# The IFF Meta Namespace

Robert E. Kent

December 21, 2007

## Contents

**1 The Meta Kernel** . . . **2**
- 1.1 Introduction . . . 2
- 1.2 Category Theory . . . 9
- 1.3 Meta Sets . . . 12
- 1.4 Meta Functions . . . 24
  - 1.4.1 Function Morphisms . . . 40

## 1.4 Meta Functions

**Basics.** There is a type set of all meta *functions*. Any meta function is itself a type function; that is, the type set of all meta functions is an (implicit) subset of the IFF set of all type functions.

```
(type.set:set function)
(forall ((function ?f)) (type.ftn:function ?f))
```

Each meta function has a unique *source* meta set and a unique *target* meta set. The source (target) map for meta functions is an (implicit) restriction of the source (target) map for type functions. The (implicit) source-target pairing map for meta functions is the (implicit) optimal restriction of the (implicit) source-target pairing map for type functions[5].

```
(type.ftn:function source)
(= (type.ftn:source source) function)
(= (type.ftn:target source) meta.set:set)
(forall ((function ?f))
    (= (source ?f) (type.ftn:source ?f)))

(type.ftn:function target)
(= (type.ftn:source target) function)
(= (type.ftn:target target) meta.set:set)
(forall ((function ?f))
    (= (target ?f) (type.ftn:target ?f)))

(forall ((type.ftn:function ?f)
        (meta.set:set (type.ftn:source ?f))
        (meta.set:set (type.ftn:target ?f)))
    (function ?f))
```

There is a binary *restriction* relation $\sqsubseteq$ between pairs of meta functions that are linked by source and target inclusions. One (*smaller*) meta function $f_1 : X_1 \rightarrow Y_1$ is a restriction of another (*larger*) meta function $f_2 : X_2 \rightarrow Y_2$, $f_1 \sqsubseteq f_2$, when (1) the source (target) of $f_1$ is a subset of the source (target) of $f_2$, $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$, and (2) the functions agree (on source elements of $f_1$), $f_1(x_1) = f_2(x_1)$ for all elements $x_1 \in X_1$; that is, the functions commute with the source/target inclusions. This implies that the source meta set of $f_1$ is a subset of the inverse image along $f_2$ of the target meta set of $f_1$: $X_1 \subseteq f_2^{-1}(Y_1)$. We name the components of a restriction relationship.

$$
\begin{array}{ccc}
X_1 & \xrightarrow{f_1} & Y_1 \\
\big\downarrow & & \big\downarrow \\
X_2 & \xrightarrow[f_2]{} & Y_2
\end{array}
$$

From one point of view, restriction can be viewed as a constraint on the larger meta function — it says that the larger function maps the source meta set of the smaller function into the target meta set of the smaller function. From another

[5] We assume a special axiom throughout the IFF: for any set-theoretically large (here, type) function $f : X \rightarrow Y$, if the source $X$ and target $Y$ are both set-theoretically small (meta) sets, then the function $f$ is a set-theoretically small (meta) function.

point of view, restriction defines the smaller function; that is, if we assume that the definition of the larger function is given (in some other axiomatization), then asserting the restriction relationship effectively defines the smaller function. This is the point of view taken when we use (only) restriction to define a particular function at one metalevel in terms of the corresponding function at a higher metalevel. The meta level restriction relation is the (implicit) abridgement of the type level restriction relation. The meta level smaller and larger component functions are restrictions of their type level counterparts, with the larger map being optimal[6]. The restriction relation is a partial order, since the subset relation is a partial order.

```
(type.rel:endorelation restriction)
(= (type.rel:set restriction) function)
(type.set:subset-relation [(type.rel:extent restriction) (type.lim.pwr2.obj:power function)])
(forall ((function ?f1) (function ?f2))
    (<=> (restriction ?f1 ?f2) (type.ftn:restriction ?f1 ?f2)))
(type.rel:partial-order restriction)

(type.ftn:function smaller)
(= (type.ftn:source smaller) (type.rel:extent restriction))
(= (type.ftn:target smaller) function)
(forall ((function ?f1) (function ?f2) (restriction ?f1 ?f2))
    (= (smaller [?f1 ?f2]) (type.ftn:smaller [?f1 ?f2])))

(type.ftn:function larger)
(= (type.ftn:source larger) (type.rel:extent restriction))
(= (type.ftn:target larger) function)
(forall ((function ?f1) (function ?f2) (restriction ?f1 ?f2))
    (= (larger [?f1 ?f2]) (type.ftn:larger [?f1 ?f2])))
(forall ((type.ftn:function ?f1) (type.ftn:function ?f2) (type.ftn:restriction ?f1 ?f2))
    (=> (function (type.ftn:larger [?f1 ?f2]))
        (function ?f1)))
```

There is a binary *optimal restriction* relation $\dot{\sqsubseteq}$ between pairs of meta functions. Optimal restriction is a subrelation of restriction. A restriction between two meta functions $f_1$ and $f_2$ is optimal, $f_1 \dot{\sqsubseteq} f_2$, when the source meta set of the smaller function $f_1$ is exactly the inverse image of the target meta set of $f_1$ along the larger function $f_2$: $X_1 = f_2^{-1}(Y_1)$. Optimal restriction is a pullback notion. We name the components of an optimal restriction relationship.

$$\begin{array}{ccc} X_1 & \xrightarrow{f_1} & Y_1 \\ \big\downarrow & & \big\downarrow \\ X_2 & \xrightarrow[f_2]{} & Y_2 \end{array}$$

The meta level optimal restriction relation is the (implicit) abridgement of the type level optimal restriction relation. The meta level smaller and larger component functions are (implicit) restrictions of their type level counterparts with

[6] We can prove the special theorem throughout the IFF: for any two set-theoretically large (here, type) functions $f_1 : X_1 \rightarrow Y_1$ and $f_2 : X_2 \rightarrow Y_2$, where $f_1$ is a restriction of $f_2$, $f_1 \sqsubseteq f_2$, if the larger function $f_2$ is a set-theoretically small (meta) function, then the smaller function $f_1$ is also a set-theoretically small (meta) function.

the larger map being optimal[7]. The optimal restriction relation is also a partial order.

```
(type.rel:endorelation optimal-restriction)
(= (type.rel:set optimal-restriction) function)
(type.rel:inclusion-relation [optimal-restriction restriction])
(type.set:inclusion-set [(type.rel:set optimal-restriction) (type.rel:set restriction)])
(forall ((function ?f1) (function ?f2))
    (<=> (optimal-restriction ?f1 ?f2) (type.ftn:optimal-restriction ?f1 ?f2)))
(type.rel:partial-order optimal-restriction)

(type.ftn:function optimal-smaller)
(= (type.ftn:source optimal-smaller) (type.rel:extent optimal-restriction))
(= (type.ftn:target optimal-smaller) function)
(forall ((function ?f1) (function ?f2) (optimal-restriction-relation [?f1 ?f2]))
    (= (optimal-smaller [?f1 ?f2]) (type.ftn:optimal-smaller [?f1 ?f2])))

(type.ftn:function optimal-larger)
(= (type.ftn:source optimal-larger) (type.rel:extent optimal-restriction))
(= (type.ftn:target optimal-larger) function)
(forall ((function ?f1) (function ?f2) (optimal-restriction-relation [?f1 ?f2]))
    (= (optimal-larger [?f1 ?f2]) (type.ftn:optimal-larger [?f1 ?f2])))
(forall ((type.ftn:function ?f1) (type.ftn:function ?f2)
         (type.ftn:optimal-restriction ?f1 ?f2))
    (=> (function (type.ftn:optimal-larger [?f1 ?f2]))
        (function ?f1)))
```

Both restriction and optimal restriction are closed under composition and identities. If $f_1 : X_1 \to Y_1$ and $g_1 : Y_1 \to Z_1$ is a composable pair of meta functions, $f_2 : X_2 \to Y_2$ and $g_2 : Y_2 \to Z_2$ is a composable pair of meta functions, $f_1$ is a (the optimal-)restriction of $f_2$ and $g_1$ is a (the optimal-)restriction of $g_2$, $f_1 \sqsubseteq (\dot{\sqsubseteq})\ f_2$ and $g_1 \sqsubseteq (\dot{\sqsubseteq})\ g_2$, then the composition $f_1 \cdot g_1 : X_1 \to Z_1$ is a (the optimal-)restriction of the composition $f_2 \cdot g_2 : X_2 \to Z_2$, $f_1 \cdot g_1 \sqsubseteq (\dot{\sqsubseteq})\ f_2 \cdot g_2$. If $X_1$ is a subset of $X_2$, $X_1 \subseteq X_2$, then the identity $1_{X_1} : X_1 \to X_1$ is a (the optimal-)restriction of the identity $1_{X_2} : X_2 \to X_2$, $1_{X_1} \sqsubseteq (\dot{\sqsubseteq})\ 1_{X_2}$.

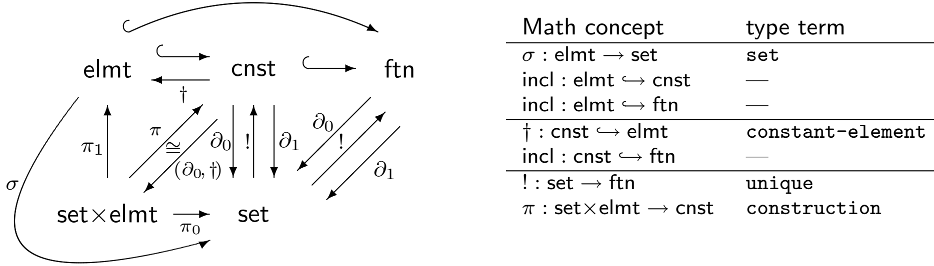
```
(forall ((function ?f1) (function ?g1) (composable ?f1 ?g1)
         (function ?f2) (function ?g2) (composable ?f2 ?g2)
        (restriction ?f1 ?f2) (restriction ?g1 ?g2))
    (restriction (composition [?f1 ?g1]) (composition [?f2 ?g2])))

(forall ((meta.set:set ?X1) (meta.set:set ?X2) (meta.set:subset ?X1 ?X2))
    (restriction (identity ?X1) (identity ?X2)))

(forall ((function ?f1) (function ?g1) (composable ?f1 ?g1)
         (function ?f2) (function ?g2) (composable ?f2 ?g2)
        (optimal-restriction ?f1 ?f2) (optimal-restriction ?g1 ?g2))
    (optimal-restriction (composition [?f1 ?g1]) (composition [?f2 ?g2])))

(forall ((meta.set:set ?X1) (meta.set:set ?X2) (meta.set:subset ?X1 ?X2))
    (optimal-restriction (identity ?X1) (identity ?X2)))
```

[7] We can prove the special theorem throughout the IFF: for any two set-theoretically large (here, type) functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$, where $f_1$ is the optimal restriction of $f_2$, $f_1 \dot{\sqsubseteq} f_2$, if the larger function $f_2$ is a set-theoretically small (meta) function, then the smaller function $f_1$ is also a set-theoretically small (meta) function.




| Math concept | type term |
|---|---|
| $\sigma$ : elmt $\rightarrow$ set | `set` |
| incl : elmt $\hookrightarrow$ cnst | — |
| incl : elmt $\hookrightarrow$ ftn | — |
| $\dagger$ : cnst $\hookrightarrow$ elmt | `constant-element` |
| incl : cnst $\hookrightarrow$ ftn | — |
| ! : set $\rightarrow$ ftn | `unique` |
| $\pi$ : set×elmt $\rightarrow$ cnst | `construction` |

Figure 1: Basic Function Kinds

Category theory represents elements-in-sets by morphisms[8]. A meta *element* $x$ in a set $X \in$ set is a meta function $1 \xrightarrow{x} X$. We use the notation $\sigma(x) = X$ or the notation $x \in X$ to denote this. There is a type set elmt of all meta elements, which is a(n implicit) subset of the type set of all functions, elmt $\subseteq$ ftn. There is a *set* function $\sigma$ : elmt $\rightarrow$ set. The set of an element is its target, $\sigma(x) = \partial_1(x)$ (outer part of Figure 1). $X$ as a set-theoretic "bag of dots" corresponds to the fiber of the inclusion function over $X$. The type set of all meta elements is the (implicit) intersection of the IFF set of all type elements and the type set of all meta functions. The meta level component set function is the (implicit) optimal restriction of the type level component set function.

```
(type.set:set element)
(type.set:subset-relation [element function])
(forall ((function ?x))
    (<=> (element ?x) (type.ftn:element ?x)))

(type.ftn:function set)
(= (type.ftn:source set) element)
(= (type.ftn:target set) meta.set:set)
(= set type.ftn:composition [(type.set:inclusion [element function]) target])
(forall ((element ?x))
    (= (set ?x) (type.ftn:set ?x)))
```

The statement that "the function $c : Y \rightarrow X$ is constant" means that there exists an element of $x \in X$ such that $c$ is the composite $c = !_Y \cdot x : Y \rightarrow 1 \rightarrow X$. We use the notation $\dagger(c) = x$ to denote the element $x$ associated with a constant function $c$. Note that there is no constraint between the set $Y$ and the element $x \in X$; they are completely independent. Conversely, given any pair $(Y, x \in X)$, consisting of a meta set $Y \in$ set and a meta element $x \in$ elmt, we can construct a constant function $!_Y \cdot x : Y \rightarrow 1 \rightarrow X$. As two special cases, any meta element $x \in$ elmt is a constant function $x : 1 \rightarrow X$, and any meta set $Y \in$ set has an associated constant function $!_Y : Y \rightarrow 1$. There is a type set cnst of all *constant* meta functions, which is a(n implicit) subset of the type set of meta functions, cnst $\subseteq$ ftn. The type set of meta elements is a(n implicit) subset of the type set of constant meta functions, elmt $\subseteq$ cnst. There is an element function $\dagger$ : cnst $\rightarrow$ elmt. The type set of constant meta functions is (implicitly) the binary product of the type set of meta sets and the type set of

[8]Note that these elements are not separate, disconnected, dissociated and isolated individuals, but have a set attached. This provides a kind of context for the element. The representation of elements as separate individuals is not needed in the natural part of the IFF. The categorical representation of elements within a context is exactly what is needed.

meta elements, $\mathsf{cnst} \cong \mathsf{set}{\times}\mathsf{elmt}$. This isomorphism is mediated by the pairing $(\partial_0, \dagger) : \mathsf{cnst} \rightarrow \mathsf{set}{\times}\mathsf{elmt}$ and the *construction* function $\pi : \mathsf{set}{\times}\mathsf{elmt} \rightarrow \mathsf{cnst}$. The type set of all meta constant functions is the (implicit) intersection of the IFF set of all type constant functions and the type set of all meta functions. The meta level constant-element function is the (implicit) restriction of the type level constant-element function. The meta level construction function is the (implicit) optimal restriction of the type level construction function.

```
(type.set:set constant-function)
(type.set:subset-relation [constant-function function])
(forall ((function ?c))
    (<=> (constant-function ?c) (type.ftn:constant-function ?c)))

(type.ftn:function constant-element)
(= (type.ftn:source constant-element) constant-function)
(= (type.ftn:target constant-element) element)
(= (type.ftn:composition [(type.set:inclusion [constant-function function]) target])
   (type.ftn:composition [constant-element set]))
(forall ((constant-function ?c))
    (= (constant-element ?c) (type.ftn:constant-element ?c)))

(type.ftn:function construction)
(= (type.ftn:source construction) (type.lim.prd2.obj:product [meta.set:set element]))
(= (type.ftn:target construction) constant-function)
(forall ((meta.set:set ?Y) (element ?x))
    (= (construction [?Y ?x]) (type.ftn:construction [?Y ?x])))
(forall ((type.set:set ?Y) (type.ftn:element ?x))
    (=> (constant-function (type.ftn:construction [?Y ?x]))
        (and (meta.set:set ?Y) (element ?x))))

(type.set:subset-relation [(type.ftn:range meta.set:unique) constant-function])
(type.set:subset-relation [element constant-function])
```

**Category Theory.** A *pair* of meta functions is *composable* when the target of the first is equal to the source of the second. The meta level composable endorelation is the (implicit) abridgement of its (implicit) type level counterpart. The extent of the meta level composable endorelation is the meta set of composable pairs of meta functions. We name the projection *factor*s of composable pairs. These component functions are (implicit) restrictions of their type level counterparts.

```
(type.rel:endorelation composable)
(= (type.rel:set composable) function)
(forall ((function ?f) (function ?g))
    (<=> (composable ?f ?g) (type.ftn:composable-pair [?f ?g])))

(type.ftn:function factor0)
(= (type.ftn:source factor0) (type.rel:extent composable))
(= (type.ftn:target factor0) function)
(forall ((function ?f) (function ?g) (composable ?f ?g))
    (= (factor0 [?f ?g]) (type.ftn:factor0 [?f ?g])))

(type.ftn:function factor1)
(= (type.ftn:source factor1) (type.rel:extent composable))
(= (type.ftn:target factor1) function)
(forall ((function ?f) (function ?g) (composable ?f ?g))
    (= (factor1 [?f ?g]) (type.ftn:factor1 [?f ?g])))
```

The *composition* of two composable meta functions $f : X \to Y$ and $g : Y \to Z$ is a meta function $f \cdot g : X \to Z$. The source of the composite is the source of the first component factor, and the target of the composite is the target of the second component factor. The composition (type) function for meta level functions is a(n implicit) restriction of the composition (IFF) function for type level functions.

```
(type.ftn:function composition)
(= (type.ftn:source composition) (type.rel:extent composable))
(= (type.ftn:target composition) function)
(forall ((function ?f) (function ?g) (composable-pair [?f ?g]))
    (= (composition [?f ?g]) (type.ftn:composition [?f ?g])))
(= (type.ftn:composition [composition source])
   (type.ftn:composition [factor0 source]))
(= (type.ftn:composition [composition target])
   (type.ftn:composition [factor1 target]))
```

Composition is associative. Any three composable meta functions $f : X \to Y$, $g : Y \to Z$ and $h : Z \to W$ satisfy the associative law $f \cdot (g \cdot h) = (f \cdot g) \cdot h$.

```
(forall ((function ?f) (function ?g) (function ?h)
        (composable ?f ?g) (composable ?g ?h))
    (= (composition [?f (composition [?g ?h])])
       (composition [(composition [?f ?g]) ?h])))
```

The composition map $\mathsf{ftn} \times_{\mathsf{set}} \mathsf{ftn} \dot{\to} \mathsf{ftn}$ is surjective (see identity below).

```
(type.ftn:surjection composition)
```

For every meta set $X$, there is a unique associated *identity* meta function $1_X : X \to X$. The meta level identity function is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function identity)
(= (type.ftn:source identity) meta.set:set)
(= (type.ftn:target identity) function)
(= (type.ftn:composition [identity source]) type.ftn:identity)
(= (type.ftn:composition [identity target]) type.ftn:identity)
(forall ((meta.set:set ?X))
    (= (identity ?X) (type.ftn:identity ?X)))
(forall ((type.set:set ?X))
    (=> (function (type.ftn:identity ?X))
        (meta.set:set ?X)))
```

Identity satisfies two unit laws with respect to composition: composition with the identity of the source (target) of a function $f : X \to Y$ returns that function: $1_X \cdot f = f = f \cdot 1_Y$.

```
(forall ((function ?f))
    (and (= (composition [(identity (source ?f)) ?f]) ?f)
         (= ?f (composition [?f (identity (target ?f))]))))
```

If we had parametric composition and delta function at the type level, and composition at the IFF level, we could express this as:

```
(= (iff:composition
      [(delta function) (type.ftn:parametric-composition [(type.ftn:composition [source identity]) type.ftn:identity])])
   type.ftn:identity)

(= (iff:composition
      [(delta function) (type.ftn:parametric-composition [type.ftn:identity (type.ftn:composition [target identity]) ])])
   type.ftn:identity)
```

For any set $X$, the identity function $1_X : X \rightarrow X$ is a bijection.

```
(forall ((meta.set:set ?X))
    (bijection (identity ?X)))
```

The identity map is injective $\mathsf{set} \overset{1}{\hookrightarrow} \mathsf{ftn}$. Hence, sets can be regarded as special functions that satisfy the unit laws.

```
(type.ftn:injection identity)
```

Application gives the definition of any function. For element $x$ and function $f$, the expression '$f(x)$' is not defined. We use application to give this meaning. A pair $(x, f)$ consisting of a meta element $x$ and a meta function $f$ is *appliable* when the set of the element is equal to the source of the function, $\sigma(x) = \partial_0(f)$. The meta level appliable relation is the (implicit) optimal restriction of its (implicit) type level counterpart. The extent of the meta level appliable relation is the meta set of appliable pairs. We name the projection *factor*s of appliable pairs. These component functions are (implicit) restrictions of their type level counterparts.

```
(type.rel:endorelation composable)
(= (type.rel:set composable) function)
(forall ((function ?f) (function ?g))
    (<=> (composable ?f ?g) (type.ftn:composable-pair [?f ?g])))

(type.ftn:function factor0)
(= (type.ftn:source factor0) (type.rel:extent composable))
(= (type.ftn:target factor0) function)
(forall ((function ?f) (function ?g) (composable ?f ?g))
    (= (factor0 [?f ?g]) (type.ftn:factor0 [?f ?g])))

(type.ftn:function factor1)
(= (type.ftn:source factor1) (type.rel:extent composable))
(= (type.ftn:target factor1) function)
(forall ((function ?f) (function ?g) (composable ?f ?g))
    (= (factor1 [?f ?g]) (type.ftn:factor1 [?f ?g])))

(type.rel:relation appliable)
(= (type.rel:set0 appliable) element)
(= (type.rel:set0 appliable) function)
(forall ((element ?x) (function ?f))
    (<=> (appliable ?x ?f) (type.ftn:appliable-pair [?x ?f])))

(type.ftn:function element-factor)
(= (type.ftn:source element-factor) (type.rel:extent appliable))
(= (type.ftn:target element-factor) element)
(forall ((element ?x) (function ?f) (appliable ?x ?f))
    (= (element-factor [?x ?f]) (type.ftn:element-factor [?x ?f])))

(type.ftn:function function-factor)
(= (type.ftn:source function-factor) (type.rel:extent appliable))
(= (type.ftn:target function-factor) function)
(forall ((element ?x) (function ?f) (appliable ?x ?f))
    (= (function-factor [?x ?f]) (type.ftn:function-factor [?x ?f])))
```

The type set of all appliable pairs is a(n implicit) subset of the type set of all composable pairs, since $\sigma(x) = \partial_0(f)$.

```
(forall ((element ?x) (function ?f) (appliable ?x ?f))
    (composable ?x ?f))
```

The *application* of an appliable pair $x \in X$ and $f : X \to Y$ is an element $(x \wr f) \in Y$. The set of the result(ing element) is the target of the function. The embedding of a result is the composition of the input element: $x \wr f = x \cdot f$ for any appliable pair $x \in X$ and $f : X \to Y$; that is, application is a restriction of composition. It is also an (implicit) restriction of its type level counterpart.

```
(type.ftn:function application)
(= (type.ftn:source application) (type.rel:extent appliable))
(= (type.ftn:target application) element)
(= (type.ftn:composition [application set])
   (type.ftn:composition [function-factor target]))
(= application
   (type.ftn:composition
       [(type.lim.pbk.mor:pullback
            [(type.ftn:inclusion [element function])
             (type.ftn:identity function)])
        composition]))
(type.ftn:restriction-relation [application composition])
(forall ((element ?x) (function ?f) (appliable ?x ?f))
    (= (application [?x ?f]) (type.ftn:application [?x ?f])))
```

There is a mixed associative law for application. Using the associative law for composition, $x \wr (f \cdot g) = (x \wr f) \wr g$ for any composable pair $f : X \to Y$ and $g : Y \to Z$ and any element $x \in X$. This corresponds to the usual pointwise definition of composition, $(f \cdot g)(x) = g(f(x))$. Hence, the associative law for composition implies the definition of application of composition.

```
(forall ((element ?x) (function ?f) (function ?g)
        (appliable ?x ?f) (composable ?f ?g))
    (= (application [?x (composition [?f ?g])])
       (application [(application [?x ?f]) ?g])))
```

There is a unit law for application. Using the unit laws for composition, $x \wr 1_X = x$ for any set $X$ and any element $x \in X$. This corresponds to the usual pointwise definition of identity, $1_X(x) = x$. Hence, the unit laws for composition imply the definition of application of identity.

```
(forall ((element ?x))
    (= (application [?x (identity (set ?x))]) ?x))
```

**Factorization.** There are special kinds of functions.

- A meta function $f : X \to Y$ is *injective* when any image value is from a unique source element; or more categorically, when for any parallel pair of meta functions (generalized elements) $x_0, x_1 : W \to X$ the equality $x_0 \cdot f = x_1 \cdot f$ implies $x_0 = x_1$ ($f$ is right-cancellable).

- Dually, a meta function $f : X \to Y$ is *surjective* when any target value is an image; or more categorically, when for any parallel pair of meta functions $y_0, y_1 : Y \to Z$ the equality $f \cdot y_0 = f \cdot y_1$ implies $y_0 = y_1$ ($f$ is left-cancellable).

- A meta function $f : X \rightarrow Y$ is *bijective* when there is a (necessarily unique *inverse*) function in the opposite direction $\hat{f} : Y \rightarrow X$ with $f \cdot \hat{f} = 1_X$ and $\hat{f} \cdot f = 1_Y$.

These are predicates (adjectives) that refer to (modify) sets (nouns), the genus, and result in subsets (noun phrases), the differentia. A meta function is an *injection* (monomorphism) when it is injective. A meta function is a *surjection* (epimorphism) when it is surjective. A meta function is a *bijection* (isomorphism) when it is bijective. The type set of all meta level injections is the differentia of the meta level injective predicate; this is the intersection of the IFF set of all type level injections and the type set of all meta level functions. The same assertions hold for surjections and bijections. The bijective predicate is the conjunction of the injective and surjective predicates; that is, a bijection is an injection that is also a surjection. There is a meta level *inverse* function on bijections that is the (implicit) optimal restriction of its type level counterpart.

```
(type.pred:predicate injective)
(type.set:set injection)
(= (type.pred:genus injective) function)
(= (type.pred:differentia injective) injection)
(forall ((function ?f))
    (<=> (injective ?f) (type.ftn:injection ?f)))

(type.pred:predicate surjective)
(type.set:set surjection)
(= (type.pred:genus surjective) function)
(= (type.pred:differentia surjective) surjection)
(forall ((function ?f))
    (<=> (surjective ?f) (type.ftn:surjection ?f)))

(type.pred:predicate bijective)
(type.set:set bijection)
(= (type.pred:genus bijective) function)
(= (type.pred:differentia bijective) bijection)
(forall ((function ?f))
    (<=> (bijective ?f) (type.ftn:bijection ?f)))
(forall ((function ?f))
    (<=> (bijective ?f)
         (and (injective ?f) (surjective ?f))))

(type.ftn:function inverse)
(= (type.ftn:source inverse) bijection)
(= (type.ftn:target inverse) bijection)
(forall ((bijection ?f))
    (= (inverse ?f) (type.ftn:inverse ?f)))
(forall ((type.ftn:bijection ?f))
    (=> (bijection (type.ftn:inverse ?f))
        (bijection ?f)))
```

Injections, surjections and bijections are closed under composition.

```
(forall ((injection ?f) (injection ?g) (composable ?f ?g))
    (injection (composition [?f ?g])))
(forall ((surjection ?f) (surjection ?g) (composable ?f ?g))
    (surjection (composition [?f ?g])))
(forall ((bijection ?f) (bijection ?g) (composable ?f ?g))
    (bijection (composition [?f ?g])))
```

For any meta function $f : X \rightarrow Y$, there is a *range* meta set $\rho_f = \mathsf{rng}(f) \subseteq Y$, defined by $\rho(f) = \{y \in Y \mid \exists_{x \in X} f(x) = y\}$. The meta level range function is an (implicit) restriction of its type level counterpart.

```
(type.ftn:function range)
(= (type.ftn:source range) function)
(= (type.ftn:target range) meta.set:set)
(forall ((function ?f))
    (and (subset (range ?f) (target ?f))
         (= (range ?f) (type.ftn:range ?f))))
```

For any meta function $f : X \rightarrow Y$, there is an *injective factor* $\iota_f = \mathsf{inj}_f : \rho(f) \rightarrow Y$, which is the inclusion of the range of $f$ into the target of $f$. The meta level injective factor function is an (implicit) restriction of its type level counterpart.

```
(type.ftn:function injective-factor)
(= (type.ftn:source injective-factor) function)
(= (type.ftn:target injective-factor) function)
(= (type.ftn:composition [injective-factor source]) range)
(= (type.ftn:composition [injective-factor target]) target)
(forall ((function ?f))
    (and (injective (injective-factor ?f))
         (= (injective-factor ?f) (type.ftn:injective-factor ?f))))
```

For any meta function $f : X \rightarrow Y$, there is a *surjective factor* $\sigma_f = \mathsf{surj}_f : X \rightarrow \rho(f)$ with the same action as $f$ (it is the target restriction of $f$ to its range). The meta level surjective factor function is an (implicit) restriction of its type level counterpart.

```
(type.ftn:function surjective-factor)
(= (type.ftn:source surjective-factor) function)
(= (type.ftn:target surjective-factor) function)
(= (type.ftn:composition [injective-factor source]) source)
(= (type.ftn:composition [injective-factor target]) range)
(forall ((function ?f))
    (and (surjective (surjective-factor ?f))
         (restriction (surjective-factor ?f) ?f)
         (= (surjective-factor ?f) (type.ftn:surjective-factor ?f))))
```

The function $f$ is the composition of its surjective factor and injective factor: $f = \sigma_f \cdot \iota_f : X \rightarrow \rho(f) \rightarrow Y$.

```
(forall ((function ?f))
    (= ?f (composition [(surjective-factor ?f) (injective-factor ?f)])))
```

The (surjective-factor, injective-factor) pair forms a surjection-injection "factorization system" for meta sets and meta functions; that is, if a meta function $f : X \rightarrow Y$ is the composition of a surjection with an injection, $f = e \cdot m : X \rightarrow Z \rightarrow Y$, then there is a (unique) "diagonal" bijection $d : \rho(f) \rightarrow Z$, such that $\sigma_f \cdot d = e$ and $d \cdot m = \iota_f$.

$$\begin{array}{ccc} X & \xrightarrow{\sigma_f} & \rho(f) \\ {\scriptstyle e}\downarrow & \swarrow{\scriptstyle d} & \downarrow{\scriptstyle \iota_f} \\ Z & \xrightarrow[m]{} & Y \end{array}$$

```
(forall ((function ?f) (surjection ?e) (injection ?m) (= ?f (composition [?e ?m]))
    (and (exists ((bijection ?d) (= (source ?d) (range ?f)) (= (target ?d) (target ?e)))
             (and (= (composition [(surjective-factor ?f) ?d]) ?e)
                  (= (composition [?d ?m]) (injective-factor ?f))))
         (forall ((bijection ?d1) (bijection ?d2)
                 (= (source ?d1) (range ?f)) (= (target ?d1) (target ?e))
                 (= (source ?d2) (range ?f)) (= (target ?d2) (target ?e))
                 (= (composition [(surjective-factor ?f) ?d1]) ?e)
                 (= (composition [(surjective-factor ?f) ?d2]) ?e)
                 (= (composition [?d1 ?m]) (injective-factor ?f))
                 (= (composition [?d2 ?m]) (injective-factor ?f)))
             (= ?d1 ?d2))))
```

Let $f : X \to Y$ be a meta function. The *kernel* of $f$ is the equivalence relation $\mathsf{ker}_f = \equiv_f$ on $X$ defined by $x_1 \equiv_f x_2$ iff $f(x_1) = f(x_2)$ for all source pairs $x_1, x_2 \in X$. The *coimage* of $f$ is the quotient of the kernel $\mathsf{coim}_f = X/{\equiv_f} = \{ [x]_{\equiv_f} \mid x \in X\}$, where $[x]_f = \{x' \in X \mid f(x') = f(x)\}$ is the equivalence class of $x$ with respect to the kernel of $f$, and the *coequalizer*[9] of $f$ is the canon of the kernel $\mathsf{coeq}_f = [\text{-}]_f = [\text{-}]_{\equiv_f} : X \to \mathsf{coim}_f$. The meta level kernel, coimage and coequalizer functions are (implicit) restrictions of their type level counterparts.

```
(type.ftn:function kernel)
(= (type.ftn:source kernel) function)
(= (type.ftn:target kernel) meta.rel:equivalence-relation)
(= (type.ftn:composition [kernel meta.rel:set]) source)
(forall ((function ?f))
    (= (kernel ?f) (type.ftn:kernel ?f)))

(type.ftn:function coimage)
(= (type.ftn:source coimage) function)
(= (type.ftn:target coimage) meta.set:set)
(= coimage (type.ftn:composition [kernel meta.rel:quotient]))
(forall ((function ?f))
    (= (coimage ?f) (type.ftn:coimage ?f)))

(type.ftn:function coequalizer)
(= (type.ftn:source coequalizer) function)
(= (type.ftn:target coequalizer) function)
(= (type.ftn:composition [coequalizer source]) source)
(= (type.ftn:composition [coequalizer target]) coimage)
(= coequalizer (type.ftn:composition [kernel meta.rel:canon]))
(forall ((function ?f))
    (and (surjective (coequalizer ?f))
         (= (coequalizer ?f) (type.ftn:coequalizer ?f))))
```

Any function $f : X \to Y$ respects its kernel $\pi_0^{\equiv_f} \cdot f = \pi_1^{\equiv_f} \cdot f$, and hence factors through its coimage $f = [\text{-}]_f \cdot \wr_f$ for some (injective) function $\mathsf{coapply}_f = \wr_f : \mathsf{coim}_f \to Y$. This factor, called the *coapplication* of $f$, is defined by $\wr_f([x]_{\equiv_f}) = f(x)$. The meta level coapplication function is an (implicit) restriction of its type level counterpart.

$$\mathsf{ext}(\equiv_f) \underset{\pi_1}{\overset{\pi_0}{\rightrightarrows}} X \xrightarrow{f} Y, \quad X \xrightarrow{[\text{-}]_f} X/{\equiv_f} \xrightarrow{\wr_f} Y$$

[9]So named because it is the coequalizer of the kernel projection pair.

```
(type.ftn:function coapplication)
(= (type.ftn:source coapplication) function)
(= (type.ftn:target coapplication) function)
(= (type.ftn:composition [coapplication source]) coimage)
(= (type.ftn:composition [coapplication target]) target)
(forall ((function ?f))
    (and (injective (coapplication ?f))
         (= ?f (composition [(coequalizer ?f) (coapplication ?f)]))
         (= (coapplication ?f) (type.ftn:coapplication ?f))))
```

The (coequalizer, coapplication) pair forms a surjection-injection "factorization system" for meta sets and meta functions; that is, if a meta function $f : X \rightarrow Y$ is the composition of a surjection with an injection, $f = e{\cdot}m : X \rightarrow Z \rightarrow Y$, then there is a (unique) "diagonal" bijection $d : \mathsf{coim}(f) \rightarrow Z$, such that $[\text{-}]_f \cdot d = e$ and $d \cdot m = \wr_f$.

$$\begin{array}{ccc} X & \xrightarrow{[\text{-}]_f} & X/{\equiv_f} \\ {\scriptstyle e}\downarrow & \swarrow{\scriptstyle d} & \downarrow{\scriptstyle \wr_f} \\ Z & \xrightarrow[m]{} & Y \end{array}$$

This implies that the coimage is naturally isomorphic to the range (image), $\mathsf{coim}_f \cong \mathsf{rng}_f$; specifically, for any source element of $x \in X$, the equivalence class $[x]_{\equiv_f} \in \mathsf{coim}_f$ corresponds to the image element $f(x) \in \mathsf{rng}_f$.

```
(forall ((function ?f))
    (meta.set:isomorphic (range ?f) (coimage ?f)))
```

**Conversion.** Any function $X \xrightarrow{f} Y$ has an associated image *predicate*, whose genus is the target of the function, whose differentia is the range of the function, and whose injection is the injective-factor of the function. The meta level predicate function is an (implicit) restriction of its type level counterpart.

```
(type.ftn:function predicate)
(= (type.ftn:source predicate) function)
(= (type.ftn:target predicate) meta.pred:predicate)
(= (type.ftn:composition [predicate meta.pred:genus]) target)
(= (type.ftn:composition [predicate meta.pred:differentia]) range)
(= (type.ftn:composition [predicate meta.pred:function]) injective-factor)
(forall ((function ?f))
    (= (predicate ?f) (type.ftn:predicate ?f)))
```

For any meta function $x : Y \xrightarrow{x} X$, there is a *unit* function 2-cell $\eta_x : x \Rightarrow \mathsf{ftn}(\mathsf{pred}(x))$ whose source is $x$, whose target $\mathsf{ftn}(\mathsf{pred}(x))$ is the function (injection) of the predicate of $x$, and whose function $\sigma_x : X \rightarrow \varepsilon(x)$ is the surjective factor of $x$ (Figure 2). The meta level unit function is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function unit)
(= (type.ftn:source unit) function)
(= (type.ftn:target unit) meta.ftn.mor:2-cell)
(= (type.ftn:composition [unit meta.ftn.mor:source]) (type.ftn:identity function))
(= (type.ftn:composition [unit meta.ftn.mor:target])
```

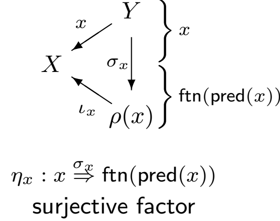



Figure 2: Unit Function 2-Cell

```
   (type.ftn:composition [predicate meta.pred:function]))
(= (type.ftn:composition [unit meta.ftn.mor:function]) surjective-factor)
(forall ((function ?f))
    (= (unit ?f) (type.ftn:unit ?f)))
(forall ((type.ftn:function ?f))
    (=> (meta.ftn.mor:2-cell (type.ftn:unit ?f))
        (function ?f)))
```

There is a function-to-*span* injection that embeds a meta function $f : X \rightarrow Y$ as a meta span $\mathsf{spn}(f) = \langle X \stackrel{1_X}{\leftarrow} X \stackrel{f}{\rightarrow} Y \rangle$. The meta level span function is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function span)
(= (type.ftn:source span) function)
(= (type.ftn:target span) meta.spn:span)
(= (type.ftn:composition [span meta.spn:function0])
   (type.ftn:composition [source identity]))
(= (type.ftn:composition [span meta.spn:function1]) (type.ftn:identity function))
(= (type.ftn:composition [span meta.spn:vertex]) source)
(forall ((function ?f))
    (= (span ?f) (type.ftn:span ?f)))
(forall ((type.ftn:function ?f))
    (=> (meta.spn:span (type.ftn:span ?f))
        (function ?f)))
```

There is a function to *relation* map that embeds a meta function $f : X \rightarrow Y$ as a total functional relation $\mathsf{rel}(f) = \hat{f} : X \rightharpoondown Y$, where $\mathsf{ext}(\mathsf{rel}(f)) = \{(x, y) \mid x \in X, y \in Y, f(x) = y\}$. The domain (codomain) of the relation is the source (target) of the function. The domain projection is an bijection, and post-composition with the original function gives the codomain projection. The relation of a function is the relation of the span of the function. The relation map $\mathsf{ftn} \rightarrow \mathsf{rel}$ is injective. The meta level relation function is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function relation)
(= (type.ftn:source relation) function)
(= (type.ftn:target relation) meta.rel:relation)
(= (type.ftn:composition [relation meta.rel:set0]) source)
(= (type.ftn:composition [relation meta.rel:set1]) target)
(= relation (type.ftn:composition [span meta.spn:relation])
(type.ftn:injective relation)
```

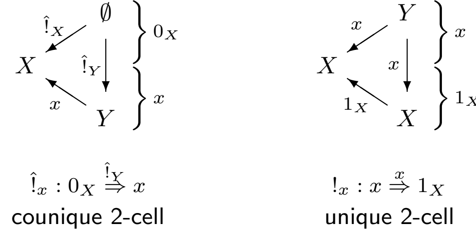



Figure 3: Counique and Unique Function 2-Cells

```
(forall ((function ?f))
    (and (meta.set:isomorphic (meta.rel:extent (relation ?f)) (source ?f))
         (bijection (meta.rel:projection0 (relation ?f)))
         (= (composition [(meta.rel:projection0 (relation ?f)) ?f])
            (meta.rel:projection1 (relation ?f)))
         (= (relation ?f) (type.ftn:relation ?f))))
(forall ((type.ftn:function ?f))
    (=> (meta.rel:relation (type.ftn:relation ?f))
        (function ?f)))
```

For any meta function $f : X \rightarrow Y$, there is a *fiber* meta function $\mathsf{fbr}(f) : Y \rightarrow \wp X$, defined as

$$\mathsf{fbr}(f)(y) \;=\; \{x \in X \mid f(x) = y\}$$

for any target element $y \in Y$. The fiber function can be defined in terms of target singleton and inverse image, $\mathsf{fbr}(f) = \{\text{-}\}_Y \cdot f^{-1} : Y \rightarrow \wp Y \rightarrow \wp X$. The fiber function of $f : X \rightarrow Y$ is the 10-fiber of the relation $\hat{f} : X \rightharpoondown Y$. The meta level fiber function is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function fiber)
(= (type.ftn:source fiber) function)
(= (type.ftn:target fiber) function)
(= (type.ftn:composition [fiber source]) target)
(= (type.ftn:composition [fiber target]) (type.ftn:composition [source meta.set:power]))
(= fiber (type.ftn:composition [relation meta.rel:fiber01]))
(forall ((function ?f))
    (and (= (fiber ?f)
            (composition [(meta.set:singleton (target ?f)) (meta.set:inverse-image ?f)]))
         (= (fiber ?f) (type.ftn:fiber ?f))))
(forall ((type.ftn:function ?f))
    (=> (function (type.ftn:fiber ?f))
        (function ?f)))
```

**Instances.** For any set $X$, the empty set and counique function form an *initial* (predicative) function $0_X = \emptyset \xrightarrow{\hat{!}_X} X$, and the identity function forms a *terminal* (predicative) function $1_X = X \xrightarrow{1_X} X$. The meta level initial and terminal functions are the (implicit) optimal restriction of their type level counterparts.

```
(type.ftn:function initial)
```

```
(= (type.ftn:source initial) meta.set:set)
(= (type.ftn:target initial) function)
(= (type.ftn:composition [initial source]) meta.set:constant-zero)
(= (type.ftn:composition [initial target]) (type.ftn:identity meta.set:set))
(= initial meta.set:counique)
(forall ((meta.set:set ?X))
    (= (initial ?X) (type.ftn:initial ?X)))
(forall ((type.set:set ?X))
    (=> (function (type.ftn:initial ?X))
        (meta.set:set ?X)))

(type.ftn:function terminal)
(= (type.ftn:source terminal) meta.set:set)
(= (type.ftn:target terminal) function)
(= (type.ftn:composition [terminal source]) (type.ftn:identity meta.set:set))
(= (type.ftn:composition [terminal target]) (type.ftn:identity meta.set:set))
(= terminal identity)
(forall ((meta.set:set ?X))
    (= (terminal ?X) (type.ftn:terminal ?X)))
(forall ((type.set:set ?X))
    (=> (function (type.ftn:terminal ?X))
        (meta.set:set ?X)))
```

For any meta function $x = (Y \xrightarrow{x} X)$, there is a *counique* meta function 2-cell $\hat{!}_x : 0_X \Rightarrow x$ and a *unique* meta function 2-cell $!_x : x \Rightarrow 1_X$ (Figure 3). These are the unique function 2-cells between their respective sources and targets. The meta level counique and unique functions are the (implicit) optimal restriction of their type level counterparts.

```
(type.ftn:function counique)
(= (type.ftn:source counique) function)
(= (type.ftn:target counique) meta.ftn.mor:2-cell)
(= (type.ftn:composition [counique meta.ftn.mor:source])
   (type.ftn:composition [target initial]))
(= (type.ftn:composition [counique meta.ftn.mor:target]) (type.ftn:identity function))
(= (type.ftn:composition [counique meta.ftn.mor:function])
   (type.ftn:composition [target meta.set:counique]))
(forall ((function ?x))
    (and (= (counique ?f) (type.ftn:counique ?f))
         (forall ((meta.ftn.mor:2-cell ?a)
                 (= (meta.ftn.mor:source ?a) (initial (target ?x)))
                 (= (meta.ftn.mor:target ?a) ?x))
             (= ?a (counique ?x)))))
(forall ((type.ftn:function ?f))
    (=> (meta.ftn.mor:2-cell (type.ftn:counique ?f))
        (function ?f)))

(type.ftn:function unique)
(= (type.ftn:source unique) function)
(= (type.ftn:target unique) meta.ftn.mor:2-cell)
(= (type.ftn:composition [unique meta.ftn.mor:source]) (type.ftn:identity function))
(= (type.ftn:composition [unique meta.ftn.mor:target])
   (type.ftn:composition [target terminal]))
(= (type.ftn:composition [unique meta.ftn.mor:function]) (type.ftn:identity function))
(forall ((function ?x))
    (and (= (unique ?f) (type.ftn:unique ?f))
         (forall ((meta.ftn.mor:2-cell ?a)
                 (= (meta.ftn.mor:source ?a) ?x)
```

```
                (= (meta.ftn.mor:target ?a) (terminal (target ?x)))
            (= ?a (unique ?x)))))
(forall ((type.ftn:function ?f))
    (=> (meta.ftn.mor:2-cell (type.ftn:unique ?f))
        (function ?f)))
```

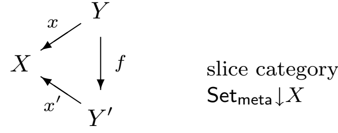



Figure 4: Function 2-Cell

### 1.4.1 Function Morphisms

**Basics.** Using the abbreviation $\mathsf{Set} = \mathsf{Set}_{\mathsf{meta}}$, a meta function *morphism* is a morphism in the comma category[10]

$$\mathsf{Set} \xleftarrow{\pi_0} (1 \downarrow 1) \xrightarrow{\pi_1} \mathsf{Set}$$

over the functorial ospan $1 : \mathsf{Set} \to \mathsf{Set} \leftarrow \mathsf{Set} : 1$. More specifically, a meta function morphism, from source function $x = (Y, X, x)$ to target function $x' = (Y', X', x')$, is a pair $(f, g)$ consisting of a function $f : Y \to Y'$ between source sets and a function $g : X \to X'$ between target sets, which satisfy the identity $f \cdot x' = x \cdot g$.

However, we do not need full generality here. Instead we restrict the definition by requiring the function $g$ to be identity. This defines the $\pi_1$-fiber over meta set $X$, or alternatively the the comma category[11]

$$\mathsf{Set} \xleftarrow{\pi_0} (1 \downarrow X) \xrightarrow{\pi_1} 1$$

over the functorial ospan $1 : \mathsf{Set} \to \mathsf{Set} \leftarrow 1 : X$. More specifically, a meta function *2-cell* (a morphism in the slice category $\mathsf{Set}/X$) $\alpha : x \overset{f}{\Rightarrow} x'$, with source function $Y \xrightarrow{x} X$ and target function $Y' \xrightarrow{x'} X$, has a meta function $f : Y \to Y'$ that commutes with source and target functions: $f \cdot x' = x$ (Figure 4); that is, a function 2-cell is a triple $\alpha = (x, f, x')$, where $(f, x')$ are a composable pair whose composition is $x = f \cdot x'$. Clearly, the source and target functions have a common target set. The meta level source, target and function functions are the (implicit) restrictions of their type level counterparts.

```
(type.set:set 2-cell)

(type.ftn:function source)
(= (type.ftn:source source) 2-cell)
(= (type.ftn:target source) meta.ftn:function)
(forall ((2-cell ?a))
    (= (source ?a) (type.ftn.mor:source ?a)))

(type.ftn:function target)
(= (type.ftn:source target) 2-cell)
(= (type.ftn:target target) meta.ftn:function)
(forall ((2-cell ?a))
    (= (target ?a) (type.ftn.mor:target ?a)))
```

[10] The comma category $(1 \downarrow 1) = \mathsf{Set}^{\to}$ is called the arrow category of $\mathsf{Set}$.

[11] The comma category $(1 \downarrow X) = \mathsf{Set}/X$ is called the slice category of $\mathsf{Set}$ over $X$.

```
(type.ftn:function function)
(= (type.ftn:source function) 2-cell)
(= (type.ftn:target function) meta.ftn:function)
(forall ((2-cell ?a))
    (= (function ?a) (type.ftn.mor:function ?a)))

(forall ((2-cell ?a))
    (and (meta.ftn:composable-pair [(function ?a) (target ?a)])
         (= (source ?a) (meta.ftn:composition [(function ?a) (target ?a)]))))
```

For any two functions (generalized elements) $x, x' \in \mathsf{ftn}_{\mathsf{meta}}$, $x$ *belongs to* $x'$, $x \sqsubseteq x'$, when there is a function 2-cell $\alpha : x \overset{p}{\Rightarrow} x'$; that is, when there exists a *proof* function[12] $p \in \mathsf{ftn}_{\mathsf{meta}}$ such that $x = p \cdot x'$. We name the component *element*s of a belonging relationship. When $x'$ is an injection, the proof $p$ is unique. The meta level belongs relation is the (implicit) abridgement of the (implicit) type level belongs relation. The meta level element component functions are restrictions of their type level counterparts. The belongs relation is a preorder, since 2-cells are closed under identities and composites.

```
(type.rel:endorelation belongs)
(= (type.rel:set belongs) meta.ftn:function)
(forall ((meta.ftn:function ?x) (meta.ftn:function ?xp))
    (<=> (belongs ?x ?xp) (type.ftn.mor:belonging [?x ?xp])))
(type.rel:preorder belongs)

(type.ftn:function element0)
(= (type.ftn:source element0) (type.rel:extent belongs))
(= (type.ftn:target element0) meta.ftn:function)
(forall ((meta.ftn:function ?x) (meta.ftn:function ?xp) (belongs ?x ?xp]))
    (= (element0 [?x ?xp]) (type.ftn.mor:element0 [?x ?xp])))

(type.ftn:function element1)
(= (type.ftn:source element1) (type.rel:extent belongs))
(= (type.ftn:target element1) meta.ftn:function)
(forall ((meta.ftn:function ?x) (meta.ftn:function ?xp) (belongs ?x ?xp]))
    (= (element1 [?x ?xp]) (type.ftn.mor:element1 [?x ?xp])))

(forall ((meta.ftn:function ?x) (meta.ftn:injection ?xp) (belongs ?x ?xp))
    (forall ((2-cell ?a1) (2-cell ?a2))
        (=> (and (= ?x (source ?a1)) (= ?xp (target ?a1)))
                 (= ?x (source ?a2)) (= ?xp (target ?a2))))
            (= ?a1 ?a2))))
```

Two functions $x, x' \in \mathsf{ftn}_{\mathsf{meta}}$ are *equivalent*, $x \equiv x'$, when each belongs to the other, $x \sqsubseteq x'$ and $x' \sqsubseteq x$. Two equivalent functions are *isomorphic*, $x \cong x'$, when there exists a bijection $p \in \mathsf{ftn}_{\mathsf{meta}}$ proving belonging. The meta level equivalent and isomorphic relations are the (implicit) abridgement of their (implicit) type level counterparts. The equivalent and isomorphic endorelations are equivalence relations (reflexive, symmetric and transitive).

```
(type.rel:endorelation equivalent)
(= (type.rel:set equivalent) meta.ftn:function)
(forall ((meta.ftn:function ?x) (meta.ftn:function ?xp))
    (and (<=> (equivalent ?x ?xp) (and (belongs ?x ?xp) (belongs ?xp ?x)))
         (<=> (equivalent ?x ?xp) (type.ftn.mor:equivalence [?x ?xp]))))
(type.rel:equivalence-relation equivalent)
```

[12] $p$ proves that $x$ belongs to $x'$. In general, there may be several such proofs.

```
(type.rel:endorelation isomorphic)
(= (type.rel:set isomorphic) meta.ftn:function)
(forall ((meta.ftn:function ?x) (meta.ftn:function ?xp))
    (and (<=> (isomorphic ?x ?xp)
              (exists ((2-cell ?a) (= ?x (source ?a)) (= ?xp (target ?a)))
                  (meta.ftn:bijection (function ?a))))
         (<=> (isomorphic ?x ?xp) (type.ftn.mor:isomorphism [?x ?xp]))))
(type.rel:equivalence-relation isomorphic)
```

**Category Theory.** A *pair* of meta function 2-cells is *composable* when the target of the first is equal to the source of the second. The meta level composable endorelation is the (implicit) optimal restriction of its (implicit) type level counterpart. The extent of the meta level composable endorelation is the meta set of composable pairs of meta 2-cells. We name the projection *factor*s of composable pairs. These component functions are (implicit) restrictions of their type level counterparts.

```
(type.rel:endorelation composable)
(= (type.rel:set composable) 2-cell)
(forall ((2-cell ?a) (2-cell ?b))
    (<=> (composable ?a ?b) (type.ftn.mor:composable-pair [?a ?b])))

(type.ftn:function factor0)
(= (type.ftn:source factor0) (type.rel:extent composable))
(= (type.ftn:target factor0) 2-cell)
(forall ((2-cell ?a) (2-cell ?b) (composable ?a ?b))
    (= (factor0 [?a ?b]) (type.ftn.mor:factor0 [?a ?b])))

(type.ftn:function factor1)
(= (type.ftn:source factor1) (type.rel:extent composable))
(= (type.ftn:target factor1) 2-cell)
(forall ((2-cell ?a) (2-cell ?b) (composable ?a ?b))
    (= (factor1 [?a ?b]) (type.ftn.mor:factor1 [?a ?b])))
```

The *composition* of two composable meta 2-cells $\alpha : x \stackrel{f}{\Rightarrow} y$ and $\beta : y \stackrel{g}{\Rightarrow} z$ is a meta 2-cell $\alpha \circ \beta : x \stackrel{f \cdot g}{\Rightarrow} z$. The source of the composite is the source of the first component factor, the target of the composite is the target of the second component factor, and the function of the composite is the composite of the functions of the component factors. The meta level composition map is a(n implicit) restriction of its type level counterpart.

```
(type.ftn:function composition)
(= (type.ftn:source composition) (type.rel:extent composable))
(= (type.ftn:target composition) 2-cell)
(= (type.ftn:composition [composition source]) (type.ftn:composition [factor0 source]))
(= (type.ftn:composition [composition target]) (type.ftn:composition [factor1 target]))
(forall ((2-cell ?a) (2-cell ?b) (composable ?a ?b))
    (and (= (function (composition [?a ?b]))
            (type.ftn:composition [(function ?a) (function ?b)]))
         (= (composition [?a ?b]) (type.ftn.mor:composition [?a ?b]))))
```

Composition is associative. Any three composable meta 2-cells $\alpha : x \Rightarrow y$, $\beta : y \Rightarrow z$ and $\gamma : z \Rightarrow w$ satisfy the associative law $\alpha \circ (\beta \circ \gamma) = (\alpha \circ \beta) \circ \gamma$.

```
(forall ((2-cell ?a) (2-cell ?b) (2-cell ?c)
        (composable ?a ?b) (composable ?b ?c))
```

```
    (= (composition [?a (composition [?b ?c])])
       (composition [(composition [?a ?b]) ?c])))
```

The composition map $\mathsf{mor}{\times_{\mathsf{ftn}}}\mathsf{mor} \xrightarrow{\circ} \mathsf{mor}$ is surjective (see identity below).

```
(type.ftn:surjection composition)
```

For every meta function $x : X \to V$, there is a unique associated *identity* meta 2-cell $1_x : x \overset{1_X}{\Rightarrow} x$. The meta level identity map is the (implicit) optimal restriction of its type level counterpart.

```
(type.ftn:function identity)
(= (type.ftn:source identity) meta.ftn:function)
(= (type.ftn:target identity) 2-cell)
(= (type.ftn:composition [identity source]) (type.ftn:identity meta.ftn:function))
(= (type.ftn:composition [identity target]) (type.ftn:identity meta.ftn:function))
(= (type.ftn:composition [identity function]) (type.ftn:composition [meta.ftn:source meta.ftn:identity]))
(forall ((meta.ftn:function ?x))
    (= (identity ?x) (type.ftn.mor:identity ?x)))
(forall ((type.ftn:function ?x))
    (=> (2-cell (type.ftn.mor:identity ?x))
        (meta.ftn:function ?x)))
```

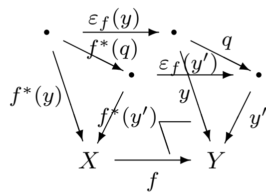
Identity satisfies two unit laws with respect to composition: composition with the identity of the source (target) of a 2-cell $\alpha : x \Rightarrow y$ returns that 2-cell: $1_x \circ \alpha = \alpha = \alpha \circ 1_y$.

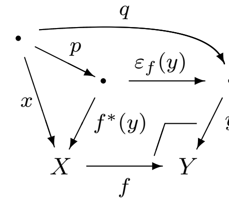
```
(forall ((2-cell ?a))
    (and (= (composition [(identity (source ?a)) ?a]) ?a)
         (= ?a (composition [?a (identity (target ?a))]))))
```

The identity map is injective $\mathsf{ftn} \overset{1}{\hookrightarrow} \mathsf{mor}$. Hence, functions can be regarded as special 2-cells that satisfy the unit laws.

```
(type.ftn:injection identity)
```

**Transformation.** Let $f : X \to Y$ be a function. Composition with $f$ defines an external *existential* quantification functor $\Sigma_f : \mathsf{Set}/X \to \mathsf{Set}/Y$ between slice categories: $x \mapsto x \cdot f$ and $(p : x \Rightarrow y) \mapsto (p : x \cdot f \Rightarrow y \cdot f)$. The existential operation is functorial: the existential quantification of the identity $1_X : X \to X$ is the identity of the existential quantification, $\Sigma_{1_X} = 1_{\mathsf{Set}/X} : \mathsf{Set}/X \to \mathsf{Set}/X$; and the existential quantification of the composition $f \cdot g : X \to Z$ is the composition of the existential quantifications, $\Sigma_{f \cdot g} = \Sigma_f \cdot \Sigma_g : \mathsf{Set}/X \to \mathsf{Set}/Z$. Since $\mathsf{Set}_{\mathsf{meta}}$ has (canonical) pullbacks, the operation of pulling back along $f$ forms an external *inverse image* functor $f^* : \mathsf{Set}/Y \to \mathsf{Set}/X$ between slice categories. An external universal quantification functor $\Pi_f : \mathsf{Set}/X \to \mathsf{Set}/Y$ can also be conceived between slice categories. The inverse (universal) image operation is also functorial. The existential functor is left adjoint to the inverse image functor and the inverse image functor is left adjoint to the universal functor $\Sigma_f \dashv (\text{-})_f^{-1} \dashv \Pi_f$. The counit $\varepsilon_f : f^* \cdot \Sigma_f \Rightarrow 1_{\mathsf{Set}/Y}$ for the first adjunction has the pullback projection $\varepsilon_f(y) = \pi_1 : X{\times_Y}\partial_0(y) \to \partial_0(y)$ as its $y^{\text{th}}$ component. Unlike the internal transformations between subset lattices, the external transformations between complete slice categories cannot be defined at this level.

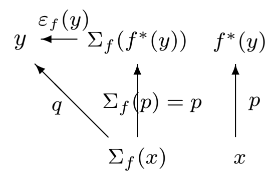



inverse image functor $f^*$ and counit $\varepsilon_f$

adjunction $\Sigma_f \dashv f^*$

$$\mathsf{Set}/X \underset{f^*}{\overset{\Sigma_f}{\rightleftarrows}} \mathsf{Set}/Y \qquad \eta_f \dashv \varepsilon_f$$



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.