

# The IFF Type Namespace

Robert E. Kent

December 18, 2007

## Contents

|          |                              |          |
|----------|------------------------------|----------|
| <b>1</b> | <b>The Type Kernel</b>       | <b>2</b> |
| 1.1      | Introduction . . . . .       | 2        |
| 1.2      | Type Sets . . . . .          | 9        |
| 1.3      | Type Functions . . . . .     | 21       |
| 1.3.1    | Function Morphisms . . . . . | 35       |
| 1.4      | Type Spans . . . . .         | 40       |
| 1.4.1    | Type Endospans . . . . .     | 55       |
| 1.4.2    | Span Morphisms . . . . .     | 57       |
| 1.5      | Type Predicates . . . . .    | 65       |
| 1.6      | Type Relations . . . . .     | 73       |
| 1.6.1    | Type Endorelations . . . . . | 85       |

|   |   |   |
|---|---|---|
| ordinary elements<br>$x \in X$                |   | generalized elements<br>$Y \xrightarrow{x} X$ |
| predicates as inclusions<br>$ p  \subseteq X$ | predicates as injections<br>$ p  \xrightarrow{p} X$ |   |
| predicate membership in terms of              |   |   |
| set membership                                | function application                                | function composition                          |
| $x \in  p $                                   | $p(x)$ when<br>$\exists y \in  p  p(y) = x$         | $\exists y : Y \rightarrow  p  y \cdot p = x$ |

Table 7: Predicates, Elements and Membership

## 1.5 Type Predicates

**Introduction.** Just as sets represent the nouns in natural language expressions, so also predicates represent the adjectives. In the expression “Michael is a young person”, the adjective “young” modifies the noun “person”. The semantics of this expression consists of a set of persons, a subset consisting of young people and an individual asserted to be a member of that subset. Predicates, elements and predicate membership have several representations in mathematics and knowledge engineering (Table 7). Predicates can be represented as subsets (inclusions) or injective functions, elements can be represented as ordinary (global) elements or generalized elements (functions), and the representation of predicate membership varies accordingly. The IFF symbolism for predicate membership, both for ordinary elements and for generalized elements, is defined in this namespace<sup>11</sup>.

**Basics.** There is a collection of all type *predicates*. Predicates are also called *parts*. The symbol ‘`predicate`’ is used to declare a type predicate. Conceptually, a type predicate is an injective type function, hence an injective IFF function. But practically, in order to parse the defined syntactic construct of predicate membership, we require the collection of type predicates to be disjoint from the collections of type sets and type functions. The collection of all type predicates is an IFF set. A type predicate can be neither a type set nor a type function. These three collections are pairwise disjoint. The collections of type sets and type functions already inherit their disjointness from the collection of IFF sets and IFF functions.

```
(iff:set predicate)
(forall ((predicate ?p))
  (and (not (type.set:set ?p))
       (not (type.ftn:function ?p))))
```

<sup>11</sup>Given any type predicate  $p : X$ , the IFF symbolism for predicate membership is `(p x)` for ordinary elements  $x \in X$  and `(member x p)` for generalized elements (functions  $x : Y \rightarrow X$ ).

Each type predicate  $p : |p| \hookrightarrow X$  has a unique *differentia* type set  $\delta(p) = \text{diff}(p) = |p|$  and a unique *genus* type set  $\gamma(p) = \text{gen}(p) = X$ . The predicate notation  $p : X$  shows a predicate  $p$  with genus  $X$ .

```
(iff:function differentia)
(= (iff:source differentia) predicate)
(= (iff:target differentia) type.set:set)
```

```
(iff:function genus)
(= (iff:source genus) predicate)
(= (iff:target genus) type.set:set)
```

A predicate with genus  $X$  generalizes a subset of  $X$ . We can recapture subsets with strictness. A predicate is *strict* when the differentia is a subset of the genus. For strict predicates, the associated injective function is an inclusion (of the differentia into the genus).

```
(iff:set strict-predicate)
(forall ((strict-predicate ?p)) (predicate ?p))
(forall ((predicate ?p))
  (<=> (strict-predicate ?p)
    (type.set:subset (differentia ?p) (genus ?p))))
(forall ((strict-predicate ?p))
  (= (function ?p) (type.set:inclusion [(differentia ?p) (genus ?p)])))
```

For each type predicate  $p : |p| \hookrightarrow X$  there is a *canonically* strict predicate  $[p] : \rho(p) \subseteq X$  of the same genus, whose differentia is the range of the injective function of  $p$ .

```
(iff:function canon)
(= (iff:source canon) predicate)
(= (iff:target canon) predicate)
(forall ((predicate ?p))
  (= (differentia (canon ?p)) (type.ftn:range (function ?p)))
  (= (genus (canon ?p)) (genus ?p))
  (= (function (canon ?p)) (type.ftn:injective-factor (function ?p))))
```

There is a binary *delimitation* relationship  $\leq$  between pairs of type predicates. One (*smaller*) type predicate  $|p| \xrightarrow{p} X$  is the delimitation of another (*larger*) type predicate  $|q| \xrightarrow{q} Y$ , denoted  $p \leq q$ , when (1) the genus of  $p$  is a subset of the genus of  $q$ ,  $X \hookrightarrow Y$ , (2) the differentia of  $p$  is a subset of the differentia of  $q$ ,  $|p| \hookrightarrow |q|$ , and (3) the function of the predicate  $p$  is the optimal restriction of the function of the predicate  $q$ . When both predicates are strict,  $p$  is the delimitation of  $q$  *iff* for all  $x \in X$ ,  $p(x)$  iff  $q(x)$ . The delimitation endorelation is a partial order (reflexive, antisymmetric and transitive).

$$\begin{array}{ccc}
 |p| & \xhookrightarrow{p} & X \\
 \downarrow & & \downarrow \\
 |q| & \xhookrightarrow{q} & Y
 \end{array}
 \quad \lrcorner$$

```
(iff:set delimitation-relation)
(forall ((delimitation-relation ?pq))
  (exists ((predicate ?p) (predicate ?q))
```

For each type predicate  $p : X$ , the differentia embeds as the subset of the genus  $\text{diff}(p) \hookrightarrow \text{gen}(p)$ , consisting of those (ordinary) elements  $x \in X$  satisfying the predicate:  $p(x)$  iff  $\exists_{y \in \text{diff}(p)} p(y) = x$ . Hence, we introduce into the IFF the predicate holds statement ‘ $(p \ x)$ ’. If the symbols ‘ $p$ ’ and ‘ $X$ ’ represent the two IFF things  $p$  and  $X$ , then the code

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

makes the declaration “ $p : Y \hookrightarrow X$ ”<sup>a</sup>, and the code

```
(X x)
```

expresses the statement that “ $x \in X$ ”. All of this follows standard IFF syntax, which, until now, was expressed in terms of set membership and function application. However, the following code

```
(p x)
```

is new. Here the symbol ‘ $p$ ’ is neither a set nor a function, and hence we can use neither set membership nor function application to define this. The predicate holds expression ‘ $(p \ x)$ ’, which states that ‘ $x$ ’ satisfies ‘ $p$ ’, serves as a shorthand for the code

```
(exists ((Y ?y))
 (= ((function p) ?y) x))
```

This follows standard IFF syntax, since it is expressed only in terms of set membership and function application. Hence, the following equivalence holds anywhere in the IFF

```
(forall ((predicate ?p) ((genus ?p) ?x))
 (<=> (?p ?x)
 (exists (((differentia ?p) ?y))
 (= ((function ?p) ?y) ?x))))
```

The notation ‘ $(p \ x)$ ’ follows the prescription: *all IFF predicates are unary*. Hence, the following nullary, binary, ternary and higher arity expressions are iff-formed

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

---

<sup>a</sup>equivalently, “ $p \in \text{pred}, \gamma(p) = X, \delta(p) = Y$ ”

Table 8: IFF Predicate Notation

```

      (= (?pq [?p ?q])))
    (forall ((predicate ?p) (predicate ?q))
      (<=> (delimitation-relation [?p ?q])
        (and (type.set:subset-relation [(genus ?p) (genus ?q)])
              (type.set:subset-relation [(differentia ?p) (differentia ?q)])
              (type.ftn:optimal-restriction-relation [(function ?p) (function ?q)]))))

    (iff:function smaller)
    (= (iff:source smaller) delimitation-relation)
    (= (iff:target smaller) predicate)
    (forall ((predicate ?p) (predicate ?q) (delimitation-relation [?p ?q]))
      (= (smaller [?p ?q] ?p))

    (iff:function larger)
    (= (iff:source larger) delimitation-relation)
    (= (iff:target larger) predicate)
    (forall ((predicate ?p) (predicate ?q) (delimitation-relation [?p ?q]))
      (= (larger [?p ?q] ?q))

    (forall ((strict-predicate ?p) (strict-predicate ?q)
            (type.set:subset-relation [(genus ?p) (genus ?q)]))
      (<=> (delimitation-relation [?p ?q])
        (forall ((genus ?p) ?x)
          (<=> (?p ?x) (?q ?x)))))

    (forall ((predicate ?p))
      (delimitation-relation [?p ?p]))
    (forall ((predicate ?p) (predicate ?q))
      (=> (and (delimitation-relation [?p ?q]) (delimitation-relation [?q ?p]))
        (= ?p ?q))
    (forall ((predicate ?p1) (predicate ?p2) (predicate ?p3))
      (=> (and (delimitation-relation [?p1 ?p2]) (delimitation-relation [?p2 ?p3]))
        (delimitation-relation [?p1 ?p3])))

```

**Conversion.** A part (predicate) of a set  $X$  has an associated type injective *function* (more generally, a monomorphism)  $\varphi(p) : |p| \hookrightarrow X$ . The source (target) of the injective function is the differentia (genus) of the predicate  $p$ .

```

(iff:function function)
(= (iff:source function) predicate)
(= (iff:target function) type.ftn:function)
(forall ((predicate ?p))
  (and (= (type.ftn:source (function ?p)) (differentia ?p))
        (= (type.ftn:target (function ?p)) (genus ?p))
        (type.ftn:injection (function ?p))))

(forall ((predicate ?p1) (predicate ?p2))
  (=> (= (function ?p1) (function ?p2)) (= ?p1 ?p2)))

```

The canon of any predicate is equal to the predicate of its injective function,  $[p] = \pi(\varphi(p))$ .

```

(forall ((predicate ?p))
  (= (canon ?p) (type.ftn:predicate (function ?p))))

```

Any predicate  $p : Y \hookrightarrow X$  has an associated *subordinate* pair  $\sigma(p) = (\rho(p), X)$ , where  $\rho(p) \cong Y$  is the range of the injective function  $p$ . This is part of the

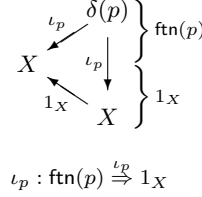


Figure 11: Injection of a Predicate

fact that the canonical restriction of predicates on genus  $X$  is isomorphic to the powerset of  $X$ .

```
(iff:function subordinate)
(= (iff:source subordinate) predicate)
(= (iff:target subordinate) type.set:subset-relation)
(forall ((predicate ?p))
  (and (= (type.set:smaller (subordinate ?p)) (type.ftn:range (function ?p)))
        (= (type.set:larger (subordinate ?p)) (genus ?p))
        (= (type.set:inclusion (subordinate ?p))
            (type.ftn:injective-factor (function ?p)))))
```

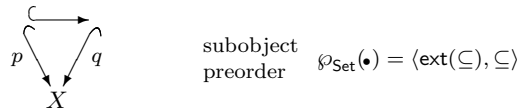
The canon of any predicate is equal to the predicate of its subordinate pair,  $[p] = \pi(\sigma(p))$ .

```
(forall ((predicate ?p))
  (= (canon ?p) (type.set:predicate (subordinate ?p))))
```

For any predicate  $p : X$ , there is an *injection* function 2-cell  $\iota_p : \text{ftn}(p) \Rightarrow 1_X$  (Figure 11), whose source  $\text{ftn}(p)$  is the function (injection) of a predicate, whose target is the terminal (predicative) function  $1_X$ , and whose function  $\iota_p : \delta(p) \hookrightarrow X$  is the injection of  $p$ .

```
(iff:function injection)
(= (iff:source injection) predicate)
(= (iff:target injection) type.ftn.mor:2-cell)
(forall ((predicate ?p))
  (and (= (type.ftn.mor:source (injection ?p)) (function ?p))
        (= (type.ftn.mor:target (injection ?p)) (type.ftn:terminal (genus ?p)))
        (= (type.ftn.mor:function (injection ?p)) (function ?p))))
```

**Subobject.** For any two type predicates  $p : X$  and  $q : Y$ ,  $p$  is included in  $q$ , denoted by  $p \subseteq q$ , when (the function of)  $p$  belongs to (the function of)  $q$ . When  $p$  is included in  $q$ , the genus of  $p$  is the genus of  $q$ ,  $X = Y$ . The inclusion endorelation on predicates is a preorder (reflexive and transitive). We name the component *parts* of an inclusion relationship; that is, there are projections  $\pi_0^{\subseteq}, \pi_1^{\subseteq} : \text{ext}(\subseteq) \rightarrow \text{pred}$  to the component parts.



```

(iff:set inclusion-relation)
(forall ((inclusion-relation ?pq))
  (exists ((predicate ?p) (predicate ?q))
    (= ?pq [?p ?q])))
(forall ((predicate ?p) (predicate ?q))
  (<=> (inclusion-relation [?p ?q])
    (type.ftn:belonging [(function ?p) (function ?q)])))

(forall ((predicate ?p) (predicate ?q))
  (=> (inclusion-relation [?p ?q])
    (= (genus ?p) (genus ?q))))

(forall ((predicate ?p))
  (inclusion-relation [?p ?p]))
(forall ((predicate ?p) (predicate ?q) (predicate ?r))
  (=> (and (inclusion-relation [?p ?q]) (inclusion-relation [?q ?r]))
    (inclusion-relation [?p ?r])))

(iff:function part0)
(= (iff:source part0) inclusion-relation)
(= (iff:target part0) predicate)
(forall ((predicate ?p) (predicate ?q) (inclusion-relation [?p ?q]))
  (= (part0 [?p ?q]) ?p))

(iff:function part1)
(= (iff:source part1) inclusion-relation)
(= (iff:target part1) predicate)
(forall ((predicate ?p) (predicate ?q) (inclusion-relation [?p ?q]))
  (= (part1 [?p ?q]) ?q))

```

The inclusion relation on predicates generalizes the inclusion relation on the powerset of  $X$ . We can recapture the latter through the canonically strict associate: for any two type predicates  $p$  and  $q$ ,  $p \subseteq q$  iff  $[p] \subseteq [q]$ .

```

(forall ((predicate ?p) (predicate ?q))
  (<=> (inclusion-relation [?p ?q])
    (inclusion-relation [(canon ?p) (canon ?q)])))

```

For any two type predicates  $p$  and  $q$ ,  $p$  is *equivalent* or *isomorphic* to  $q$ , denoted by  $p \equiv q$ , when  $p$  and  $q$  are included in each other,  $p \subseteq q$  and  $q \subseteq p$ . When  $p$  is equivalent to  $q$ , the differentia of  $p$  is isomorphic to the differentia of  $q$ ,  $|p| \cong |q|$ , and  $p$  and  $q$  factor through each other via the inverses. For any two type predicates  $p$  and  $q$ ,  $p$  is equivalent to  $q$  iff the canonically strict associates are equal:  $p \equiv q$  iff  $[p] = [q]$ . The equivalence endorelation on type predicates is an equivalence relation (reflexive, symmetric and transitive). Note: for any type predicate  $p$ , the collection of type predicates equivalent to  $p$ , the equivalence class  $[p]$ , is not necessarily locally small.

```

(iff:set equivalence) (iff:set isomorphism) (= isomorphism equivalence)
(forall ((equivalence ?pq))
  (exists ((predicate ?p) (predicate ?q))
    (= ?pq [?p ?q])))
(forall ((predicate ?p) (predicate ?q))
  (<=> (equivalence [?p ?q])
    (and (inclusion-relation [?p ?q])
      (inclusion-relation [?q ?p]))))
(forall ((predicate ?p) (predicate ?q))
  (<=> (equivalence [?p ?q])
    (and (inclusion-relation [?p ?q])
      (inclusion-relation [?q ?p]))))

```

```

(= (canon ?p) (canon ?q)))

(forall ((predicate ?p))
  (equivalence [?p ?p]))
(forall ((predicate ?p) (predicate ?q))
  (=> (equivalence [?p ?q]) (equivalence [?q ?p])))
(forall ((predicate ?p) (predicate ?q) (predicate ?r))
  (=> (and (equivalence [?p ?q]) (equivalence [?q ?r]))
      (equivalence [?p ?r])))

```

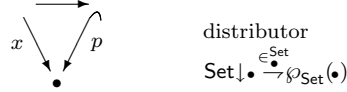
Any predicate is equivalent to its canonically strict associate,  $p \equiv [p]$ . Hence, for any type predicate  $p$ , the equivalence class of  $p$  has the canon as canonical representative.

```

(forall ((predicate ?p))
  (equivalence [?p (canon ?p)]))

```

For any type function (generalized element)  $x \xrightarrow{\bullet} \bullet$  and any type predicate (part)  $\xrightarrow{p} \bullet$ ,  $x$  is a *member* of  $p$ ,  $x \in p$ , when  $x$  belongs to (the function of)  $p$ ; that is, when there is a proof function  $h$  such that  $x = h \cdot p$ . As noted when defining the belonging relation, the proof function  $h$  is unique. The membership relation from type functions to type predicates is closed with respect to function belonging on the left and predicate inclusion on the right. We name the component *function* and *part* of a membership relationship; that is, there are projections  $\pi_0^\in \text{ : ext}(\in) \rightarrow \text{ftn}$  and  $\pi_1^\in \text{ : ext}(\in) \rightarrow \text{pred}$  to the components.



```

(iff:set membership)
(forall ((membership ?xp))
  (exists ((type.ftn:function ?x) (predicate ?p))
    (= ?xp [?x ?p])))
(forall ((type.ftn:function ?x) (predicate ?p))
  (<=> (membership [?x ?p])
      (type.ftn:belonging [?x (function ?p)])))

(forall ((type.ftn:function ?x) (predicate ?p) (membership [?x ?p]))
  (and (forall ((type.ftn:function ?y) (type.ftn:belonging [?y ?x]))
        (membership [?y ?p]))
      (forall ((predicate ?q) (inclusion-relation [?p ?q]))
        (membership [?x ?q]))))

(iff:function element)
(= (iff:source element) membership)
(= (iff:target element) type.ftn:function)
(forall ((type.ftn:function ?x) (predicate ?p) (membership [?x ?p]))
  (= (element [?x ?p]) ?x))

(iff:function part)
(= (iff:source part) membership)
(= (iff:target part) predicate)
(forall ((type.ftn:function ?x) (predicate ?p) (membership [?x ?p]))
  (= (part [?x ?p]) ?p))

```

**Fact 2** *Inclusion is equivalent to universal implication of membership*

$$p \subseteq q \text{ iff } \forall_{x \in X} (x \in p \Rightarrow x \in q) \quad \subseteq = \in \setminus \in$$

Hence, the usual relationship holds between inclusion and membership.

```
(forall ((predicate ?p) (predicate ?q))
  (<=> (inclusion-relation [?p ?q])
    (forall ((type.ftn:function ?x))
      (=> (membership [?x ?p]) (membership [?x ?q])))))
```

We name the unique *proof* function for membership.

```
(iff:function proof)
(= (iff:source proof) membership)
(= (iff:target proof) type.ftn:function)
(forall ((type.ftn:function ?x) (predicate ?p) (membership [?x ?p]))
  (and (= (type.ftn:source (proof [?x ?p])) (type.ftn:source ?x))
    (= (type.ftn:target (proof [?x ?p])) (differentia ?p))
    (= ?x (type.ftn:composition [(proof [?x ?p]) (function ?p)]))))
```

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.