ftn:target projection0) type.ftn:function)
(forall ((type.set:set ?Y))
  (and (= (type.ftn:source (projection0 ?Y)) (power ?Y))
        (= (type.ftn:target (projection0 ?Y)) ?Y)
        (= (projection0 ?Y) (type.dgm.pr.mor:function0 (projection-function-pair ?Y)))))

(iff:ftn:function projection1)
(= (iff:ftn:source projection1) type.set:set)
(= (iff:ftn:target projection1) type.ftn:function)
(forall ((type.set:set ?Y))
  (and (= (type.ftn:source (projection1 ?Y)) (power ?Y))
        (= (type.ftn:target (projection1 ?Y)) ?Y)
        (= (projection1 ?Y) (type.dgm.pr.mor:function1 (projection-function-pair ?Y)))))

(iff:function pairing)
(= (iff:source pairing) cone)
(= (iff:target pairing) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (pairing ?c)) (set ?c))
        (= (type.ftn:target (pairing ?c)) (power (cone-set ?c)))
        (= (pairing ?c) (type.lim.prd2.obj:pairing ?c))))

```

Morphisms.

type.lim.pwr2.mor

Powers. The binary power operation is extended to morphisms. For any type function $f : X \rightarrow Y$, the binary *power* type function $f^2 : X^2 \rightarrow Y^2$ is defined in terms of the binary product function for the function pair $\Delta(f)$. Given any function $f : X \rightarrow Y$, there is a binary power function $f^2 : X^2 \rightarrow Y^2$ defined using projections: $\pi_X \cdot \Delta(f) = \Delta(f^2) \cdot \pi_Y$.

```

(iff:function power)
(= (iff:source power) type.ftn:function)
(= (iff:target power) type.ftn:function)
(forall ((type.ftn:function ?f) (type.set:set ?X) (type.set:set ?Y))
  (= (type.ftn:source ?f) ?X) (= (type.ftn:target ?f) ?Y))
  (and (= (type.ftn:source (power ?f)) (type.lim.pwr2.obj:power ?X))
        (= (type.ftn:target (power ?f)) (type.lim.pwr2.obj:power ?Y))
        (= (power ?f) (type.lim.prd2.mor:product (type.dgm.pr.mor:constant ?f)))
        (= (type.dgm.pr.mor:composition
            [(type.lim.pwr2.obj:projection-function-pair ?X)

```

```
(type.dgm.pr.mor:delta ?f)])  
(type.dgm.pr.mor:composition  
 [(type.dgm.pr.mor:delta (power ?f))  
  (type.lim.pwr2.obj:projection-function-pair ?Y)]))
```

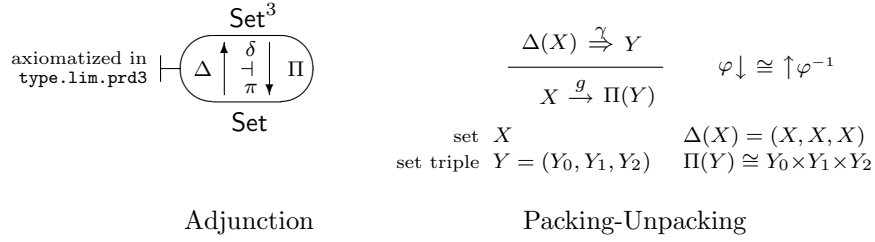


Figure 19: Ternary Product

1.8.4 Type Ternary Products

`type.lim.prd3`

The essential properties of the ternary product construction (see Figure 14...) are illustrated in Figure 19. The horizontal bar designates a natural bijection between the cone above the bar and the mediator below the bar. Let X be a set with $\Delta(X) = (X, X, X)$, and let $Y = (Y_0, Y_1, Y_2)$ be a set triple with $\Pi(Y) = Y_0 \times Y_1 \times Y_2$. A natural packing (tripling) process (φ) assigns a mediator $X \rightarrow \Pi(Y)$ below the bar to every cone $\Delta(X) \Rightarrow Y$ above the bar. A natural unpacking (components) process (φ^{-1}) assigns a cone $\Delta(X) \Rightarrow Y$ above the bar to every mediator $X \rightarrow \Pi(Y)$ below the bar. These two processes are inverse to each other. The packing process is realized by application of the product (Π) operation and then composition on the left with the delta function (a component of the unit δ)

$$g = X \xrightarrow{\delta_X} \Pi(\Delta(X)) \xrightarrow{\Pi(\gamma)} \Pi(Y).$$

The unpacking process is realized by application of the constant (Δ) operation and then composition on the right with the projection function triple (a component of the counit π)

$$\gamma = \Delta(X) \xrightarrow{\Delta(g)} \Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y.$$

Objects.

`type.lim.prd3.obj`

A ternary product *cone* is an object in the comma category

$$\mathbf{Set} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \mathbf{Set}^3$$

over the functorial ospan $\Delta : \mathbf{Set} \rightarrow \mathbf{Set}^3 \leftarrow \mathbf{Set}^3 : 1_{\mathbf{Set}^3}$. More specifically, a cone is a triple (X, Y, γ) consisting of a vertex set X , a set triple Y , and a function triple $\gamma = (g_0, g_1, g_2) : \Delta(X) \rightarrow Y$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\widehat{\text{cone}} = \{(X, Y, \gamma) \mid X \in \text{set}, Y \in \text{set}^3, \gamma \in \text{ftn}^3, \partial_0(\gamma) = \Delta(X), \partial_1(\gamma) = Y\}, \text{ or}$$

$$\text{cone} = \{(X, \gamma) \mid X \in \text{set}, \gamma \in \text{ftn}^3, \partial_0(\gamma) = \Delta(X)\} \subseteq \text{set} \times \text{ftn}^3.$$

The map $\widehat{\text{cone}} \rightarrow \text{cone}$ is just projection; the map $\text{cone} \rightarrow \widehat{\text{cone}}$ is defined by $(X, \gamma) \mapsto (X, \partial_1(\gamma), \gamma)$. Cones are packed in the natural isomorphism axiomatization of the constant-product adjunction (expressing universality of the ternary product operation). We define the parts of a cone. The cones that are axiomatized here are concrete. They can be referenced as follows.

```
(type.set:set ?X)
(type.dgm.trp.mor:function-triple ?g012)
(= (type.dgm.trp.mor:source ?g012) (type.dgm.trp.obj:constant ?X))

(type.ftn:function ?g)
(= ?g (type.lim.prd3.obj:tripling [?X ?g012]))
```

Here is the axiomatization.

```
(iff:set cone)
(forall ((cone ?c))
  (exists ((type.set:set ?X) (type.dgm.trp.mor:function-triple ?g))
    (= ?c [?X ?g])))
(forall ((type.set:set ?X) (type.dgm.trp.mor:function-triple ?g))
  (<=> (cone [?X ?g])
    (= (type.dgm.trp.mor:source ?g) (type.dgm.trp.obj:constant ?X))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((type.set:set ?X) (type.dgm.trp.mor:function-triple ?g) (cone [?X ?g]))
  (= (set [?X ?g]) ?X))

(iff:function function-triple)
(= (iff:source function-triple) cone)
(= (iff:target function-triple) type.dgm.trp.mor:function-triple)
(forall ((type.set:set ?X) (type.dgm.trp.mor:function-triple ?g) (cone [?X ?g]))
  (= (function-triple [?X ?g]) ?g))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c))
  (= (function0 ?c) (type.dgm.trp.mor:function0 (function-triple ?c))))

(iff:function function1)
(= (iff:source function1) cone)
(= (iff:target function1) type.ftn:function)
(forall ((cone ?c))
  (= (function1 ?c) (type.dgm.trp.mor:function1 (function-triple ?c))))

(iff:function function2)
(= (iff:source function2) cone)
(= (iff:target function2) type.ftn:function)
(forall ((cone ?c))
  (= (function2 ?c) (type.dgm.trp.mor:function2 (function-triple ?c))))

(forall ((cone ?c))
  (= (type.dgm.trp.obj:source (function-triple ?c))
```

```

(type.dgm.trp.mor:constant (set ?c)))

(iff:function cone-set-triple)
(= (iff:source cone-set-triple) cone)
(= (iff:target cone-set-triple) type.dgm.trp.obj:set-triple)
(forall ((cone ?g)
  (= (cone-set-triple ?g) (type.dgm.trp.mor:target (function-triple ?g))))

```

Mediators. A ternary product *mediator* is an object in the comma category

$$\text{Set} \xleftarrow{\pi_0} (1 \downarrow \Pi) \xrightarrow{\pi_1} \text{Set}^3$$

over the functorial ospan $1_{\text{Set}} : \text{Set} \rightarrow \text{Set} \leftarrow \text{Set}^3 : \Pi$. More specifically, a mediator is a triple (X, Y, g) consisting of a set X , a set triple Y , and a function $g : X \rightarrow \Pi(Y)$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{mdtr}} &= \{(X, Y, g) \mid X \in \text{set}, Y \in \text{set}^3, g \in \text{ftn}, \partial_0(g) = X, \partial_1(g) = \Pi(Y)\}, \text{ or} \\ \text{mdtr} &= \{(g, Y) \mid g \in \text{ftn}, Y \in \text{set}^3, \partial_1(g) = \Pi(Y)\} \subseteq \text{ftn} \times \text{set}^3. \end{aligned}$$

The map $\widehat{\text{mdtr}} \rightarrow \text{mdtr}$ is just projection; the map $\text{mdtr} \rightarrow \widehat{\text{mdtr}}$ is defined by $(g, Y) \mapsto (\partial_0(g), Y, g)$. Mediators are unpacked in the natural isomorphism axiomatization of the constant-product adjunction (expressing universality of the ternary product operation). We define the parts of a mediator. Although the products used to define mediators are abstract, the mediators themselves are concrete in the sense that they can be referenced as follows.

```

(type.ftn:function ?g)
(type.dgm.trp.obj:set-triple ?Y)
(= (type.ftn:target ?g) (type.lim.prd3.obj:product ?Y))

(type.dgm.pr.mor:function-triple ?g012)
(= ?g012 (type.lim.prd3.obj:components [?g ?Y]))

```

Here is the axiomatization.

```

(iff:set mediator)
(forall ((mediator ?m)
  (exists ((type.ftn:function ?g) (type.dgm.trp.obj:set-triple ?Y))
    (= ?m [?g ?Y])))
(forall ((type.ftn:function ?g) (type.dgm.trp.obj:set-triple ?Y))
  (<=> (mediator [?g ?Y])
    (= (type.ftn:target ?g) (product ?Y))))

(iff:function function)
(= (iff.ftn:source function) mediator)
(= (iff.ftn:target function) type.ftn:function)
(forall ((type.ftn:function ?g) (type.dgm.trp.obj:set-triple ?Y) (mediator [?g ?Y]))
  (= (function [?g ?Y]) ?g))

(iff:function set-triple)
(= (iff:source set-triple) mediator)
(= (iff:target set-triple) type.dgm.trp.obj:set-triple)
(forall ((type.ftn:function ?g) (type.dgm.trp.obj:set-triple ?Y) (mediator [?g ?Y]))

```

$$\begin{array}{ccc}
\frac{\Delta(X) \xrightarrow{1} \Delta(X)}{X \xrightarrow{\delta_X} \Pi(\Delta(X))} & & \frac{\Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y}{\Pi(Y) \xrightarrow{1} \Pi(Y)} \\
\text{set } X & & \Delta(X) = (X, X, X) \\
\text{set pair } Y = (Y_0, Y_1, Y_2) & & \Pi(Y) = Y_0 \times Y_1 \times Y_2 \\
\text{Delta Mediator} & & \text{Projection Cone}
\end{array}$$

Figure 20: Ternary Product Unit and Counit

```

(= (set-triple [?g ?Y] ?Y))

(forall ((mediator ?m))
  (= (type.ftn:target (function ?m)) (product (set-triple ?m))))

(iff:function mediator-set)
(= (iff:source mediator-set) mediator)
(= (iff:target mediator-set) type.set:set)
(forall ((mediator ?m))
  (= (mediator-set ?m) (type.ftn:source (function ?m))))

```

Delta Unit. Given any type set X , the delta map constructs the type mediator (Figure 20) with function $\delta_X : X \rightarrow \Pi(\Delta(X)) \cong X \times X \times X$, which is the packing of the cone whose function triple $1 : \Delta(X) \rightarrow \Delta(X)$ is the identity on the constant set triple over X (or the constant function triple over the identity function on X) – the delta mediator is the packing of three copies of the identity function. This delta function is a bijection. For any set X , the delta function δ_X is the X^{th} -component of the delta natural transformation $\delta : 1_{\mathbf{Set}} \Rightarrow \Delta \circ \Pi$, the unit of the constant-product adjunction $\Delta \dashv \Pi$. Naturality means that for any function $f : X \rightarrow Y$ we have the commuting diagram $\delta_X \cdot \Pi(\Delta(f)) = f \cdot \delta_Y$, where $\Pi(\Delta(f)) \cong f^3 = f \times f \times f$. Delta naturality is a special case of packing naturality.

```

(iff:function delta-cone)
(= (iff:source delta-cone) type.set:set)
(= (iff:target delta-cone) cone)
(forall ((type.set:set ?X))
  (and (= (set (delta-cone ?X)) ?X)
        (= (function-triple (delta-cone ?X))
            (type.dgm.trp.mor:identity (type.dgm.trp.obj:constant ?X)))))

(iff:function delta)
(= (iff:source delta) type.set:set)
(= (iff:target delta) mediator)
(forall ((type.set:set ?X))
  (= (delta ?X) (packing (delta-cone ?X))))

(iff.ftn:function delta-function)
(= (iff:source delta-function) type.set:set)
(= (iff:target delta-function) type.ftn:function)
(forall ((type.set:set ?X))

```

```

    (and (= (type.ftn:source (delta-function ?X)) ?X)
         (= (type.ftn:target (delta-function ?X))
            (type.lim.prd3.obj:product (type.dgm.trp.obj:constant ?X)))
         (= (delta-function ?X) (function (delta ?X))))

(forall ((type.ftn:function ?f))
  (= (type.ftn:composition
     [(delta-function (type.ftn:source ?f))
      (type.lim.prd3.mor:product (type.dgm.trp.mor:constant ?f))])
     (type.ftn:composition [?f (delta-function (type.ftn:target ?f))])))

```

Projection Counit. Given any type set triple $Y = (Y_0, Y_1, Y_2)$, the projection map constructs the type cone (Figure 20) with function triple $\pi_Y = (\pi_{Y_0}, \pi_{Y_1}, \pi_{Y_2}) : \Delta(\Pi(Y)) \rightarrow Y$, which is the unpacking of the mediator whose function $1 : \Pi(Y) \rightarrow \Pi(Y)$ is the identity function for the ternary product of the given type set triple – the projection cone is the unpacking of the product identity. This projection function is a bijection. For any set triple $Y = (Y_0, Y_1, Y_2)$, the projection function triple π_Y is the Y^{th} -component of the projection natural transformation $\pi : \Pi \circ \Delta \Rightarrow 1_{\mathbf{Set}^3}$, the counit of the constant-product adjunction $\Delta \dashv \Pi$. Naturality means that for any function triple $f = (f_0, f_1, f_2) : X = (X_0, X_1, X_2) \rightarrow (Y_0, Y_1, Y_2) = Y$ we have the commuting diagram $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$, where $\Pi(f) \cong f_0 \times f_1 \times f_2$. In the morphism namespace, the definition of the product of function triples follows this. Projection naturality is a special case of unpacking naturality.²⁰

```

(iff:function projection-mediator)
(= (iff:source projection-mediator) type.dgm.trp.obj:set-triple)
(= (iff:target projection-mediator) mediator)
(forall ((type.dgm.trp.obj:set-triple ?Y))
  (and (= (function (projection-mediator ?Y)) (type.ftn:identity (product ?Y)))
        (= (set-triple (projection-mediator ?Y)) ?Y)))

(iff:function projection)
(= (iff:source projection) type.dgm.trp.obj:set-triple)
(= (iff:target projection) cone)
(forall ((type.dgm.trp.obj:set-triple ?Y))
  (= (projection ?Y) (unpacking (projection-mediator ?Y))))

(iff.ftn:function product)
(= (iff:source product) type.dgm.trp.obj:set-triple)
(= (iff:target product) type.set:set)

```

²⁰Here we define *product*, *projection* and *mediating* maps

$$\begin{aligned}
\Pi &: \mathbf{set}^3 \rightarrow \mathbf{set} \\
\pi &: \mathbf{set}^3 \rightarrow \mathbf{ftn}^3 \\
\langle - \rangle &: \mathbf{cone} \rightarrow \mathbf{ftn}
\end{aligned}$$

For any type set triple $Y = (Y_0, Y_1, Y_2)$, the product type set $\Pi(Y) \cong Y_0 \times Y_1 \times Y_2$ is a (not necessarily the Cartesian) ternary product in the category \mathbf{Set} of type sets and type functions: this and the ternary product projection type function triple $\pi : \Delta(\Pi(Y)) \rightarrow Y$ form a cone that is universal over the given type set triple. For any set triple Y , the structure $(\Pi(Y), \pi)$, consisting of ternary product set and projection function triple, is a universal arrow from the constant functor $\Delta : \mathbf{Set} \rightarrow \mathbf{Set}^3$ to the set triple Y . In more detail, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a limiting cone such that to every cone $x : \Delta(X) \rightarrow Y$ there is a unique *mediating* function $\langle x \rangle : X \rightarrow \Pi(Y)$ satisfying $\Delta(\langle x \rangle) \cdot \pi = x$. More concisely, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a terminal object in the slice category $(\Delta \downarrow Y)$.

```

(forall ((type.dgm.trp.obj:set-triple ?Y)
  (= (product ?Y) (set (projection ?Y))))

(iff.ftn:function projection-function-triple)
(= (iff.source projection-function-triple) type.dgm.trp.obj:set-triple)
(= (iff.target projection-function-triple) type.dgm.trp.mor:function-triple)
(forall ((type.dgm.trp.obj:set-triple ?Y)
  (and (= (type.dgm.trp.mor:source (projection-function-triple ?Y))
    (type.dgm.trp.obj:constant (product ?Y)))
    (= (type.dgm.trp.mor:target (projection-function-triple ?Y)) ?Y)
    (= (projection-function-triple ?Y) (function-triple (projection ?Y)))))

(iff.ftn:function projection0)
(= (iff.ftn:source projection0) type.dgm.trp.obj:set-triple)
(= (iff.ftn:target projection0) type.ftn:function)
(forall ((type.dgm.trp.obj:set-triple ?Y)
  (and (= (type.ftn:source (projection0 ?Y)) (product ?Y))
    (= (type.ftn:target (projection0 ?Y)) (type.dgm.trp.obj:set0 ?Y))
    (= (projection0 ?Y) (type.dgm.trp.mor:function0 (projection-function-triple ?Y)))))

(iff.ftn:function projection1)
(= (iff.ftn:source projection1) type.dgm.trp.obj:set-triple)
(= (iff.ftn:target projection1) type.ftn:function)
(forall ((type.dgm.trp.obj:set-triple ?Y)
  (and (= (type.ftn:source (projection1 ?Y)) (product ?Y))
    (= (type.ftn:target (projection1 ?Y)) (type.dgm.trp.obj:set1 ?Y))
    (= (projection1 ?Y) (type.dgm.trp.mor:function1 (projection-function-triple ?Y)))))

(iff.ftn:function projection2)
(= (iff.ftn:source projection2) type.dgm.trp.obj:set-triple)
(= (iff.ftn:target projection2) type.ftn:function)
(forall ((type.dgm.trp.obj:set-triple ?Y)
  (and (= (type.ftn:source (projection2 ?Y)) (product ?Y))
    (= (type.ftn:target (projection2 ?Y)) (type.dgm.trp.obj:set2 ?Y))
    (= (projection2 ?Y) (type.dgm.trp.mor:function2 (projection-function-triple ?Y)))))

(forall ((type.dgm.trp.mor:function-triple ?f)
  (= (type.dgm.trp.mor:composition
    [(projection-function-triple (type.dgm.trp.mor:source ?f)) ?f])
    (type.dgm.trp.mor:composition
    [(type.dgm.trp.mor:constant (type.lim.prd3.mor:product ?f))
    (projection-function-triple (type.dgm.trp.mor:target ?f))])))

```

Packing and Unpacking. Any cone can be packed into a mediator (Figure 19). For any cone γ with set X and function triple $\gamma = (g_0, g_1, g_2) : \Delta(X) \Rightarrow Y$, there is a mediator $g = \langle \gamma \rangle = \langle g_0, g_1, g_2 \rangle : X \rightarrow \Pi(Y)$, which triples the images of the original three functions. The packing (tripling) process is realized by application of the product (Π) operator to the function triple (g_0, g_1, g_2) and then composition on the left with the delta function δ_X (a component of the unit δ):

$$g = \delta_X \cdot \Pi(\gamma) = \delta_X \cdot g_0 \times g_1 \times g_2 : X \rightarrow \Pi(\Delta(X)) \rightarrow \Pi(Y).$$

Any mediator can be unpacked into a cone. For any mediator g with set triple $Y = (Y_0, Y_1, Y_2)$ and function $g : X \rightarrow \Pi(Y) = Y_0 \times Y_1 \times Y_2$, there is a cone

$\gamma = g_{012} = (g_0, g_1, g_2) : \Delta(X) \rightarrow Y$, which separates the mediator into three component functions. The unpacking (components) process is realized by application of the constant (Δ) operator to the function g and then composition on the right with the function triple π_Y (a component of the counit π):

$$\gamma = \Delta(g) \cdot \pi_Y = (g \cdot \pi_{Y_0}, g \cdot \pi_{Y_1}, g \cdot \pi_{Y_2}) : \Delta(X) \Rightarrow \Delta(\Pi(Y)) \Rightarrow Y.$$

These two processes are inverse to each other: $\langle \gamma \rangle_{012} = \gamma$ for each cone γ and $\langle g_{012} \rangle = g$ for each mediator g . This bijection is natural in the components for cones and mediators. This means that the diagram in Figure 21 is commutative for any function $f : X' \rightarrow X$ and any function triple $(h_0, h_1, h_1) : (Y_0, Y_1, Y_2) \rightarrow (Y'_0, Y'_1, Y'_2)$.

```
(iff.ftn:function packing)
(= (iff.ftn:source packing) cone)
(= (iff.ftn:target packing) mediator)
(forall ((cone ?c))
  (and (= (function (packing ?c))
          (type.ftn:composition
            [(delta-function (set ?c))
             (type.lim.prd3.mor:product (function-triple ?c))]))
        (= (set-triple (packing ?c)) (cone-set-triple ?c))))

(iff:function tripling)
(= (iff:source tripling) cone)
(= (iff:target tripling) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (tripling ?c)) (set ?c))
        (= (type.ftn:target (tripling ?c))
            (type.lim.prd3.obj:product (cone-set-triple ?c)))
        (= (tripling ?c) (function (packing ?c)))))

(iff.ftn:function unpacking)
(= (iff.ftn:source unpacking) mediator)
(= (iff.ftn:target unpacking) cone)
(forall ((mediator ?m))
  (and (= (set (unpacking ?m)) (mediator-set ?m))
        (= (function-triple (unpacking ?m))
            (type.dgm.trp.mor:composition
              [(type.dgm.trp.mor:constant (function ?m))
               (projection-function-triple (set-triple ?m))])))))

(iff:function components)
(= (iff:source components) mediator)
(= (iff:target components) type.dgm.trp.mor:function-triple)
(forall ((mediator ?m))
  (and (= (type.dgm.trp.mor:source (components ?m))
          (type.dgm.trp.obj:constant (mediator-set ?m)))
        (= (type.dgm.trp.mor:target (components ?m)) (set-triple ?m))
        (= (components ?m) (function-triple (unpacking ?m)))))

(forall ((cone ?c))
  (= (unpacking (packing ?c)) ?c))
(forall ((mediator ?m))
  (= (packing (unpacking ?m)) ?m))

(forall ((cone ?c) (cone ?d) (function ?f))
```

$$\begin{array}{ccc}
X & Y = (Y_0, Y_1, Y_2) & \text{Set}^2[\Delta(X), Y] \xrightarrow{\varphi_{X,Y}} \text{Set}[X, \Pi(Y)] \\
f \uparrow & \downarrow h = (h_0, h_1, h_2) & \Delta(f) \cdot (-) \cdot h \downarrow \varphi_{X',Y'} \downarrow f \cdot (-) \cdot \Pi(h) \\
X' & Y' = (Y'_0, Y'_1, Y'_2) & \text{Set}^2[\Delta(X'), Y'] \xrightarrow{\varphi_{X',Y'}} \text{Set}[X', \Pi(Y')] \\
& & (f \cdot g_0 \cdot h_0, f \cdot g_1 \cdot h_1, f \cdot g_2 \cdot h_2) \quad f \cdot g \cdot \Pi(h)
\end{array}$$

Figure 21: Ternary Product Naturality

```

(= (target ?f) (set ?c))
(= (source ?f) (set ?d))
(= (function-triple ?d)
    (type.dgm.trp.mor:composition
     [(type.dgm.trp.mor:constant ?f) (function-triple ?c)]))
(= (tripling ?d)
    (type.ftn:composition [?f (tripling ?c)]))

(forall ((cone ?c) (cone ?d) (type.dgm.trp.mor:function-triple ?h)
         (= (type.dgm.trp.mor:source ?h) (cone-set-triple ?c))
         (= (type.dgm.trp.mor:target ?h) (cone-set-triple ?d))
         (= (function-triple ?d)
            (type.dgm.trp.mor:composition [(function-triple ?c) ?h])))
(= (tripling ?d)
    (type.ftn:composition [(tripling ?c) (type.lim.prd3.mor:product ?h)]))

```

Morphisms.

type.lim.prd3.mor

Products. The ternary product operation is extended to morphisms. Given any function triple $f : X \rightarrow Y$, there is a ternary product function $\Pi(f) : \Pi(X) \rightarrow \Pi(Y)$ defined using projections: $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$.

```

(iff:function product)
(= (iff:source product) type.dgm.trp.mor:function-triple)
(= (iff:target product) type.ftn:function)
(forall ((type.dgm.trp.mor:function-triple ?f)
         (type.dgm.trp.obj:set-triple ?X) (type.dgm.trp.obj:set-triple ?Y)
         (= (type.dgm.trp.mor:source ?f) ?X) (= (type.dgm.trp.mor:target ?f) ?Y))
 (and (= (type.ftn:source (product ?f)) (type.lim.prd3.obj:product ?X))
       (= (type.ftn:target (product ?f)) (type.lim.prd3.obj:product ?Y))
       (= (type.dgm.trp.mor:composition
           [(type.lim.prd3.obj:projection-function-triple ?X) ?f])
          (type.dgm.trp.mor:composition
           [(type.dgm.trp.mor:delta (product ?f))
            (type.lim.prd2.obj:projection-function-triple ?Y)]))))

```

1.8.5 Type Ternary Powers

type.lim.pwr3

The type ternary power monad $\langle \Delta \circ \Pi, \delta, \Delta \Pi \rangle : \text{Set} \rightarrow \text{Set}$ is sometimes useful.

Objects.

type.lim.pwr3.obj

Cones. A type ternary power *cone* is a ternary product cone $\Delta(X) \xrightarrow{\cong} \Delta(Y)$, whose underlying set triple is the constant for some type set Y .

```
(iff:set cone)
(forall ((cone ?c)) (type.lim.prd3.obj:cone ?c))
(forall ((type.lim.prd3.obj:cone ?c))
  (<=> (cone ?c)
    (exists ((type.set:set ?Y))
      (= (type.lim.prd3.obj:cone-set-triple ?c) (type.dgm.trp.obj:constant ?Y))))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((cone ?c))
  (= (set ?c) (type.lim.prd3.obj:set ?c)))

(iff:function function-triple)
(= (iff:source function-triple) cone)
(= (iff:target function-triple) type.ftn:function)
(forall ((cone ?c))
  (= (function-triple ?c) (type.lim.prd3.obj:function-triple ?c)))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c))
  (= (function0 ?c) (type.dgm.trp.mor:function0 (function-triple ?c))))

(iff:function function1)
(= (iff:source function1) cone)
(= (iff:target function1) type.ftn:function)
(forall ((cone ?c))
  (= (function1 ?c) (type.dgm.trp.mor:function1 (function-triple ?c))))

(iff:function function2)
(= (iff:source function2) cone)
(= (iff:target function2) type.ftn:function)
(forall ((cone ?c))
  (= (function2 ?c) (type.dgm.trp.mor:function2 (function-triple ?c))))

(iff:function cone-set)
(= (iff:source cone-set) cone)
(= (iff:target cone-set) type.set:set)
(forall ((cone ?c))
  (= (type.dgm.trp.obj:constant (cone-set ?c)) (type.lim.prd3.obj:cone-set-triple ?c)))
```

Monad. For any type set Y , the ternary *power* type set Y^3 and the ternary power *projection* type function triple $\pi_Y : \Delta(Y^3) \rightarrow \Delta(Y)$ are defined in terms of the ternary product set and ternary product projection function triple of the type set triple $\Delta(Y)$. For any ternary power cone $\Delta(X) \xrightarrow{\cong} \Delta(Y)$, the ternary power *tripling* type function $\langle \gamma \rangle = (g_0, g_1, g_2) : X \rightarrow Y^3 = \Pi(\Delta(Y))$ is defined

in terms of the tripling of the cone as ternary product cone. The power set is abstract.

```

(iff:function power)
(= (iff:source power) type.set:set)
(= (iff:target power) type.set:set)
(forall ((type.set:set ?Y))
  (= (power ?Y) (type.lim.prd3.obj:product (type.dgm.trp.obj:constant ?Y))))

(iff:function projection-function-triple)
(= (iff:source projection-function-triple) type.set:set)
(= (iff:target projection-function-triple) type.dgm.trp.mor:function-triple)
(forall ((type.set:set ?Y))
  (and (= (type.dgm.trp.mor:source (projection-function-triple ?Y))
          (type.dgm.trp.obj:constant (power ?Y)))
        (= (type.dgm.trp.mor:target (projection-function-triple ?Y))
          (type.dgm.trp.obj:constant ?Y))
        (= (projection-function-triple ?Y)
          (type.lim.prd3.obj:projection-function-triple (type.dgm.trp.obj:constant ?Y)))))

(iff.ftn:function projection0)
(= (iff.ftn:source projection0) type.set:set)
(= (iff.ftn:target projection0) type.ftn:function)
(forall ((type.set:set ?Y))
  (and (= (type.ftn:source (projection0 ?Y)) (power ?Y))
        (= (type.ftn:target (projection0 ?Y)) ?Y)
        (= (projection0 ?Y) (type.dgm.trp.mor:function0 (projection-function-triple ?Y)))))

(iff.ftn:function projection1)
(= (iff.ftn:source projection1) type.set:set)
(= (iff.ftn:target projection1) type.ftn:function)
(forall ((type.set:set ?Y))
  (and (= (type.ftn:source (projection1 ?Y)) (power ?Y))
        (= (type.ftn:target (projection1 ?Y)) ?Y)
        (= (projection1 ?Y) (type.dgm.trp.mor:function1 (projection-function-triple ?Y)))))

(iff.ftn:function projection2)
(= (iff.ftn:source projection2) type.set:set)
(= (iff.ftn:target projection2) type.ftn:function)
(forall ((type.set:set ?Y))
  (and (= (type.ftn:source (projection2 ?Y)) (power ?Y))
        (= (type.ftn:target (projection2 ?Y)) ?Y)
        (= (projection2 ?Y) (type.dgm.trp.mor:function2 (projection-function-triple ?Y)))))

(iff:function tripling)
(= (iff:source tripling) cone)
(= (iff:target tripling) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (tripling ?c)) (set ?c))
        (= (type.ftn:target (tripling ?c)) (power (cone-set ?c)))
        (= (tripling ?c) (type.lim.prd3.obj:tripling ?c))))

```

Morphisms.

type.lim.pwr3.mor

Powers. The ternary power operation is extended to morphisms. For any type function $f : X \rightarrow Y$, the ternary *power* type function $f^3 : X^3 \rightarrow Y^3$ is defined in terms of the ternary product function for the function triple $\Delta(f)$. Given any function $f : X \rightarrow Y$, there is a ternary power function $f^3 : X^3 \rightarrow Y^3$ defined using projections: $\pi_X \cdot \Delta(f) = \Delta(f^3) \cdot \pi_Y$.

```
(iff:function power)
(= (iff:source power) type.ftn:function)
(= (iff:target power) type.ftn:function)
(forall ((type.ftn:function ?f) (type.set:set ?X) (type.set:set ?Y)
  (= (type.ftn:source ?f) ?X) (= (type.ftn:target ?f) ?Y))
  (and (= (type.ftn:source (power ?f)) (type.lim.pwr3.obj:power ?X))
    (= (type.ftn:target (power ?f)) (type.lim.pwr3.obj:power ?Y))
    (= (power ?f) (type.lim.prd3.mor:product (type.dgm.trp.mor:constant ?f)))
    (= (type.dgm.trp.mor:composition
      [(type.lim.pwr3.obj:projection-function-triple ?X)
       (type.dgm.trp.mor:delta ?f)])
      (type.dgm.trp.mor:composition
        [(type.dgm.trp.mor:delta (power ?f))
         (type.lim.pwr3.obj:projection-function-triple ?Y)]))))))
```

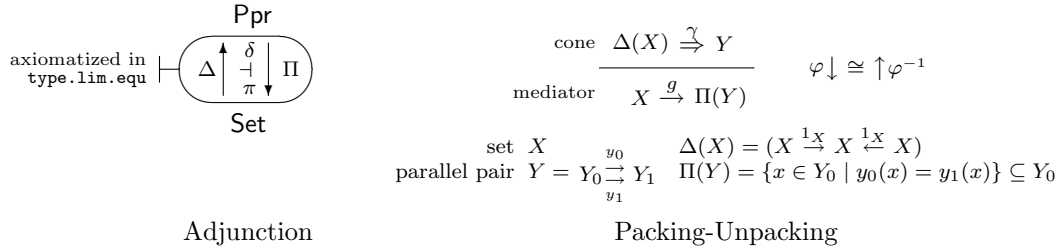


Figure 22: Equalizer

1.8.6 Type Equalizers

`type.lim.equ`

The essential properties of the equalizer construction (see Figure 14.=) are illustrated in Figure 22. The horizontal bar designates a natural bijection between the cone above the bar and the mediator below the bar. Let X be a set with constant parallel pair of functions $\Delta(X) = 1_X, 1_X : X \rightarrow X$, and let $Y = y_0, y_1 : Y_0 \rightarrow Y_1$ be a parallel pair of functions with equalizer set $\Pi(Y)$ and injection $\iota_Y = \pi_{Y_0} : \Pi(Y) \hookrightarrow Y_0$ (the 0th component of the projection parallel pair morphism)²¹. A natural packing process (φ) assigns a mediator $X \rightarrow \Pi(Y)$ below the bar to every cone $\Delta(X) \Rightarrow Y$ above the bar. A natural unpacking process (φ^{-1}) assigns a cone $\Delta(X) \Rightarrow Y$ above the bar to every mediator $X \rightarrow \Pi(Y)$ below the bar. These two processes are inverse to each other. The packing process is realized by application of the equalizer (Π) operation and then composition on the left with the delta bijective function (a component of the unit δ)

$$g = X \xrightarrow{\delta_X} \Pi(\Delta(X)) \xrightarrow{\Pi(\gamma)} \Pi(Y).$$

The unpacking process is realized by application of the constant (Δ) operation and then composition on the right with the projection parallel pair morphism (a component of the counit π)

$$\gamma = \Delta(X) \xrightarrow{\Delta(g)} \Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y.$$

Objects.

`type.lim.equ.obj`

²¹The canonical equalizer has the equalizer set $\{y_0 = y_1\} = \{x \in Y_0 \mid y_0(x) = y_1(x)\} \subseteq Y_0$ and inclusion injection $\{y_0 = y_1\} \hookrightarrow Y_0$. The equalizer type set $\Pi(Y)$ is abstract, and is not necessarily equal, but only isomorphic, to the canonical equalizer $\Pi(Y) \cong \{y_0 = y_1\}$. Hence, the equalizer set is axiomatically underspecified. When the canonical equalizer is needed at some point in a lower or more peripheral level, then additional axiomatization will be given there.

Cones. An equalizer *cone* is an object in the comma category

$$\text{Set} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \text{Ppr}$$

over the functorial ospan $\Delta : \text{Set} \rightarrow \text{Ppr} \leftarrow \text{Ppr} : 1_{\text{Ppr}}$. More specifically, a cone is a triple (X, Y, γ) consisting of a vertex set X , a parallel pair Y , and a parallel pair morphism $\gamma = (g_0, g_1) : \Delta(X) \rightarrow Y$. By definition of such a morphism, the first component function $g_1 : X \rightarrow Y_1$ is redundant – it is the composition of the zeroth component function $g_0 : X \rightarrow Y_0$ with either function component (y_0 or y_1) of the target parallel pair. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{cone}} &= \{(X, Y, \gamma) \mid X \in \text{set}, Y \in \text{ppr}, \gamma \in \text{ppr-mor}, \partial_0(\gamma) = \Delta(X), \partial_1(\gamma) = Y\}, \text{ or} \\ \text{cone} &= \{(X, \gamma) \mid X \in \text{set}, \gamma \in \text{ppr-mor}, \Delta(X) = \partial_0(\gamma)\} \subseteq \text{set} \times \text{ppr-mor}. \end{aligned}$$

The map $\widehat{\text{cone}} \rightarrow \text{cone}$ is just projection; the map $\text{cone} \rightarrow \widehat{\text{cone}}$ is defined by $(X, \gamma) \mapsto (X, \partial_1(\gamma), \gamma)$. Cones are packed in the natural isomorphism axiomatization of the constant-pullback adjunction (expressing universality of the pullback operation). We name the projections. The cones that are axiomatized here are concrete. They can be referenced as follows.

```
(type.set:set ?X)
(type.dgm.ppr.mor:parallel-pair-morphism ?g01)
(= (type.dgm.ppr.mor:source ?g01) (type.dgm.ppr.obj:constant ?X))

(type.ftn:function ?g)
(= ?g (type.lim.equ.obj:parting [?X ?g01]))
```

Here is the axiomatization.

```
(iff:set cone)
(forall ((cone ?c))
  (exists ((type.set:set ?X) (type.dgm.ppr.mor:parallel-pair-morphism ?g))
    (= ?c [?X ?g])))
(forall ((type.set:set ?X) (type.dgm.ppr.mor:parallel-pair-morphism ?g))
  (<=> (cone [?X ?g])
    (= (type.dgm.ppr.mor:source ?g) (type.dgm.ppr.obj:constant ?X))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((type.set:set ?X) (type.dgm.ppr.mor:parallel-pair-morphism ?g) (cone [?X ?g]))
  (= (set [?X ?g]) ?X))

(iff:function parallel-pair-morphism)
(= (iff:source parallel-pair-morphism) cone)
(= (iff:target parallel-pair-morphism) type.dgm.ppr.mor:parallel-pair-morphism)
(forall ((type.set:set ?X) (type.dgm.ppr.mor:parallel-pair-morphism ?g) (cone [?X ?g]))
  (= (parallel-pair-morphism [?X ?g]) ?g))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c))
```

```

    (= (function0 ?c) (type.dgm.ppr.mor:function0 (parallel-pair-morphism ?c)))

(forall ((cone ?c))
  (= (type.dgm.ppr.mor:source (parallel-pair-morphism ?c))
     (type.dgm.ppr.obj:constant (set ?c))))

(iff:function cone-parallel-pair)
(= (iff:source cone-parallel-pair) cone)
(= (iff:target cone-parallel-pair) type.dgm.ppr.obj:parallel-pair)
(forall ((cone ?c))
  (= (cone-parallel-pair ?c) (type.dgm.ppr.mor:target (parallel-pair-morphism ?c))))

```

Mediators. An equalizer *mediator* is an object in the comma category

$$\text{Set} \xleftarrow{\pi_0} (1 \downarrow \Pi) \xrightarrow{\pi_1} \text{Ppr}$$

over the functorial ospan $1_{\text{Set}} : \text{Set} \rightarrow \text{Set} \leftarrow \text{Ppr} : \Pi$. More specifically, a mediator is a triple (X, Y, g) consisting of a set X , a parallel pair $Y = y_0, y_1 : Y_0 \rightarrow Y_1$, and a function $g : X \rightarrow \Pi(Y)$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{mdtr}} &= \{(X, Y, g) \mid X \in \text{set}, Y \in \text{ppr}, g \in \text{ftn}, \partial_0(g) = X, \partial_1(g) = \Pi(Y)\}, \text{ or} \\ \text{mdtr} &= \{(g, Y) \mid g \in \text{ftn}, Y \in \text{ppr}, \partial_1(g) = \Pi(Y)\} \subseteq \text{ftn} \times \text{ppr}. \end{aligned}$$

The map $\widehat{\text{mdtr}} \rightarrow \text{mdtr}$ is just projection; the map $\text{mdtr} \rightarrow \widehat{\text{mdtr}}$ is defined by $(g, Y) \mapsto (\partial_0(g), Y, g)$. Mediators are unpacked in the natural isomorphism axiomatization of the constant-pullback adjunction (expressing universality of the pullback operation). We define the parts of a mediator. Although the pullbacks used to define mediators are abstract, the mediators themselves are concrete in the sense that they can be referenced as follows.

```

(type.ftn:function ?g)
(type.dgm.ppr.obj:parallel-pair ?Y)
(= (type.ftn:target ?g) (type.lim.equ.obj:equalizer ?Y))

(type.ftn:function ?g01)
(= ?g01 (type.lim.equ.obj:injection [?g ?Y]))

```

Here is the axiomatization.

```

(iff:set mediator)
(forall ((mediator ?m))
  (exists ((type.ftn:function ?g) (type.dgm.ppr.obj:parallel-pair ?Y))
    (= ?m [?g ?Y])))
(forall ((type.ftn:function ?g) (type.dgm.ppr.obj:parallel-pair ?Y))
  (<=> (mediator [?g ?Y])
       (= (type.ftn:target ?g) (equalizer ?Y))))

(iff:function function)
(= (iff.ftn:source function) mediator)
(= (iff.ftn:target function) type.ftn:function)
(forall ((type.ftn:function ?g) (type.dgm.ppr.obj:parallel-pair ?Y) (mediator [?g ?Y]))
  (= (function [?g ?Y]) ?g))

```

```

(iff:function parallel-pair)
(= (iff:source parallel-pair) mediator)
(= (iff:target parallel-pair) type.dgm.ppr.obj:parallel-pair)
(forall ((type.ftn:function ?g) (type.dgm.ppr.obj:parallel-pair ?Y) (mediator [?g ?Y]))
  (= (parallel-pair [?g ?Y]) ?Y))

(forall ((mediator ?m))
  (= (type.ftn:target (function ?m)) (pullback (parallel-pair ?m))))

(iff:function mediator-set)
(= (iff:source mediator-set) mediator)
(= (iff:target mediator-set) type.set:set)
(forall ((mediator ?m))
  (= (mediator-set ?m) (type.ftn:source (function ?m))))

```

Delta Unit. Given any type set X , the delta map constructs the type mediator (Figure 23) with function $\delta_X : X \rightarrow \Pi(\Delta(X)) \cong \{1_X, 1_X\} = X$, which is the packing of the cone whose parallel pair morphism $1 : \Delta(X) \rightarrow \Delta(X)$ is the identity on the constant parallel pair over X (or the constant parallel pair morphism over the identity function on X) – the delta mediator is the packing of the constant identity. This delta function is a bijection. For any set X , the delta function δ_X is the X^{th} -component of the delta natural transformation $\delta : 1_{\text{Set}} \Rightarrow \Delta \circ \Pi$, the unit of the constant-pullback adjunction $\Delta \dashv \Pi$. Naturality means that for any function $f : X \rightarrow Y$ we have the commuting diagram $\delta_X \cdot \Pi(\Delta(f)) = f \cdot \delta_Y$. Delta naturality is a special case of packing naturality.

```

(iff:function delta-cone)
(= (iff:source delta-cone) type.set:set)
(= (iff:target delta-cone) cone)
(forall ((type.set:set ?X))
  (and (= (set (delta-cone ?X)) ?X)
    (= (parallel-pair-morphism (delta-cone ?X))
      (type.dgm.ppr.mor:identity (type.dgm.ppr.obj:constant ?X))))))

(iff:function delta)
(= (iff:source delta) type.set:set)
(= (iff:target delta) mediator)
(forall ((type.set:set ?X))
  (= (delta ?X) (packing (delta-cone ?X))))

(iff.ftn:function delta-function)
(= (iff:source delta-function) type.set:set)
(= (iff:target delta-function) type.ftn:function)
(forall ((type.set:set ?X))
  (and (= (type.ftn:source (delta-function ?X)) ?X)
    (= (type.ftn:target (delta-function ?X))
      (type.lim.equ.obj:equalizer (type.dgm.ppr.obj:constant ?X)))
    (= (delta-function ?X) (function (delta ?X)))))

(forall ((type.ftn:function ?f))
  (= (type.ftn:composition
    [(delta-function (type.ftn:source ?f))
     (type.lim.equ.mor:equalizer (type.dgm.ppr.mor:constant ?f))])
    (type.ftn:composition [?f (delta-function (type.ftn:target ?f))])))

```

Projection Counit. Given any type parallel pair $Y = y_0, y_1 : Y_0 \rightarrow Y_1$, the projection map constructs the type projection cone (Figure 23) with parallel-pair morphism $\pi_Y = (\pi_{Y_0}, \pi_{Y_1}) : \Delta(\Pi(Y)) \rightarrow Y$ (with $\pi_{Y_1} = \pi_{Y_0} \cdot y_0 = \pi_{Y_0} \cdot y_1$), which is the unpacking of the mediator whose function $1 : \Pi(Y) \rightarrow \Pi(Y)$ is the identity on the equalizer of the parallel-pair Y (or the equalizer function over the identity parallel pair morphism on Y) – the projection cone is the unpacking of the equalizer identity. This projection map is a bijection. For any parallel-pair $Y = y_0, y_1 : Y_0 \rightarrow Y_1$, the projection parallel pair morphism $\pi_Y : \Delta(\Pi(Y)) \rightarrow Y$ is the Y^{th} -component of the projection natural transformation $\pi : \Pi \circ \Delta \Rightarrow \mathbf{1}_{\mathbf{Ppr}}$, the counit of the constant-equalizer adjunction $\Delta \dashv \Pi$. Naturality means that for any parallel-pair morphism $f : X \rightarrow Y$ we have the commuting diagram $\pi_X \cdot \hat{f} = \Delta(\Pi(f)) \cdot \pi_Y$. In the morphism namespace, the definition of the equalizer of parallel pair morphisms follows this. Projection naturality is a special case of unpacking naturality.²²

```
(iff:function projection-mediator)
(= (iff:source projection-mediator) type.dgm.ppr.obj:parallel-pair)
(= (iff:target projection-mediator) mediator)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (and (= (function (projection-mediator ?Y)) (type.ftn:identity (equalizer ?Y)))
    (= (parallel-pair (projection-mediator ?Y)) ?Y)))

(iff:function projection)
(= (iff:source projection) type.dgm.ppr.obj:parallel-pair)
(= (iff:target projection) cone)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (= (projection ?Y) (unpacking (projection-mediator ?Y))))

(iff.ftn:function equalizer)
(= (iff:source equalizer) type.dgm.ppr.obj:parallel-pair)
(= (iff:target equalizer) type.set:set)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (= (equalizer ?Y) (set (projection ?Y))))

(iff.ftn:function projection-parallel-pair-morphism)
(= (iff:source projection-parallel-pair-morphism) type.dgm.ppr.obj:parallel-pair)
(= (iff:target projection-parallel-pair-morphism) type.dgm.ppr.mor:parallel-pair-morphism)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (and (= (type.dgm.ppr.mor:source (projection-parallel-pair-morphism ?Y))
    (type.dgm.ppr.obj:constant (equalizer ?Y)))
    (= (type.dgm.ppr.mor:target (projection-parallel-pair-morphism ?Y)) ?Y)
    (= (projection-parallel-pair-morphism ?Y) (parallel-pair-morphism (projection ?Y)))))

(iff.ftn:function projection-function-pair)
```

²²Here we define *equalizer*, *projection* and *mediating* maps

$$\begin{aligned} \Pi &: \text{ppr} \rightarrow \text{set} \\ \pi &: \text{ppr} \rightarrow \text{ppr-mor} \\ \langle - \rangle &: \text{cone} \rightarrow \text{ftn} \end{aligned}$$

which satisfy the following. For any parallel pair Y , the structure $(\Pi(Y), \pi)$, consisting of *equalizer* set $\Pi(Y)$ and *projection* parallel pair morphism $\pi : \Delta(\Pi(Y)) \rightarrow Y$, is a universal arrow from the constant functor $\Delta : \text{Set} \rightarrow \text{Ppr}$ to the parallel pair Y . In more detail, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a limiting cone such that to every cone $x : \Delta(X) \rightarrow Y$ there is a unique *mediating* function $\langle x \rangle : X \rightarrow \Pi(Y)$ satisfying $\Delta(\langle x \rangle) \cdot \pi = x$. More concisely, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a terminal object in the slice category $(\Delta \downarrow Y)$.

$$\begin{array}{ccc}
\frac{\Delta(X) \xrightarrow{1} \Delta(X)}{X \xrightarrow{\delta_X} \Pi(\Delta(X))} & & \frac{\Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y}{\Pi(Y) \xrightarrow{1} \Pi(Y)} \\
\text{set } X & & \Delta(X) = X \begin{array}{c} \xrightarrow{1_X} \\ \rightrightarrows \\ \xrightarrow{1_X} \end{array} X \\
\text{parallel pair } Y = Y_0 \begin{array}{c} \xrightarrow{y_0} \\ \rightrightarrows \\ \xrightarrow{y_1} \end{array} Y_1 & & \Pi(Y) \cong \{y_0=y_1\} \subset Y_0 \\
\text{Delta Mediator} & & \text{Projection Cone}
\end{array}$$

Figure 23: Equalizer Unit and Counit

```

(= (iff:source projection-function-pair) type.dgm.ppr.obj:parallel-pair)
(= (iff:target projection-function-pair) type.dgm.pr.mor:function-pair)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (and (= (type.dgm.pr.mor:source (projection-function-pair ?Y))
    (type.dgm.pr.obj:constant (equalizer ?Y)))
    (= (type.dgm.pr.mor:target (projection-function-pair ?Y))
    (type.dgm.ppr.obj:set-pair ?Y))
    (= (projection-function-pair ?Y)
    (type.dgm.ppr.mor:function-pair (projection-parallel-pair-morphism ?Y)))))

(iff.ftn:function projection0)
(= (iff.ftn:source projection0) type.dgm.ppr.obj:parallel-pair)
(= (iff.ftn:target projection0) type.ftn:function)
(forall ((type.dgm.ppr.obj:parallel-pair ?Y))
  (and (= (type.ftn:source (projection0 ?Y)) (equalizer ?Y))
    (= (type.ftn:target (projection0 ?Y)) (type.dgm.pr.obj:set0 ?Y))
    (= (projection0 ?Y) (type.dgm.pr.mor:function0 (projection-function-pair ?Y)))))

(forall ((type.dgm.ppr.mor:parallel-pair-morphism ?f))
  (= (type.dgm.ppr.mor:composition
    [(projection-parallel-pair-morphism (type.dgm.ppr.mor:source ?f)) ?f])
    (type.dgm.ppr.mor:composition
    [(type.dgm.ppr.mor:constant (type.lim.pbk.mor:equalizer ?f))
    (projection-parallel-pair-morphism (type.dgm.ppr.mor:target ?f))])))

```

Packing and Unpacking. Any cone can be packed into a mediator (Figure 22). For any cone γ with set X and parallel pair morphism $\gamma = (g_0, g_1) : \Delta(X) \Rightarrow Y$ (so that $g_1 = g_0 \cdot y_0 = g_0 \cdot y_1$), there is a mediator $g = \langle \gamma \rangle : X \rightarrow \Pi(Y)$, which restricts the zeroth component function to the equalizer part of Y_0 : $g_0 = X \xrightarrow{g} \Pi(Y) \xrightarrow{\pi_{Y_0}} Y_0$. The packing (parting) process is realized by application of the equalizer (Π) operator to the parallel pair morphism γ and then composition on the left with the delta function δ_X (a component of the unit δ):

$$g = \delta_X \cdot \Pi(\gamma) : X \rightarrow \Pi(\Delta(X)) \rightarrow \Pi(Y).$$

Any mediator can be unpacked into a cone. For any mediator g with parallel pair $Y = y_0, y_1 : Y_0 \rightarrow Y_1$ and function $g : X \rightarrow \Pi(Y) \cong \{y_0 = y_1\}$, there is a cone $\gamma = g_{01} = (g_0, g_1) : \Delta(X) \rightarrow Y$, which “separates” the mediator into

component functions. The unpacking process is realized by application of the constant (Δ) operator to the function g and then composition on the right with the parallel pair morphism π_Y (a component of the counit π):

$$\gamma = \Delta(g) \cdot \pi_Y : \Delta(X) \Rightarrow \Delta(\Pi(Y)) \Rightarrow Y.$$

These two processes are inverse to each other: $\langle \gamma \rangle_{01} = \gamma$ for each cone γ and $\langle g_{01} \rangle = g$ for each mediator g . This bijection is natural in the components for cones and mediators. This means that the diagram in Figure 27 is commutative for any function $f : X' \rightarrow X$ and any parallel pair morphism $h = (h_0, h_1) : Y = (y_0, y_1 : Y_0 \rightarrow Y_1) \rightarrow (y'_0, y'_1 : Y'_0 \rightarrow Y'_1) = Y'$.

```
(iff.ftn:function packing)
(= (iff.ftn:source packing) cone)
(= (iff.ftn:target packing) mediator)
(forall ((cone ?c))
  (and (= (function (packing ?c))
        (type.ftn:composition
          [(delta-function (set ?c))
           (type.lim.equ.mor:equalizer (parallel-pair-morphism ?c))]))
    (= (parallel-pair (packing ?c)) (cone-parallel-pair ?c))))

(iff:function parting)
(= (iff:source parting) cone)
(= (iff:target parting) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (parting ?c)) (set ?c))
        (= (type.ftn:target (parting ?c)) (type.lim.equ.obj:equalizer (cone-parallel-pair ?c)))
        (= (parting ?c) (function (packing ?c)))))

(iff.ftn:function unpacking)
(= (iff.ftn:source unpacking) mediator)
(= (iff.ftn:target unpacking) cone)
(forall ((mediator ?m))
  (and (= (set (unpacking ?m)) (mediator-set ?m))
        (= (parallel-pair-morphism (unpacking ?m))
            (type.dgm.ppr.mor:composition
              [(type.dgm.ppr.mor:constant (function ?m))
               (projection-parallel-pair-morphism (parallel-pair ?m))])))

(forall ((cone ?c))
  (= (unpacking (packing ?c)) ?c))
(forall ((mediator ?m))
  (= (packing (unpacking ?m)) ?m))

(forall ((cone ?c) (cone ?d) (function ?f))
  (= (target ?f) (set ?c))
    (= (source ?f) (set ?d))
    (= (parallel-pair-morphism ?d)
        (type.dgm.ppr.mor:composition
          [(type.dgm.ppr.mor:constant ?f) (parallel-pair-morphism ?c)])))
(= (parting ?d)
    (type.ftn:composition [?f (parting ?c)])))

(forall ((cone ?c) (cone ?d) (type.dgm.ppr.mor:parallel-pair-morphism ?h))
  (= (type.dgm.ppr.mor:source ?h) (cone-parallel-pair ?c))
    (= (type.dgm.ppr.mor:target ?h) (cone-parallel-pair ?d))
```

$$\begin{array}{ccc}
\begin{array}{c} X \\ f \uparrow \\ X' \end{array} & \begin{array}{c} Y \\ h \Downarrow (h_0, h_1) \\ Y' \end{array} & \begin{array}{ccc}
\begin{array}{c} (g_0, g_1) \\ \mathbf{Ppr}[\Delta(X), Y] \end{array} & \begin{array}{c} \xrightarrow{\varphi_{X,Y}} \\ \xleftarrow{\quad} \end{array} & \begin{array}{c} g \\ \mathbf{Set}[X, \Pi(Y)] \end{array} \\
\Delta(f) \cdot (-) \cdot h \downarrow & & \downarrow f \cdot (-) \cdot \Pi(h) \\
\begin{array}{c} \mathbf{Ppr}[\Delta(X'), Y'] \end{array} & \begin{array}{c} \xrightarrow{\varphi_{X',Y'}} \\ \xleftarrow{\quad} \end{array} & \begin{array}{c} \mathbf{Set}[X', \Pi(Y')] \end{array} \\
(f \cdot g_0 \cdot h_0, f \cdot g_1 \cdot h_1) & & f \cdot g \cdot \Pi(h)
\end{array}
\end{array}$$

Figure 24: Equalizer Naturality

```

(= (parallel-pair-morphism ?d)
   (type.dgm.ppr.mor:composition [(parallel-pair-morphism ?c) ?h]))
(= (parting ?d)
   (type.ftn:composition [(parting ?c) (type.lim.equ.mor:equalizer ?h)]))

```

Morphisms.

`type.lim.equ.mor`

Equalizers. The equalizer operation is extended to morphisms. Given any parallel pair morphism $\gamma = (g_0, g_1) : X \rightrightarrows Y$, there is an equalizer function $\Pi(\gamma) : \Pi(X) \rightarrow \Pi(Y)$ defined using projections: $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$.

```

(iff.ftn:function equalizer)
(= (iff.ftn:source equalizer) type.dgm.ppr.mor:parallel-pair-morphism)
(= (iff.ftn:target equalizer) type.ftn:function)
(forall ((type.dgm.ppr.mor:parallel-pair-morphism ?f))
  (and (= (type.ftn:source (equalizer ?f))
         (type.lim.equ.obj:equalizer (type.dgm.ppr.mor:source ?f)))
       (= (type.ftn:target (equalizer ?f))
         (type.lim.equ.obj:equalizer (type.dgm.ppr.mor:target ?f)))
       (= (type.dgm.ppr.mor:composition
          [(type.lim.equ.obj:projection-parallel-pair-morphism (type.dgm.ppr.mor:source ?f)) ?f])
         (type.dgm.ppr.mor:composition
          [(type.dgm.ppr.mor:delta (equalizer ?f))
           (type.lim.equ.obj:projection-parallel-pair-morphism (type.dgm.ppr.mor:target ?f)]))))))

```

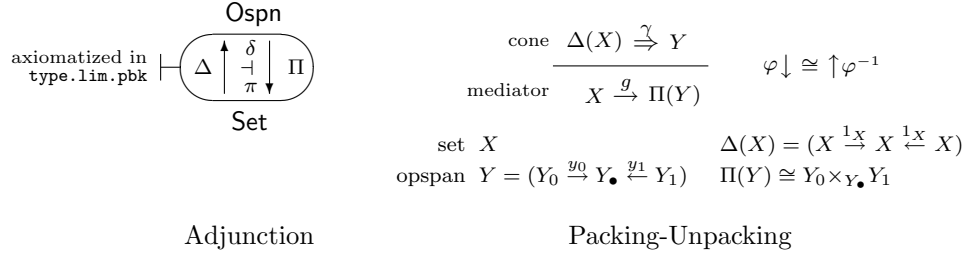


Figure 25: Pullback

1.8.7 Type Pullbacks

`type.lim.pbk`

The essential properties of the pullback construction (see Figure 14.V) are illustrated in Figure 25. The horizontal bar designates a natural bijection between the cone above the bar and the mediator below the bar. Let X be a set with constant opspan $\Delta(X) = (1_X : X \rightarrow X \leftarrow X : 1_X)$, and let $Y = (y_0 : Y_0 \rightarrow Y_\bullet \leftarrow Y_1 : y_1)$ be an opspan with pullback set $\Pi(Y)$ ²³. A natural packing (pairing) process (φ) assigns a mediator $X \rightarrow \Pi(Y)$ below the bar to every cone $\Delta(X) \Rightarrow Y$ above the bar. A natural unpacking (components) process (φ^{-1}) assigns a cone $\Delta(X) \Rightarrow Y$ above the bar to every mediator $X \rightarrow \Pi(Y)$ below the bar. These two processes are inverse to each other. The packing process is realized by application of the pullback (Π) operation and then composition on the left with the delta function (a component of the unit δ)

$$g = X \xrightarrow{\delta_X} \Pi(\Delta(X)) \xrightarrow{\Pi(\gamma)} \Pi(Y).$$

The unpacking process is realized by application of the constant (Δ) operation and then composition on the right with the projection opspan morphism (a component of the counit π)

$$\gamma = \Delta(X) \xrightarrow{\Delta(g)} \Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y.$$

Objects.

`type.lim.pbk.obj`

²³The canonical (Cartesian) pullback has pullback set $Y_0 \times_{Y_\bullet} Y_1 = \{(b_0, b_1) \mid b_0 \in Y_0, b_1 \in Y_1, y_0(b_0) = y_1(b_1)\} \subseteq Y_0 \times Y_1$ with pullback projection functions $\pi_0 : Y_0 \times_{Y_\bullet} Y_1 \rightarrow Y_0 : (b_0, b_1) \mapsto b_0$ and $\pi_1 : Y_0 \times_{Y_\bullet} Y_1 \rightarrow Y_1 : (b_0, b_1) \mapsto b_1$. The pullback type set $\Pi(Y)$ is abstract, and is not necessarily equal, but only isomorphic, to the canonical (Cartesian) pullback $\Pi(Y) \cong Y_0 \times_{Y_\bullet} Y_1$. Hence, the pullback set is axiomatically underspecified. When the canonical pullback is needed at some point in a lower or more peripheral level, then additional axiomatization will be given there. For example, the cone set and the mediator set at any level are canonical (Cartesian) pullback sets. In the meta level of the metashell and the generic level of the natural part, this fact will be explicitly axiomatized with additional axioms.

Cones. A pullback *cone* is an object in the comma category

$$\text{Set} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \text{Ospn}$$

over the functorial opspan $\Delta : \text{Set} \rightarrow \text{Ospn} \leftarrow \text{Ospn} : 1_{\text{Ospn}}$. More specifically, a cone is a triple (X, Y, γ) consisting of a vertex set X , an opspan Y , and an opspan morphism $\gamma : \Delta(X) \rightarrow Y$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{cone}} &= \{(X, Y, \gamma) \mid X \in \text{set}, Y \in \text{ospn}, \gamma \in \text{ospn-mor}, \partial_0(\gamma) = \Delta(X), \partial_1(\gamma) = Y\}, \text{ or} \\ \text{cone} &= \{(X, \gamma) \mid X \in \text{set}, \gamma \in \text{ospn-mor}, \Delta(X) = \partial_0(\gamma)\} \subseteq \text{set} \times \text{ospn-mor}. \end{aligned}$$

The map $\widehat{\text{cone}} \rightarrow \text{cone}$ is just projection; the map $\text{cone} \rightarrow \widehat{\text{cone}}$ is defined by $(X, \gamma) \mapsto (X, \partial_1(\gamma), \gamma)$. Cones are packed in the natural isomorphism axiomatization of the constant-pullback adjunction (expressing universality of the pullback operation). We name the projections. The cones that are axiomatized here are concrete. They can be referenced as follows.

```
(type.set:set ?X)
(type.dgm.ospn.mor:opspan-morphism ?g01)
(= (type.dgm.ospn.mor:source ?g01) (type.dgm.ospn.obj:constant ?X))

(type.ftn:function ?g)
(= ?g (type.lim.pbk.obj:pairing [?X ?g01]))
```

Here is the axiomatization.

```
(iff:set cone)
(forall ((cone ?c)
  (exists ((type.set:set ?X) (type.dgm.ospn.mor:opspan-morphism ?g)
    (= ?c [?X ?g])))
  (forall ((type.set:set ?X) (type.dgm.ospn.mor:opspan-morphism ?g)
    (<=> (cone [?X ?g])
      (= (type.dgm.ospn.mor:source ?g) (type.dgm.ospn.obj:constant ?X))))))

(iff:function set)
(= (iff:source set) cone)
(= (iff:target set) type.set:set)
(forall ((type.set:set ?X) (type.dgm.ospn.mor:opspan-morphism ?g) (cone [?X ?g]))
  (= (set [?X ?g]) ?X))

(iff:function opspan-morphism)
(= (iff:source opspan-morphism) cone)
(= (iff:target opspan-morphism) type.dgm.ospn.mor:opspan-morphism)
(forall ((type.set:set ?X) (type.dgm.ospn.mor:opspan-morphism ?g) (cone [?X ?g]))
  (= (opspan-morphism [?X ?g]) ?g))

(iff:function function0)
(= (iff:source function0) cone)
(= (iff:target function0) type.ftn:function)
(forall ((cone ?c)
  (= (function0 ?c) (type.dgm.ospn.mor:function0 (opspan-morphism ?c))))))

(iff:function function1)
(= (iff:source function1) cone)
```

```

(= (iff:target function1) type.ftn:function)
(forall ((cone ?c))
  (= (function1 ?c) (type.dgm.ospn.mor:function1 (opspan-morphism ?c))))

(forall ((cone ?c))
  (= (type.dgm.ospn.mor:source (opspan-morphism ?c))
    (type.dgm.ospn.obj:constant (set ?c))))

(iff:function cone-opspan)
(= (iff:source cone-opspan) cone)
(= (iff:target cone-opspan) type.dgm.ospn.obj:opspan)
(forall ((cone ?c))
  (= (cone-opspan ?c) (type.dgm.ospn.mor:target (opspan-morphism ?c))))

```

For any cone $\Delta(X) \xrightarrow{\gamma} Y$ over ospan Y , there is an opposite cone $\Delta(X) \xrightarrow{\gamma^{\text{op}}} Y^{\text{op}}$ over the opposite ospan Y^{op} .

```

(iff.ftn:function opposite)
(= (iff.ftn:source opposite) cone)
(= (iff.ftn:target opposite) cone)
(forall ((cone ?c))
  (and (= (set (opposite ?c)) (set ?c))
        (= (opspan-morphism (opposite ?c)) (type.dgm.ospn.mor:opposite (opspan-morphism ?c)))
        (= (cone-opspan (opposite ?c)) (type.dgm.ospn.obj:opposite (cone-opspan ?c)))))

```

Mediators. A pullback *mediator* is an object in the comma category

$$\text{Set} \xrightarrow{\pi_0} (1 \downarrow \Pi) \xrightarrow{\pi_1} \text{Ospn}$$

over the functorial ospan $1_{\text{Set}} : \text{Set} \rightarrow \text{Set} \leftarrow \text{Ospn} : \Pi$. More specifically, a mediator is a triple (X, Y, g) consisting of a set X , an ospan $Y = (y_0 : Y_0 \rightarrow Y_\bullet \leftarrow Y_1 : y_1)$, and a function $g : X \rightarrow \Pi(Y) \cong Y_0 \times_{Y_\bullet} Y_1$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned} \widehat{\text{mdtr}} &= \{(X, Y, g) \mid X \in \text{set}, Y \in \text{ospn}, g \in \text{ftn}, \partial_0(g) = X, \partial_1(g) = \Pi(Y)\}, \text{ or} \\ \text{mdtr} &= \{(g, Y) \mid g \in \text{ftn}, Y \in \text{ospn}, \partial_1(g) = \Pi(Y)\} \subseteq \text{ftn} \times \text{ospn}. \end{aligned}$$

The map $\widehat{\text{mdtr}} \rightarrow \text{mdtr}$ is just projection; the map $\text{mdtr} \rightarrow \widehat{\text{mdtr}}$ is defined by $(g, Y) \mapsto (\partial_0(g), Y, g)$. Mediators are unpacked in the natural isomorphism axiomatization of the constant-pullback adjunction (expressing universality of the pullback operation). We define the parts of a mediator. Although the pullbacks used to define mediators are abstract, the mediators themselves are concrete in the sense that they can be referenced as follows.

```

(type.ftn:function ?g)
(type.dgm.ospn.obj:opspan ?Y)
(= (type.ftn:target ?g) (type.lim.pbk.obj:pullback ?Y))

(type.dgm.pr.mor:function-pair ?g01)
(= ?g01 (type.lim.pbk.obj:components [?g ?Y]))

```

Here is the axiomatization.

```

(iff:set mediator)
(forall ((mediator ?m))
  (exists ((type.ftn:function ?g) (type.dgm.ospn.obj:opspan ?Y))
    (= ?m [?g ?Y])))
(forall ((type.ftn:function ?g) (type.dgm.ospn.obj:opspan ?Y))
  (<=> (mediator [?g ?Y])
    (= (type.ftn:target ?g) (pullback ?Y))))

(iff:function function)
(= (iff.ftn:source function) mediator)
(= (iff.ftn:target function) type.ftn:function)
(forall ((type.ftn:function ?g) (type.dgm.ospn.obj:opspan ?Y) (mediator [?g ?Y]))
  (= (function [?g ?Y]) ?g))

(iff:function opspan)
(= (iff:source opspan) mediator)
(= (iff:target opspan) type.dgm.ospn.obj:opspan)
(forall ((type.ftn:function ?g) (type.dgm.ospn.obj:opspan ?Y) (mediator [?g ?Y]))
  (= (opspan [?g ?Y]) ?Y))

(forall ((mediator ?m))
  (= (type.ftn:target (function ?m)) (pullback (opspan ?m))))

(iff:function mediator-set)
(= (iff:source mediator-set) mediator)
(= (iff:target mediator-set) type.set:set)
(forall ((mediator ?m))
  (= (mediator-set ?m) (type.ftn:source (function ?m))))

```

Delta Unit. Given any type set X , the delta map constructs the type mediator (Figure 26) with function $\delta_X : X \rightarrow \Pi(\Delta(X)) \cong X \times_X X$, which is the packing of the cone whose opspan morphism $1 : \Delta(X) \rightarrow \Delta(X)$ is the identity on the constant opspan over X (or the constant opspan morphism over the identity function on X) – the delta mediator is the packing of the constant identity. This delta function is a bijection. For any set X , the delta function δ_X is the X^{th} -component of the delta natural transformation $\delta : 1_{\mathbf{Set}} \Rightarrow \Delta \circ \Pi$, the unit of the constant-pullback adjunction $\Delta \dashv \Pi$. Naturality means that for any function $f : X \rightarrow Y$ we have the commuting diagram $\delta_X \cdot \Pi(\Delta(f)) = f \cdot \delta_Y$. Delta naturality is a special case of packing naturality.

```

(iff:function delta-cone)
(= (iff:source delta-cone) type.set:set)
(= (iff:target delta-cone) cone)
(forall ((type.set:set ?X))
  (and (= (set (delta-cone ?X)) ?X)
    (= (opspan-morphism (delta-cone ?X))
      (type.dgm.ospn.mor:identity (type.dgm.ospn.obj:constant ?X)))))

(iff:function delta)
(= (iff:source delta) type.set:set)
(= (iff:target delta) mediator)
(forall ((type.set:set ?X))
  (= (delta ?X) (packing (delta-cone ?X))))

(iff.ftn:function delta-function)
(= (iff:source delta-function) type.set:set)
(= (iff:target delta-function) type.ftn:function)

```

```

(forall ((type.set:set ?X))
  (and (= (type.ftn:source (delta-function ?X)) ?X)
        (= (type.ftn:target (delta-function ?X))
            (type.lim.pbk.obj:pullback (type.dgm.ospn.obj:constant ?X)))
        (= (delta-function ?X) (function (delta ?X)))))

(forall ((type.ftn:function ?f))
  (= (type.ftn:composition
      [(delta-function (type.ftn:source ?f))
       (type.lim.pbk.mor:pullback (type.dgm.ospn.mor:constant ?f))])
     (type.ftn:composition [?f (delta-function (type.ftn:target ?f))])))

```

Projection Counit. Given any type opspan $Y = (Y_0 \xrightarrow{y_0} Y_\bullet \xleftarrow{y_1} Y_1)$, the projection map constructs the type projection cone (Figure 26) with opspan morphism $\pi_Y = ((\pi_{Y_0}, \pi_{Y_1}), \pi_{Y_\bullet}) : \Delta(\Pi(Y)) \rightarrow Y$ (where $\pi_{Y_\bullet} = \pi_{Y_0} \cdot y_0 = \pi_{Y_1} \cdot y_1$), which is the unpacking of the mediator whose function $1 : \Pi(Y) \rightarrow \Pi(Y)$ is the identity on the pullback of the opspan Y (or the pullback function over the identity opspan morphism on Y) – the projection cone is the unpacking of the pullback identity. This projection map is a bijection. For any opspan $Y = (Y_0 \xrightarrow{x_0} Y_\bullet \xleftarrow{x_1} Y_1)$, the projection opspan morphism $\pi_Y : \Delta(\Pi(Y)) \rightarrow Y$ is the Y^{th} -component of the projection natural transformation $\pi : \Pi \circ \Delta \Rightarrow \mathbf{1}_{\mathbf{Ospn}}$, the counit of the constant-pullback adjunction $\Delta \dashv \Pi$. Naturality means that for any opspan morphism $f : X \rightarrow Y$ we have the commuting diagram $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$. In the morphism namespace, the definition of the pullback of opspan morphisms follows this. Projection naturality is a special case of unpacking naturality.²⁴

```

(iff:function projection-mediator)
(= (iff:source projection-mediator) type.dgm.ospn.obj:opspan)
(= (iff:target projection-mediator) mediator)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (function (projection-mediator ?Y)) (type.ftn:identity (pullback ?Y)))
        (= (opspan (projection-mediator ?Y)) ?Y)))

(iff:function projection)
(= (iff:source projection) type.dgm.ospn.obj:opspan)
(= (iff:target projection) cone)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (= (projection ?Y) (unpacking (projection-mediator ?Y))))

(iff.ftn:function pullback)
(= (iff:source pullback) type.dgm.ospn.obj:opspan)
(= (iff:target pullback) type.set:set)

```

²⁴Here we define *pullback*, *projection* and *mediating* maps

$$\begin{aligned}
\Pi &: \text{ospn} \rightarrow \text{set} \\
\pi &: \text{ospn} \rightarrow \text{ospn-mor} \\
\langle - \rangle &: \text{cone} \rightarrow \text{ftn}
\end{aligned}$$

which satisfy the following. For any opspan Y , the structure $(\Pi(Y), \pi)$, consisting of *pullback* set $\Pi(Y)$ and *projection* opspan morphism $\pi : \Delta(\Pi(Y)) \rightarrow Y$, is a universal arrow from the constant functor $\Delta : \mathbf{Set} \rightarrow \mathbf{Ospn}$ to the opspan Y . In more detail, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a limiting cone such that to every cone $x : \Delta(X) \rightarrow Y$ there is a unique *mediating* function $\langle x \rangle : X \rightarrow \Pi(Y)$ satisfying $\Delta(\langle x \rangle) \cdot \pi = x$. More concisely, $\pi : \Delta(\Pi(Y)) \rightarrow Y$ is a terminal object in the slice category $(\Delta \downarrow Y)$.

```

(forall ((type.dgm.ospn.obj:opspan ?Y))
  (= (pullback ?Y) (set (projection ?Y))))

(iff.ftn:function projection-opspan-morphism)
(= (iff:source projection-opspan-morphism) type.dgm.ospn.obj:opspan)
(= (iff:target projection-opspan-morphism) type.dgm.ospn.mor:opspan-morphism)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (type.dgm.ospn.mor:source (projection-opspan-morphism ?Y))
    (type.dgm.ospn.obj:constant (pullback ?Y)))
    (= (type.dgm.ospn.mor:target (projection-opspan-morphism ?Y)) ?Y)
    (= (projection-opspan-morphism ?Y) (opspan-morphism (projection ?Y)))))

(iff.ftn:function projection-function-pair)
(= (iff:source projection-function-pair) type.dgm.ospn.obj:opspan)
(= (iff:target projection-function-pair) type.dgm.pr.mor:function-pair)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (type.dgm.pr.mor:source (projection-function-pair ?Y))
    (type.dgm.pr.obj:constant (pullback ?Y)))
    (= (type.dgm.pr.mor:target (projection-function-pair ?Y))
    (type.dgm.ospn.obj:set-pair ?Y))
    (= (projection-function-pair ?Y)
    (type.dgm.ospn.mor:function-pair (projection-opspan-morphism ?Y)))))

(iff.ftn:function projection0)
(= (iff.ftn:source projection0) type.dgm.ospn.obj:opspan)
(= (iff.ftn:target projection0) type.ftn:function)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (type.ftn:source (projection0 ?Y)) (pullback ?Y))
    (= (type.ftn:target (projection0 ?Y)) (type.dgm.ospn.obj:set0 ?Y))
    (= (projection0 ?Y) (type.dgm.pr.mor:function0 (projection-function-pair ?Y)))))

(iff.ftn:function projection1)
(= (iff.ftn:source projection1) type.dgm.ospn.obj:opspan)
(= (iff.ftn:target projection1) type.ftn:function)
(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (type.ftn:source (projection1 ?Y)) (pullback ?Y))
    (= (type.ftn:target (projection1 ?Y)) (type.dgm.ospn.obj:set1 ?Y))
    (= (projection1 ?Y) (type.dgm.pr.mor:function1 (projection-function-pair ?Y)))))

(forall ((type.dgm.ospn.mor:opspan-morphism ?f))
  (= (type.dgm.ospn.mor:composition
    [(projection-opspan-morphism (type.dgm.ospn.mor:source ?f)) ?f])
    (type.dgm.ospn.mor:composition
    [(type.dgm.ospn.mor:constant (type.lim.pbk.mor:pullback ?f))
    (projection-opspan-morphism (type.dgm.ospn.mor:target ?f))])))

```

For any opspan Y with associated set pair $\text{pr}(Y)$ and parallel pair $\text{ppr}(Y)$, the equalizer of $\text{ppr}(Y)$ is the pullback set $\Pi(Y) = \Pi(\text{ppr}(Y))$, and the function pair composition of the constant of the equalizer injection of $\text{ppr}(Y)$ with the binary product projection of $\text{pr}(Y)$ is the pullback projection $\pi_Y = \Delta(\iota_{\text{ppr}(Y)}) \cdot \pi_{\text{pr}(Y)} : \Delta(\Pi(Y)) = \Delta(\Pi(\text{ppr}(Y))) \rightarrow \Delta(\Pi(\text{pr}(Y))) \rightarrow Y$.

```

(forall ((type.dgm.ospn.obj:opspan ?Y))
  (and (= (pullback ?Y)
    (type.lim.equ.obj:equalizer (type.dgm.ospn.obj:parallel-pair ?Y)))
    (= (projection-function-pair ?Y)
    (type.dgm.pr.mor:composition

```

$$\begin{array}{ccc}
\frac{\Delta(X) \xrightarrow{1} \Delta(X)}{X \xrightarrow{\delta_X} \Pi(\Delta(X))} & & \frac{\Delta(\Pi(Y)) \xrightarrow{\pi_Y} Y}{\Pi(Y) \xrightarrow{1} \Pi(Y)} \\
\text{set } X & & \Delta(X) = (X \xrightarrow{1_X} X \xleftarrow{1_X} X) \\
\text{opspan } Y = (Y_0 \xrightarrow{y_0} Y_\bullet \xleftarrow{y_1} Y_1) & & \Pi(Y) \cong Y_0 \times_{Y_\bullet} Y_1 \\
\text{Delta Mediator} & & \text{Projection Cone}
\end{array}$$

Figure 26: Pullback Unit and Counit

```

[(type.dgm.pr.mor:constant (type.dgm.pr.mor:function0
  (type.lim.equ.obj:projection-function-pair (type.dgm.ospn.obj:parallel-pair ?Y))))
 (type.lim.prd2.obj:projection-function-pair (type.dgm.ospn.obj:set-pair ?Y))]]))

```

For any opspan $X = (x_0 : X_0 \rightarrow X_\bullet \leftarrow X_1 : x_1)$, there is a *tau* or *twist* cone

$(\pi_0 : X_0 \leftarrow \Pi(X^{\text{op}}) \rightarrow X_1 : \pi_1)$ whose opspan morphism $\Delta(\Pi(X^{\text{op}})) \xrightarrow{\pi_{X^{\text{op}}}} X$ is the opposite of the projection opspan morphism of X^{op} .

```

(iff.ftn:function tau-cone)
(= (iff.ftn:source tau-cone) type.dgm.ospn.obj:opspan)
(= (iff.ftn:target tau-cone) cone)
(forall ((type.dgm.ospn.obj:opspan ?X))
  (and (= (set (tau-cone ?X)) (pullback (type.dgm.ospn.obj:opposite ?X)))
    (= (opspan-morphism (tau-cone ?X))
      (type.dgm.ospn.mor:opposite (projection-opspan-morphism (type.dgm.ospn.obj:opposite ?X)))))
    (= (cone-opspan (tau-cone ?X)) ?X)))

```

For convenience, we name the injection $\iota : \Pi(Y) \hookrightarrow \Pi(Y_0, Y_1)$ from the pullback to the binary product of the set pair of Y . The pullback projections are the composition of this injection with the binary product projections of the set pair of Y , $\pi_Y = \Delta(\iota) \cdot \pi_{(Y_0, Y_1)}$ (or $\pi_{Y_i} = \iota \cdot \pi_{Y_i}^{Y_0, Y_1}$ for $i = 0, 1$).

```

(iff:function injection-cone)
(= (iff:source injection-cone) type.dgm.ospn.obj:opspan)
(= (iff:target injection-cone) type.lim.prd2.obj:cone)
(forall ((type.dgm.ospn.obj:opspan ?X))
  (and (= (type.lim.prd2.obj:set (injection-cone ?X)) (pullback ?X))
    (= (type.lim.prd2.obj:function-pair (injection-cone ?X))
      (projection-function-pair ?X))))

(iff:function injection)
(= (iff:source injection) type.dgm.ospn.obj:opspan)
(= (iff:target injection) type.ftn:function)
(forall ((type.dgm.ospn.obj:opspan ?X))
  (and (= (type.ftn:source (injection ?X)) (pullback ?X))
    (= (type.ftn:target (injection ?X))
      (type.lim.prd2.obj:product (type.dgm.ospn.obj:set-pair ?X)))
    (= (injection ?X) (type.lim.prd2.obj:pairing (injection-cone ?X)))))

(forall ((type.dgm.ospn.obj:opspan ?X))
  (= (projection-function-pair ?X)
    (type.dgm.pr.mor:composition

```

```
[(type.dgm.pr.mor:constant (injection ?X))
 (type.lim.prd2.obj:projection-function-pair (type.dgm.ospn.obj:set-pair ?X)))]))
```

Packing and Unpacking. Any cone can be packed into a mediator (Figure 25). For any cone γ with set X and opspan morphism $\gamma = ((g_0, g_1), g_\bullet) : \Delta(X) \Rightarrow Y$ (so that $g_\bullet = g_0 \cdot y_0 = g_1 \cdot y_1$), there is a mediator $g = \langle \gamma \rangle : X \rightarrow \Pi(Y)$, which combines the images of the component functions g_0 and g_1 (by factoring the product pairing $\langle g_0, g_1 \rangle : X \rightarrow \Pi(Y_0, Y_1)$ through the pullback). The packing (pairing) process is realized by application of the pullback (Π) operator to the opspan morphism γ and then composition on the left with the delta function δ_X (a component of the unit δ):

$$g = \delta_X \cdot \Pi(\gamma) : X \rightarrow \Pi(\Delta(X)) \rightarrow \Pi(Y).$$

Any mediator can be unpacked into a cone. For any mediator g with opspan $Y = (y_0 : Y_0 \rightarrow Y_\bullet \leftarrow Y_1 : y_1)$ and function $g : X \rightarrow \Pi(Y) \cong Y_0 \times_{Y_\bullet} Y_1$, there is a cone $\gamma = g_{01\bullet} = ((g_0, g_1), g_\bullet) : \Delta(X) \rightarrow Y$, which separates the mediator into component functions. The unpacking (components) process is realized by application of the constant (Δ) operator to the function g and then composition on the right with the opspan morphism π_Y (a component of the counit π):

$$\gamma = \Delta(g) \cdot \pi_Y : \Delta(X) \Rightarrow \Delta(\Pi(Y)) \Rightarrow Y.$$

These two processes are inverse to each other: $\langle \gamma \rangle_{01\bullet} = \gamma$ for each cone γ and $\langle g_{01\bullet} \rangle = g$ for each mediator g . This bijection is natural in the components for cones and mediators. This means that the diagram in Figure 27 is commutative for any function $f : X' \rightarrow X$ and any opspan morphism $h = ((h_0, h_1), h_\bullet) : Y = (y_0 : Y_0 \rightarrow Y_\bullet \leftarrow Y_1 : y_1) \rightarrow (y'_0 : Y'_0 \rightarrow Y'_\bullet \leftarrow Y'_1 : y'_1) = Y'$.

```
(iff.ftn:function packing)
(= (iff.ftn:source packing) cone)
(= (iff.ftn:target packing) mediator)
(forall ((cone ?c))
  (and (= (function (packing ?c))
          (type.ftn:composition
            [(delta-function (set ?c))
             (type.lim.pbk.mor:pullback (opspan-morphism ?c))]))
        (= (opspan (packing ?c)) (cone-opspan ?c))))

(iff:function pairing)
(= (iff:source pairing) cone)
(= (iff:target pairing) type.ftn:function)
(forall ((cone ?c))
  (and (= (type.ftn:source (pairing ?c)) (set ?c))
        (= (type.ftn:target (pairing ?c))
            (type.lim.pbk.obj:pullback (cone-opspan ?c)))
        (= (pairing ?c) (function (packing ?c)))))

(iff.ftn:function unpacking)
(= (iff.ftn:source unpacking) mediator)
(= (iff.ftn:target unpacking) cone)
(forall ((mediator ?m))
  (and (= (set (unpacking ?m)) (mediator-set ?m))
```

$$\begin{array}{ccc}
X & & Y \\
f \uparrow & & h \downarrow \\
X' & & Y'
\end{array}
\quad
\begin{array}{ccc}
& & \begin{array}{ccc}
((g_0, g_1), g_\bullet) & \varphi_{X,Y} & g \\
\text{Ospan}[\Delta(X), Y] & \xrightarrow{\quad} & \text{Set}[X, Y_0 \times_{Y_\bullet} Y_1] \\
& \downarrow \Delta(f) \cdot (-) \cdot h & \downarrow \varphi_{X',Y'} \\
& \text{Ospan}[\Delta(X'), Y'] & \xrightarrow{\quad} \text{Set}[X', Y'_0 \times_{Y'_\bullet} Y'_1] \\
& \downarrow ((f \cdot g_0 \cdot h_0, f \cdot g_1 \cdot h_1), f \cdot g_\bullet \cdot h_0) & \downarrow f \cdot (-) \cdot \Pi(h) \\
& & f \cdot g \cdot \Pi(h)
\end{array}
\end{array}$$

Figure 27: Pullback Naturality

```

(= (opspan-morphism (unpacking ?m))
  (type.dgm.ospn.mor:composition
    [(type.dgm.ospn.mor:constant (function ?m))
     (projection-opspan-morphism (opspan ?m))]))

(iff:function components)
(= (iff:source components) mediator)
(= (iff:target components) type.dgm.ospn.mor:opspan-morphism)
(forall ((mediator ?m))
  (and (= (type.dgm.ospn.mor:source (components ?m))
          (type.dgm.ospn.obj:constant (mediator-set ?m)))
        (= (type.dgm.ospn.mor:target (components ?m)) (opspan ?m))
        (= (components ?m) (opspan-morphism (unpacking ?m)))))

(forall ((cone ?c))
  (= (unpacking (packing ?c)) ?c))
(forall ((mediator ?m))
  (= (packing (unpacking ?m)) ?m))

(forall ((cone ?c) (cone ?d) (function ?f)
          (= (target ?f) (set ?c))
          (= (source ?f) (set ?d))
          (= (opspan-morphism ?d)
            (type.dgm.ospn.mor:composition
              [(type.dgm.ospn.mor:constant ?f) (opspan-morphism ?c)])))
  (= (pairing ?d)
     (type.ftn:composition [?f (pairing ?c)])))

(forall ((cone ?c) (cone ?d) (type.dgm.ospn.mor:opspan-morphism ?h)
          (= (type.dgm.ospn.mor:source ?h) (cone-opspan ?c))
          (= (type.dgm.ospn.mor:target ?h) (cone-opspan ?d))
          (= (opspan-morphism ?d)
            (type.dgm.ospn.mor:composition [(opspan-morphism ?c) ?h])))
  (= (pairing ?d)
     (type.ftn:composition [(pairing ?c) (type.lim.pbk.mor:pullback ?h)])))

```

For any opspan $X = (x_0 : X_0 \rightarrow X_\bullet \leftarrow X_1 : x_1)$, there is a *tau* or *twist* bijection $\tau_X : \Pi(X^{\text{op}}) \xrightarrow{\cong} \Pi(X)$ from the pullback of the opposite opspan $X^{\text{op}} = (x_1 : X_1 \rightarrow X_\bullet \leftarrow X_0 : x_0)$ to the pullback of X . This is the pullback pairing of the tau cone $(\pi_0 : X_0 \leftarrow \Pi(X^{\text{op}}) \rightarrow X_1 : \pi_1)$ over the opspan X .

```

(iff.ftn:function tau)
(= (iff.ftn:source tau) type.dgm.ospn.obj:opspan)
(= (iff.ftn:target tau) type.ftn:function)
(forall ((type.dgm.ospn.obj:opspan ?X)
          (and (= (type.ftn:source (tau ?X)) (type.lim.pbk.obj:pullback (type.dgm.ospn.obj:opposite ?X))
                (= (type.ftn:target (tau ?X)) type.ftn:function))))

```

```

(= (type.ftn:target (tau ?X)) (type.lim.pbk.obj:pullback ?X))
(= (tau ?X) (pairing (tau-cone ?X)))
(type.ftn:bijection (tau ?X)))

```

Morphisms.

`type.lim.pbk.mor`

Pullbacks. The pullback operation is extended to morphisms. Given any opspan morphism $f = ((f_0, f_1), f_\bullet) : X = (X_0 \xrightarrow{x_0} X_\bullet \xleftarrow{x_1} X_1) \rightarrow (Y_0 \xrightarrow{y_0} Y_\bullet \xleftarrow{y_1} Y_1) = Y$, there is a pullback function $\Pi(f) : \Pi(X) \rightarrow \Pi(Y)$ defined using projections: $\pi_X \cdot f = \Delta(\Pi(f)) \cdot \pi_Y$ (that is, $\pi_{X_0} \cdot f_0 = \Pi(f) \cdot \pi_{Y_0}$ and $\pi_{X_1} \cdot f_1 = \Pi(f) \cdot \pi_{Y_1}$).

```

(iff:function pullback)
(= (iff:source pullback) type.dgm.ospn.mor:opspan-morphism)
(= (iff:target pullback) type.ftn:function)
(forall ((type.dgm.ospn.mor:opspan-morphism ?f))
  (and (= (type.ftn:source (pullback ?f))
    (type.lim.pbk.obj:pullback (type.dgm.ospn.mor:source ?f)))
    (= (type.ftn:target (pullback ?f))
    (type.lim.pbk.obj:pullback (type.dgm.ospn.mor:target ?f)))
    (= (type.dgm.ospn.mor:composition
      [(type.lim.pbk.obj:projection-opspan-morphism (type.dgm.ospn.mor:source ?f)) ?f])
    (type.dgm.ospn.mor:composition
      [(type.dgm.ospn.mor:delta (pullback ?f))
        (type.lim.pbk.obj:projection-opspan-morphism (type.dgm.ospn.mor:target ?f)]))))))

```

Our two standard references for the IFF are the books: *Sets for Mathematics* (2003) by F. William Lawvere and Robert Rosebrugh [1] and *Categories for the Working Mathematician* (1971) by Saunders Mac Lane [2].

References

- [1] F. William Lawvere and Robert Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.
- [2] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.