The IFF Namespace

Robert E. Kent

February 7, 2006

Contents

1 IFF Namespace

Namespace Prefix

Technical:	`iff`
Recommended:	`iff`

The terminology of the IFF namespace, a small-sized namespace at the very top of the IFF architecture, is listed in Table 1. This consists of only 19 terms. There is a collection of all IFF *thing*s. IFF things fall into four basic collections: there is a collection set of IFF *set*s, there is a collection pred of IFF *predicate*s, there is a collection ftn of (unary) IFF *function*s, and there is a collection rel of (binary) IFF *relation*s. All four basic collections are distinct; an IFF predicate is not an IFF set, an IFF function is neither an IFF set nor an IFF predicate, and an IFF relation is neither an IFF set, an IFF predicate nor an IFF function; that is, these collections are pairwise disjoint. Some of the components of these four basic collections are also introduced here. There is a *genus* map from predicates to sets, there are *source* and *target* maps from functions to sets, and there is a domain or *zeroth* map and a codomain or *first* map from relations to sets. There is also a binary *intersection* operation on sets. Things of the same nature may be related. There is a *subset* partial order on IFF sets, there is a *delimitation* partial order on IFF predicates, there are *restriction* and *optimal-restriction* partial orders on IFF functions, and there is an *abridgment* partial order on IFF relations. The five endorelations subset, delimitation, restriction,

$$\begin{tikzcd}
 & \textsf{function} & \textsf{set} & \textsf{relation} & \\
\textsf{generic} & \textsf{ftn}_\infty \arrow[r, shift left=2, "\partial_0^\infty"] \arrow[r, shift right=2, "\partial_1^\infty"'] \arrow[r, dotted, no head] \arrow[dr, phantom, "\llcorner", very near end] & \textsf{set}_\infty \arrow[r, dotted, no head] & \textsf{rel}_\infty \arrow[l, shift left=2, "\sigma_0^\infty"'] \arrow[l, shift right=2, "\sigma_1^\infty"] \arrow[dl, phantom, "\lrcorner", very near end] & \infty = \texttt{ur} \\
 & \vdots \arrow[u, hook] & \vdots \arrow[u, hook] & \vdots \arrow[u, hook] & \vdots \\
\textsf{very large} & \textsf{ftn}_3 \arrow[u, hook] \arrow[r, shift left=2, "\partial_0^3"] \arrow[r, shift right=2, "\partial_1^3"'] \arrow[r, dotted, no head] \arrow[dr, phantom, "\llcorner", very near end] & \textsf{set}_3 \arrow[u, hook] \arrow[r, dotted, no head] & \textsf{rel}_3 \arrow[u, hook] \arrow[l, shift left=2, "\sigma_0^3"'] \arrow[l, shift right=2, "\sigma_1^3"] \arrow[dl, phantom, "\lrcorner", very near end] & 3 = \texttt{vlrg} \\
\textsf{large} & \textsf{ftn}_2 \arrow[u, hook] \arrow[r, shift left=2, "\partial_0^2"] \arrow[r, shift right=2, "\partial_1^2"'] \arrow[r, dotted, no head] \arrow[dr, phantom, "\llcorner", very near end] & \textsf{set}_2 \arrow[u, hook] \arrow[r, dotted, no head] & \textsf{rel}_2 \arrow[u, hook] \arrow[l, shift left=2, "\sigma_0^2"'] \arrow[l, shift right=2, "\sigma_1^2"] \arrow[dl, phantom, "\lrcorner", very near end] & 2 = \texttt{lrg} \\
\textsf{small} & \textsf{ftn}_1 \arrow[u, hook] \arrow[r, shift left=2, "\partial_0^1"] \arrow[r, shift right=2, "\partial_1^1"'] \arrow[r, dotted, no head] & \textsf{set}_1 \arrow[u, hook] \arrow[r, dotted, no head] & \textsf{rel}_1 \arrow[u, hook] \arrow[l, shift left=2, "\sigma_0^1"'] \arrow[l, shift right=2, "\sigma_1^1"] & 1 = \texttt{sml}
\end{tikzcd}$$

Figure 1: Metastack Structural Framework

optimal-restriction and abridgment (and the binary intersection operation) are important for building the vertical aspect (establishing the preservation properties) of the metastack structural framework illustrated in Figure 1. Finally, there are three elements used for indexing: '`0`', '`1`' and '`2`'.

Like other parts of the IFF metashell, the IFF namespace is closely related to the IFF grammar (see the document entitled "The IFF Syntax"), which specifies the correct form for IFF expressions. The IFF grammar can be used for type correctness in set membership, predicate invocation, function application and relation invocation: an error should be signaled, (1) if a set or predicate symbol is being used as a function (in a term) or a relation (in a relational expression), (2) if a function symbol is being used as a set or predicate (in a declaration), a set component for functions and relations or as a relation (in a relational expression), and (3) if a relation symbol is being used as a set or predicate (in a declaration), a set component for functions and relations or as a function (in a term). As is explained below, the IFF grammar can and should be used to check for well-formed-ness of set membership, predicate invocation, function application and relation invocation: set membership, predicate invocation, function application and relation invocation are all represented in prefix notation.

1.1 IFF Things

Things exist. There is a set of all IFF *thing*s. Any IFF set, IFF predicate, (unary) IFF function or (binary) IFF relation is an IFF thing. But there can be no two of these.

```
(forall (?x (thing ?x))
    (and (not (and (set ?x) (function ?x)))
         (not (and (set ?x) (predicate ?x)))
         (not (and (set ?x) (relation ?x)))
         (not (and (predicate ?x) (function ?x)))
```

Set	Function	Relation
thing		
set	binary-intersection	subset
predicate	genus	delimitation
function	source target	(optimal-)restriction
relation	set0 set1	abridgment
0 1 2		

Technical and Recommended Prefix : `iff`

Table 1: The IFF Namespace

```
        (not (and (predicate ?x) (relation ?x)))
        (not (and (function ?x) (relation ?x)))))
```

We assert the existence of (at least) three distinct canoncial objects (things): 0, 1 and 2. These are used for indexing.

```
(thing 0) (thing 1) (thing 2)
(not (= 0 1)) (not (= 0 2)) (not (= 1 2))
```

1.2 IFF Sets

There is a set of all IFF *sets*. The symbol '`set`' is used to declare an IFF set. Any IFF set is an IFF thing. Any element of an IFF set is an IFF thing.

```
(forall (?X (set ?X)) (thing ?X))
(forall (?X (set ?X) ?x (?X ?x)) (thing ?x))
```

If the symbols '`X`' and '`x`' represent the two IFF things X and x, then

```
(set X)
```

declares X to be an IFF set, and

```
(X x)
```

expresses the statement that "$x \in X$". However, the following nullary, binary, ternary and higher arity expressions are ill-formed

```
(X)
(X x0 x1)
(X x0 x1 x2)
...
```

For any pair of IFF sets X_0 and X_1, there is a binary *intersection* IFF set defined by $X_0 \cap X_1 = \{y \mid y \in X_0 \text{ and } y \in X_1\}$. The symbol '`binary-intersection`' is used to express application of the binary intersection (total, functional) operation, which takes a pair of IFF sets and returns an IFF set.

```
(forall (?X0 (thing ?X0) ?X1 (thing ?X1) ?Y (thing ?Y)
        (= ?Y (binary-intersection [?X0 ?X1])))
    (and (set ?X0) (set ?X1) (set ?Y)))
(forall (?X0 (set ?X0) ?X1 (set ?X1))
    (and (exists (?Y (set ?Y))
             (= (binary-intersection [?X0 ?X1]) ?Y))
         (forall (?Y0 (set ?Y0) ?Y1 (set ?Y1)
                 (= (binary-intersection [?X0 ?X1]) ?Y0)
                 (= (binary-intersection [?X0 ?X1]) ?Y1))
             (= ?Y0 ?Y1))))
(forall (?X0 (set ?X0) ?X1 (set ?X1))
    (and (subset (binary-intersection [?X0 ?X1]) ?X0)
         (subset (binary-intersection [?X0 ?X1]) ?X1)))
(forall (?X0 (set ?X0) ?X1 (set ?X1) ?x (?X0 ?x) (?X1 ?x))
    ((binary-intersection [?X0 ?X1]) ?x))
```

There is a binary *subset* endorelation $\subseteq$ between pairs of IFF sets. Pointwise, the subset relationship $X \subseteq Y$ holds when

- all elements of the smaller set $x \in X$ are elements of the larger set $x \in Y$.

The subset endorelation on IFF sets is a partial order (reflexive, antisymmetric and transitive), since identities are inclusions and inclusions are close under composition. The symbol '`subset`' is used to assert the subset relationship between pairs of IFF sets.

```
(forall (?X (thing ?X) ?Y (thing ?Y) (subset ?X ?Y))
    (and (set ?X) (set ?Y)))
(forall (?X (set ?X) ?Y (set ?Y))
    (<=> (subset ?X ?Y)
         (forall (?x (?X ?x)) (?Y ?x))))
(forall (?X (set ?X))
    (subset ?X ?X))
(forall (?X (set ?X) ?Y (set ?Y))
    (=> (and (subset ?X ?Y) (subset ?Y ?X))
        (= ?X ?Y)))
(forall (?X (set ?X) ?Y (set ?Y) ?Z (set ?Z))
    (=> (and (subset ?X ?Y) (subset ?Y ?Z))
        (subset ?X ?Z)))
```

1.3 IFF Predicates

There is a set of all IFF *predicate*s. Predicates are also called "parts". The symbol '`predicate`' is used to declare an IFF predicate. Any IFF predicate is an IFF thing.

```
(forall (?p (predicate ?p)) (thing ?p))
```

A part (predicate) of a set X is an injection $p : |p| \hookrightarrow X$. Each IFF predicate p has a unique *genus* IFF set X. The predicate notation $p : X$ shows an IFF predicate p with genus IFF set X. The notation $\gamma(p) = \mathsf{gen}(p) = X$ is also used to assert the genus. The symbol '`genus`' is used to express application of the genus (total, functional) operation, which takes an IFF predicate and returns an IFF set.

```
(forall (?p (thing ?p) ?X (thing ?X) (= ?X (genus ?p)))
    (and (predicate ?p) (set ?X)))
(forall (?p (predicate ?p))
    (and (exists (?X (set ?X))
             (= (genus ?p) ?X))
         (forall (?X0 (set ?X0) ?X1 (set ?X1)
                  (= (genus ?p) ?X0) (= (genus ?p) ?X1))
             (= ?X0 ?X1))))
```

If the symbols '`p`' and '`X`' represent the two IFF things p and X, then

```
(predicate p)
(set X) (= (genus p) X)
```

makes the declaration "$p : X$" (or "$\gamma(r) = X$"), and

```
(X x)
(p x)
```

expresses the statement that "$p(x)$". Hence, '`(p x)`' is a well-formed formula, which follows the prescription: *all IFF predicates are unnary.* However, the following nullary, binary, ternary and higher arity expressions are iff-formed

```
(p)
(p x0 x1)
(p x0 x1 x2)
...
```

There is a binary *delimitation* endorelation $\leq$ between pairs of IFF predicates. One IFF predicate $p : X$ is a *delimitation* of another IFF predicate $q : Y$, $p \leq q$, when the genus IFF set of p is a subset of the genus IFF set of q, $X \subseteq Y$, and pointwise

- the two predicates are equivalent $p(x)$ iff $q(x)$ on all p-genus elements $x \in X$.

From one point of view, delimitation defines the smaller predicate; that is, if we assume that the definition of the larger predicate is given (in some other axiomatization), then asserting the delimitation relationship effectively defines the smaller predicate. This is the point of view taken when we use (only) delimitation to define a particular predicate at one metalevel in terms of the similar predicate at the next higher metalevel. The delimitation endorelation is a partial order (reflexive, antisymmetric and transitive). The symbol '`delimitation`' is used to assert the delimitation relationship between pairs of IFF predicates.

```
(forall (?p (thing ?p) ?q (thing ?q) (delimitation ?p ?q))
    (and (predicate ?p) (predicate ?q)))
(forall (?p (predicate ?p) ?q (predicate ?q))
     (<=> (delimitation ?p ?q)
          (and (subset (genus ?p) (genus ?q))
               (forall (?x ((genus ?p) ?x))
                   (<=> (?p ?x) (?q ?x))))))
(forall (?p (predicate ?p))
    (delimitation ?p ?p))
(forall (?p (predicate ?p) ?q (predicate ?q))
    (=> (and (delimitation ?p ?q) (delimitation ?q ?p))
        (= ?p ?q)))
(forall (?p (predicate ?p) ?q (predicate ?q) ?r (predicate ?r))
    (=> (and (delimitation ?p ?q) (delimitation ?q ?r))
        (delimitation ?p ?r)))
```

1.4 IFF Functions

There is a set of all IFF *functions*. The symbol '`function`' is used to declare an IFF function. Any IFF function is an IFF thing.

```
(forall (?f (function ?f)) (thing ?f))
```

Each IFF function has a unique *source* IFF set and a unique *target* IFF set. The function notation $f : X \to Y$ shows an IFF function f with source IFF set X and target IFF set Y. The notation $\partial_0(f) = X$ and $\partial_1(f) = Y$ is also used to assert the source and target. The symbols '`source`' and '`target`' are used to express application of the source and target (total, functional) operations, which take an IFF function and return an IFF set.

```
(forall (?f (thing ?f) ?X (thing ?X) (= ?X (source ?f)))
    (and (function ?f) (set ?X)))
(forall (?f (function ?f))
    (and (exists (?X (set ?X))
             (= (source ?f) ?X))
         (forall (?X0 (set ?X0) ?X1 (set ?X1)
                 (= (source ?f) ?X0) (= (source ?f) ?X1))
             (= ?X0 ?X1))))

(forall (?f (thing ?f) ?Y (thing ?Y) (= ?Y (target ?f)))
    (and (function ?f) (set ?Y)))
(forall (?f (function ?f))
    (and (exists (?Y (set ?Y))
             (= (target ?f) ?Y))
         (forall (?Y0 (set ?Y0) ?Y1 (set ?Y1)
                 (= (target ?f) ?Y0) (= (target ?f) ?Y1))
             (= ?Y0 ?Y1))))
```

Any IFF function is required to be total (defined on all source elements) and functional (have only one value).

Note 1 *This axiom uses quantification over variables in function position.*

```
(forall (?f (function ?f))
    (forall (?x ((source ?f) ?x))
        (and (exists (?y ((target ?f) ?y))
                 (= (?f ?x) ?y))
             (forall (?y0 ((target ?f) ?y0) ?y1 ((target ?f) ?y1)
                     (= (?f ?x) ?y0) (= (?f ?x) ?y1))
                 (= ?y0 ?y1)))))
```

If the symbols '`f`', '`X`' and '`Y`' represent the three IFF things f, X and Y, then

```
(function f)
(set X) (= (source f) X)
(set Y) (= (target f) Y)
```

makes the declaration "$f : X \to Y$" (or "$\partial_0(f) = X$" and "$\partial_1(f) = Y$"), and

```
(X x) (X y)
(= y (f x))
```

expresses the equality statement that "$y = f(x)$". Hence, '`(f x)`' is a well-formed term, which follows the prescription: *all IFF functions are unary.* However, the following nullary, binary, ternary and higher arity expressions are iff-formed

```
(= y (f))
(= y (f x0 x1))
(= y (f x0 x1 x2))
...
```

The IFF tuple notation can be used to express n-ary functions

```
(= y (f [x0 x1 ...]))
```

There is a binary *restriction* relationship $\sqsubseteq$ between pairs of IFF functions that are linked by source and target inclusions. One (smaller) IFF function $f_1 : X_1 \rightarrow Y_1$ is a restriction of another (larger) IFF 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 X_1), $f_1(x_1) = f_2(x_1)$ for all elements $x_1 \in X_1$. This implies that the source type set of f_1 is a subset of the inverse image of the target type set of f_1 along f_2: $X_1 \subseteq f_2^{-1}(Y_1)$. From one point of view, restriction can be viewed as a constraint on the larger IFF function — it says that the larger function maps the source IFF set of the smaller function into the target IFF set of the smaller function. From another 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 similar function at the next higher metalevel. The restriction relation is a partial order (reflexive, antisymmetric and transitive), since subset is a partial order and property (2) above holds. The symbol '`restriction`' is used to assert the restriction relationship between pairs of IFF functions.

```
(forall (?f1 (thing ?f) ?f2 (thing ?f2) (restriction ?f1 ?f2))
    (and (function ?f1) (function ?f2)))
(forall (?f1 (function ?f1) ?f2 (function ?f2))
    (<=> (restriction ?f1 ?f2)
         (and (subset (source ?f1) (source ?f2))
              (subset (target ?f1) (target ?f2))
              (forall (?x1 ((source ?f1) ?x1))
                  (= (?f1 ?x1) (?f2 ?x1))))))
(forall (?f (function ?f))
    (restriction ?f ?f))
(forall (?f1 (function ?f1) ?f2 (function ?f2))
    (=> (and (restriction ?f1 ?f2) (restriction ?f2 ?f1))
        (= ?f1 ?f2)))
(forall (?f1 (function ?f1) ?f2 (function ?f2) ?f3 (function ?f3))
    (=> (and (restriction ?f1 ?f2) (restriction ?f2 ?f3))
        (restriction ?f1 ?f3)))
```

We reiterate a fact of order. The source and target functions are monotonic: if two functions satisfy the restriction order, $f \sqsubseteq g$, then their source (target) sets satisfy the subset order, $\partial_0(f) \subseteq \partial_0(g)$ ($\partial_1(f) \subseteq \partial_1(g)$).

```
(forall (?f1 (function ?f1) ?f2 (function ?f2) (restriction ?f1 ?f2))
    (and (subset (source ?f1) (source ?f2))
         (subset (target ?f1) (target ?f2))))
```

There is a binary *optimal restriction* relationship $\dot{\sqsubseteq}$ between pairs of IFF functions. A restriction between two IFF functions f_1 and f_2 is optimal, $f_1 \dot{\sqsubseteq} f_2$, when the source IFF set of the smaller function is exactly the inverse image of the target IFF set of the smaller function along the larger function: $X_1 = f_2^{-1}(Y_1)$. Pointwise,

- the two functions agree $f_1(x_1) = f_2(x_1)$ on all f_1-source elements $x_1 \in X_1$; and
- any f_2-source element $x_2 \in X_2$ that maps into the f_1-target, $f_2(x_2) \in Y_1$, is actually an f_1-source element $x_2 \in X_1$.

The optimal restriction relation is also a partial order (reflexive, antisymmetric and transitive). The symbol '`optimal-restriction`' is used to assert the optimal restriction relationship between pairs of IFF functions.

```
(forall (?f1 (thing ?f) ?f2 (thing ?f2) (optimal-restriction ?f1 ?f2))
    (and (function ?f1) (function ?f2)))
(forall (?f1 (function ?f1) ?f2 (function ?f2) (optimal-restriction ?f1 ?f2))
    (restriction ?f1 ?f2))
(forall (?f1 (function ?f1) ?f2 (function ?f2) (restriction ?f1 ?f2))
    (<=> (optimal-restriction ?f1 ?f2)
         (forall (?x2 ((source ?f2) ?x2))
             (=> ((target ?f1) (?f2 ?x2))
                 ((source ?f1) ?x2)))))
(forall (?f (function ?f))
    (optimal-restriction ?f ?f))
(forall (?f1 (function ?f1) ?f2 (function ?f2))
    (=> (and (optimal-restriction ?f1 ?f2) (optimal-restriction ?f2 ?f1))
        (= ?f1 ?f2)))
(forall (?f1 (function ?f1) ?f2 (function ?f2) ?f3 (function ?f3))
    (=> (and (optimal-restriction ?f1 ?f2) (optimal-restriction ?f2 ?f3))
        (optimal-restriction ?f1 ?f3)))
```

We reiterate some facts of order. If two functions satisfy the optimal restriction order, $f \dot{\sqsubseteq} g$, then they satisfy the restriction order, $f \sqsubseteq g$. The source and target functions are monotonic: if two functions satisfy the optimal restriction order, $f \dot{\sqsubseteq} g$, then their source (target) sets satisfy the subset order, $\partial_0(f) \subseteq \partial_0(g)$ $(\partial_1(f) \subseteq \partial_1(g))$.

```
(forall (?f1 (function ?f1) ?f2 (function ?f2) (optimal-restriction ?f1 ?f2))
    (restriction ?f1 ?f2))
(forall (?f1 (function ?f1) ?f2 (function ?f2) (optimal-restriction ?f1 ?f2))
    (and (subset (source ?f1) (source ?f2))
         (subset (target ?f1) (target ?f2))))
```

1.5 IFF Relations

There is a set of all IFF *relations*. The symbol '`relation`' is used to declare an IFF relation. Any IFF relation is an IFF thing.

```
(forall (?r (relation ?r)) (thing ?r))
```

Each IFF relation has a unique domain or *zeroth* IFF set and a unique codomain or *first* IFF set. The relation notation $r : X_0 \rightharpoondown X_1$ shows an IFF relation r with domain or zeroth IFF set X_0 and codomain or first IFF set X_1. The notation

$\sigma_0(r) = X_0$ and $\sigma_1(r) = X_1$ is also used to assert the source and target. The symbols '`set0`' and '`set1`' is used to express application of the zeroth and first (total, functional) operations, which take an IFF relation and return an IFF set.

```
(forall (?r (thing ?r) ?X0 (thing ?X0) (= ?X0 (set0 ?r)))
    (and (relation ?r) (set ?X0)))
(forall (?r (relation ?r))
    (and (exists (?X (set ?X))
             (= (set0 ?r) ?X))
         (forall (?X0 (set ?X0) ?X1 (set ?X1)
                  (= (set0 ?r) ?X0) (= (set0 ?r) ?X1))
             (= ?X0 ?X1))))

(forall (?r (thing ?r) ?X1 (thing ?X1) (= ?X1 (set1 ?r)))
    (and (relation ?r) (set ?X1)))
(forall (?r (relation ?r))
    (and (exists (?X (set ?X))
             (= (set1 ?r) ?X))
         (forall (?X0 (set ?X0) ?X1 (set ?X1)
                  (= (set1 ?r) ?X0) (= (set1 ?r) ?X1))
             (= ?X0 ?X1))))
```

If the symbols '`r`', '`X0`' and '`X1`' represent the three IFF things r, X_0 and X_1, then

```
(relation r)
(set X0) (= (set0 r) X0)
(set X1) (= (set1 r) X1)
```

makes the declaration "$r : X_0 \rightharpoondown X_1$" (or "$\sigma_0(r) = X_0$" and "$\sigma_1(r) = X_1$"), and

```
(X0 x0) (X1 x1)
(r x0 x1)
```

expresses the statement that "$r(x_0, x_1)$". Hence, '`(r x0 x1)`' is a well-formed formula, which follows the prescription: *all IFF relations are binary.* However, the following nullary, unary, ternary and higher arity expressions are iff-formed

```
(r)
(r x0)
(r x0 x1 x2)
...
```

The notion of abridgment (Merriam-Webster) is that of "a shortened form of a work retaining the general sense and unity of the original". There is a binary *abridgment* relationship $\preceq$ between pairs of IFF relations. One IFF relation $r : X_0 \rightharpoondown X_1$ is an abridgment of another IFF relation $s : Y_0 \rightharpoondown Y_1$, $r \preceq s$, when the domain IFF set X_0 and the codomain IFF set X_1 of r are IFF subsets of the domain IFF set Y_0 and the codomain IFF set Y_1 of s, respectively, $X_0 \subseteq Y_0$ and $X_1 \subseteq Y_1$, and pointwise

- the two relations are equivalent $r(x_0, x_1)$ iff $s(x_0, x_1)$ on all r-component pairs $x_0 \in X_0$ and $x_1 \in X_1$.

The abridgment relation is a partial order (reflexive, antisymmetric and transitive). The symbol '`abridgment`' is used to assert the abridgment relationship between pairs of IFF relations.

```
(forall (?r (thing ?r) ?s (thing ?s) (abridgment ?r ?s))
    (and (relation ?r) (relation ?s)))
(forall (?r (relation ?r) ?s (relation ?s))
     (<=> (abridgment ?r ?s)
          (and (subset (set0 ?r) (set0 ?s))
               (subset (set1 ?r) (set1 ?s))
               (forall (?x0 ((set0 ?r) ?x0) ?x1 ((set1 ?r) ?x1))
                   (<=> (?r ?x0 ?x1) (?s ?x0 ?x1))))))
(forall (?r (relation ?r))
    (abridgment ?r ?r))
(forall (?r (relation ?r) ?s (relation ?s))
    (=> (and (abridgment ?r ?s) (abridgment ?s ?r))
        (= ?r ?s)))
(forall (?r (relation ?r) ?s (relation ?s) ?t (relation ?t))
    (=> (and (abridgment ?r ?s) (abridgment ?s ?t))
        (abridgment ?r ?t)))
```

We reiterate a fact of order. The domain and codomain functions are monotonic: if two relations satisfy the abridgment order, $r \preceq s$, then their domain (codomain) sets satisfy the subset order, $\sigma_0(r) \subseteq \sigma_0(s)$ $(\sigma_1(r) \subseteq \sigma_1(s))$.

```
(forall (?r (relation ?r) ?s (relation ?s) (abridgment ?r ?s))
    (and (subset (set0 ?r) (set0 ?s))
         (subset (set1 ?r) (set1 ?s))))
```

A Composites and Determination

A *composite* thing $x : \mathcal{X}$ is a thing with *component*s. Each of these components is represented as a function $c : \mathcal{X} \to \mathcal{Y}_c$ from the type of the thing $\mathcal{X}$ to its own component type $\mathcal{Y}_c$. A set of components $\mathcal{C}$ *determines* a set (of things of type) $\mathcal{X}$

$$\mathcal{C} \vdash \mathcal{X}$$

when $\forall_{x,x' \in \mathcal{X}} \forall_{c \in \mathcal{C}} \, (c(x) = c(x')) \Rightarrow (x = x')$. Abstractly, this means that the tupling function $(c)_{c \in \mathcal{C}} : \mathcal{X} \to \Pi_{c \in \mathcal{C}} \mathcal{Y}_c$ is an injection. Here are some examples.

- A pair of things of types X_0 and X_1 is a set pair (X_0, X_1). The set (type) of set pairs $\mathcal{X} = \mathsf{set}^2$ is determined

  $$\{\pi_0, \pi_0\} \vdash \mathsf{set}^2$$

  by the two projection functions $\mathcal{C} = \{\pi_0 : \mathsf{set}^2 \to \mathsf{set}, \pi_1 : \mathsf{set}^2 \to \mathsf{set}\}$.

- The set (type) of categories $\mathcal{X} = \mathsf{cat}$ is determined

  $$\langle \mathsf{obj}, \mathsf{mor}, \partial_0, \partial_1, \cdot, 1 \rangle \vdash \mathsf{cat}$$

for each category C by the collection consisting of: the object set $\mathsf{obj}(C)$, the morphism set $\mathsf{mor}(C)$, the source function $\partial_0^C : \mathsf{mor}(C) \to \mathsf{obj}(C)$, the target function $\partial_1^C : \mathsf{mor}(C) \to \mathsf{obj}(C)$, the composition function $\cdot_C : \mathsf{mor}(C) \times_{\mathsf{obj}(C)} \mathsf{mor}(C) \to \mathsf{obj}(C)$ and the identity function $1_C : \mathsf{obj}(C) \to \mathsf{mor}(C)$.

- But $\mathcal{X} = \mathsf{cat}$ is also determined

$$\langle \mathsf{gph}, \mu, \eta \rangle \vdash \mathsf{cat}.$$

for each category C by the collection consisting of: the underlying graph $|C| = \mathsf{gph}(C) = \langle \mathsf{obj}(C), \mathsf{mor}(C), \partial_0^C, \partial_1^C \rangle$, with the identity object function $1_{\mathsf{obj}}(C)$, where the set (type) of graphs gph has the determination

$$\langle \mathsf{obj}, \mathsf{mor}, \partial_0, \partial_1 \rangle \vdash \mathsf{gph},$$

the composition graph morphism $\mu_C : |C| \otimes |C| \to |C|$ with the identity object function $1_{\mathsf{obj}}(C)$, and the identity graph morphism $\eta_C : 1_{\mathsf{obj}}(C) \to |C|$ with the identity object function $1_{\mathsf{obj}}(C)$, where the set (type) of graph morphisms $\mathsf{gph\text{-}mor}$ has the determination of their object/morphism functions

$$\langle \cdot \rangle \vdash \mu,$$

$$\langle 1 \rangle \vdash \eta.$$

The last example shows both that determination has an order structure and that there is more than one way to "package" a concept. We would like to axiomatize this notion. For this, it would be useful to have a simple listing set-builder notation $\{\cdots\}$ in the IFF.
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.