The IFF Upper Core (meta) Ontology

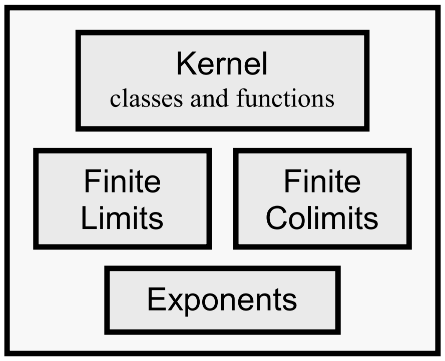


Figure 1: IFF-UCO Architecture

In the current phase, a revised version of the IFF Upper Core (meta) Ontology (IFF-UCO) is being designed. This axiomatization will be in adjunctive form. The IFF-UCO plays a central role in the IFF axiomatization – it is the most referenced ontology at the metalevel. The architecture of the IFF-UCO is illustrated in Figure 1. The IFF-UCO kernel consists of the axiomatization for classes and functions. At the upper metalevel the axiomatization for relations, which includes order-theoretic concepts, has branched off into an ontology in its on right. The kernel and finite limits namespace axiomatizations are essentially copies of those in the IFF Top Core (Basic KIF) (meta) Ontology (IFF-TCO). These axiomatizations rely heavily upon the subcollection, restriction and abridgement relations for collections, functions and binary relations, respectively. The finite colimits namespace is a categorical dual to the finite limits namespace. The exponents namespace axiomatization is new.

Exponents

Table 1 lists 24 terms for 22 basic concepts in the exponents namespace of the IFF-UCO meta-ontology.

Table 1: Terminology for Exponents in the Upper Core (meta) Ontology

SET.OBJ *exponents*	`binary-hom`	`class0 binary-function`
	`unary-hom`	`unary-function class-value`
		`product-with constant-identity-binary-hom constant-identity`
		`exponent` `evaluation-unary-hom evaluation`
		`packing = currying unpacking = inverse-currying`
		`hom composition identity`
SET.FTN		`product-with` `exponent`
		`hom`

General Discussion

As Mac Lane explains, much of the basic theory of categories is represented by enriched categories; that is, categories enriched in a closed category. A Cartesian-closed category is a special case of a closed category. Set, the most basic category of sets and set functions, is Cartesian-closed. Here we lift this structure from the IFF lower metalevel to the IFF upper metalevel. That is, we axiomatize the IFF Upper Core (meta) Ontology to be a Cartesian-closed structure. Hence, the IFF metastack, the core architecture of the IFF, is a multi-tiered structure anchored in the core meta-ontologies in the top, upper and lower metalevels. The top core is axiomatized to have (specific) finite limits, the upper core is axiomatized to be Cartesian closed and the lower core (corresponding to the ordinary large category Set) is axiomatized to be a topoi.

Loosely speaking, a category C is Cartesian-closed when any morphism defined on a product of two objects can be naturally identified with a morphism defined on one of the factors. These categories are particularly important in logic and the theory of programming. More precisely, a category C is Cartesian-closed when it satisfies the following three properties.

- There is a terminal C-object 1

  $\mathsf{1}[0, 0] \cong \mathsf{C}[A, 1]$.
- For any two C-objects A and B, there is a binary product $A \times B$ in C

  $\mathsf{C}\times\mathsf{C}[(A, A), (B, C)] \cong \mathsf{C}[A, B\times C]$
- For every C-object A, the endo functor $(\text{-})\times A$ on C has a right adjoint

  $\mathsf{C}[A\times B, C] \cong \mathsf{C}[A, C^B]$

The product-exponent adjunction $(\text{-})\times B \dashv (\text{-})^B$ is concentrated in the *evaluation* morphism $\varepsilon_X : (X^B) \times B \rightarrow X$ for each pair of C-objects B and X, which is natural in X and universal from $(\text{-})\times B$ to X. The right adjoint of $(\text{-})\times B$, applied to the object X, can be written in the hom notation as $hom_{\mathsf{C}}(B, X)$, the arrow notation as $B \Rightarrow X$ or the exponential notation as X^B. The adjointness condition means that the set of C-morphisms from $X\times B$ to Y is naturally identified with the set of morphisms from X to Y^B, for any three C-objects B, X and Y. That is, in Cartesian-closed categories, a binary morphism (a morphism of two variables) can always be represented as a unary morphism (a morphism of one variable). In some contexts, this is known as *currying*; it has led to the realization that lambda calculus can be formulated in any Cartesian-closed category. Certain Cartesian-closed categories called topoi have been proposed as a general setting for mathematics. Since the product functor is left adjoint to the exponent functor with evaluation counit, by the adjunction parameter theorem of category theory, (Mac Lane p. 100) the map $(B, A) \rightarrow A^B$ is (the object function of) a *hom* bifunctor $\mathsf{C}^{\mathrm{op}}\times\mathsf{C} \rightarrow \mathsf{C}$.

The following product and exponent laws hold in a Cartesian-closed category C. Other laws such as $(A^B)^C = (A^C)^B$ follow from these: $(A^B)^C \cong A^{(B\times C)} \cong A^{(C\times B)} \cong (A^C)^B$.

Product Laws		
$A\times(B\times C) \cong (A\times B)\times C$	product associative law	"The product is associative."
$A\times B \cong B\times A$	product commutative law	"The product is commutative."
$A\times 1 \cong A$	product unit law	"The unit is the unit for product."

Exponent Laws		
$A^1 \cong A$	exponent unit law	"Any object to the unit power is that object."
$1^A \cong 1$	base unit law (unit terminality)	adjunction $\mathsf{1}[0, 0] \cong \mathsf{C}[A, 1]$
$(A\times B)^C \cong A^C\times A^C$	base product law (product mediator)	adjunction $\mathsf{C}\times\mathsf{C}[(A, A), (B, C)] \cong \mathsf{C}[A, B\times C]$
$(A^B)^C \cong A^{(B\times C)}$	base exponent law (currying)	adjunction $\mathsf{C}[A\times B, C] \cong \mathsf{C}[A, C^B]$

Axiomatization

The Object Aspect

`SET.OBJ`

$$\begin{tikzcd}
\mathsf{Cls} \arrow[d, shift right=2, "(-)\times B"'] \arrow[d, phantom, "\overset{\eta}{\underset{\varepsilon}{\dashv}}"] \\
\mathsf{Cls} \arrow[u, shift right=2, "(-)^{B}"']
\end{tikzcd}$$

Figure 2a: Adjunction

$$\frac{X \longrightarrow Y^{B}}{X \times B \longrightarrow Y} \qquad \downarrow \cong \uparrow$$

Figure 2b: Exponentiation

The essential properties of the exponentiation or “mapping class” construction are illustrated in Figure 2b. The horizontal bar designates a natural bijection between the unary hom above the bar and the binary hom below the bar. A natural *inverse currying* (*unpacking*) process assigns a binary function $X \times B \rightarrow Y$ below the bar to every unary hom (function) $X \rightarrow Y^{B}$ above the bar. A natural *currying* (*packing*) process assigns a unary hom (function) $X \rightarrow Y^{B}$ above the bar to every binary function $X \times B \rightarrow Y$ below the bar. These two processes are mutually inverse. The inverse currying process is realized by application of the product-with $(\text{-}) \times B$ construction and then composition on the right with the evaluation (application) function (a component of the counit ε). The currying process is realized by application of the exponent $(\text{-})^{B}$ construction and then composition on the left with the constant-identity function (a component of the unit η).

For a fixed class B, the binary-hom collection consists of functions with product-with-B source class. A *binary-hom* is a constrained pair (X, g) consisting of a class X and a binary function $g : X \times B \rightarrow Y$, where the source of g is the product-with-B class applied to the class X. Hence, the binary-hom collection is a sub-collection of the function collection with two related associated KIF functions mapping to the left source component class and to the function itself.

```
(1)  (UR$morphism binary-hom)
     (= (UR$source binary-hom) class)
     (= (UR$target binary-hom) KIF.COL$collection)

(2)  (forall (?B (class ?B))
         (KIF.COL$subcollection (binary-hom ?B) SET.FTN$function))

(3)  (UR$morphism class0)
     (= (UR$source class0) class)
     (= (UR$target class0) KIF.FTN$function)
     (= (UR$composition [class0 KIF.FTN$source]) binary-hom)
     (forall (?B (class ?B))
         (= (KIF.FTN$target (class0 ?B)) SET.OBJ$class))

(4)  (UR$morphism binary-function)
     (= (UR$source binary-function) class)
     (= (UR$target binary-function) KIF.FTN$function)
     (= (UR$composition [binary-function KIF.FTN$source]) binary-hom)
     (forall (?B (class ?B))
         (and (= (KIF.FTN$target (binary-function ?B)) SET.FTN$function)
              (= (binary-function ?B)
                 (KIF.COL$inclusion [(binary-hom ?B) SET.FTN$function]))))

(5)  (= (UR$composition [(binary-function ?B) KIF.FTN$source])
        (UR$composition [(class0 ?B) (product-with ?B)]))
```

For any class B, the *unary-hom* collection consists of functions with exponent-with-B target. A unary-hom consists of a function $f : X \rightarrow Y^{B}$ and a class Y whose exponent with B is the target of the function.

```
(6)  (UR$morphism unary-hom)
     (= (UR$source unary-hom) class)
     (= (UR$target unary-hom) KIF.COL$collection)

(7)  (forall (?B (class ?B))
        (KIF.COL$subcollection (unary-hom ?B) SET.FTN$function))

 (8)  (UR$morphism unary-function)
      (= (UR$source unary-function) class)
      (= (UR$target unary-function) KIF.FTN$function)
      (= (UR$composition [unary-function KIF.FTN$source]) unary-hom)
      (forall (?B (class ?B))
          (and (= (KIF.FTN$target (unary-function ?B)) SET.FTN$function)
               (= (unary-function ?B)
                  (KIF.COL$inclusion [(unary-hom ?B) SET.FTN$function]))))

 (9)  (UR$morphism class-value)
      (= (UR$source class-value) class)
      (= (UR$target class-value) KIF.FTN$function)
      (= (UR$composition [class-value KIF.FTN$source]) unary-hom)
      (forall (?B (class ?B))
          (= (KIF.FTN$target (class-value ?B)) SET.OBJ$class))

(10)  (= (UR$composition [(class-value ?B) (exponent ?B)])
         (UR$composition [(unary-function ?B) KIF.FTN$target]))
```

$$\frac{X \xrightarrow{\eta_X} (X\times B)^B}{X\times B \xrightarrow{1_{X\times B}} X\times B}$$

Figure 3a: Constant-Identity Unary-hom

$$\frac{Y^B \xrightarrow{1} Y^B}{Y^B\times B \xrightarrow{\varepsilon_Y} Y}$$

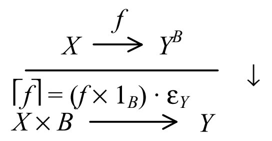
Figure 3b: Evaluation Binary-hom

Right adjoint and unit. Given any class B, the *product-with* KIF function constructs the product class $(X)\times B = X\times B$ for any class X. Given any class B, the *constant-identity* KIF-function constructs the unary function (Figure 3a) $\eta_X : X \rightarrow (X\times B)^B$ for any class X, which maps any element $x \in X$ to the function $B \rightarrow X\times B : b \rightarrow (x, b)$ constant on 1st coordinate and identity on 2nd coordinate; this is the currying of the binary product identity function $1_{X\times B} : X\times B \rightarrow X\times B$ – the constant-identity unary function is the currying of the binary product identity function.

```
(11)  (UR$morphism product-with)
      (= (UR$source product-with) class)
      (= (UR$target product-with) KIF.FTN$function)

(12)  (forall (?B (class ?B))
          (and (= (KIF.FTN$source (product-with ?B)) class)
               (= (KIF.FTN$target (product-with ?B)) class)
               (forall (?X (class ?X))
                   (= ((product-with ?B) ?X)
                      (SET.LIM.PRD2.OBJ$binary-product [?X ?B])))))

(13)  (UR$morphism constant-identity-binary-hom)
      (= (UR$source constant-identity-binary-hom) class)
      (= (UR$target constant-identity-binary-hom) KIF.FTN$function)

(14)  (forall (?B (class ?B))
          (and (= (KIF.FTN$source (constant-identity-binary-hom ?B)) class)
               (= (KIF.FTN$target (constant-identity-binary-hom ?B)) binary-hom)
               (= (KIF.FTN$composition [(constant-identity-binary-hom ?B) (class0 ?B)])
                  (KIF.FTN$identity class))
               (= (KIF.FTN$composition [(constant-identity-binary-hom ?B) (binary-function ?B)])
                  (KIF.FTN$composition [(product-with ?B) SET.FTN$identity]))))

(15)  (UR$morphism constant-identity)
      (= (UR$source constant-identity) class)
      (= (UR$target constant-identity) KIF.FTN$function)

(16)  (forall (?B (class ?B))
          (and (= (KIF.FTN$source (constant-identity ?B)) class)
               (= (KIF.FTN$target (constant-identity ?B)) unary-hom)
               (= (KIF.FTN$composition [(constant-identity ?B) (class-value ?B)])
                  (product-with ?B))
               (= (constant-identity ?B)
                 (KIF.FTN$composition
                     [(constant-identity-binary-hom ?B) (currying ?B)])))))
```

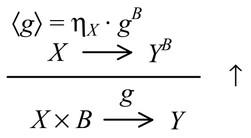
Left adjoint and counit. Given any class B, for any collection X, the *exponent* class X^B and *evaluation* function $\varepsilon_X : X^B \times B \rightarrow X$ are defined. This is the exponent in the quasi-category of classes and class functions. These form a binary-hom that is universal over the given class. Given any class X, the *exponent* KIF-function constructs the class X^B that is the *exponent* of the given class. And the *evaluation* KIF-function constructs the binary hom (Figure 3b), which is the inverse currying of the unary (hom) identity function $1 : X^B \rightarrow X^B$ for the exponent of the given class – the evaluation function is the inverse currying of the exponent identity.

```
(17) (UR$morphism exponent)
     (= (UR$source exponent) class)
     (= (UR$target exponent) KIF.FTN$function)

(18) (forall (?B (class ?B))
         (and (= (KIF.FTN$source (exponent ?B)) class)
              (= (KIF.FTN$target (exponent ?B)) class)
              (forall (?X (class ?X))
                  (and (KIF$subcollection ((exponent ?B) ?X) SET.FTN$function)
                       (forall (?f (SET.FTN$function ?f))
                           (<=> (((exponent ?B) ?X) ?f)
                                (and (= (SET.FTN$source ?f) ?B)
                                     (= (SET.FTN$target ?f) ?X))))))))

(19) (UR$morphism evaluation-unary-hom)
     (= (UR$source evaluation-unary-hom) class)
     (= (UR$target evaluation-unary-hom) KIF.FTN$function)

(20) (forall (?B (class ?B))
         (and (= (KIF.FTN$source (evaluation-unary-hom ?B)) class)
              (= (KIF.FTN$target (evaluation-unary-hom ?B)) unary-hom)
              (= (KIF.FTN$composition [(evaluation-unary-hom ?B) (class-value ?B)])
                 (KIF.FTN$identity class))
              (= (KIF.FTN$composition [(evaluation-unary-hom ?B) (unary-function ?B)])
                 (KIF.FTN$composition [(exponent ?B) SET.FTN$identity]))))

(21) (UR$morphism evaluation)
     (= (UR$source evaluation) class)
     (= (UR$target evaluation) KIF.FTN$function)

(22) (forall (?B (class ?B))
         (and (= (KIF.FTN$source (evaluation ?B)) class)
              (= (KIF.FTN$target (evaluation ?B)) binary-hom)
              (= (KIF.FTN$composition [(evaluation ?B) (class0 ?B)])
                 (exponent ?B))
              (= (evaluation ?B)
                 (KIF.FTN$composition
                     [(evaluation-unary-hom ?B) (inverse-currying ?B)]))))
```



$$\begin{array}{c} X \xrightarrow{f} Y^B \\ \hline \lceil f \rceil = (f \times 1_B) \cdot \varepsilon_Y \\ X \times B \xrightarrow{\qquad} Y \end{array} \;\downarrow$$

Figure 16b: unpacking or inverse-currying



$$\begin{array}{c} \langle g \rangle = \eta_X \cdot g^B \\ X \xrightarrow{\qquad} Y^B \\ \hline X \times B \xrightarrow{g} Y \end{array} \;\uparrow$$

Figure 16b: packing or currying

The inverse currying of a unary function with exponential target unpacks it into a binary function. For any unary function $f : X \rightarrow Y^B$ with associated value class Y, there is an *inverse currying* binary function $\lceil f \rceil : X \times B \rightarrow Y$ with associated class X, which is the unpacking and flattening of the unary function into a binary function. The inverse currying process is realized by application of the product-with $(\text{-}) \times 1_B$ construction to the function f and then composition on the right with the evaluation function ε_Y (a component of the counit ε):

$$\lceil f \rceil = (f \times 1_B) \cdot \varepsilon_Y : (X \times B) \rightarrow (Y^B \times B) \rightarrow Y.$$

The currying of a binary function with product source packs it into a unary function. For any binary function $g : X \times B \rightarrow Y$ with associated class X, there is a *currying* unary function $\langle g \rangle : X \rightarrow Y^B$ with associated value class Y, which is the packing of the parameter class as an exponent. The currying process is realized by application of the exponent $(\text{-})^B$ construction to the function g and then composition on the left with the constant-identity function η_X (a component of the unit η):

$$\langle g \rangle = \eta_X \cdot g^B : X \rightarrow (X \times B)^B \rightarrow Y^B.$$

These two processes are inverse to each other: $\lceil \langle g \rangle \rceil = g$ for each binary function g and $\langle \lceil f \rceil \rangle = f$ for each unary function f.

```
(23) (UR$morphism packing) (UR$morphism currying) (= currying packing)
     (= (UR$source packing) class)
     (= (UR$target packing) KIF.FTN$function)

(24) (forall (?B (class ?B))
         (and (= (KIF.FTN$source (packing ?B)) (binary-hom ?B))
              (= (KIF.FTN$target (packing ?B)) (unary-hom ?B))
              (= (KIF.FTN$composition [(packing ?B) (class-value ?B)])
                 (KIF.FTN$composition [(binary-function ?B) SET.FTN$target]))
              (= (KIF.FTN$composition [(packing ?B) (unary-function ?B)])
                 (SET.FTN$composition-parameterized
                     [(KIF.FTN$composition [(KIF.FTN$composition
                          [(class0 ?B) (constant-identity ?B)]) (unary-function ?B)])
                      (SET.FTN$exponent ?B)])))))

(25) (UR$morphism unpacking) (UR$morphism inverse-currying) (= inverse-currying unpacking)
     (= (UR$source unpacking) class)
     (= (UR$target unpacking) KIF.FTN$function)

(26) (forall (?B (class ?B))
         (and (= (KIF.FTN$source (unpacking ?B)) (unary-hom ?B))
              (= (KIF.FTN$target (unpacking ?B)) (binary-hom ?B))
              (= (KIF.FTN$composition [(unpacking ?B) (class0 ?B)])
                 (KIF.FTN$composition [(unary-function ?B) SET.FTN$source]))
              (= (KIF.FTN$composition [(unpacking ?B) (binary-function ?B)])
                 (SET.FTN$composition-parameterized
                     [(SET.FTN$product-with ?B)
                      (KIF.FTN$composition [(KIF.FTN$composition
                          [(class-value ?B) (evaluation ?B)]) (binary-function ?B)])])))))

(27) (forall (?B (class ?B))
         (and (= (UR$composition [(packing ?B) (unpacking ?B)]) (UR$identity (binary-hom ?B)))
              (= (UR$composition [(unpacking ?B) (packing ?B)]) (UR$identity (unary-hom ?B)))))
```

Additional Concepts and Properties

$$\begin{tikzcd}
X \arrow[d, "g"'] & X^{\wedge} \times B \arrow[l, "\varepsilon^B{}_X"'] \arrow[d, "(g^B)\times 1_B"] & X \arrow[d, "h"] \\
Y & Y^{\wedge} \times B \arrow[l, "\varepsilon^B{}_Y"] & Y
\end{tikzcd}$$

Figure 4: Naturality of Evaluation

$$\begin{tikzcd}
Y & Y^{\wedge} \times B \arrow[l, "\varepsilon^B{}_Y"'] & Y^B \\
 & X \times B \arrow[ul, "g"] \arrow[u, "f \times 1_B"'] & X \arrow[u, "f"']
\end{tikzcd}$$

Figure 5: Adjunction

$$\begin{tikzcd}
Y_1{}^{(X_1 \times B)} \arrow[r, "\gamma_B(X_1, C_1)", "\cong"'] \arrow[d, "h^{(g\times 1)}"'] & (Y_1{}^B)^{X_1} \arrow[d, "(h^1)^g"'] & Y_1 \arrow[d, "h"] & X_1 \\
Y_2{}^{(X_2 \times B)} \arrow[r, "\cong", "\gamma^B(X_2, Y_2)"'] & (Y_2{}^B)^{X_2} & Y_2 & X_2 \arrow[u, "g"']
\end{tikzcd}$$

Figure 6: Naturality of Curry

Given any class B, for any class X the evaluation function $\varepsilon^B{}_X : (X^B) \times B \rightarrow X$ is the Xth component of a quasi-natural transformation $\varepsilon^B : (\text{-})^B \times B \rightarrow (\text{-}) = id$ (Figure 4). The evaluation ε^B is the counit of a quasi-adjunction (Figure 5) between the product and exponent quasi-functors. The curry operation is natural (Figure 6). Although we express the latter, it needs two functions not yet defined, the product-with-B and exponent-B extended from classes (B) to functions; and the form of the curry operation is too specific.

```
(28) (forall (?B (class ?B) ?X (class ?X) ?Y (class ?Y)
                 ?h ((hom [?X ((exponent ?B) ?Y)]) ?h))
         (and ((hom [((product-with ?B) ?X) ?Y]) ((inverse-curry ?B) ?f))
                (= ((inverse-curry ?B) ?f)
                   (SET.FTN$composition [((product-with ?B) ?f) ((evaluation ?B) ?Y)]))
                (=> (exists (?g ((hom [((product-with ?B) ?X) ?Y]) ?g)
                               (= ?g (SET.FTN$composition [((product-with ?B) ?f) ((evaluation ?B) ?Y)])))
                    (= ?g ((inverse-curry ?B) ?f))))

(29) (forall (?B (class ?B) ?g (function ?g))
         (= (SET.FTN$composition [((evaluation ?B) (source ?g)) ?g])
            (SET.FTN$composition
               [((product-with ?B) ((exponent ?B) ?g)) ((evaluation ?B) (target ?g))])))

(30) (forall (?B (class ?B) ?X1 (class ?X1) ?X2 (class ?X2) ?Y1 (class ?Y1) ?Y2 (class ?Y2)
                ?g (((exponent ?X2) ?X1) ?g) ?h (((exponent ?Y1) ?Y2) ?h))
         (= (SET.FTN$composition
               [((curry ?B) [?X1 ?Y1]) ((exponent ((product-with ?B) ?g)) ?h)])
            (SET.FTN$composition
               [((exponent ?g) ((product-with ?B) ?h)) ((curry ?B) [?X2 ?Y2])])))
```

As remarked above, for any Cartesian-close category C, the map (A, B) → A^B is (the object function of) a hom bifunctor $C^{op}×C$ → C. Translating this to the IFF upper metalevel, we assert the existence of a *hom* KIF function *hom* : *cls*×*cls* → *cls*, which maps any pair of classes A and B to the hom class *hom*(A, B), the class of all functions from class A to class B.

```
(31) (KIF.FTN$function hom)
     (= (KIF.FTN$source hom) (KIF.LIM.PRD2.OBJ$binary-product [class class]))
     (= (KIF.FTN$target hom) class)

(32) (forall (?A (class ?A) ?B (class ?B))
         (and (class (hom [?A ?B]))
              (KIF.COL$subcollection (hom [?A ?B]) SET.FTN$function)
              (= (hom [?A ?B]) ((exponent ?A) ?B))))
```

Using hom function, we can mimic class function composition and class function identity. There is a KIF composition function ∘ = *comp* : *cls*×*cls*×*cls* → *ftn*, where for any three classes A, B and C there is a class *composition* function $\circ_{A,B,C}$ = *comp*(A, B, C) : $B^A×C^B → C^A$. There is a KIF composition 1 = *id* : *cls* → *ftn*, where for any class A there is a class *identity* function 1_A = *id*(A) : 1 → A^A. Composition is associative and identity satisfies two unit laws.

```
(33) (KIF.FTN$function composition)
     (= (KIF.FTN$source composition) (KIF.LIM.PRD3.OBJ$ternary-product [class class class]))
     (= (KIF.FTN$target composition) SET.FTN$function)

(34) (forall (?A (class ?A) ?B (class ?B) ?C (class ?C))
         (and (= (SET.FTN$source (composition [?A ?B ?C]))
                 (SET.LIM.PRD2.OBJ$binary-product [((exponent ?A) ?B) ((exponent ?B) ?C)]))
              (= (SET.FTN$target (composition [?A ?B ?C])) ((exponent ?A) ?C))
              (forall (?f (((exponent ?A) ?B) ?f)
                       ?g (((exponent ?B) ?C) ?g))
              (= ((composition [?A ?B ?C]) [?f ?g]) (SET.FTN$composition [?f ?g])))))

(35) (KIF$function identity)
     (= (KIF$source identity) class)
     (= (KIF$target identity) SET.FTN$function)

(36) (forall (?A (class ?A))
         (and (= (SET.FTN$source (identity ?A)) unit)
              (= (SET.FTN$target (identity ?A)) ((exponent ?A) ?A))
              (= ((identity ?A) 0)
                 (SET.FTN$identity ?A))))

(37) (forall (?A (class ?A) ?B (class ?B) ?C (class ?C) ?D (class ?D)
                ?f (((exponent ?A) ?B) ?f) ?g (((exponent ?B) ?C) ?g) ?h (((exponent ?C) ?D) ?h))
         (= ((composition [?A ?C ?D]) [((composition [?A ?B ?C]) [?f ?g]) ?h])
            ((composition [?A ?B ?D]) [?f ((composition [?B ?C ?D]) [?g ?h])])))

(38) (forall (?A (class ?A) ?B (class ?B)
                ?f (((exponent ?A) ?B) ?f))
         (and (= ((composition [?A ?A ?B]) [((identity ?A) 0) ?f]) ?f)
              (= ((composition [?A ?B ?B]) [?f ((identity ?B) 0)]) ?f)))
```

The Morphic Aspect

SET.FTN

The product-with and exponent constructions are extended to morphisms. For any class B, given any function $f : X \rightarrow Y$, there is a product-with-B function $(f \times 1_B) : (X \times B) \rightarrow (Y \times B)$, which applies f to the 1st component and is identity on the 2nd component. For any class B, given any function $g : X \rightarrow Y$, there is an exponent function $g^B : X^B \rightarrow Y^B$ defined via right composition with g: $g^B(x) = x \cdot g$ for any function $x \in X^B$.

```
(1) (UR$morphism product-with)
    (= (UR$source product-with) SET.OBJ$class)
    (= (UR$target product-with) KIF.FTN$function)

(2) (forall (?B (class ?B))
        (and (= (KIF.FTN$source (product-with ?B)) function)
             (= (KIF.FTN$target (product-with ?B)) function)
             (= (KIF.FTN$composition [(product-with ?B) source])
                (KIF.FTN$composition [source (product-with ?B)]))
             (= (KIF.FTN$composition [(product-with ?B) target])
                (KIF.FTN$composition [target (product-with ?B)]))
             (forall (?f (function ?f))
                 (= ((product-with ?B) ?f)
                    (SET.LIM.PRD2.MOR$binary-product [?f (identity ?B)])))))

(3) (UR$morphism exponent)
    (= (UR$source exponent) SET.OBJ$class)
    (= (UR$target exponent) KIF.FTN$function)

(4) (forall (?B (class ?B))
        (and (= (KIF.FTN$source (exponent ?B)) function)
             (= (KIF.FTN$target (exponent ?B)) function)
             (= (KIF.FTN$composition [(exponent ?B) source])
                (KIF.FTN$composition [source (exponent ?B)]))
             (= (KIF.FTN$composition [(exponent ?B) target])
                (KIF.FTN$composition [target (exponent ?B)]))
             (forall (?g (function ?g))
                      ?x (((exponent ?B) (source ?g)) ?x))
                 (= (((exponent ?B) ?g) ?x)
                    (composition [?x ?g])))))
```

Additional Concepts and Properties

There is a *hom* KIF function *hom* : *ftn*×*ftn* → *ftn*, which maps any pair of class functions g : C → A and h : B → D to the class function $hom(g, h)$: hom(A, B) → hom(C, D), which is define by two-sided composition: $hom(g, h)(f) = (h^A(f))^C(g) = h^C(f^C(g)) = g \cdot f \cdot h$ for any class function f : A → B.

```
(5) (KIF.FTN$function hom)
    (= (KIF.FTN$source hom) (KIF.LIM.PRD2.OBJ$binary-product [function function]))
    (= (KIF.FTN$target hom) function)

(6) (forall (?g (function ?g) ?h (function ?h))
        (and (= (source (hom [?g ?h])) (SET.OBJ$hom [(target ?g) (source ?h)]))
             (= (target (hom [?g ?h])) (SET.OBJ$hom [(source ?g) (target ?h)]))
             (forall (?f ((SET.OBJ$hom [(target ?g) (source ?h)]) ?f))
                 (= ((hom [?g ?h]) ?f)
                    (((exponent (source ?g)) ?h) (((exponent (source ?g)) ?f) ?g))))))
```