

The IFF Type Namespace

Robert E. Kent

October 30, 2008

Contents

1	The Structural Component	2
1.1	Introduction	2
1.2	Type Sets	9
0.2	Type Functions	14
0.2.1	Function Morphisms	28
1.4	Type Spans	40
1.4.1	Type Endospans	55
1.4.2	Span Morphisms	57
1.5	Type Predicates	65
1.6	Type Relations	73
1.6.1	Type Endorelations	85

1.2 Type Sets

Basics. There is an IFF object set of all type *sets*.

```
iff.obj:object set;
```

There is also an IFF object set² of all type *set pairs*.

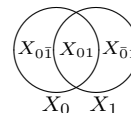
```
iff.obj:object set-pair = iff.prd2:product(set,set);
iff.mor:morphism set-pair twist(set-pair) = iff.prd2:twist(set,set);
iff.mor:morphism set set0 (set-pair) = iff.prd2:projection0(set,set);
iff.mor:morphism set set1 (set-pair) = iff.prd2:projection1(set,set);
```

There is a binary *isomorphic* endorelation between pairs of type sets, that are linked by a bijection². 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.rel:equivalence-relation isomorphic(set);
```

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.mor:morphism set binary-union (set-pair);
iff.mor:morphism set binary-intersection (set-pair);
iff.mor:morphism set difference (set-pair);
```

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

$$\begin{aligned} X_0 &\subseteq X_0 \cup X_1 \\ X_1 &\subseteq X_0 \cup X_1 \\ X_0 \cap X_1 &\subseteq X_0 \\ X_0 \cap X_1 &\subseteq X_1 \\ X_0 \setminus X_1 &\subseteq X_0 \end{aligned}$$

```
subset(set0,binary-union)
subset(set1,binary-union)
subset(binary-intersection,set0)
subset(binary-intersection,set1)
subset(difference,set0)
```

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

²Here we follow and quote from the discussion on the topic of isomorphism 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.

```

binary-union      = twist ∘ binary-union;
binary-intersection = twist ∘ binary-intersection;
⟨binary-union, 1(set-pair)⟩ ∘ binary-union
= ⟨1(set-pair), binary-union⟩ ∘ binary-union;
⟨binary-intersection, 1(set-pair)⟩ ∘ binary-intersection
= ⟨1(set-pair), binary-intersection⟩ ∘ binary-intersection;

( $X_0 \setminus X_1$ ) ∪  $X_0$  =  $X_0$ 
( $X_0 \setminus X_1$ ) ∩  $X_0$  = ( $X_0 \setminus X_1$ )
( $X_0 \setminus X_1$ ) ∪  $X_1$  =  $X_0 \cup X_1$ 
( $X_0 \setminus X_1$ ) ∩  $X_1$  =  $\emptyset$ 

⟨difference, set0⟩ ∘ binary-union      = set0;
⟨difference, set0⟩ ∘ binary-intersection = difference;
⟨difference, set1⟩ ∘ binary-union      = binary-union;
⟨difference, set1⟩ ∘ binary-intersection = iff:unique(set-pair) ∘ empty;

if  $X \subseteq Y$ ,
then  $X \cup Y = Y$ ,  $X \cap Y = X$  and  $X \setminus Y = \emptyset$ 
hence,  $X \cup X = X$ ,  $X \cup \emptyset = X$ ,
 $X \cap X = X$ ,  $X \cap \emptyset = \emptyset$ ,
 $X \setminus X = \emptyset$ ,  $X \setminus \emptyset = X$ 

subclusion ∘ binary-union      = larger;
subclusion ∘ binary-intersection = smaller;
subclusion ∘ difference        = iff:unique(set-pair) ∘ empty;
⟨1(set),1(set)⟩ ∘ binary-union      = 1(set);
⟨1(set),1(set)⟩ ∘ binary-intersection = 1(set);
⟨1(set),1(set)⟩ ∘ difference        = constant-zero;
⟨1(set),constant-zero⟩ ∘ binary-union      = 1(set);
⟨1(set),constant-zero⟩ ∘ binary-intersection = constant-zero;
⟨1(set),constant-zero⟩ ∘ difference        = 1(set);

```

Instances. Here are some basic sets: $\mathbf{0}$ = zero = \emptyset , the initial empty set; $\mathbf{1}$ = one, the terminal set with one element; $\mathbf{2}$ = two = $\mathbf{1} + \mathbf{1}$ and $\mathbf{3}$ = three = $\mathbf{1} + \mathbf{1} + \mathbf{1}$. zero and one have several synonyms. two and three are often used for indexing. $\mathbf{0}$ has the universal property that for any set X there is only one function $\mathbf{0} \rightarrow X$, and $\mathbf{1}$ has the universal property that for any set X there is only one function $X \rightarrow \mathbf{1}$. The following inclusions (subset relationships) hold for instances: $\text{zero} \subseteq X$ for any set X , and $\text{one} \subset \text{two} \subset \text{three}$.

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

```

iff.mor:element set zero = type.col:initial = initial = empty;
iff.mor:element set one  = type.lim:terminal = terminal;
iff.mor:element set two  = ⟨one,one⟩ ∘ type.copr2.obj:coproduct;
iff.mor:element set three = ⟨one,one,one⟩ ∘ type.copr3.obj:coproduct;
subset(constant-zero,1(set)); subset(one,two); subset(two,three);
one ≠ two; two ≠ three;

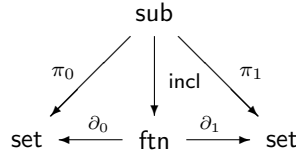
```

```

iff.mor:morphism set constant-zero (set) = iff.unique(set) ◦ zero;
iff.mor:morphism set constant-one (set) = iff.unique(set) ◦ one;
iff.mor:morphism type.ftn:function counique (set) = type.colim:counique;
counique ◦ type.ftn:source = constant-zero;
counique ◦ type.ftn:target = 1(set);
iff.mor:morphism type.ftn:function unique (set) = type.lim:unique;
unique ◦ type.ftn:source = 1(set);
unique ◦ type.ftn:target = constant-one;

```

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 $\text{sub} = \text{ext}(\subseteq)$ denote the set of type subordinate pairs. We name the components of a subset relationship. There are projections $\pi_0^{\subseteq} : \text{sub} \rightarrow \text{set}$ and $\pi_1^{\subseteq} : \text{sub} \rightarrow \text{set}$ to the *smaller* and *larger* components. For each subordinate pair of type sets $Y \subseteq X$, there is an *inclusion* injection $\iota_{Y,X} = \text{incl}_{Y,X} : Y \hookrightarrow X : x \mapsto x$, 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 composition³.



```

iff.rel:partial-order subset (set);
iff.obj:object subordinate = iff.rel:extent(subset);
iff.mor:morphism set smaller (subordinate) = iff.rel:projection0(subset);
iff.mor:morphism set larger (subordinate) = iff.rel:projection1(subset);
iff.mor:morphism set-pair subclusion (subordinate) = iff.rel:monomorphism(subset);
iff.mor:morphism type.ftn:function inclusion (subordinate);
inclusion ◦ type.ftn:source = smaller;
inclusion ◦ type.ftn:target = larger;

```

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

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

³We 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 $\text{inj} : \text{sub} \rightarrow \text{ftn}$ such that $\text{inj}_{Y,X} : Y \hookrightarrow X$ is an injection for any subordinate pair (Y, X) . Furthermore, we could assume that the chosen 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 $\text{inj}_{Z,Y} \cdot \text{inj}_{Y,X} \cong \text{inj}_{Z,X}$, but not necessarily an identity. And we need an identity, to show transitivity for delimitation, restriction and abridgment.

```

subordinate = iff.equ:equalizer(binary-union,set1);
subordinate = iff.equ:equalizer(binary-intersection,set0);
subordinate = iff.equ:equalizer(difference,iff.unique(set-pair) o zero);

```

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

```

iff.mor:morphism type.pred:predicate predicate (subordinate);
predicate o type.pred:genus      = larger;
predicate o type.pred:differentia = smaller;
predicate o type.pred:function   = inclusion;

```

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

```

predicate o type.pred:subordinate = 1(subordinate);

```

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

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

The IFF power function $\wp : \text{set} \rightarrow \text{set}$ is the (implicit) 10-fiber of the subset relation. It is a monomorphism, since $\wp X_1 = \wp X_2$ implies $X_1 = \cup(\wp X_1) = \cup(\wp X_2) = X_2$ for any two sets $X_1, X_2 \in \text{set}$.

```

iff.mor:monomorphism set power (set);

```

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

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

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

```

iff.mor:morphism type.ftn:function singleton (set);
singleton o type.ftn:source = 1(set);
singleton o type.ftn:target = power;
(type.ftn:injective singleton);

```

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, $\exists_{\{-\}_X} \cdot \cup_X = 1_{\wp X} : \wp X \rightarrow \wp \wp X \rightarrow \wp X$.

```

iff.mor:morphism type.ftn:function union (set);
union o type.ftn:source = power o power;
union o type.ftn:target = power;
(type.ftn:surjective union);

```

```

[singleton o exists, union] = [power];

```

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

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

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

```
[power o union, union] = [union o type.ftn:power, union];
[singleton o type.ftn:power, union] = [power];
[power o singleton, union] = [power];
```

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, $\exists_{\{-\}_X} \cdot \cap_X = 1_{\wp X} : \wp X \rightarrow \wp\wp X \rightarrow \wp X$.

```
iff.mor:morphism type.ftn:function intersection) (set) = meet;
intersection o type.ftn:source = power o power;
intersection o type.ftn:target = power;
(type.ftn:surjective intersection);
```

```
[singleton o exists, intersection] = [power];
```

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

$$y \in \exists_f A \equiv \begin{cases} \text{“there exists something } x \text{ in } X \\ \text{that satisfies } x \in A \text{ and} \\ \text{goes to } y \text{ under } f : f(x) = y\text{”} \end{cases}$$

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

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

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

$$y \in \forall_f A \equiv \begin{cases} \text{“for all things } x \text{ in } X \\ \text{for which } f(x) = y, \\ x \in A \text{ holds”} \end{cases}$$

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

$$\begin{array}{ccc} & \xrightarrow{\exists_f} & \\ \wp X & \xleftarrow{f^{-1}} & \wp Y \\ & \xrightarrow{\forall_f} & \end{array}$$

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.mor:morphism type.ftn:function exists (type.ftn:function);
exists o type.ftn:source = type.ftn:source o power;
exists o type.ftn:target = type.ftn:target o power;

iff.mor:morphism type.ftn:function inverse-image (type.ftn:function);
inverse-image o type.ftn:source = type.ftn:target o power;
inverse-image o type.ftn:target = type.ftn:source o power;

iff.mor:morphism type.ftn:function forall (type.ftn:function);
forall o type.ftn:source = type.ftn:source o power;
forall o type.ftn:target = type.ftn:target o power;

subset([type.ftn:source], [exists, inverse-image]);
subset([inverse-image, exists], [type.ftn:target]);
[[exists, inverse-image], exists] = [type.ftn:source];
[[inverse-image, exists], inverse-image] = [type.ftn:target];

subset([type.ftn:target], [inverse-image, forall]);
subset([forall, inverse-image], [type.ftn:source]);
[[inverse-image, forall], forall] = [type.ftn:target];
[[forall, inverse-image], forall] = [type.ftn:source];

```

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.