The IFF Type Namespace

Robert E. Kent

December 20, 2007

Contents

0.1 Type Sets 2
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

0.1 Type Sets

Basics. There is an IFF set set of all type *set*s. Any type set is itself an IFF set; that is, the IFF set of all type sets is an (implicit) subset of the set of all IFF sets.

```
(iff:set set)
(forall ((set ?X)) (iff:set ?X))
```

There is a binary *isomorphic* endorelation between pairs of type sets, that are linked by an bijection[1]. Since there is no notion of binary relation specified at the IFF level, at the type level we use the extent set of the isomorphism relation[2] to indirectly specify it. The isomorphic endorelation is an equivalence relation (reflexive, symmetric and transitive), since identities are bijections, bijections have an inverse and bijections are closed under composition.

```
(iff:set isomorphic-relation)
(forall ((isomorphic-relation ?XY)) (type.dgm.pr.obj:set-pair ?XY))
(forall ((set ?X) (set ?Y))
    (<=> (isomorphic-relation [?X ?Y])
         (exists ((type.ftn:bijection ?f))
             (and (= ?X (type.ftn:source ?f)) (= ?Y (type.ftn:target ?f))))))
(forall ((set ?X))
    (isomorphic-relation [?X ?X]))
(forall ((set ?X) (set ?Y))
    (=> (isomorphic-relation [?X ?Y])
        (isomorphic-relation [?Y ?X])))
(forall ((set ?X) (set ?Y) (set ?Z))
    (=> (and (isomorphic-relation [?X ?Y]) (isomorphic-relation [?Y ?Z]))
        (isomorphic-relation [?X ?Z])))
```

The following is taken from the book *Sets for Mathematics* (2003) by Lawvere and Rosebrugh.

Fact 1 (Galileo) *The set* $\mathsf{Natno} = \aleph = \{0, 1, 2, \cdots\}$ *of all natural numbers is isomorphic to the set* $\mathsf{Sqr} = \{0, 2, 4, \cdots\} \subset \mathsf{Natno}$ *of all square whole numbers (a proper subset).*

Proof. Use the squaring function $(\text{-}) : \mathsf{Natno} \xrightarrow{\sim} \mathsf{Sqr}$ and its inverse the square root function $\surd : \mathsf{Sqr} \xrightarrow{\sim} \mathsf{Natno}$. ∎

This observation by Galileo was generalized into a definition by Dedekind.

Definition 1 (Dedekind) *A set X is* finite *when all injections on X are bijections. A set X is* infinite *when there is at least one injection on X that is not surjective.*

---

[1] Here we follow and quote from the discussion on the topic of isomorphism and Dedekind finiteness as presented in Lawvere and Rosebrugh [1]. "The notation $f : X \xrightarrow{\sim} Y$ means that f is an isomorphism. One type set X is *isomorphic* to a type set Y when there is at least one isomorphism (type bijection) from X to Y. This definition of isomorphism is used in all categories, but in a category of abstract sets and arbitrary functions the two type sets X and Y are said to be *equinumerous* or to *have the same cardinality*." As Lawvere and Rosebrugh point out, the isomorphism of abstract sets offers a method to study equinumerosity without counting, a fact systematically used by Cantor.

[2] We use this idea of "extent serving as proxy" for other relations and orders, such as subset, delimitation, (optimal-)restriction, abridgment, function (element) belonging, predicate (part) inclusion and the member relations between functions and predicates and between spans and relations. This is a small part of the bootstrapping mechanism of the IFF metashell.

These are predicates (adjectives) that refer to (modify) sets (nouns), the genus, and result in subsets (noun phrases), the differentia.

```
(iff:set finite-set)
(forall ((finite-set ?X)) (set ?X))
(forall ((set ?X))
    (<=> (finite-set ?X)
         (forall ((type.ftn:injection ?f)
                 (= (type.ftn:source ?f) ?X)
                 (= (type.ftn:target ?f) ?X))
             (type.ftn:bijection ?f))))

(iff:set infinite-set)
(forall ((infinite-set ?X)) (set ?X))
(forall ((set ?X))
    (<=> (infinite-set ?X)
         (not (finite-set ?X))))
```

A type set X is relatively *small* when it is a meta set. That is, the predicate "small" has as its genus the set of all type sets and as its differentia the set of all meta sets.

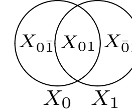
```
(iff:set small-set)
(forall ((small-set ?X)) (set ?X))
(forall ((set ?X))
   (<=> (small-set ?X)
        (meta.set:set ?X)))
```

Any finite set is relatively small.

```
(forall ((finite-set ?X)) (small-set ?X))
```

For any pair of type sets X_0 and X_1, there are *binary union*, *binary intersection* and (binary) *difference* type sets defined as follows.

$$\begin{aligned} X_0 \cup X_1 &= \{y \mid y \in X_0 \text{ or } y \in X_1\} &= X_{0\bar{1}} \cup X_{01} \cup X_{\bar{0}1} \\ X_0 \cap X_1 &= \{y \mid y \in X_0 \text{ and } y \in X_1\} &= X_{01} \\ X_0 \setminus X_1 &= \{y \mid y \in X_0 \text{ and } y \notin X_1\} &= X_{0\bar{1}} \end{aligned}$$




```
(iff:function binary-union)
(= (iff:source binary-union) type.dgm.pr.obj:set-pair)
(= (iff:target binary-union) set)
(forall ((set ?X0) (set ?X1))
    (and (subset-relation [?X0 (binary-union [?X0 ?X1])])
         (subset-relation [?X1 (binary-union [?X0 ?X1])])
         (forall (((binary-union [?X0 ?X1]) ?x))
             (or (?X0 ?x) (?X1 ?x)))))

(iff:function binary-intersection)
(= (iff:source binary-intersection) type.dgm.pr.obj:set-pair)
(= (iff:target binary-intersection) set)
(forall ((set ?X0) (set ?X1))
    (and (subset-relation [(binary-intersection [?X0 ?X1]) ?X0])
         (subset-relation [(binary-intersection [?X0 ?X1]) ?X1])))
(forall ((set ?X0) (set ?X1) (?X0 ?x) (?X1 ?x))
    ((binary-intersection [?X0 ?X1]) ?x))

(iff:function difference)
```

```
(= (iff:source difference) type.dgm.pr.obj:set-pair)
(= (iff:target difference) set)
(forall ((set ?X0) (set ?X1))
    (and (subset-relation [(difference [?X0 ?X1]) ?X0])
         (forall (((difference [?X0 ?X1]) ?x))
             (not (?X1 ?x)))
         (forall ((?X0 ?x) (not (?X1 ?x)))
             ((difference [?X0 ?X1]) ?x))))
```

Here are a few of the many properties that relate these operations:

$$
\begin{aligned}
X_0 \cup X_1 &= X_1 \cup X_0 \\
(X_0 \cup X_1) \cup X_2 &= X_0 \cup (X_1 \cup X_2) \\
X_0 \cap X_1 &= X_1 \cap X_0 \\
(X_0 \cap X_1) \cap X_2 &= X_0 \cap (X_1 \cup X_2) \\
(X_0 \setminus X_1) \cup X_1 &= X_0 \cup X_1 \\
(X_0 \setminus X_1) \cap X_1 &= \emptyset
\end{aligned}
$$

$$
\begin{array}{rl}
\text{if} & X \subseteq Y, \\
\text{then} & X \cup Y = Y,\ X \cap Y = X \text{ and } X \setminus Y = \emptyset \\
\text{hence,} & X \cup X = X,\ X \cup \emptyset = X, \\
& X \cap X = X,\ X \cap \emptyset = \emptyset, \\
& X \setminus \emptyset = X,\ X \setminus X = \emptyset
\end{array}
$$

```
(forall ((set ?X0) (set ?X1))
    (and (= (binary-union [?X0 ?X1]) (binary-union [?X1 ?X0]))
         (= (binary-intersection [?X0 ?X1]) (binary-intersection [?X1 ?X0]))
         (= (binary-union [(difference [?X0 ?X1]) ?X1]) ?X1)
         (= (binary-intersection [(difference [?X0 ?X1]) ?X1]) zero)))

(forall ((set ?X0) (set ?X1) (set ?X2))
    (and (= (binary-union [(binary-union [?X0 ?X1]) ?X2])
            (binary-union [?X0 (binary-union [?X1 ?X2])]))
         (= (binary-intersection [(binary-intersection [?X0 ?X1]) ?X2])
            (binary-intersection [?X0 (binary-intersection [?X1 ?X2])]))))

(forall ((set ?X0) (set ?X1))
    (=> (subset-relation [?X0 ?X1])
        (and (= (binary-union [?X0 ?X1]) ?X1)
             (= (binary-intersection [?X0 ?X1]) ?X0)
             (= (difference [?X0 ?X1]) empty))))

(forall ((set ?X))
    (and (= (binary-union [?X ?X]) ?X) (= (binary-union [?X empty]) ?X)
         (= (binary-intersection [?X ?X]) ?X) (= (binary-intersection [?X empty]) empty)
         (= (difference [?X empty]) ?X) (= (difference [?X ?X]) empty)))
```

Instances. Here are some basic sets: $\mathbf{0} = \mathsf{zero} = \emptyset$, the initial empty set; $\mathbf{1} = \mathsf{one}$, the terminal set with one element; $\mathbf{2} = \mathsf{two} = \mathbf{1} + \mathbf{1}$ and $\mathbf{3} = \mathsf{three} = \mathbf{1} + \mathbf{1} + \mathbf{1}$. zero and one have several synonyms. two and three are often used for indexing. Nothing is in zero. The canonical object in one is denoted $0 \in \mathsf{one}$. The canonical object in two, but not one, is denoted $1 \in \mathsf{two} \backslash \mathsf{one}$. The canonical object in three, but not two, is denoted $2 \in \mathsf{three} \backslash \mathsf{two}$. These three canonical objects are distinct: $0 \neq 1$, $1 \neq 2$ and $0 \neq 2$. $\mathbf{0}$ has the universal property that for any set X there is only one function $\mathbf{0} \to X$, and $\mathbf{1}$ has the universal property that for any set X there is only one function $X \to \mathbf{1}$. There is a *constant zero*

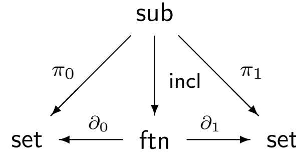
function $\Delta_{\mathsf{zero}} : \mathsf{set} \stackrel{!_{\mathsf{set}}}{\to} \mathbf{1} \stackrel{\mathsf{zero}}{\to} \mathsf{set}$, which maps any meta set X to the meta set $\mathsf{zero} = 0$. There is a *constant one* function $\Delta_{\mathsf{one}} : \mathsf{set} \stackrel{!_{\mathsf{set}}}{\to} \mathbf{1} \stackrel{\mathsf{one}}{\to} \mathsf{set}$, which maps any meta set X to the meta set $\mathsf{one} = 1$. For any type set X, there is a *counique* type function $\hat{!}_X : \emptyset \to X$ and a *unique* type function $!_X : X \to 1$. These are the unique functions between their respective sources and targets.

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

(set zero) (set initial) (= initial zero) (set empty) (= empty initial)
(forall ((zero ?x)) (not (zero ?x)))

(set one) (set terminal) (= terminal one) (one 0)
(forall ((one ?x)) (= ?x 0))

(set two) (two 0) (two 1)
(forall ((two ?x)) (or (= ?x 0) (= ?x 1)))

(set three) (three 0) (three 1) (three 2)
(forall ((three ?x)) (or (= ?x 0) (= ?x 1) (= ?x 2)))

(iff:function constant-zero)
(= (iff:source constant-zero) set)
(= (iff:target constant-zero) set)
(forall ((set ?X))
    (= (constant-zero ?X) zero))

(iff:function constant-one)
(= (iff:source constant-one) set)
(= (iff:target constant-one) set)
(forall ((set ?X))
    (= (constant-one ?X) one))

(iff:function counique)
(= (iff:source counique) set)
(= (iff:target counique) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (counique ?X)) zero)
         (= (type.ftn:target (counique ?X)) ?X)
         (forall ((type.function ?f)
                 (= (type.ftn:source ?f) zero)
                 (= (type.ftn:target ?f) ?X))
             (= ?f (counique ?X)))))

(iff:function unique)
(= (iff:source unique) set)
(= (iff:target unique) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (unique ?X)) ?X)
         (= (type.ftn:target (unique ?X)) one)
         (forall ((type.function ?f)
                 (= (type.ftn:source ?f) ?X)
                 (= (type.ftn:target ?f) one))
             (= ?f (unique ?X)))))
```

The following inclusions (subset relationships) hold for instances: $\mathsf{zero} \subseteq X$ for any set X, and $\mathsf{one} \subset \mathsf{two} \subset \mathsf{three}$.

```
(forall ((set ?X)) (subset-relation [zero ?X]))
(subset-relation [one two]) (not (= one two))
(subset-relation [two three]) (not (= two three))
```

Subobject. There is a binary *subset* relation $\subseteq$ on type sets. In the subset relationship $Y \subseteq X$, all elements of the smaller type set Y are members of the larger type set X. A pair of type sets is subordinate when it satisfies the subset relation. Let $\mathsf{sub} = \mathsf{ext}(\subseteq)$ denote the set of type subordinate pairs. We name the components of a subset relationship. There are projections $\pi_0^{\subseteq} : \mathsf{sub} \rightarrow \mathsf{set}$ and $\pi_1^{\subseteq} : \mathsf{sub} \rightarrow \mathsf{set}$ to the *smaller* and *larger* components. For each subordinate pair of type sets $Y \subseteq X$, there is an *inclusion* injection $\iota_{Y,X} = \mathsf{incl}_{Y,X} : Y \hookrightarrow X : x \mapsto x$, whose source is the smaller type set and whose target is the larger type set. The subset relation on type sets is a partial order (reflexive, antisymmetric and transitive), since identities are inclusions and inclusions are closed under composition[3].




```
(iff:set subset-relation)
(forall ((subset-relation ?YX)) (type.dgm.pr.obj:set-pair ?YX))
(forall ((set ?Y) (set ?X))
    (<=> (subset-relation [?Y ?X])
         (forall ((?Y ?x)) (?X ?x))))

(iff:function smaller)
(= (iff:source smaller) subset-relation)
(= (iff:target smaller) set)
(forall ((set ?Y) (set ?X) (subset-relation [?Y ?X]))
    (= (smaller [?Y ?X]) ?Y))

(iff:function larger)
(= (iff:source larger) subset-relation)
(= (iff:target larger) set)
(forall ((set ?Y) (set ?X) (subset-relation [?Y ?X]))
    (= (larger [?Y ?X]) ?X))

(iff:function inclusion)
(= (iff:source inclusion) subset-relation)
(= (iff:target inclusion) type.ftn:function)
```

[3]We might be interested in extending the subset relation to a subobject relation. If we say that Y is a subobject of X, we mean that there is an injection $Y \hookrightarrow X$. There could be more than one of these. However, there is at least one and we could choose and name a canonical one of these. With this assumption, there would be a function $\mathsf{inj} : \mathsf{sub} \rightarrow \mathsf{ftn}$ such that $\mathsf{inj}_{Y,X} : Y \hookrightarrow X$ is an injection for any subordinate pair (Y, X). Furthermore, we could assume that the choosen injection is the inclusion when $Y \subseteq X$. This would allow us to conservatively extend the delimitation, restriction and abridgment relations. The main problem is that the chosen injections do not strictly obey transitivity; that is, for subordinate pairs (Z, Y) and (Y, X), we would have an isomorphism $\mathsf{inj}_{Z,Y} \cdot \mathsf{inj}_{Y,X} \cong \mathsf{inj}_{Z,X}$, but not necessarily an identity. And we need an identity, to show transitivity for delimitation, restriction and abridgment.

```
(forall ((set ?Y) (set ?X) (subset-relation [?Y ?X]))
    (and (= (type.ftn:source (inclusion [?Y ?X])) ?Y)
         (= (type.ftn:target (inclusion [?Y ?X])) ?X)
         (forall ((?Y ?y))
             (= ((inclusion [?Y ?X]) ?y) ?y))))

(forall ((set ?X))
    (subset-relation [?X ?X]))
(forall ((set ?Y) (set ?X))
    (=> (and (subset-relation [?Y ?X]) (subset-relation [?X ?Y]))
        (= ?Y ?X)))
(forall ((set ?Z) (set ?Y) (set ?X))
    (=> (and (subset-relation [?Z ?Y]) (subset-relation [?Y ?X]))
        (subset-relation [?Z ?X])))
```

The subset order can be defined in terms of Boolean operations.

$$\begin{aligned} Y \subseteq X \ \text{ iff } \ & Y \cap X = Y \\ \text{iff } \ & Y \cup X = X \\ \text{iff } \ & Y \setminus X = \emptyset \end{aligned}$$

```
(forall ((set ?Y) (set ?X))
    (<=> (subset-relation [?Y ?X])
         (= (binary-intersection [?Y ?X]) ?Y)))

(forall ((set ?Y) (set ?X))
    (<=> (subset-relation [?Y ?X])
         (= (binary-union [?Y ?X]) ?X)))

(forall ((set ?Y) (set ?X))
    (<=> (subset-relation [?Y ?X])
         (= (difference [?Y ?X]) empty)))
```

For every subordinate pair $Y \subseteq X$, there is a type *predicate* $\mathsf{pred}_{Y,X} : Y \hookrightarrow X$, whose genus is X, whose differentia is Y and whose function is the inclusion $\mathsf{incl}_{Y,X} : Y \hookrightarrow X$.

```
(iff:function predicate)
(= (iff:source predicate) subordinate)
(= (iff:target predicate) predicate)
(forall ((subordinate ?YX))
    (and (= (type.pred:genus (predicate ?YX)) (smaller ?YX))
         (= (type.pred:differentia (predicate ?YX)) (larger ?YX))
         (= (type.pred:function (predicate ?YX)) (inclusion ?YX))))
```

Any subordinate pair is identical to the subordinate pair of its predicate.

```
(forall ((subset-relation ?YX))
    (= ?YX (type.pred:subordinate (predicate ?YX))))
```

For any type set X, there is a *power* type set $\wp X$ consisting of the set of all subsets of X

$$\wp X \ = \ \{Y \in \mathsf{set} \mid Y \subseteq X\}.$$

The IFF power function $\wp : \mathsf{set} \rightarrow \mathsf{set}$ is the (implicit) 10-fiber of the subset relation. It is an injection, since $\wp X_1 = \wp X_2$ implies $X_1 = \cup(\wp X_1) = \cup(\wp X_2) =$

X_2 for any two sets $X_1, X_2 \in \mathsf{set}$. For convenience, here we name the product of the pairing of the power function. For any type set X, a *power pair* is a pair of X-subsets $(Y_0, Y_1) \in \wp X^2 = \wp X \times \wp X$.

$$\begin{aligned} \wp X^0 &= \mathsf{one} \\ \wp X^1 &= \wp X \\ \wp X^2 &= \wp X \times \wp X \end{aligned}$$

```
(iff:function power)
(= (iff:source power) set)
(= (iff:target power) set)
(forall ((set ?X) ((power ?X) ?Y)) (set ?Y))
(forall ((set ?X) (set ?Y))
    (<=> ((power ?X) ?Y)
         (subset-relation [?Y ?X])))

(iff:function power-pair)
(= (iff:source power-pair) set)
(= (iff:target power-pair) set)
(forall ((set ?X))
    (= (power-pair ?X) (type.lim.pwr2.obj:power (power ?X))))
```

Let X be any type set.

- The subset order on sets restricts to the power $\wp X$. There is a binary *leq* relationship $\subseteq_X$ between pairs of X-subsets in $\wp X$. One X-subset $Y \subseteq X$ is less than or equal to another X-subset $Z \subseteq X$, denoted $Y \subseteq_X Z$, when Y is a subset of Z: $Y \subseteq_X Z$ iff $Y \subseteq Z$. This local inclusion relation is a partial order (reflexive, antisymmetric and transitive).

- There is a *bottom* X-subset $\ulcorner \bot_X \urcorner : \Delta_{\mathsf{one}}(X) = 1 \to \wp X$. The bottom X-subset $\bot$ is the X-subset $\emptyset \subseteq X$ whose inclusion is the counique function $!_X : \emptyset \hookrightarrow X$; that is, for all elements $x \in X$, not $\bot(x)$.

- There is a *top* X-subset $\ulcorner \top_X \urcorner : \Delta_{\mathsf{one}}(X) = 1 \to \wp X$. The top X-subset $\top$ is the X-subset $X \subseteq X$ whose inclusion is the identity $1_X : X \hookrightarrow X$; that is, for all elements $x \in X$, $\top(x)$.

- There is a *binary join* (*disjunction*) operation $\vee_X : \wp X \times \wp X \to \wp X$. For any two X-subset $Y \subseteq X$ and $Z \subseteq X$, the binary join is the X-subset $Y \vee Z = \{x \in X \mid x \in Y \text{ or } x \in Z\} \subseteq X$.

- There is a *binary meet* (*conjunction*) operation $\wedge_X : \wp X \times \wp X \to \wp X$. For any two X-subset $Y \subseteq X$ and $Z \subseteq X$, the binary meet is the X-subset $Y \wedge Z = \{x \in X \mid x \in Y \text{ and } x \in Z\} \subseteq X$.

- There is a *minus* operation $\backslash_X : \wp X \times \wp X \to \wp X$. For any two X-subsets $Y \subseteq X$ and $Z \subseteq X$, the minus is the X-subset $Y \backslash Z = \{x \in X \mid x \in Y \text{ or } x \notin Z\} \subseteq X$. Clearly, $Y \setminus Z = Y \wedge \neg Z$.

- There is a *complement* (*negation*) operation $\neg_X : \wp X \to \wp X$. For any X-subset $Y \subseteq X$, the complement is the X-subset $\neg Y = \{x \in X \mid x \notin Y\} \subseteq X$, the minus of the "universe" X with Y, $\neg Y = X \setminus Y$.

- There is a *implication* operation $\Rightarrow_X : \wp X \times \wp X \rightarrow \wp X$. For any two X-subsets $Y \subseteq X$ and $Z \subseteq X$, the implication is the X-subset $Y \Rightarrow Z = \{x \in X \mid x \in Y \text{ implies } x \in Z\} = \{x \in X \mid x \in Z \text{ or } x \notin Y\} \in X$. Clearly, $Y \Rightarrow Z = \neg(Y \setminus Z) = \neg(Y \wedge \neg Z) = \neg Y \vee Z$.

```
(iff:function leq)
(= (iff:source leq) set)
(= (iff:target leq) type.rel:endorelation)
(forall ((set ?X))
    (and (= (type.rel:set (leq ?X)) (power ?X))
         (forall (((power ?X) ?Y) ((power ?X) ?Z))
             (<=> ((leq ?X) ?Y ?Z)
                  (subset-relation [?Y ?Z])))))
(forall ((set ?X))
    (type.rel:partial-order (leq ?X)))
(forall ((set ?X) ((power ?X) ?Y) ((power ?X) ?Z))
    (<=> ((leq ?X) ?Y ?Z)
         (= ((binary-meet ?X) [?Y ?Z]) ?Y)))

(iff:function bottom)
(= (iff:source bottom) set)
(= (iff:target bottom) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (bottom ?X)) (constant-one ?X))
         (= (type.ftn:target (bottom ?X)) (power ?X))
         (= ((bottom ?X) iff:0) empty)))

(iff:function top)
(= (iff:source top) set)
(= (iff:target top) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (top ?X)) (constant-one ?X))
         (= (type.ftn:target (top ?X)) (power ?X))
         (= ((top ?X) iff:0) ?X)))

(iff:function binary-join)
(= (iff:source binary-join) set)
(= (iff:target binary-join) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (binary-join ?X)) (power-pair ?X))
         (= (type.ftn:target (binary-join ?X)) (power ?X))
         (forall (((power ?X) ?Y) ((power ?X) ?Z))
             (= ((binary-join ?X) [?Y ?Z]) (binary-union [?Y ?Z])))))

(iff:function binary-meet)
(= (iff:source binary-meet) set)
(= (iff:target binary-meet) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (binary-meet ?X)) (power-pair ?X))
         (= (type.ftn:target (binary-meet ?X)) (power ?X))
         (forall (((power ?X) ?Y) ((power ?X) ?Z))
             (= ((binary-meet ?X) [?Y ?Z]) (binary-intersection [?Y ?Z])))))

(iff:function minus)
(= (iff:source minus) set)
(= (iff:target minus) type.ftn:function)
(forall ((set ?X))
```

```
    (and (= (type.ftn:source (minus ?X)) (power-pair ?X))
         (= (type.ftn:target (minus ?X)) (power ?X))
         (forall (((power ?X) ?Y) ((power ?X) ?Z))
             (and (= ((minus ?X) [?Y ?Z]) (difference [?Y ?Z]))
                  (= ((minus ?X) [?Y ?Z])
                     ((binary-meet ?X) [?Y ((complement ?X) ?Z)]))))))

(iff:function complement)
(= (iff:source complement) set)
(= (iff:target complement) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (complement ?X)) (power ?X))
         (= (type.ftn:target (complement ?X)) (power ?X))
         (forall (((power ?X) ?Y))
             (and (= ((complement ?X) ?Y) (difference [?X ?Y]))
                  (= ((complement ?X) ?Y) ((minus ?X) [(top ?X) ?Y]))))))

(iff:function implication)
(= (iff:source implication) set)
(= (iff:target implication) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (implication ?X)) (power-pair ?X))
         (= (type.ftn:target (implication ?X)) (power ?X))
         (forall (((power ?X) ?Y) ((power ?X) ?Z))
             (and (= ((implication ?X) [?Y ?Z])
                     (difference [?X (difference [?Y ?Z])]))
                  (= ((implication ?X) [?Y ?Z])
                     ((complement ?X) ((difference ?X) [?Y ?Z])))
                  (= ((implication ?X) [?Y ?Z])
                     ((binary-join ?X) [((complement ?X) ?Y) ?Z]))))))
```

For any type set X, the *singleton* type function $\{\text{-}\}_X : X \rightarrow \wp X$ is defined as

$$\{x\}_X \; = \; \{x\}$$

for any element $x \in \wp X$. The singleton operation $\{\text{-}\}_X$ is an injection.

```
(iff:function singleton)
(= (iff:source singleton) set)
(= (iff:target singleton) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (singleton ?X)) ?X)
         (= (type.ftn:target (singleton ?X)) (power ?X))
         (forall ((?X ?x) (?X ?y))
             (<=> (((singleton ?X) ?x) ?y) (= ?y ?x)))
         (type.ftn:injection (singleton ?X))))
```

For any type set X, the (bounded) *union* (or *join*) type function $\cup_X : \wp\wp X \rightarrow \wp X$ is defined as

$$\cup_X(Z) \; = \; \{x \in X \mid \exists_{Y \in Z} \, x \in Y\}$$

for any family of subsets $Z \in \wp\wp X$. The union operation $\cup_X$ is a surjection, since for any $Y \in \wp X$ we have $\{Y\} \in \wp\wp X$ and $\cup_X(\{Y\}) = Y$; that is, $\exists_{\{\text{-}\}_X} \cdot \cup_X = 1_{\wp X} : \wp X \rightarrow \wp\wp X \rightarrow \wp X$.

```
(iff:function union) (iff:function join) (= join union)
(= (iff:source union) set)
(= (iff:target union) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (union ?X)) (power (power ?X)))
         (= (type.ftn:target (union ?X)) (power ?X))
         (forall (((power (power ?X)) ?Z) (?X ?x))
             (<=> (((union ?X) ?Z) ?x)
                  (exists ((?Z ?Y)) (?Y ?x))))
         (type.ftn:surjection (union ?X))))

(forall ((set ?X) ((power ?X) ?Y))
    (= (type.ftn:composition [(exists (singleton ?X)) (union ?X)])
       (type.ftn:identity (power ?X))))
```

The triple $\langle \wp, \cup, \{\text{-}\} \rangle$ forms a(n implicit) monad. This means that $\wp : \mathsf{Set} \rightarrow \mathsf{Set}$ is a(n implicit) functor, $\cup : \wp \circ \wp \Rightarrow \wp$ and $\{\text{-}\} : 1_{\mathsf{Set}} \Rightarrow \wp$ are (implicit) natural transformations, and these satisfy the associative law and two unit laws

$$\begin{aligned} \cup_{\wp(X)} \cdot \cup_X &= \wp(\cup_X) \cdot \cup_X \\ \wp(\{\text{-}\}_X) \cdot \cup_X &= 1_{\wp(X)} \\ \{\text{-}\}_{\wp(X)} \cdot \cup_X &= 1_{\wp(X)} \end{aligned}$$

for all sets $X \in \mathsf{set}$.

```
(forall ((set ?X))
    (and (= (type.ftn:composition [(union (power ?X)) (union ?X)])
            (type.ftn:composition [(type.ftn:power (union ?X)) (union ?X)]))
         (= (type.ftn:composition [(type.ftn:power (singleton ?X)) (union ?X)])))
            (type.ftn:identity (power ?X)))
         (= (type.ftn:composition [(singleton (power ?X)) (union ?X)])))
            (type.ftn:identity (power ?X)))))
```

For any type set X, the *intersection* (or *meet*) type function $\cap_X : \wp\wp X \rightarrow \wp X$ is defined as

$$\cap_X(Z) \;=\; \{x \in X \mid \forall_{Y \in Z}\, x \in Y\}$$

for any family of subsets $Z \in \wp\wp X$. The intersection operation $\cap_X$ is a surjection, since for any $Y \in \wp X$ we have $\{Y\} \in \wp\wp X$ and $\cap_X(\{Y\}) = Y$; that is, $\exists_{\{\text{-}\}_X} \cdot \cap_X = 1_{\wp X} : \wp X \rightarrow \wp\wp X \rightarrow \wp X$.

```
(iff:function intersection) (iff:function meet) (= meet intersection)
(= (iff:source intersection) set)
(= (iff:target intersection) type.ftn:function)
(forall ((set ?X))
    (and (= (type.ftn:source (intersection ?X)) (power (power ?X)))
         (= (type.ftn:target (intersection ?X)) (power ?X))
         (forall (((power (power ?X)) ?Z) (?X ?x))
             (<=> (((intersection ?X) ?Z) ?x)
                  (forall ((?Z ?Y)) (?Y ?x))))
         (type.ftn:surjection (intersection ?X))))

(forall ((set ?X) ((power ?X) ?Y))
    (= (type.ftn:composition [(exists (singleton ?X)) (intersection ?X)])
       (type.ftn:identity (power ?X))))
```

Let $f : X \rightarrow Y$ be any type function. There is an internal *exist*ential quantification, (direct) image or (*power*) type function $\exists_f : \wp X \rightarrow \wp Y$ defined by

$$y \in \exists_f A \;\equiv\; \left\{ \begin{array}{l} \text{“there exists something } x \text{ in } X \\ \text{that satisfies } x \in A \textit{ and} \\ \text{goes to } y \text{ under } f : f(x) = y\text{”} \end{array} \right.$$

or

$$\exists_f(A) \;=\; \{y \in Y \mid \exists_{x \in X}(x \in A \wedge f(x) = y)\}$$

for any source subset $A \subseteq X$. There is an internal *inverse image* type function

$$f^{-1} : \wp Y \rightarrow \wp X$$

defined by

$$\begin{array}{rcl} f^{-1}(B) & = & \{x \in X \mid \exists_{y \in B}(f(x) = y)\} \\ & = & \{x \in X \mid f(x) \in B\} \end{array}$$

for any target subset $B \subseteq Y$. There is an internal universal quantification (*forall*) type function $\forall_f : \wp X \rightarrow \wp Y$ defined by

$$y \in \forall_f A \;\equiv\; \left\{ \begin{array}{l} \text{“for all things } x \text{ in } X \\ \text{for which } f(x) = y, \\ x \in A \text{ holds”} \end{array} \right.$$

or

$$\forall_f(A) \;=\; \{y \in Y \mid \forall_{x \in X}(f(x) = y \Rightarrow A(x))\}$$

for any source subset $A \subseteq X$. The existential (inverse image, universal) function is monotonic. The existential quantification operation is functorial: the existential quantification of the identity $1_X : X \rightarrow X$ is the identity of the power, $\exists_{1_X} = 1_{\wp X} : \wp X \rightarrow \wp X$; and the existential quantification of the composition $f \cdot g : X \rightarrow Y \rightarrow Z$ is the composition of the existential quantifications, $\exists_{f \cdot g} = \exists_f \cdot \exists_g : \wp X \rightarrow \wp Z$. The inverse image (universal quantification) operation is also functorial. In the diagram

$$\wp X \;\; \begin{array}{c} \xrightarrow{\exists_f} \\ \xleftarrow[f^{-1}]{} \\ \xrightarrow[\forall_f]{} \end{array} \;\; \wp Y$$

we have the adjunctions $\exists_f \dashv f^{-1} \dashv \forall_f$; that is, the existential quantification is left adjoint to the inverse image and the inverse image is left adjoint to the universal quantification.

```
(iff:function exists) (iff:function power) (= power exists)
(= (iff:source exists) type.ftn:function)
(= (iff:target exists) type.ftn:function)
(forall ((type.ftn:function ?f))
    (and (= (type.ftn:source (exists ?f)) (power (type.ftn:source ?f)))
         (= (type.ftn:target (exists ?f)) (power (type.ftn:target ?f)))
         (forall (((power (type.ftn:source ?f)) ?A) ((type.ftn:target ?f) ?y))
```

```
            (<=> (((exists ?f) ?A) ?y)
                 (exists (((type.ftn:source ?f) ?x))
                     (and (?A ?x) (= (?f ?x) ?y)))))))

(iff:function inverse-image)
(= (iff:source inverse-image) type.ftn:function)
(= (iff:target inverse-image) type.ftn:function)
(forall ((type.ftn:function ?f))
    (and (= (type.ftn:source (inverse-image ?f)) (power (type.ftn:target ?f)))
         (= (type.ftn:target (inverse-image ?f)) (power (type.ftn:source ?f)))
         (forall (((power (type.ftn:target ?f)) ?B) ((type.ftn:source ?f) ?x))
             (<=> (((inverse-image ?f) ?B) ?x)
                  (?B (?f ?x))))))

(iff:function forall)
(= (iff:source forall) type.ftn:function)
(= (iff:target forall) type.ftn:function)
(forall ((type.ftn:function ?f))
    (and (= (type.ftn:source (forall ?f)) (power (type.ftn:source ?f)))
         (= (type.ftn:target (forall ?f)) (power (type.ftn:target ?f)))
         (forall (((power (type.ftn:source ?f)) ?A) ((type.ftn:target ?f) ?y))
             (<=> (((forall ?f) ?A) ?y)
                  (forall (((type.ftn:source ?f) ?x))
                      (=> (= (?f ?x) ?y) (?A ?x)))))))

(forall ((type.ftn:function ?f))
    (and (forall (((power (type.ftn:source ?f)) ?X0)
                  ((power (type.ftn:source ?f)) ?X1)
                 (subset-relation [?X0 ?X1]))
             (subset-relation [((exists ?f) ?X0) ((exists ?f) ?X1)]))
         (forall (((power (type.ftn:target ?f)) ?Y0)
                  ((power (type.ftn:target ?f)) ?Y1)
                 (subset-relation [?Y0 ?Y1]))
             (subset-relation [((inverse-image ?f) ?Y0) ((inverse-image ?f) ?Y1)]))
         (forall (((power (type.ftn:source ?f)) ?X0)
                  ((power (type.ftn:source ?f)) ?X1)
                 (subset-relation [?X0 ?X1]))
             (subset-relation [((forall ?f) ?X0) ((forall ?f) ?X1)]))))

(forall ((type.ftn:function ?f))
    (and (forall (((power (type.ftn:source ?f)) ?X)
                  ((power (type.ftn:target ?f)) ?Y))
             (<=> (subset-relation [((exists ?f) ?X) ?Y])
                  (subset-relation [?X ((inverse-image ?f) ?Y)])))
         (forall (((power (type.ftn:source ?f)) ?X)
                  ((power (type.ftn:target ?f)) ?Y))
             (<=> (subset-relation [((inverse-image ?f) ?Y) ?X])
                  (subset-relation [?Y ((forall ?f) ?X)])))))

(forall ((set ?X))
    (= (exists (type.ftn:identity ?X)) (type.ftn:identity (power ?X))))

(forall ((type.ftn:function ?f) (type.ftn:function ?g) (composable-pair [?f ?g]))
    (= (exists (type.ftn:composition [?f ?g]))
       (type.ftn:composition [(exists ?f) (exists ?g)])))
```

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.