

The IFF Core (meta) Ontology

Robert E. Kent

November 10, 2005

Contents

1	The Ur Level	2
1.1	Basics	3
1.1.1	Ur Sets	5
1.1.2	Ur Functions	7
1.1.3	Ur Relations	11
1.2	Diagrams	14
1.2.1	Ur Set Pairs	15
1.3	Limits	18
1.3.1	Ur Terminal Set	18
1.3.2	Ur Binary Products	20
1.3.3	Old Binary Product, Projections and Pairing	25
2	The Generic Levels	28
2.1	Axiomatization	28
2.1.1	Level n Sets	29
2.1.2	Level n Functions	32
2.1.3	Level n Relations	36
2.1.4	Level n Binary Products	40

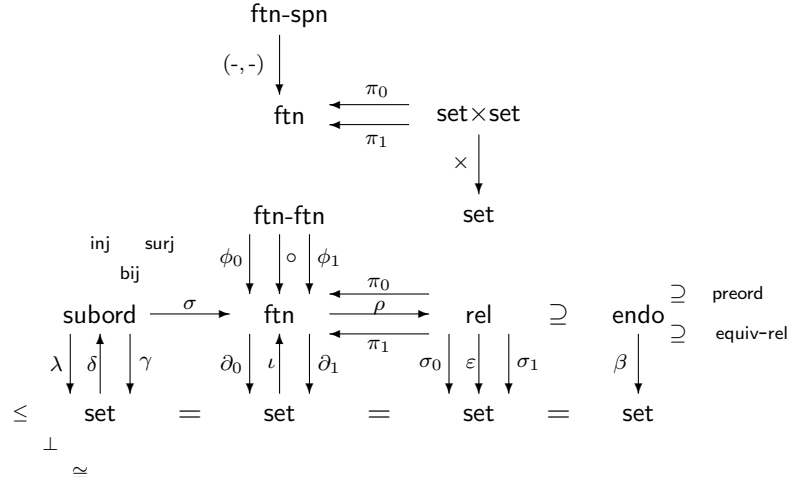


Figure 1: The Ur Architecture

Ur \supset	Col \supset	Cls \supset	Set
<i>category theory</i>			
<i>finite limits</i>			
<i>finite colimits + exponents</i>			
			<i>general limits/colimits</i>

Figure 2: The IFF Metastack

1 The Ur Level

Figure 2 illustrates the IFF Metastack (the IFF core hierarchy) The IFF Ur^1 meta-ontology (IFF-UR) is a tiny ontology at the very top of the IFF metastack (the IFF core metalevel hierarchy) above all finite metalevels. Figure 1 illustrates the architecture for the IFF-UR. In a sense, the Ur meta-ontology, which fills out and completes the metashell, contains generic sets, functions and relations. The metashell functions as a simple typing mechanism for the metastack, and hence makes no ontological commitments. It is in the Ur meta-ontology that ontological commitments start being made. Here we assert the existence of sets, functions, relations, etc. The various finite levels of the IFF metastack extend up to the Ur meta-ontology.

The terminology of the Ur metalanguage is listed in Table 1, which consists of only seventy-one terms representing sixty-one concepts (ten synonymies): twenty-one generic sets, consisting of three basic sets (**set**, **function = tuple** and **relation**), two derived sets (**function-function** and **function-cone**), four subsets (**endorelation**, **preorder**, **partial-order**, **equivalence-relation**) and four specific sets (**zero = nothing = null =**

¹original; primitive; prototypical

empty = initial, one = unit = terminal, two, three), and seven predicates (injection, surjection, bijection, reflexive, symmetric, antisymmetric and transitive); thirty-two generic functions; and eight generic endorelations, consisting of three binary endorelations on sets (subset, disjoint and isomorphic), four binary endorelations on functions (restriction, optimal-restriction, pairable and composable), and one binary relations on relations (abridgment). The subset, restriction, optimal-restriction and abridgment binary relations (and the binary intersection operation) are important for building the vertical aspect (establishing the preservation properties) of the IFF core hierarchy (the “IFF metastack”).

1.1 Basics

Namespace Prefix

Categorical: core

There is a category $\text{Set}_\infty = \langle \text{set}_\infty, \text{ftn}_\infty, \circ_\infty, 1_\infty \rangle$, whose object set is the type set of Ur sets, whose morphism set is the type set of Ur functions, whose composition is the composition operation on Ur functions, and whose identity is the identity operation from Ur sets to Ur functions.

```
(meta.cat:category Set)
(= (meta.cat:object Set) ur.set:set)
(= (meta.cat:morphism Set) ur.ftn:function)
(= (meta.cat:composition Set) ur.ftn:composition)
(= (meta.cat:identity Set) ur.ftn:identity)
```

There is a *power* endofunctor $\wp_\infty : \text{Set}_\infty \rightarrow \text{Set}_\infty$, whose object function is the power function on sets and whose morphism function is the power (direct-image) function on functions.

```
(ur.func:functor power)
(= (ur.func:source power) Set)
(= (ur.func:target power) Set)
(= (ur.func:object power) ur.set:power)
(= (ur.func:morphism power) ur.ftn:power) = direct-image
```

There is a *singleton* natural transformation $\{-\} : 1_{\text{Set}_\infty} \Rightarrow \wp_\infty : \text{Set}_\infty \rightarrow \text{Set}_\infty$, from the identity functor on Set_∞ to the power functor on Set_∞ .

```
(meta.nat:natural-transformation singleton)
(= (meta.nat:source-functor singleton) (ur.func:identity Set))
(= (meta.nat:target-functor singleton) power)
(= (meta.nat:source-category singleton) Set)
(= (meta.nat:target-category singleton) Set)
(= (meta.nat:component singleton) ur.set:singleton)
```

There is a *union* natural transformation $\cup : \wp_\infty^2 \circ \wp_\infty \Rightarrow \wp_\infty : \text{Set}_\infty \rightarrow \text{Set}_\infty$, from the power squared functor on Set_∞ to the power functor on Set_∞ .

```
(meta.nat:natural-transformation union)
(= (meta.nat:source-functor union)
   (meta.func:composition [power power]))
(= (meta.nat:target-functor union) power)
(= (meta.nat:source-category union) Set)
(= (meta.nat:target-category union) Set)
(= (meta.nat:component union) ur.set:union)
```


There is a *power* monad $\wp = \langle \wp, \{-\}, \cup \rangle$ on Set_∞ , whose endofunctor is the power functor, whose unit is the singleton natural transformation and whose unit is the union natural transformation

```
(meta.mnd:monad Power)
(= (meta.mnd:category) Set)
(= (meta.mnd:functor) power)
(= (meta.mnd:unit) singleton)
(= (meta.mnd:multiplication) union)
```

Injections, surjections and bijections are closed under composition.

```
((meta.cat:subcategory Set) ur.set:set ur.ftn:injection)
((meta.cat:subcategory Set) ur.set:set ur.ftn:surjection)
((meta.cat:subcategory Set) ur.set:set ur.ftn:bijection)
```

The complete Ur theory is expressed in the following axioms. The axioms specify the five basic kinds or sorts of things: sets, subordinate pairs of sets, functions, composable pairs of functions and relations; plus relation subkinds of endorelations, preorders and equivalence relations; and pairable pairs of functions. The axioms are partitioned accordingly.

1.1.1 Ur Sets

Namespace Prefix

Technical: `∞.set`
Recommended: `ur.set`
Categorical: `core.obj`

Ur *sets* are generic sets. The set of all Ur sets is a metaset. Any Ur set is itself a metaset. The metaset of Ur sets is not a Ur set.

```
(meta.set:set set)
(type:subset set meta.set:set)
(not (set set))
```

There is a binary Ur *subset* relationship between pairs of Ur sets, where all elements of one are elements of the other. The Ur subset relation is the abridgment of the meta subset relation. The Ur subset relation is a partial order (reflexive, antisymmetric and transitive), since identities are inclusions and inclusions are close under composition.

```
(meta.rel:endorelation subset)
(= (meta.rel:base subset) set)
(type:abridgment subset meta.set:subset)
(meta.rel:partial-order subset)
```

The two Ur sets in a subset relationship are linked by a injection. For each pair of Ur sets that satisfy the subset relation, there is an *inclusion* injection whose source is the smaller set and whose target is the larger set. The Ur inclusion function is the optimal-restriction of the meta inclusion function.

```
(meta.ftn:injection inclusion)
(= (meta.ftn:source inclusion) (meta.rel:extent ur.set:subset))
(= (meta.ftn:target inclusion) ur.ftn:injection)
(type:optimal-restriction inclusion meta.set:inclusion)
```

There is a binary *disjointness* relation between pairs of sets, which have no elements in common. The Ur disjointness relation is the abridgment of the meta disjointness relation. The disjointness relation is clearly symmetric.

```
(meta.rel:endorelation disjoint)
(= (meta.rel:base disjoint) set)
(type:abridgment disjoint meta.set:disjoint)
(meta.rel:symmetric disjoint)
```

Here we follow 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 set X is *isomorphic* to another set Y when there is at least one isomorphism from X to Y . This definition of isomorphic is used in all categories, but in a category of abstract sets and arbitrary functions the two sets X and Y are said to be *equinumerous* or *have the same cardinality*. The isomorphism of abstract sets offers a method to study equinumerosity without counting (hence, it also works for infinite sets). This fact was used systematically by Cantor. There is a binary *isomorphic* relation between pairs of sets, that are linked by an bijection. The Ur isomorphic relation is the abridgment of the meta isomorphic relation. The isomorphic relation is an equivalence relation (reflexive, symmetric and transitive), since identities are bijections, bijections have an inverse and bijections are closed under composition.

```
(meta.rel:endorelation isomorphic)
(= (meta.rel:base isomorphic) set)
(type:abridgment isomorphic meta.set:isomorphic)
(meta.rel:equivalence-relation isomorphic)
```

Fact 1 (Galileo) *The set \mathbb{N} of all natural numbers (nonnegative whole numbers) is isomorphic to the set \mathbf{Sqr} of all square whole numbers (a proper subset).*

Proof. Use the squaring function $(-)^2 : \mathbb{N} \xrightarrow{\sim} \mathbf{Sqr}$ and its inverse the square root function $\sqrt{} : \mathbf{Sqr} \xrightarrow{\sim} \mathbb{N}$. ■ 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.*

The Ur infinite set is the binary intersection of the meta infinite set with *set_{inf}*.

```
(meta.set:set infinite)
(meta.set:subset infinite set)
(= infinite (type:binary-intersection [infinite meta.set:infinite])
```

```
(meta.set:set finite)
(meta.set:subset finite set)
(= finite (difference [set infinite]))
```

Here are some basic finite sets: **zero** = $\{\}$ = \emptyset , **one** = $\{0\}$, **two** = $\{0, 1\}$ and **three** = $\{0, 1, 2\}$. **zero** and **one** have several synonyms. **two** and **three** are often used for indexing. Nothing is in **zero**. Three canonical objects have been used in specifying the other basic sets: 0, 1 and 2. Clearly, we have the following inclusions (subset relationships): **zero** \subseteq X for any set X , and **one** \subset **two** \subset **three**.

```
(set zero) (set nothing) (set null) (set empty) (set initial)
(= zero meta.set:zero) (= zero nothing) (= nothing null) (= null empty) (= empty initial)
```

```
(set one) (set unit) (set terminal)
(= one meta.set:one) (= one unit) (= unit terminal)
```

```
(set two) (= two meta.set:two)
```

```
(set three) (= three meta.set:three)
```

For any pair of Ur sets X_0 and X_1 , there is a binary union Ur set $X_0 \cup X_1$ and a binary intersection Ur set $X_0 \cap X_1$. Binary union is the optimal restriction of meta binary union, whereas binary intersection is only the restriction of meta binary intersection.

```
(meta.ftn:function binary-union)
(= (meta.ftn:source binary-union) ur.pr:set-pair)
(= (meta.ftn:target binary-union) set)
(type:optimal-restriction binary-union meta.set:binary-union)
```

```
(meta.ftn:function binary-intersection)
(= (meta.ftn:source binary-intersection) ur.pr:set-pair)
(= (meta.ftn:target binary-intersection) set)
(type:restriction binary-intersection meta.set:binary-intersection)
```

For any Ur set X , there is a *power* Ur set $\wp X$, which consists of the set of all subsets of X . The Ur power function is the optimal restriction of the meta power function.

```
(meta.ftn:function power)
(= (meta.ftn:source power) set)
(= (meta.ftn:target power) set)
(type:optimal-restriction power meta.set:power)
```

1.1.2 Ur Functions

Namespace Prefix

Technical: `∞.ftn`
Recommended: `ur.ftn`
Categorical: `core.mor`

Ur *functions* are generic functions. Any Ur function is itself a metafunction. The function notation $f : X \rightarrow Y$, shows a function f with source set X and target set Y . Functions are also called *tuples*.

```
(meta.set:set function) (meta.set:set tuple) (= tuple function)
(type:subset function meta.ftn:function)
```

Each Ur function $f : X \rightarrow Y$ has a unique *source* set $\partial_0(f) = X$ and a unique *target* set $\partial_1(f) = Y$. The Ur source (target) function is a restriction of the meta source (target) function. For any Ur sets $X, Y \in \mathbf{set}_\infty$ and any metafunction $f \in \mathbf{ftn}_\bullet$, if $\partial_0(f) = X$ and $\partial_1(f) = Y$, then f is a Ur function $f \in \mathbf{ftn}_\infty$. This means that the Ur source-target pairing function is the optimal restriction of the meta source-target pairing function. As a tuple, the source is called the *arity* of the tuple and the target is called the *type* of the tuple. For any tuple (function) $x : I \rightarrow A$, the tuple notation $x = (i \in I \mid x_i \in A) = (\dots, x_i, \dots)$ is useful. Since tuples are synonymous with functions, we can use function application to index tuples: the components in the tuple x are represented by $x(i)$ for $i \in I$.

```

(meta.ftn:function source) (meta.ftn:function arity) (= arity source)
(= (meta.ftn:source source) function)
(= (meta.ftn:target source) ur.set:set)
(type:restriction source meta.ftn:source)

(meta.ftn:function target) (meta.set:set type) (= type target)
(= (meta.ftn:source target) function)
(= (meta.ftn:target target) ur.set:set)
(type:restriction target meta.ftn:target)

(type:optimal-restriction
  (meta.lim:pairing [source target])
  (type:pairing [meta.ftn:source meta.ftn:target]))

```

For any pair of Ur sets X and Y , there is an Ur set $\text{hom}_\infty[X, Y]$ consisting of all functions $f : X \rightarrow Y$ with source X and target Y . The hom Ur function is the restriction of the hom metafunction. The hom function is a partition, since it is discriminating and comprehensive. The latter means that the metaset of all Ur functions is the (disjoint) union of all homsets: $\text{ftn} = \cup_{X, Y} \text{hom}[X, Y]$.

```

(meta.ftn:function hom)
(= (meta.ftn:source hom) ur.pr:set-pair))
(= (meta.ftn:target hom) (meta.set:power function))
(type:restriction hom meta.set:hom)
(meta.ftn:partition hom)
(= function ((meta.set:union function) (meta.ftn:range hom)))

```

Two functions are *composable* when the target of the first is equal to the source of the second. The Ur composable relation is the abridgment of the meta composable relation.

```

(meta.rel:endorelation composable)
(= (meta.rel:base composable) function)
(type:abridgment composable meta.ftn:composable)

```

We name the extent and projections of the composable endorelation.

```

(meta.set:set function-function)
(= function-function (meta.rel:extent composable))

(meta.ftn:function function0)
(= (meta.ftn:source function0) function-function)
(= (meta.ftn:target function0) function)
(= function0 (meta.rel:projection0 composable))

(meta.ftn:function function1)
(= (meta.ftn:source function1) function-function)
(= (meta.ftn:target function1) function)
(= function1 (meta.rel:projection1 composable))

```

The *composition* of two composable functions $f : A \rightarrow B$ and $g : B \rightarrow C$ is a function $f \cdot g : A \rightarrow C$. Hence, the source of the composite is the source of the first component, and the target of the composite is the target of the second component. The Ur composition function is a restriction of the meta composition function.

```

(meta.ftn:surjection composition)
(= (meta.ftn:source composition) function-function)
(= (meta.ftn:target composition) function)

```

```

(= (meta.ftn:composition [composition source])
   (meta.ftn:composition [function0 source]))
(= (meta.ftn:composition [composition target])
   (meta.ftn:composition [function1 target]))
(type:restriction composition meta.ftn:composition)

```

For every set, there is a unique associated *identity* function. The Ur identity function is the optimal-restriction of the meta identity function. Composition with the identity of the source (target) of a function returns that function. Hence, composition is surjective. Because of uniqueness, identity is injective, and sets could be regarded as special functions that satisfy the unit laws.

```

(meta.ftn:injection identity)
(= (meta.ftn:source identity) ur.set:set)
(= (meta.ftn:target identity) function)
(= (meta.ftn:composition [identity source]) (meta.ftn:identity ur.set:set))
(= (meta.ftn:composition [identity target]) (meta.ftn:identity ur.set:set))
(type:optimal-restriction identity meta.ftn:identity)

```

An Ur function $f : A \rightarrow B$ is an *injection* when any image value is from a unique source element; or more categorically, when for any two parallel functions $h_0, h_1 : D \rightarrow A$ the equality $h_0 \cdot f = h_1 \cdot f$ implies $h_0 = h_1$. The Ur injection metaset is the binary intersection of the meta injection type set with the Ur function metaset. An Ur function $f : A \rightarrow B$ is an *surjection* when any target value is an image; or more categorically, when for any two parallel functions $g_0, g_1 : B \rightarrow C$ the equality $f \cdot g_0 = f \cdot g_1$ implies $g_0 = g_1$. The Ur surjection metaset is the binary intersection of the meta surjection type set with the Ur function metaset. An Ur function $f : A \rightarrow B$ is a *bijection* when there is a(n *inverse*) function (in the opposite direction) $g : B \rightarrow A$ with $f \cdot g = 1_A$ and $g \cdot f = 1_B$. The Ur bijection metaset is the binary intersection of the meta bijection type set with the Ur function metaset. Any bijection has a unique inverse. The Ur inverse function is the optimal restriction of the meta inverse function.

```

(meta.set:set injection)
(= injection (type:binary-intersection [meta.ftn:injection function]))

(meta.set:set surjection)
(= surjection (type:binary-intersection [meta.ftn:surjection function]))

(meta.set:set bijection)
(= bijection (type:binary-intersection [meta.ftn:bijection function]))

(meta.ftn:function inverse)
(= (meta.ftn:source inverse) bijection)
(= (meta.ftn:target inverse) bijection)
(type:optimal-restriction inverse meta.ftn:inverse)

```

For sets and functions, a bijection is exactly a function that is both injective and surjective.

```

(= (bijection (meta.set:binary-intersection [injection surjection]))

```

There is a *function to relation* injection that embeds functions as relations. The domain (codomain) of the relation of a function is its source (target). The domain projection is an bijection, and post-composition with the original function gives the codomain projection. The Ur function to relation injection is the optimal restriction of the meta function to relation injection.

```

(meta.ftn:injection ftn2rel)
(= (meta.ftn:source ftn2rel) function)
(= (meta.ftn:target ftn2rel) ur.rel:relation)
(type:optimal-restriction ftn2rel meta.ftn:ftn2rel)

```

For any two Ur sets X and Y and any element $y \in Y$, there is a *constant* y function $\Delta_{X,Y}(y) : X \rightarrow Y : x \mapsto y$. The Ur constant type function is a restriction of the meta constant type function.

```

(meta.ftn:function constant)
(= (meta.ftn:source constant) ur.pr2.obj:set-pair)
(= (meta.ftn:target constant) meta.ftn:function)
(type:restriction constant meta.ftn:constant)

```

There is a binary restriction relationship between pairs of Ur functions that are linked by source and target inclusions. The Ur restriction relation is the abridgment of the meta restriction relation. One function $f_1 : X_1 \rightarrow Y_1$ is a *restriction* of another function $f_2 : X_2 \rightarrow Y_2$ when the source (target) X_1 (Y_1) is a subset of the source (target) X_2 (Y_2) and the functions agree (on source elements of X_1); that is, the functions commute with the domain/target inclusions. Restriction can be viewed as a constraint on the larger function — it says that the larger function maps the source set of the smaller function into the target set of the smaller function. The restriction relation is a partial order (reflexive, antisymmetric and transitive), since identities are injections, subset is antisymmetric, and injections are closed under composition.

```

(meta.rel:endorelation restriction)
(= (meta.rel:base restriction) function)
(type:abridgment restriction meta.ftn:restriction)

```

```

(meta.ftn:function source-pair)
(= (meta.ftn:source source-pair) (meta.rel:extent restriction))
(= (meta.ftn:target source-pair) (meta.rel:extent ur.set:subset))
(meta.ftn:restriction source-pair (meta.ftn:function-square source))

```

```

(meta.ftn:function target-pair)
(= (meta.ftn:source target-pair) (meta.rel:extent restriction))
(= (meta.ftn:target target-pair) (meta.rel:extent ur.set:subset))
(meta.ftn:restriction target-pair (meta.ftn:function-square target))

```

```

((meta.rel:parametric ur.set:subset)
  (meta.ftn:composition [(meta.rel:projection0 restriction) source])
  (meta.ftn:composition [(meta.rel:projection1 restriction) source]))
(meta.rel:parametric ur.set:subset)
  (meta.ftn:composition [(meta.rel:projection0 restriction) target])
  (meta.ftn:composition [(meta.rel:projection1 restriction) target]))
(= ((meta.rel:pairing composable)
    [(meta.rel:projection0 restriction)
     (meta.ftn:composition [target-pair ur.set:inclusion])]))
  ((meta.rel:pairing composable)
    [(meta.ftn:composition [source-pair ur.set:inclusion])
     (meta.rel:projection1 restriction)]))

```

```

(meta.rel:partial-order restriction)

```

There is a binary optimal restriction relationship between pairs of Ur functions. The Ur optimal restriction relation is the abridgment of the meta optimal restriction relation. A restriction between two functions f_1 and f_2 is *optimal*

when the source set of the smaller function is exactly the inverse image of the target of the smaller function along the larger function. The optimal restriction relation is a partial order (reflexive, antisymmetric and transitive).

```
(meta.rel:endorelation optimal-restriction)
(= (meta.rel:base optimal-restriction) function)
(type:abridgment optimal-restriction meta.ftn:optimal-restriction)
(meta.rel:partial-order optimal-restriction)
```

1.1.3 Ur Relations

Namespace Prefix

Technical: `∞.rel`
Recommended: `ur.rel`
Categorical: `core.rel`

Since the Ur ontology gives not just a morphic framework, but also a relational framework, we include some additional terminology and axiomatization. Ur *relations* are generic (binary) relations. Any Ur relation is itself a metarelation.

```
(meta.set:set relation)
(meta.set:subset relation meta.rel:relation)
```

Each Ur relation r has a unique domain or zeroth Ur set X_0 and a unique codomain or first Ur set X_1 . The zeroth (first) set Ur function is a restriction of the meta zeroth (first) set function. The zeroth-first set pairing Ur function is the optimal restriction of the meta zeroth-first set pairing function.

```
(meta.ftn:function set0)
(= (meta.ftn:source set0) relation)
(= (meta.ftn:target set0) ur.set:set)
(type:restriction set0 meta.rel:set0)

(meta.ftn:function set1)
(= (meta.ftn:source set1) relation)
(= (meta.ftn:target set1) ur.set:set)
(type:restriction set1 meta.rel:set1)

(meta.ftn:optimal-restriction
 (meta.lim:pairing [set0 set1])
 (type:pairing [meta.rel:set0 meta.rel:set1]))
```

Each Ur relation has a unique *extent* Ur set. The extent is the subset of the binary product of the two component sets (domain and codomain) $\text{ext}(r) \subseteq X_0 \times X_1$, consisting of those pairs that satisfy the relation $(x_0, x_1) \in \text{ext}(r)$ iff $r(x_0, x_1)$. The Ur extent function is a restriction of the meta extent function.

```
(meta.ftn:function extent)
(= (meta.ftn:source extent) relation)
(= (meta.ftn:target extent) ur.set:set)
(type:restriction extent meta.rel:extent)
```

For any Ur relation, since the extent is a subset of the binary product of domain and codomain, there are two unique *projection* functions. These relational projections are the composition of the inclusion with the product projections. The Ur projection functions are a restriction of the meta projection functions. The extent set and projection functions implicitly associate a span with a relation.

```

(meta.ftn:function projection0)
(= (meta.ftn:source projection0) relation)
(= (meta.ftn:target projection0) ur.ftn:function)
(= (meta.ftn:composition [projection0 ur.ftn:source]) extent)
(= (meta.ftn:composition [projection0 ur.ftn:target]) set0)
(type:restriction projection0 meta.rel:projection0)
(= projection0
  (meta.ftn:composition
    [(meta.rel:pairing ur.ftn:composable)
     [inclusion
      (meta.ftn:composition
        [meta.lim:pairing [set0 set1]) ur.lim.prd2:projection0]]])
    ur.ftn:composition]))

```

```

(meta.ftn:function projection1)
(= (meta.ftn:source projection1) relation)
(= (meta.ftn:target projection1) ur.ftn:function)
(= (meta.ftn:composition [projection1 ur.ftn:source]) extent)
(= (meta.ftn:composition [projection1 ur.ftn:target]) set1)
(type:restriction projection1 meta.rel:projection1)
(= projection1
  (meta.ftn:composition
    [(meta.rel:pairing ur.ftn:composable)
     [inclusion
      (meta.ftn:composition
        [meta.lim:pairing [set0 set1]) ur.lim.prd2:projection1]]])
    ur.ftn:composition]))

```

For any Ur relation, the two projections are pairable. The pairing of the projections $(\pi_0, \pi_1) : \text{ext}(r) \rightarrow X_0 \times X_1$ is the *inclusion* of the extent into the product of the component sets.

```

(meta.rel:parametric ur.prd2:pairable) projection0 projection1)

(meta.ftn:function inclusion)
(= (meta.ftn:source inclusion) relation)
(= (meta.ftn:target inclusion) ur.ftn:function)
(= (meta.ftn:composition [inclusion ur.ftn:source]) extent)
(= (meta.ftn:composition [inclusion ur.ftn:target])
  (meta.ftn:composition [(meta.lim:pairing [set0 set1]) ur.prd2:product]))
(= inclusion
  (meta.ftn:composition
    [(meta.rel:pairing ur.prd2:pairable) projection0 projection1) ur.prd2:pairing]))
(= inclusion
  (meta.ftn:composition
    [(meta.rel:pairing ur.set:subset)
     [extent (meta.ftn:composition [(meta.lim:pairing [set0 set1]) ur.prd2:product]]])
     ur.set:inclusion]))

```

Any Ur relation r can be used in a *parametric* fashion with a pairable pair of Ur functions $f_0 : X \rightarrow \text{set}_0(r)$ and $f_1 : X \rightarrow \text{set}_1(r)$ that have a common Ur source set X and the Ur target sets $\text{set}_0(r)$ and $\text{set}_1(r)$. The parametric relation \hat{r} holds for f_0 and f_1 , $\hat{r}(f_0, f_1)$, when $r(f_0(x), f_1(x))$ for each element $x \in X$. Clearly, the extent of the parametric relation \hat{r} is contained in the function-cone metaset.

```

(type:function parametric)
(= (type:source parametric) relation)

```

```

(= (type:target parametric) meta.rel:endorelation)
(forall (?r (relation ?r))
  (and (= (meta.rel:base (parametric ?r)) ur.ftn:function)
    (forall (?f0 (ur.ftn:function ?f0) ?f1 (ur.ftn:function ?f1))
      (<=> ((parametric ?r) ?f0 ?f1)
        ((meta.rel:parametric ?r) ?f0 ?f1))))))
(forall (?r (relation ?r)
  ?f0 (meta.ftn:function ?f0) ?f1 (meta.ftn:function ?f1))
  (<=> ((meta.rel:extent (parametric ?r)) [?f0 ?f1])
    (and (ur.set:span [?f0 ?f1])
      ((type:extent (meta.rel:parametric ?r)) [?f0 ?f1]))))

```

There is a *mediator* function

$$(-, -)_r : \text{ext}(\hat{r}) \rightarrow \text{ftn} : (f_0, f_1) \mapsto f : X \rightarrow \text{ext}(r) \subseteq \text{set}_0(r) \times \text{set}_1(r),$$

which gives the pairing of an r -parametric pair (f_0, f_1) restricted to $\text{ext}(r)$.

```

(meta.ftn:function pairing)
(= (meta.ftn:source pairing) relation)
(= (meta.ftn:target pairing) meta.ftn:function)
(forall (?r (relation ?r))
  (and (= (meta.ftn:source (pairing ?r)) (meta.rel:extent (parametric ?r)))
    (= (meta.ftn:target (pairing ?r)) ur.ftn:function)
    (forall (?f0 (ur.ftn:funtion ?f0) ?f1 (ur.ftn:funtion ?f1))
      ((parametric ?r) ?f0 ?f1)
      (and (= (ur.ftn:source ((pairing ?r) [?f0 ?f1])) (ur.ftn:source ?f0))
        (= (ur.ftn:target ((pairing ?r) [?f0 ?f1])) (extent ?r))
        (= ((pairing ?r) [?f0 ?f1]) ((meta.rel:pairing ?r) [?f0 ?f1]))
        (= (ur.ftn:composition [((pairing ?r) [?f0 ?f1]) (pairing ?r)])
          (ur.prd2.obj:pairing [?f0 ?f1]))))))))

```

There is a binary *abridgment* relationship between pairs of Ur relations, The Ur abridgment meta endorelation is the abridgment of the meta abridgment type endorelation. The abridgment relation is a partial order (reflexive, antisymmetric and transitive).

```

(meta.rel:endorelation abridgment)
(= (meta.rel:base abridgment) relation)
(type:abridgment abridgment meta.rel:abridgment)
(meta.rel:partial-order abridgment)

```

Ur Endorelations. Ur *endorelations* are special Ur relations, whose domain and codomain are identical. This set is called the underlying *base* set. The Ur endorelation metaset is the type binary intersection of the meta endorelation type set with the relation metaset. The Ur base function is the optimal restriction of the meta base function.

```

(meta.set:set endorelation)
(= endorelation (type:binary-intersection [meta.rel:endorelation relation]))

(meta.ftn:function base)
(= (meta.ftn:source base) endorelation)
(= (meta.ftn:target base) ur.set:set)
(type:optimal-restriction base meta.rel:base)

```

We have unary predicates (metasets) to express the fact that an Ur endorelation is *reflexive*, *symmetric*, *antisymmetric* or *transitive*.

```

(meta.set:set reflexive)
(= reflexive (type:binary-intersection [meta.rel:reflexive endorelation]))

(meta.set:set symmetric)
(= symmetric (type:binary-intersection [meta.rel:symmetric endorelation]))

(meta.set:set antisymmetric)
(= antisymmetric (type:binary-intersection [meta.rel:antisymmetric endorelation]))

(meta.set:set transitive)
(= transitive (type:binary-intersection [meta.rel:transitive endorelation]))

```

We can declare special Ur endorelations called *preorders*, *partial-orders* and *equivalence relations*. Preorders are reflexive and transitive. Partial orders are preorders that are antisymmetric. Equivalence relations are reflexive, symmetric and transitive.

```

(meta.set:set preorder)
(meta.set:subset preorder endorelation)
(= preorder (meta.set:binary-intersection [reflexive transitive]))

(meta.set:set partial-order)
(meta.set:subset partial-order endorelation)
(= partial-order (meta.set:binary-intersection [preorder antisymmetric]))

(meta.set:set equivalence-relation)
(meta.set:subset equivalence-relation endorelation)
(= equivalence-relation (meta.set:binary-intersection [preorder symmetric]))

```

Any reflexive Ur endorelation has a *delta* function from its base to its extent. The Ur delta meta function is the optimal restriction of the meta delta type function.

```

(meta.ftn:function delta)
(= (meta.ftn:source delta) reflexive)
(= (meta.ftn:target delta) ur.ftn:function)
(= (meta.ftn:composition [delta ur.ftn:source]) base)
(= (meta.ftn:composition [delta ur.ftn:target]) extent)
(type:optimal-restriction delta meta.rel:delta)

```

If the endorelation r is reflexive, symmetric, antisymmetric, transitive, pre-ordered, partially-ordered or an equivalence relation, then so is the parametric metarelation \hat{r} .

```

(forall (?r (reflexive ?r)) (meta.rel:reflexive (parameteric ?r)))
((type.pred:parametric meta.rel:reflexive)
  ((restricted-to parametric) (meta.pred:differentia reflexive)))

(forall (?r (symmetric ?r)) (meta.rel:symmetric (parameteric ?r)))
(forall (?r (transitive ?r)) (meta.rel:transitive (parameteric ?r)))
(forall (?r (antisymmetric ?r)) (meta.rel:antisymmetric (parameteric ?r)))
(forall (?r (preorder ?r)) (meta.rel:preorder (parameteric ?r)))
(forall (?r (partial-order ?r)) (meta.rel:partial-order (parameteric ?r)))
(forall (?r (equivalence-relation ?r)) (meta.rel:equivalence-relation (parameteric ?r)))

```

1.2 Diagrams

Abstractly, a finite diagram is a graph morphism from a finite (shape) graph into the level $n + 1$ graph of level $n + 1$ sets and their functions. However, here

we do not assume an axiomatization for finiteness (or natural numbers). We only introduce those diagrams used in lower metalevels $\leq n$ for specifying finite limits. Diagrams are determined by their shape, set and function components. These diagrams include (Figure ????) the empty diagram, collection pairs and triples, parallel pairs of functions, opspans and multi-opspans; their limits are called, the terminal collection, binary products, ternary products, equalizers, pullbacks and multipullbacks, respectively.

1.2.1 Ur Set Pairs

Namespace Prefix

Technical: `oo.pr`
Recommended: `ur.pr`
Categorical: `core.pr`

Objects. A pair of Ur sets (X_0, X_1) is the appropriate base diagram for a binary product. Set pairs are also used by binary coproducts. Intuitively, the *set pair* metaset `set-pair = set × set` $\cong \text{set}^2$ is the binary power of `set`. Set pairs are isomorphic to set tuples with arity two.

```
(meta.set:set set-pair}
(= set-pair (meta.set:set-square ur.set:set))
```

```
(meta.ftn:function set0)
(= (meta.ftn:source set0) set-pair)
(= (meta.ftn:target set0) ur.set:set)
(= set0 (meta.lim:square0 ur.set:set))
```

```
(meta.ftn:function set1)
(= (meta.ftn:source set1) set-pair)
(= (meta.ftn:target set1) ur.set:set)
(= set1 (meta.lim:square1 ur.set:set))
```

A pair of sets has an associated Ur *opspan*. The *opvertex* is the terminal Ur set, and the component functions are the unique Ur functions for the component sets.

```
(meta.ftn:function opspan)
(= (meta.ftn:source opspan) set-pair)
(= (meta.ftn:target opspan) ur.opsn.obj:opspan)
(= (meta.ftn:composition [opspan ur.ospn.obj:set0]) set0)
(= (meta.ftn:composition [opspan ur.ospn.obj:set1]) set1)
(= (meta.ftn:composition [opspan ur.ospn.obj:function0])
    (meta.ftn:composition [set0 ur.trm:unique]))
(= (meta.ftn:composition [opspan ur.ospn.obj:function1])
    (meta.ftn:composition [set1 ur.trm:unique]))
(= (meta.ftn:composition [opspan ur.ospn.obj:opvertex])
    ((meta.ftn:constant [set-pair ur.set:set]) ur.trm:terminal))
```

Morphisms. A pair of Ur functions (f_0, f_1) is the appropriate base diagram for a morphism of binary products. Function pairs are also used as binary coproduct morphisms. The *function pair* metaset $\text{ftn-pair} = \text{ftn}^2 \cong \text{ftn} \times \text{ftn}$ is the binary power of ftn . In use, it is important to note that this is a concrete notion; which means that we can use the notation ‘ $[f_0, f_1]$ ’ for function pairs. Any function pair (f_0, f_1) has a source (target) pair of Ur sets $(\partial_0(f_0), \partial_0(f_1))$ ($(\partial_1(f_0), \partial_1(f_1))$) consisting of the sources (targets) of its function components. Special function pairs are used as the diagrams for pullbacks, and the cones for binary products and pullbacks.

```
(meta.set:set function-pair}
(= function-pair (meta.set:set-square ur.ftn:function))

(meta.ftn:function function0)
(= (meta.ftn:source function0) function-pair)
(= (meta.ftn:target function0) ur.ftn:function)
(= function0 (meta.lim:square0 ur.ftn:function))

(meta.ftn:function function1)
(= (meta.ftn:source function1) function-pair)
(= (meta.ftn:target function1) ur.ftn:function)
(= function0 (meta.lim:square0 ur.ftn:function))

(meta.ftn:function source)
(= (meta.ftn:source source) function-pair)
(= (meta.ftn:target source) ur.pr.obj:set-pair)
(= (meta.ftn:composition [source ur.pr.obj:set0]
    (meta.ftn:composition [function0 ur.ftn:source])))
(= (meta.ftn:composition [source ur.pr.obj:set1]
    (meta.ftn:composition [function1 ur.ftn:source])))

(meta.ftn:function target)
(= (meta.ftn:source target) function-pair)
(= (meta.ftn:target target) ur.pr.obj:set-pair)
(= (meta.ftn:composition [target ur.pr.obj:set0]
    (meta.ftn:composition [function0 ur.ftn:target])))
(= (meta.ftn:composition [target ur.pr.obj:set1]
    (meta.ftn:composition [function1 ur.ftn:target])))
```

Function pairs can be composed. Two function pairs are composable when the target of the first is equal to the source of the second. The *composition* of two composable function pairs $f : (X_0, X_1) \rightarrow (Y_0, Y_1)$ and $g : (Y_0, Y_1) \rightarrow (Z_0, Z_1)$ is the function pair $f \cdot g : (X_0, X_1) \rightarrow (Z_0, Z_1)$, whose components are defined by $(f \cdot g)_0 = f_0 \cdot g_0$ and $(f \cdot g)_1 = f_1 \cdot g_1$. For any set pair $X = (X_0, X_1)$, there is an *identity* function pair.

Endorelations of composability, such as that below, appear in various places in the core axiomatization. One way to eliminate the quantifier below is to recognize that we are in the presence of a category. This indicates that we should use category theory axioms to do this elimination. Then we also need graph axiomatization all the way up, and composition and composable are part of the horizontal graph multiplication. This implies that we need pullback and multipullback (finite limit) axioms also. The axioms in the meta namespace then become those for a finitely complete category. Examples include: Set_∞ , the Ur category of sets and functions,

```
(ur.cat:category Set)
(= (ur.cat:object Set) ur.set:set)
(= (ur.cat:morphism Set) ur.ftn:function)
(= (ur.cat:source Set) ur.ftn:source)
(= (ur.cat:target Set) ur.ftn:target)
(= (ur.cat:composable Set) ur.ftn:composable)
(= (ur.cat:composable-pair Set) ur.ftn:composable-pair)
(= (ur.cat:composition Set) ur.ftn:identity)
```

and Set_∞^2 , the Ur category of set-pairs and function-pairs.

```
(ur.cat:category Set-Pair)
(= (ur.cat:object Set-Pair) ur.pr.obj:set-pair)
(= (ur.cat:morphism Set-Pair) ur.pr.mor:function-pair)
(= (ur.cat:source Set-Pair) ur.pr.mor:source)
(= (ur.cat:target Set-Pair) ur.pr.mor:target)
(= (ur.cat:composable Set-Pair) ur.pr.mor:composable)
(= (ur.cat:composable-pair Set-Pair) ur.pr.mor:composable-pair)
(= (ur.cat:composition Set-Pair) ur.pr.mor:identity)
```

Functors, such as the projection functors $\pi_0, \pi_1 : \text{Set}_\infty^2 \rightarrow \text{Set}_\infty$

```
(ur.func:functor Projection0)
(= (ur.func:source Projection0) Set-Pair)
(= (ur.func:target Projection0) Set)
(= (ur.cat:object Projection0) ur.pr.obj:set0)
(= (ur.cat:morphism Projection0) ur.pr.mor:function0)
```

may also be important to complete the definition.

```
(meta.rel:endorelation composable)
(= (meta.rel:base composable) function-pair)
(forall (?f (function-pair ?f) ?g (function-pair ?g))
  (<=> (composable ?f ?g)
    (= (ur.ftn:target ?f) (ur.ftn:source ?g))))

(meta.set:set composable-pair)
(= composable-pair (meta.rel:extent composable))

(meta.ftn:function function-pair0)
(= (meta.ftn:source function-pair0) composable-pair)
(= (meta.ftn:target function-pair0) function-pair)
(= function-pair0 (meta.rel:projection0 composable))

(meta.ftn:function function-pair1)
(= (meta.ftn:source function-pair1) composable-pair)
(= (meta.ftn:target function-pair1) function-pair)
(= function-pair1 (meta.rel:projection1 composable))

(meta.ftn:function composable0)
(= (meta.ftn:source composable0) composable-pair)
(= (meta.ftn:target composable0) ur.ftn:function-function)
(meta.ftn:restriction composable0 (meta.ftn:function-square function0))

(meta.ftn:function composable1)
(= (meta.ftn:source composable1) composable-pair)
(= (meta.ftn:target composable1) ur.ftn:function-function)
(meta.ftn:restriction composable1 (meta.ftn:function-square function1))
```

```

(meta.ftn:function composition)
(= (meta.ftn:source composition) composable-pair)
(= (meta.ftn:target composition) function-pair)
(= (meta.ftn:composition [composition source])
    (meta.ftn:composition [function-pair0 source]))
(= (meta.ftn:composition [composition target])
    (meta.ftn:composition [function-pair1 target]))
(= (meta.ftn:composition [composition function0])
    (meta.ftn:composition [composable0 ur.ftn:composition]))
(= (meta.ftn:composition [composition function1])
    (meta.ftn:composition [composable1 ur.ftn:composition]))

(meta.ftn:function identity)
(= (meta.ftn:source identity) ur.pr.obj:set-pair)
(= (meta.ftn:target identity) function-pair)
(= (meta.ftn:composition [identity source]) (meta.ftn:identity ur.pr.obj:set-pair))
(= (meta.ftn:composition [identity target]) (meta.ftn:identity ur.pr.obj:set-pair))
(= (meta.ftn:composition [identity function0])
    (meta.ftn:composition [ur.pr.obj:set0 ur.ftn:identity]))
(= (meta.ftn:composition [identity function1])
    (meta.ftn:composition [ur.pr.obj:set1 ur.ftn:identity]))

```

1.3 Limits

In this section, we axiomatize various finite limits: terminal object, binary product, ternary product, equalizer, pullback and multipullback. We do not axiomatize general finite limits, since the terminology for these is not used in the lower levels. Instead, the terminology for particular finite limits is used there. Hence, in selecting which finite limit terminology to specify, we utilize the principle of *conceptual warrant*, an extension of the librarianship notion of literary warrant, which means evidence for or token of authorization. The terms appearing in any meta-ontology exert authority, and hence should require some warrant for their existence. It is also important to establish the connections between the particular limits specified, where each can be defined in terms of some of the others (Figure 12). Any set pair is an opspan, hence binary product cones are special pullback cones and binary products and their projections are special pullbacks and their projections. Any opspan has an underlying set pair, any pullback cone has an underlying binary product cone, and the pullback of an opspan is a subset of the binary product of the set pair underlying the opspan.

1.3.1 Ur Terminal Set

Namespace Prefix

Technical:	∞.trm
Recommended:	ur.trm
Categorical:	core.trm

$$\begin{array}{ccc}
\mathbf{1} = \text{Set}_n^\emptyset & & \\
\Delta \uparrow \begin{array}{c} \delta \\ \dashv \\ \pi \end{array} \downarrow \mathbf{1} & & \begin{array}{c} X \longrightarrow 1 \\ \hline \bullet \longrightarrow \bullet \end{array} \downarrow \cong \uparrow \\
\text{Set}_n & &
\end{array}$$

Adjunction

Packing/Unpacking

We rename the unit set *one* here, and regard this as the terminal set. This is the finite limit of the empty diagram.

```
(ur.set:set terminal)
(= terminal ur.set:one)
```

There is only one diagram of shape `null` – the empty diagram. It contains no sets and no functions. There is only one morphism for the empty diagram – the identity. Hence, the category of empty diagrams is the unit category `1`. A cone for the empty diagram is just a single set. Hence, we identify the metaset of cones over the empty diagram with the metaset of Ur sets. A mediator for the empty diagram is just a function $X \rightarrow 1$. The packing function maps a set X to a function $X \rightarrow 1$. The unpacking function maps a function $X \rightarrow 1$ to its source set X . The requirement that packing and unpacking are inverse to each other, means that the unit set is a “terminal” set – for any set X , there is exactly one (mediating) function $X \rightarrow 1$ from X to unit. Given any set X , the delta metafunction constructs the mediator with function $\delta_X : X \rightarrow 1$, which is the packing of the cone $1_\bullet : \Delta(X) = \bullet \rightarrow \bullet$, consists of the unique function for the given set – the delta mediator is the packing of the identity function. For any cone (set) X , there is a packing mediator $\langle X \rangle : X \rightarrow 1$. The packing process is realized by application of the terminal (`1`) operator to the identity morphism $1_\bullet : \Delta(X) = \bullet \rightarrow \bullet$ (yielding the identity function on the terminal set) and then composition on the left with the delta function δ_X (a component of the unit δ): $\langle X \rangle : X \rightarrow 1$. These two processes are inverse to each other: $\langle X \rangle_\emptyset = X$ for each cone (set) X and $\langle f_\emptyset \rangle = f$ for each mediator $f : X \rightarrow 1$.

```
(meta.ftn:function unique)
(= (meta.ftn:source unique) ur.set:set)
(= (meta.ftn:target unique) ur.ftn:function)
(= (meta.ftn:composition [unique ur.ftn:source])
   (meta.ftn:identity ur.set:set))
(= (meta.ftn:composition [unique ur.ftn:target])
   ((meta.ftn:constant [ur.set:set ur.set:set]) terminal))
(forall (?X (ur.set:set ?X))
  ?f (ur.ftn:function ?f)
    (= (ur.ftn:source ?f) ?X)
    (= (ur.ftn:target ?f) terminal))
  (= ?f (unique ?X)))
```

For any set X a *global element* (specializing a notion from category theory) is a function $x : 1 \rightarrow X$. The set of global elements and “ordinary” elements are isomorphic. A global element is an element of a “unary product”. Hence, the membership notion is subsumed into the function notion. In this namespace we do not define the bijective function (part of the isomorphism) that maps the elements of a set to the associated global elements. That element function is defined in the exponent namespace.

```

(meta.ftn:function global-element)
(= (meta.ftn:source global-element) ur.set:set)
(= (meta.ftn:target global-element) ur.set:set)
(forall (?X (ur.set:set ?X))
  (and (ur.set:isomorphic ?X (global-element ?X))
    (meta.set:subset (global-element ?X) ur.ftn:function)
    (= (global-element ?X)
      (the (?Y (ur.set:set ?Y))
        (forall (?x (ur.ftn:function ?x))
          (<=> ((global-element ?X) ?x)
            (and (= (ur.ftn:source ?x) terminal)
              (= (ur.ftn:target ?x) ?X))))))))))

```

1.3.2 Ur Binary Products

Namespace Prefix

Technical: ∞.prd2
Recommended: ur.prd2
Categorical: core.prd2.obj

$$\begin{array}{ccc}
 \text{Set}_n^2 & & \\
 \Delta \left| \begin{array}{c} \delta \\ + \\ \pi \end{array} \right| \times & \frac{\Delta(X) \rightarrow (Y_0, Y_1)}{X \rightarrow Y_0 \times Y_1} & \downarrow \cong \uparrow \\
 \text{Set}_n & &
 \end{array}$$

Adjunction

Packing/Unpacking

Objects. The essential properties of the binary product construction are illustrated above right. The horizontal bar designates a natural bijection between the *cone* above the bar and the *mediator* below the bar. A natural *pairing* (*packing*) process assigns a mediator $X \rightarrow Y_0 \times Y_1$ below the bar to every cone $\Delta(X) = (X, X) \rightarrow (Y_0, Y_1)$ above the bar. A natural *components* (*unpacking*) process assigns a cone $\Delta(X) = (X, X) \rightarrow (Y_0, Y_1)$ above the bar to every mediator $X \rightarrow Y_0 \times Y_1$ below the bar. These two processes are inverse to each other. The pairing process is realized by application of the product (\times) operation and then composition on the left with the delta function (a component of the unit δ). The components process is realized by application of the constant (Δ) operation and then composition on the right with the projection function pair (a component of the counit π).

The *cone* metaset consists of function pairs with constant (over diagram shape graph 2) source set pair. Binary product cones are treated both abstractly and concretely. Abstractly, a binary product cone determines a constrained pair (X, g) consisting of a set (or vertex) X and a function pair $g : \Delta(X) \rightarrow (Y_0, Y_1)$, where the source of g is the constant set pair over the set X . Concretely, a binary product cone is a function pair (g_0, g_1) , which share a common source set: $g_0 : X \rightarrow Y_0$ and $g_1 : X \rightarrow Y_1$. Hence, the binary product cone metaset is

a subset of the metaset of function pairs, with two related associated metafunctions mapping to the common source set and representing the cone to function pair inclusion morphism. In applications, such as the IFF Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since the pairing function (below) can be concretely referenced as `'(ur.prd2.obj:pairing [?g0 ?g1])'` rather than needing to specify a partial function from function pairs to mediators.

```
(meta.set:set cone) (meta.set:set span) (= span cone)
(meta.set:subset cone ur.pr.mor:function-pair)

(meta.ftn:function set) (meta.ftn:function vertex) (= vertex set)
(= (meta.ftn:source set) cone)
(= (meta.ftn:target set) ur.set:set)

(meta.ftn:function function-pair)
(= (meta.ftn:source function-pair) cone)
(= (meta.ftn:target function-pair) ur.pr.mor:function-pair)
(= function-pair (meta.set:inclusion [cone ur.pr.mor:function-pair]))

(= (meta.ftn:composition [function-pair ur.pr.mor:source])
    (meta.ftn:composition [set constant]))

(meta.ftn:function function0)
(= (meta.ftn:source function0) cone)
(= (meta.ftn:target function0) ur.ftn:function)
(= function0 (meta.ftn:composition [function-pair ur.pr.mor:function0]))

(meta.ftn:function function1)
(= (meta.ftn:source function1) cone)
(= (meta.ftn:target function1) ur.ftn:function)
(= function1 (meta.ftn:composition [function-pair ur.pr.mor:function1]))
```

The *mediator* metaset consists of functions with product target. A mediator consists of a function $f : X \rightarrow Y_0 \times Y_1$ and a set pair (Y_0, Y_1) , whose binary product is the target of the function. We introduce two convenience terms for the component sets Y_0 and Y_1 .

```
(meta.set:set mediator)

(meta.ftn:function function)
(= (meta.ftn:source function) mediator)
(= (meta.ftn:target function) ur.ftn:function)

(meta.ftn:function set-pair)
(= (meta.ftn:source set-pair) mediator)
(= (meta.ftn:target set-pair) ur.pr.obj:set-pair)

(= (meta.ftn:composition [set-pair product])
    (meta.ftn:composition [function ur.ftn:target]))

(meta.ftn:function set0)
(= (meta.ftn:source set0) mediator)
(= (meta.ftn:target set0) ur.set:set)
(= set0 (meta.ftn:composition [set-pair ur.pr.obj:set0]))

(meta.ftn:function set1)
(= (meta.ftn:source set1) mediator)
```

```
(= (meta.ftn:target set1) ur.set:set)
(= set1 (meta.ftn:composition [set-pair ur.pr.obj:set1])
```

Right adjoint and unit.

$$\frac{X \xrightarrow{1_X} X, X \xrightarrow{1_X} X}{X \xrightarrow{\delta_X} X \times X} \qquad \frac{Y_0 \times Y_1 \xrightarrow{\pi_0} Y_0, Y_0 \times Y_1 \xrightarrow{\pi_1} Y_1}{Y_0 \times Y_1 \xrightarrow{1} Y_0 \times Y_1}$$

Delta Mediator

Projection Cone

Given any Ur set X , the constant metafunction constructs the Ur set pair $\Delta(X) = (X, X)$ that is constantly (over diagram shape graph 2) the given set. Given any Ur set X , the delta metafunction constructs the mediator (above left) with Ur function $\delta_X : X \rightarrow (X \times X)$, which is the pairing of the cone whose function pair $(1_X, 1_X) : (X, X) \rightarrow (X, X)$ consists of the identity function for the given set pair – the delta mediator is the pairing of two copies of the identity function.

```
(meta.ftn:function constant)
(= (meta.ftn:source constant) ur.set:set)
(= (meta.ftn:target constant) ur.pr.obj:set-pair)
(= (meta.ftn:composition [constant ur.pr.obj:set0])
  (meta.ftn:identity ur.set:set))
(= (meta.ftn:composition [constant ur.pr.obj:set1])
  (meta.ftn:identity ur.set:set))

(meta.ftn:function delta-cone)
(= (meta.ftn:source delta-cone) ur.set:set)
(= (meta.ftn:target delta-cone) cone)
(= (meta.ftn:composition [delta-cone set]) (meta.ftn:identity ur.set:set))
(= (meta.ftn:composition [delta-cone function0]) ur.ftn:identity)
(= (meta.ftn:composition [delta-cone function1]) ur.ftn:identity)

(meta.ftn:function delta)
(= (meta.ftn:source delta) ur.set:set)
(= (meta.ftn:target delta) mediator)
(= (meta.ftn:composition [delta set0]) (meta.ftn:identity ur.set:set))
(= (meta.ftn:composition [delta set1]) (meta.ftn:identity ur.set:set))
(= delta (meta.ftn:composition [delta-cone packing]))
```

Left adjoint and counit. For any set pair (X_0, X_1) , the binary Cartesian product set $X_0 \times X_1$ and binary product projection functions $\pi_0 : X_0 \times X_1 \rightarrow X_0$ and $\pi_1 : X_0 \times X_1 \rightarrow X_1$ are defined. This is the binary product in \mathbf{Set}_∞ , the Ur category of sets and functions. These form a cone that is universal over the given set pair. Given any set pair (X_0, X_1) , the limit metafunction constructs the set $X_0 \times X_1$ that is the binary product of the given set pair. And the projection metafunction constructs the cone (above right), which is the components $\pi = 1_{01}$ of the mediator whose function $1_{X_0 \times X_1} : X_0 \times X_1 \rightarrow X_0 \times X_1$ is the identity function for the binary product of the given set pair – the projection cone is the components of the product identity. We introduce two convenience terms for the component projections. The binary product and the binary product projections

are concrete, since set-pair is concrete (by definition of ‘meta.set:set-square’, ‘meta.lim:square0’ and ‘meta.lim:square1’ in the meta namespace).

```
(meta.ftn:function projection-mediator)
(= (meta.ftn:source projection-mediator) ur.pr.obj:set-pair)
(= (meta.ftn:target projection-mediator) mediator)
(= (meta.ftn:composition [projection-mediator function])
    (meta.ftn:composition [limit ur.ftn:identity]))
(= (meta.ftn:composition [projection-mediator set-pair])
    (meta.ftn:identity ur.pr.obj:set-pair))

(meta.ftn:function projection)
(= (meta.ftn:source projection) ur.pr.obj:set-pair)
(= (meta.ftn:target projection) cone)
(= projection (meta.ftn:composition [projection-mediator unpacking]))

(meta.ftn:function limit) (meta.ftn:function product) (= product limit)
(= (meta.ftn:source limit) ur.pr.obj:set-pair)
(= (meta.ftn:target limit) ur.set:set)
(= limit (meta.ftn:composition [projection set]))

(meta.ftn:function projection0)
(= (meta.ftn:source projection0) ur.pr.obj:set-pair)
(= (meta.ftn:target projection0) ur.ftn:function)
(= (meta.ftn:composition [projection0 ur.ftn:source]) limit)
(= (meta.ftn:composition [projection0 ur.ftn:target]) ur.pr.obj:set0)
(= projection0 (meta.ftn:composition [projection function0]))

(meta.ftn:function projection1)
(= (meta.ftn:source projection1) ur.pr.obj:set-pair)
(= (meta.ftn:target projection1) ur.ftn:function)
(= (meta.ftn:composition [projection1 ur.ftn:source]) limit)
(= (meta.ftn:composition [projection1 ur.ftn:target]) ur.pr.obj:set1)
(= projection1 (meta.ftn:composition [projection function1]))
```

Packing and Unpacking.

$$\frac{X \xrightarrow{g_0} Y_0, X \xrightarrow{g_1} Y_1}{X \xrightarrow{(g_0, g_1)} Y_0 \times Y_1} \quad \downarrow \quad \frac{X \xrightarrow{f \cdot \pi_0} Y_0, X \xrightarrow{f \cdot \pi_1} Y_1}{X \xrightarrow{f} Y_0 \times Y_1} \quad \uparrow$$

packing or pairing

unpacking or components

The *pairing* of a cone packs it into a mediator. For any cone g with set X and function pair $(g_0, g_1) : \Delta(X) = (X, X) \rightarrow (Y_0, Y_1)$, there is a pairing mediator $(g) = (g_0, g_1) : X \rightarrow Y_0 \times Y_1$, which pairs the images of the original two functions. The packing (pairing) process is realized by application of the product (\times) operator to the function pair (g_0, g_1) and then composition on the left with the delta function δ_X (a component of the unit δ): $(g) = (g_0, g_1) = \delta_X \cdot (g_0 \times g_1) : X \rightarrow X \times X \rightarrow Y_0 \times Y_1$.

The *components* of a mediator unpack it into a cone. For any mediator f with set pair (Y_0, Y_1) and function $f : X \rightarrow Y_0 \times Y_1 = \times(Y_0, Y_1)$, there is a components cone $f_{01} = (f_0, f_1) : \Delta(X) \rightarrow (Y_0, Y_1)$, which separates the mediator into two component functions. The unpacking (components) process is realized by application of the constant (Δ) operator to the function f and then composition on the right with the function pair of the projection cone $\pi_{(Y_0, Y_1)}$ (a component of the counit π): $f_{01} = (f_0, f_1) = (f, f) \cdot \pi_{(Y_0, Y_1)} = (f \cdot \pi_0, f \cdot \pi_1) : (X, X) \rightarrow (Y_0 \times Y_1, Y_0 \times Y_1) \rightarrow (Y_0, Y_1)$.

These two processes are inverse to each other: $(g)_{01} = g$ for each cone g and $(f_{01}) = f$ for each mediator f .

```
(meta.ftn:function packing) (meta.ftn:function pairing) (= pairing packing)
(= (meta.ftn:source packing) cone)
(= (meta.ftn:target packing) mediator)
(= (meta.ftn:composition [packing set-pair])
    (meta.ftn:composition [function-pair ur.pr.mor:target]))
(= (meta.ftn:composition [packing function])
    ((meta.rel:pairing ur.ftn:composable)
     [(meta.ftn:composition [set delta])
      (meta.ftn:composition [function-pair ur.prd2.mor:product])]))

(meta.ftn:function unpacking) (meta.ftn:function components) (= components unpacking)
(= (meta.ftn:source unpacking) mediator)
(= (meta.ftn:target unpacking) cone)
(= (meta.ftn:composition [unpacking set])
    (meta.ftn:composition [function ur.ftn:source]))
(= (meta.ftn:composition [unpacking function-pair])
    ((meta.rel:pairing ur.pr.mor:composable)
     [(meta.ftn:composition [function ur.prd2.mor:constant])
      (meta.ftn:composition [(meta.ftn:composition [
        set-pair projection]) function-pair])]))

(= (meta.ftn:composition [packing unpacking]) (meta.ftn:identity cone))
(= (meta.ftn:composition [unpacking packing]) (meta.ftn:identity mediator))
```

Using the opspan of a set pair, we can show that the notion of a binary product can be based upon pullbacks and the terminal set. We do this by proving the theorem that the pullback of this opspan is the binary product set, and the pullback projections are the binary product projections. We can also prove the theorem that the pairing of a set pair is the pullback pairing of the associated opspan.

```
(= product      (meta.ftn:composition [ur.pr.obj:opspan ur.pbk.obj:pullback]))
(= projection0 (meta.ftn:composition [ur.pr.obj:opspan ur.pbk.obj:projection0]))
(= projection1 (meta.ftn:composition [ur.pr.obj:opspan ur.pbk.obj:projection1]))
(= pairing      (meta.ftn:composition [ur.pr.obj:opspan ur.pbk.obj:pairing]))
```

Morphisms.

$$\begin{array}{ccc}
 X_0 \times X_1 & \xrightarrow{g_0 \times g_1} & Y_0 \times Y_1 \\
 \pi_k \downarrow & k = 0, 1 & \downarrow \pi_k \\
 X_k & \xrightarrow{g_k} & Y_k
 \end{array}$$

The constant and binary product operations are extended to morphisms. Given any function $f : X \rightarrow Y$, there is a constant function pair $(f, f) : (X, X) \rightarrow (Y, Y)$, whose component functions are both f . Given any function pair $(g_0, g_1) : (X_0, X_1) \rightarrow (Y_0, Y_1)$, there is a binary product function $(g_0 \times g_1) : (X_0 \times X_1) \rightarrow (Y_0 \times Y_1)$ defined using projections: $(g_0 \times g_1) \cdot \pi_0 = \pi_0 \cdot g_0$ and $(g_0 \times g_1) \cdot \pi_1 = \pi_1 \cdot g_1$ (above diagram).

```
(meta.ftn:function constant)
(= (meta.ftn:source constant) ur.ftn:function)
(= (meta.ftn:target constant) ur.pr.mor:function-pair)
(= (meta.ftn:composition [constant ur.pr.mor:source])
    (meta.ftn:composition [ur.ftn:source ur.prd2.obj:constant]))
(= (meta.ftn:composition [constant ur.pr.mor:target])
    (meta.ftn:composition [ur.ftn:target ur.prd2.obj:constant]))
(= (meta.ftn:composition [constant ur.pr.mor:function0])
    (meta.ftn:identity ur.ftn:function))
(= (meta.ftn:composition [constant ur.pr.mor:function1])
    (meta.ftn:identity ur.ftn:function))

(meta.ftn:function limit) (meta.ftn:function product) (= product limit)
(= (meta.ftn:source limit) ur.pr.mor:function-pair)
(= (meta.ftn:target limit) ur.ftn:function)
(= (meta.ftn:composition [limit ur.ftn:source])
    (meta.ftn:composition [ur.pr.mor:source ur.prd2.obj:limit]))
(= (meta.ftn:composition [limit ur.ftn:target])
    (meta.ftn:composition [ur.pr.mor:target ur.prd2.obj:limit]))
(= ((meta.rel:pairing ur.ftn:composable)
    [limit (meta.ftn:composition [ur.pr.mor:target ur.prd2.obj:projection0])])
    ((meta.rel:pairing ur.ftn:composable)
    [(meta.ftn:composition [ur.pr.mor:source ur.prd2.obj:projection0])
     ur.pr.mor:function0]))
(= ((meta.rel:pairing ur.ftn:composable)
    [limit (meta.ftn:composition [ur.pr.mor:target ur.prd2.obj:projection1])])
    ((meta.rel:pairing ur.ftn:composable)
    [(meta.ftn:composition [ur.pr.mor:source ur.prd2.obj:projection1])
     ur.pr.mor:function1]))
```

1.3.3 Old Binary Product, Projections and Pairing

Pairs of Ur sets can be combined into a binary product set. The binary product of two sets X_0 and X_1 is the set of all pairs of elements $X_0 \times X_1 = \{(x_0, x_1) \mid x_0 \in X_0, x_1 \in X_1\}$.

```
(meta.ftn:function product)
(= (meta.ftn:source product) ur.pr.obj:set-pair)
(= (meta.ftn:target product) ur.set:set)
(forall (?X0 ?X1 (ur.pr.obj:set-pair [?X0 ?X1]))
  (and (forall (?w ((product [?X0 ?X1]) ?w))
        (exists (?x0 (?X0 ?x0) ?x1 (?X ?x1))
          (= ?w [?x0 ?x1])))
        (forall (?x0 (?X0 ?x0) ?x1 (?X ?x1))
          ((product [?X0 ?X1] [?x0 ?x1])))))
```

This product comes equipped with two projection functions, $\pi_0 : X_0 \times X_1 \rightarrow X_0 : (x_0, x_1) \mapsto x_0$ and $\pi_1 : X_0 \times X_1 \rightarrow X_1 : (x_0, x_1) \mapsto x_1$.

```

(meta.ftn:function projection0)
(= (meta.ftn:source projection0) ur.pr.obj:set-pair)
(= (meta.ftn:target projection0) ur.ftn:function)
(= (meta.ftn:composition [projection0 ur.ftn:source]) product)
(= (meta.ftn:composition [projection0 ur.ftn:target]) set0)
(forall (?X0 ?X1 (ur.pr.obj:set-pair [?X0 ?X1])
        ?x0 (?X0 ?x0) ?x1 (?X1 ?x1))
      (= ((projection0 [?X0 ?X1]) [?x0 ?x1]) ?x1))

```

```

(meta.ftn:function projection1)
(= (meta.ftn:source projection1) ur.pr.obj:set-pair)
(= (meta.ftn:target projection1) ur.ftn:function)
(= (meta.ftn:composition [projection1 ur.ftn:source]) product)
(= (meta.ftn:composition [projection1 ur.ftn:target]) set1)
(forall (?X0 ?X1 (ur.pr.obj:set-pair [?X0 ?X1])
        ?x0 (?X0 ?x0) ?x1 (?X1 ?x1))
      (= ((projection1 [?X0 ?X1]) [?x0 ?x1]) ?x1))

```

Two functions are *pairable* when the source of the first is equal to the source of the second.

```

(meta.rel:relation pairable)
(= (meta.rel:set0 pairable) ur.ftn:function)
(= (meta.rel:set1 pairable) ur.ftn:function)
(forall (?f (ur.ftn:function ?f) ?g (ur.ftn:function ?g))
      (<=> (pairable ?f ?g)
          (= (ur.ftn:source ?f) (ur.ftn:source ?g))))

```

We name the extent and projections of the pairable endorelation.

```

(meta.set:set function-cone)
(= function-cone (meta.rel:extent pairable))

(meta.ftn:function function0)
(= (meta.ftn:source function0) function-cone)
(= (meta.ftn:target function0) ur.ftn:function)
(= (meta.ftn:composition [function0 ur.ftn:source]) vertex)
(= (meta.ftn:composition [function0 ur.ftn:target]) set0)
(= function0 (meta.rel:projection0 pairable))

(meta.ftn:function function1)
(= (meta.ftn:source function1) function-cone)
(= (meta.ftn:target function1) ur.ftn:function)
(= (meta.ftn:composition [function0 ur.ftn:source]) vertex)
(= (meta.ftn:composition [function0 ur.ftn:target]) set0)
(= function0 (meta.rel:projection0 pairable))

```

Pairs of functions with common source have a unique pairing function, whose source is their common source and whose target is the binary product of their target sets. Composition of pairing with product projections results in the original pair of functions. The Ur pairing function is the restriction of the meta pairing function. We use the definite description operator to define pairing.

```

(meta.ftn:function pairing)
(= (meta.ftn:source pairing) function-cone)
(= (meta.ftn:target pairing) ur.ftn:function)
(forall (?f0 (ur.ftn:function ?f0) ?f1 (ur.ftn:function ?f1) (pairable ?f0 ?f1))
      (= (pairing [?f0 ?f1]) (meta.lim:pairing [?f0 ?f1])))

```

```

(forall (?f0 (ur.ftn:function ?f0) ?f1 (ur.ftn:function ?f1) (pairable ?f0 ?f1))
  (= (pairing [?f0 ?f1])
    (the (?f (ur.ftn:function ?f))
      (forall (?Y0 (ur.set:set ?Y0) (= ?Y0 (ur.ftn:target ?f0))
        ?Y1 (ur.set:set ?Y1) (= ?Y1 (ur.ftn:target ?f1)))
        (and (= (ur.ftn:source ?f) (ur.ftn:source ?f0))
          (= (ur.ftn:target ?f) (product [?Y0 ?Y1]))
          (= ?f0 (ur.ftn:composition [?f (projection0 [?Y0 ?Y1])]))
          (= ?f1 (ur.ftn:composition [?f (projection1 [?Y0 ?Y1])]))))))))

```

Given two set pairs (X_0, X_1) and (Y_0, Y_1) , if the components are subset ordered $X_0 \subseteq Y_0$ and $X_1 \subseteq Y_1$, then the products are subset ordered $X_0 \times X_1 \subseteq Y_0 \times Y_1$.

```

(forall (?X0 (ur.set:set ?X0) ?X1 (ur.set:set ?X1)
  ?Y0 (ur.set:set ?Y0) ?Y1 (ur.set:set ?Y1)
  (ur.set:subset ?X0 ?Y0) (ur.set:subset ?X1 ?Y1))
  (ur.set:subset (product [?X0 ?X1]) (product [?Y0 ?Y1])))

```

	Sets	Functions	Relations
set	<u>set</u> finite infinite	hom pair triple union <u>intersection</u> <u>power</u>	<u>subset</u> disjoint isomorphic
	subordinate	lesser greater inclusion reflex	
ftn	<u>function</u> = tuple function-function injection surjection bijection	<u>source</u> = arity <u>target</u> = type function0 function1 composition identity inverse ftn2rel	(<u>optimal</u> -) <u>restriction</u> composable
	relation	set0 set1 extent inclusion projection0 projection1 parametric pairing = mediator	<u>abridgment</u>
rel	endorelation reflexive antisymmetric symmetric transitive preorder partial-order equivalence-relation	base delta	
prd2	function-cone	<u>product</u> projection0 projection1 function0 function1 <u>pairing</u>	pairable

Technical Prefix : ∞
Recommended Prefix : ur

Table 2: The Level n Terminology

2 The Generic Levels

The terminology of the generic meta-ontology is listed in Table 2, which consists of only seventy-one terms representing sixty-one concepts (ten synonymies): twenty-one generic sets, consisting of three basic sets (set, function = tuple and relation), one auxiliary set (subordinate), two derived sets (function-function and function-cone), four subsets (endorelation, preorder, partial-order, equivalence-relation) and four specific sets (zero = nothing = null = empty = initial, one = unit = terminal, two, three), and seven predicates (injection, surjection, bijection, reflexive, symmetric, antisymmetric and transitive); thirty-two generic functions; and eight generic endorelations, consisting of three binary endorelations on sets (subset, disjoint and isomorphic), four binary endorelations on functions (restriction, optimal-restriction, pairable and composable), and one binary relations on relations (abridgment). The subset, restriction and abridgment binary relations are important for the preservation properties of the IFF core hierarchy (the “IFF metastack”).

2.1 Axiomatization

Namespace Prefix

Categorical: #n.Core

Alternate: #n.Set

The initial capital letter in ‘#n.Core’ makes this refer to the IFF Core (meta) Ontology (IFF-CORE). The alternate namespace prefix ‘#n.Core’ refers to the Set category for which this document is an axiomatization.

```
((#n+1).Cat:category Set}
(= ((#n+1).Cat:object Set) #n.set:set)
(= ((#n+1).Cat:morphism Set) #n.ftn:function)
(= ((#n+1).Cat:composition Set) #n.ftn:composition)
(= ((#n+1).Cat:identity Set) #n.ftn:identity)
```

The level n theory is expressed in the following axioms. The axioms specify the three basic kinds or sorts of things: sets, functions and relations. The axioms are partitioned accordingly. The axioms also specify one auxiliary kind or sort of things (subordinate pairs of sets), several derived kinds or sorts of things (composable pairs of functions and pairable pairs of functions), and relation subkinds (endorelations, preorders and equivalence relations).

At times there may a question at what level to place an itme (set, function or relation). Here are some guidelines. We start with sets at level n . A function or a relation, whose components (source, target, zeroth or first set) are at level n , is also at level n . At level n the basic items, such as set, function, relation, endorelation, etc., are all sets at level $n + 1$. At level n the composite items, such as binary product, function cone, etc., whose componenets are n level, are all sets at level $n + 1$.

2.1.1 Level n Sets

Namespace Prefix

Technical:	#n.set
Recommended:	#n.set
Categorical:	#n.Core.obj
Alternate:	#n.Set.obj

The *set* of all level n sets is a level $n + 1$ set. All level n sets are also level $n + 1$ sets; they are also generic sets. The level $n + 1$ set of level n sets is not a level n set.

```
((#n+1).set:set set)
((#n+2).set:subset set (#n+1).set:set)
(meta:subset set ur.set:set)
(not (set set))
```

There is a binary level n *subset* endorelationship between pairs of sets, where all elements of one are elements of the other. The level n subset endorelation is the abridgment of the level $n + 1$ and Ur subset endorelations. The subset relation is a partial order (reflexive, antisymmetric and transitive), since identities are injections and injections are close under composition.

```
((#n+1).rel:endorelation subset)
(= ((#n+1).rel:base subset) set)
((#n+2).rel:abridgment subset (#n+1).set:subset)
(meta:abridgment subset ur.set:subset)
((#n+1).rel:partial-order subset)
```

A pair of level n sets is *subordinate* when they satisfy the subset relation. The level n subordinate set is the extent of the level $n + 1$ subset relation. The level n subordinate set is also the intersection of the level $n + 1$ subordinate set with the second power of the set of all level n sets; and the intersection of the Ur subordinate set with the second power of the set of all level n sets. There are (level $n + 1$) projection functions to the lesser and greater components. For each subordinate pair of level n sets, there is an inclusion injection whose source is the lesser set and whose target is the greater set.

```
((#n+1).set:set subordinate)
(= subordinate ((#n+1).rel:extent subset))
((#n+1).set:subset subordinate ((#n+1).prd2:product [set set]))
(= subordinate ((#n+2).set:intersection
  [(#n+1).set:subordinate ((#n+1).prd2:product [set set])]))
(= subordinate (meta:intersection
  [ur.set:subordinate ((#n+1).prd2:product [set set])]))

((#n+1).ftn:function lesser)
(= ((#n+1).ftn:source lesser) subordinate)
(= ((#n+1).ftn:target lesser) set)
((#n+2).ftn:restriction lesser (#n+1).set:lesser)
(meta:restriction lesser ur.set:lesser)

((#n+1).ftn:function greater)
(= ((#n+1).ftn:source greater) subordinate)
(= ((#n+1).ftn:target greater) set)
((#n+2).ftn:restriction greater (#n+1).set:greater)
(meta:restriction greater ur.set:greater)

((#n+1).ftn:injection inclusion)
(= ((#n+1).ftn:source inclusion) subordinate)
(= ((#n+1).ftn:target inclusion) #n.ftn:injection)
((#n+2).ftn:restriction inclusion (#n+1).set:inclusion)
(meta:restriction inclusion ur.set:inclusion)
```

Every level n set forms a subordinate pair with itself.

```
((#n+1).ftn:function reflex)
(= ((#n+1).ftn:source reflex) set)
(= ((#n+1).ftn:target reflex) subordinate)
((#n+2).ftn:restriction reflex (#n+1).set:reflex)
(meta:restriction reflex ur.set:reflex)
```

There is a binary *disjointness* relation between pairs of sets, which have no elements in common. The level n disjointness endorelation is the abridgment of the level $n + 1$ and Ur disjointness endorelations. The disjointness relation is clearly symmetric.

```
((#n+1).rel:endorelation disjoint)
(= ((#n+1).rel:base disjoint) set)
((#n+2).rel:abridgment disjoint (#n+1).set:disjoint)
(meta:abridgment disjoint ur.set:disjoint)
((#n+1).rel:symmetric disjoint)
```

The notation $f : X \xrightarrow{\sim} Y$ means that f is an isomorphism. One set X is *isomorphic* to another set Y when there is at least one isomorphism from X to Y . This definition of isomorphic is used in all categories, but in a category of abstract sets and arbitrary functions the two sets X and Y are said to be *equinumerous* or *have the same cardinality*. There is a binary isomorphic relation between pairs of

sets, that are linked by an bijection. The level n isomorphic endorelation is the abridgment of the level $n+1$ isomorphic endorelation; and the abridgment of the Ur isomorphic endorelation. The isomorphic relation is an equivalence relation (reflexive, symmetric and transitive), since identities are bijections, bijections have an inverse and bijections are closed under composition.

```
((#n+1).rel:endorelation isomorphic)
(= ((#n+1).rel:base isomorphic) set)
((#n+2).rel:abridgment isomorphic (#n+1).set:isomorphic)
(meta:abridgment isomorphic ur.set:isomorphic)
((#n+1).rel:equivalence-relation isomorphic)
```

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. There is a level n set of infinite sets. The level n infinite set is the intersection of the level $n+1$ infinite set with the set of all level n sets; and the intersection of the Ur infinite set with the set of all level n sets.

```
((#n+1).set:set infinite)
((#n+1).set:subset infinite set)
(= infinite ((#n+2).set:intersection [(#n+1).set:infinite set]))
(= infinite meta:intersection [ur.set:infinite set]))
```

```
((#n+1).set:set finite)
((#n+1).set:subset finite set)
(= finite (#n+1).set:finite)
(= finite ur.set:finite)
```

For any particular level n set X , a *pair* of X -things (x_0, x_1) is an element in the binary Cartesian product $X^2 \cong X \times X$ (the implicit second Cartesian power of X); and, a *triple* of X -things (x_0, x_1, x_2) is an element in the (implicit) ternary Cartesian product or third Cartesian power $X^3 \cong X \times X \times X$. Pairs and triples are special cases of generic tuples (of X -things). More explicitly, an X -pair is a tuple with arity two and type X , and an X -triple is a tuple with arity three and type X .

```
((#n+2).ftn:function pair)
(= ((#n+2).ftn:source pair) set)
(= ((#n+2).ftn:target pair) (#n+1).set:set)
((#n+3).ftn:restriction pair (#n+1).set:pair)
(forall (?X (set ?X)) (= (pair ?X) (ur.set:pair ?X)))
(forall (?X (set ?X))
  (and ((#n+1).set:subset (pair ?X) #n.ftn:tuple)
        (isomorphic (pair ?X) (#n.prd2:product [?X ?X]))
        (= (pair ?X) (#n.ftn:hom [two ?X])))))
```

```
((#n+2).ftn:function triple)
(= ((#n+2).ftn:source triple) set)
(= ((#n+2).ftn:target triple) (#n+1).set:set)
((#n+3).ftn:restriction triple (#n+1).set:triple)
(forall (?X (set ?X)) (= (triple ?X) (ur.set:triple ?X)))
(forall (?X (set ?X))
  (and ((#n+1).set:subset (triple ?X) #n.ftn:tuple)
        (isomorphic (triple ?X) (#n.prd3:product [?X ?X ?X]))
        (= (triple ?X) (#n.ftn:hom [three ?X])))))
```

For any pair of level n sets X_0 and X_1 , there is a level n binary union set $X_0 \cup X_1$ and a level n binary intersection set $X_0 \cap X_1$. Binary union is the optimal restriction of level $n+1$ and Ur binary union, whereas binary intersection is only the restriction of level $n+1$ and Ur binary intersection.

```

((#n+1).ftn:function union)
(= ((#n+1).ftn:source union) ((#n+1).prd2:product [set set]))
(= ((#n+1).ftn:target union) set)
((#n+2).ftn:optimal-restriction union (#n+1).set:union)
(meta:optimal-restriction union ur.set:union)

((#n+1).ftn:function intersection)
(= ((#n+1).ftn:source intersection) ((#n+1).prd2:product [set set]))
(= ((#n+1).ftn:target intersection) set)
((#n+2).ftn:restriction intersection (#n+1).set:intersection)
(meta:restriction intersection ur.set:intersection)

```

For any level n set X , there is a *power* level n set $\wp X$, which consists of the set of all subsets of X . Power is the optimal restriction of level $n + 1$ and Ur power.

```

(meta:function power)
(= (meta:source power) set)
(= (meta:target power) set)
((#n+2).ftn:optimal-restriction power (#n+1).set:power)
(meta:optimal-restriction power ur.set:power)

```

2.1.2 Level n Functions

Namespace Prefix

Technical: #n.ftn
Recommended: #n.ftn
Categorical: #n.core.mor

The set of all level n *functions* is a level $n + 1$ set. Functions are also called *tuples*. Any level n function is both a level $n + 1$ function and an Ur function.

```

((#n+1).set:set function) ((#n+1).set:set tuple) (= tuple function)
((#n+2).set:subset function (#n+1).ftn:function)
(meta:subset function ur.ftn:function)

```

The function notation $f : X \rightarrow Y$, shows a function f with source set X and target set Y . Each function has a unique *source* set and a unique *target* set. As a tuple, the source is called the *arity* of the tuple and the target is called the *type* of the tuple. For any tuple (function) $x : I \rightarrow A$, the tuple notation $x = (i \in I \mid x_i \in A) = (\dots, x_i, \dots)$ is useful. Since tuples are synonymous with functions, we can use function application to index tuples: the components in the tuple x are represented by $x(i)$ for $i \in I$. The level n source (target) function is a restriction of the level $n + 1$ source (target) function, and also is a restriction of the Ur source (target) function. The level n source-target pairing function is the optimal restriction of the level $n + 1$ source-target pairing function, and also is the optimal restriction of the Ur source-target pairing function.

```

((#n+1).ftn:function source) ((#n+1).ftn:function arity) (= arity source)
(= ((#n+1).ftn:source source) function)
(= ((#n+1).ftn:target source) #n.set:set)
((#n+2).ftn:restriction source (#n+1).ftn:source)
(meta:restriction source ur.set:source)

```

```

((#n+1).ftn:function target) ((#n+1).ftn:function type) (= type target)
(= ((#n+1).ftn:source target) function)
(= ((#n+1).ftn:target target) #n.set:set)

```

```
((#n+2).ftn:restriction target (#n+1).ftn:target)
(meta:restriction target ur.set:target)
```

```
((#n+2).ftn:optimal-restriction
  ((#n+1).prd2:pairing [source target])
  ((#n+2).prd2:pairing [(#n+1).ftn:source (#n+1).ftn:target]))
(meta:optimal-restriction
  ((#n+1).prd2:pairing [source target])
  (meta:pairing [ur.ftn:source ur.ftn:target]))
```

For any pair of level n sets X and Y , there is a level $n + 1$ set $\text{hom}[X, Y]$, consisting of all functions $f : X \rightarrow Y$. The level n hom function is the restriction of its level $n + 1$ and Ur counterparts.

```
((#n+2).ftn:function hom)
(= ((#n+2).ftn:source hom) ((#n+1).prd2:product [#n.set:set #n.set:set]))
(= ((#n+2).ftn:target hom) ((#n+1).set:power function))
((#n+2).ftn:restriction hom (#n+1).ftn:hom)
(meta:restriction hom ur.ftn:hom)
```

Two functions are *composable* when the target of the first is equal to the source of the second. The level n composable endorelation is an abridgment of the level $n + 1$ composable endorelation and the Ur composable endorelation.

```
((#n+1).rel:endorelation composable)
(= ((#n+1).rel:base composable) function)
((#n+2).rel:abridgment composable (#n+1).ftn:composable)
(meta:abridgment composable ur.set:composable)
```

We name the level n extent and projection functions of the composable endorelation. The abridgment properties above mean that the level n extent is the intersection of the second power of the set of all level n functions with the level $n + 1$ extent set and the Ur extent set. The level n projection functions are the restrictions of their level $n + 1$ and Ur counterparts.

```
((#n+1).set:set function-function)
(= function-function ((#n+1).rel:extent composable))
(= function-function
  (intersection [(#n+1).prd2:product [function function]
    (#n+1).ftn:function-function]))
(= function-function
  (intersection [(#n+1).prd2:product [function function]
    ur.ftn:function-function]))
```

```
((#n+1).ftn:function function0)
(= ((#n+1).ftn:source function0) function-function)
(= ((#n+1).ftn:target function0) function)
((#n+1).ftn:restriction function0 (#n+1).ftn:function0)
(meta:restriction function0 ur.ftn:function0)
```

```
((#n+1).ftn:function function1)
(= ((#n+1).ftn:source function1) function-function)
(= ((#n+1).ftn:target function1) function)
((#n+1).ftn:restriction function1 (#n+1).ftn:function0)
(meta:restriction function1 ur.ftn:function1)
```

The *composition* of two composable functions $f : A \rightarrow B$ and $g : B \rightarrow C$ is a function $f \cdot g : A \rightarrow C$. Hence, the source of the composite is the source of the first component, and the target of the composite is the target of the second component. The level n composition function is the restriction of its level $n + 1$ and Ur counterparts.

```

((#n+1).ftn:function composition)
(= ((#n+1).ftn:source composition) function-function)
(= ((#n+1).ftn:target composition) function)
(= ((#n+1).ftn:composition [composition source])
  ((#n+1).ftn:composition [function0 source]))
(= ((#n+1).ftn:composition [composition target])
  ((#n+1).ftn:composition [function1 target]))
((#n+1).ftn:restriction composition (#n+1).ftn:composition)
(meta:restriction composition ur.ftn:composition)

```

For every level n set, there is a unique associated level n *identity* function. Composition with the identity of the source (target) of a function returns that function. Hence, composition is surjective. Because of uniqueness, identity is injective, and sets could be regarded as special functions that satisfy the unit laws. The level n identity function is the optimal restriction of the level $n + 1$ identity function, and also is the optimal restriction of the Ur identity function.

```

((#n+1).ftn:function identity)
(= ((#n+1).ftn:source identity) #n.set:set)
(= ((#n+1).ftn:target identity) function)
(= ((#n+1).ftn:composition [identity source])
  ((#n+1).ftn:identity #n.set:set))
(= ((#n+1).ftn:composition [identity target])
  ((#n+1).ftn:identity #n.set:set))
((#n+1).ftn:injection identity)
((#n+1).ftn:surjection composition)
((#n+2).ftn:optimal-restriction identity (#n+1).ftn:identity)
(meta:optimal-restriction identity ur.ftn:identity)

```

A level n function $f : A \rightarrow B$ is an *injection* when any image value is from a unique source element; or more categorically, when for any two parallel level n functions $h_0, h_1 : D \rightarrow A$ the equality $h_0 \cdot f = h_1 \cdot f$ implies the equality $h_0 = h_1$. The set of all level n injections is the intersection of the set of all level n functions with the set of all level $n + 1$ injections, and with the set of all Ur injections. An level n function $f : A \rightarrow B$ is an *surjection* when any target value is an image; or more categorically, when for any two parallel level n functions $g_0, g_1 : B \rightarrow C$ the equality $f \cdot g_0 = f \cdot g_1$ implies the equality $g_0 = g_1$. The set of all level n surjections is the intersection of the set of all level n functions with the set of all level $n + 1$ surjections, and with the set of all Ur surjections. An level n function $f : A \rightarrow B$ is an *bijection* when there is a(n *inverse*) level n function (in the opposite direction) $g : B \rightarrow A$ with $f \cdot g = 1_A$ and $g \cdot f = 1_B$. The set of all level n bijections is the intersection of the set of all level n functions with the set of all level $n + 1$ bijections, and with the set of all Ur bijections. Inverses are unique. The level n inverse function is the optimal restriction of the level $n + 1$ inverse function, and also is the optimal restriction of the Ur inverse function.

```

((#n+1).set:set injection)
((#n+1).set:subset injection function)
(= injection ((#n+1).set:intersection [function (#n+1).ftn:injection]))
(= injection (meta:intersection [function ur.ftn:injection]))

((#n+1).set:set surjection)
((#n+1).set:subset surjection function)
(= surjection ((#n+1).set:intersection [function (#n+1).ftn:surjection]))
(= surjection (meta:intersection [function ur.ftn:surjection]))

((#n+1).set:set bijection)

```

```

((#n+1).set:subset bijection function)
(= bijection ((#n+1).set:intersection [injection surjection]))
(= bijection ((#n+1).set:intersection [function (#n+1).ftn:bijection]))
(= bijection (meta:intersection [function ur.ftn:bijection]))

((#n+1).ftn:function inverse)
(= ((#n+1).ftn:source inverse) bijection)
(= ((#n+1).ftn:target inverse) bijection)
((#n+2).ftn:optimal-restriction inverse (#n+1).ftn:inverse)
(meta:optimal-restriction inverse ur.ftn:inverse)

(= ((#n+1).ftn:composition [inverse inverse])
    (#n+1).ftn:identity bijection))

```

For abstract sets and arbitrary functions, a bijection is exactly a function that is both injective and surjective.

```
(= bijection ((#n+1).set:intersection [injection surjection]))
```

Injections, surjections and bijections are closed under composition.

```

(forall (?f (injection ?f) ?g (injection ?g) (composable ?f ?g))
  (injection (composition [?f ?g])))
(forall (?f (surjection ?f) ?g (surjection ?g) (composable ?f ?g))
  (surjection (composition [?f ?g])))
(forall (?f (bijection ?f) ?g (bijection ?g) (composable ?f ?g))
  (bijection (composition [?f ?g])))

```

The identity is both a injection and an surjection; hence also, an bijection.

```

(forall (?X (#n.set:set ?X))
  (and (injection (identity ?X))
        (surjection (identity ?X))
        (bijection (identity ?X))))

```

There is a *function-to-relation* injection that embeds level n functions as level n relations. The domain (codomain) of the relation of a function is its source (target). The domain projection is an bijection, and post-composition with the original function gives the codomain projection. The level n function-to-relation function is the optimal restriction of the level $n+1$ function-to-relation function, and also is the optimal restriction of the Ur function-to-relation function.

```

((#n+1).ftn:function ftn2rel)
(= ((#n+1).ftn:source ftn2rel) function)
(= ((#n+1).ftn:target ftn2rel) #n.rel:relation)
((#n+2).ftn:optimal-restriction ftn2rel (#n+1).ftn:ftn2rel)
(meta:optimal-restriction ftn2rel ur.ftn:ftn2rel)

```

One level n function $f_1 : X_1 \rightarrow Y_1$ is a *restriction* of another level n function $f_2 : X_2 \rightarrow Y_2$ when the source (target) X_1 (Y_1) is a subset of the source (target) X_2 (Y_2) and the functions agree (on source elements of X_1); that is, the functions commute with the domain/target inclusions. Restriction can be viewed as a constraint on the larger function — it says that the larger function maps the source set of the smaller function into the target set of the smaller function. There is a binary restriction relationship between pairs of functions that are linked by source and target inclusions. The restriction relation is a partial order (reflexive, antisymmetric and transitive), since identities are injections, subset is antisymmetric, and injections are closed under composition. The level n restriction endorelation is the abridgment of the level $n+1$ restriction endorelation;

and the abridgment of the Ur restriction endorelation. A restriction between two functions f_1 and f_2 is *optimal* when the source set of the smaller function is exactly the inverse image of the target of the smaller function along the larger function. The optimal restriction relation is a partial order. The level n optimal-restriction endorelation is the abridgment of the level $n + 1$ optimal-restriction endorelation; and the abridgment of the Ur optimal-restriction endorelation.

```
((#n+1).rel:endorelation restriction)
(= ((#n+1).rel:base restriction) function)
((#n+2).rel:abridgment restriction (#n+1).ftn:restriction)
(meta:abridgment restriction ur.ftn:restriction)
((#n+1).rel:partial-order restriction)

((#n+1).rel:endorelation optimal-restriction)
(= ((#n+1).rel:base optimal-restriction) function)
((#n+2).rel:abridgment optimal-restriction (#n+1).ftn:optimal-restriction)
(meta:abridgment optimal-restriction ur.ftn:optimal-restriction)
((#n+1).rel:partial-order optimal-restriction)
```

2.1.3 Level n Relations

Namespace Prefix

Technical: #n.rel
Recommended: #n.rel
Categorical: #n.core.rel

The set of all level n *relations* is a level $n + 1$ set. Any level n relation is both a level $n + 1$ relation and an Ur relation.

```
((#n+1).set:set relation)
((#n+2).set:subset relation (#n+1).rel:relation)
(meta:subset relation ur.ftn:relation)
```

Each relation r has a unique domain or *zeroth set* X_0 and a unique codomain or *first set* X_1 . The level n zeroth (first) set function is the restriction of the level $n + 1$ zeroth (first) set function, and also is the restriction of the Ur zeroth (first) set function. The level n zeroth-first set pairing function is the optimal restriction of the level $n + 1$ zeroth-first set pairing function, and also is the optimal restriction of the Ur zeroth-first set pairing function.

```
((#n+1).ftn:function set0)
(= ((#n+1).ftn:source set0) relation)
(= ((#n+1).ftn:target set0) #n.set:set)
((#n+2).ftn:restriction set0 (#n+1).rel:set0)
(meta:restriction set0 ur.rel:set0)

((#n+1).ftn:function set1)
(= ((#n+1).ftn:source set1) relation)
(= ((#n+1).ftn:target set1) #n.set:set)
((#n+2).ftn:restriction set1 (#n+1).rel:set1)
(meta:restriction set1 ur.rel:set1)

((#n+2).ftn:optimal-restriction
  ((#n+1).prd2:pairing [set0 set1])
  ((#n+2).prd2:pairing [(#n+1).rel:set0 (#n+1).rel:set1]))
(meta:optimal-restriction
  ((#n+1).prd2:pairing [set0 set1])
  (meta:pairing [ur.rel:set0 ur.rel:set1]))
```

Each level n relation has a unique level n *extent* set. The extent is the subset of the binary product of the two component objects (domain and codomain) $\text{ext}(r) \subseteq X_0 \times X_1$, consisting of those pairs that satisfy the relation $(x_0, x_1) \in \text{ext}(r)$ iff $r(x_0, x_1)$. The level n extent function is the restriction of its level $n + 1$ and Ur counterparts.

```
((#n+1).ftn:function extent)
(= ((#n+1).ftn:source extent) relation)
(= ((#n+1).ftn:target extent) #n.set:set)
((#n+2).ftn:restriction extent (#n+1).rel:extent)
(meta:restriction extent ur.rel:extent)
```

For any relation, since the extent is a subobject of the binary product of domain and codomain, there are two unique projection functions. These relational projections are the composition of the inclusion with the product projections. The level n zeroth (first) projection function is the restriction of the level $n + 1$ zeroth (first) projection function, and also is the restriction of the Ur zeroth (first) projection function. The extent set and projection functions implicitly associate a span with a relation.

```
((#n+1).ftn:function projection0)
(= ((#n+1).ftn:source projection0) relation)
(= ((#n+1).ftn:target projection0) #n.ftn:function)
(= ((#n+1).ftn:composition [projection0 #n.ftn:source]) extent)
(= ((#n+1).ftn:composition [projection0 #n.ftn:target]) set0)
((#n+2).ftn:restriction projection0 (#n+1).rel:projection0)
(meta:restriction projection0 ur.rel:projection0)
```

```
((#n+1).ftn:function projection1)
(= ((#n+1).ftn:source projection1) relation)
(= ((#n+1).ftn:target projection1) #n.ftn:function)
(= ((#n+1).ftn:composition [projection1 #n.ftn:source]) extent)
(= ((#n+1).ftn:composition [projection1 #n.ftn:target]) set1)
((#n+2).ftn:restriction projection1 (#n+1).rel:projection1)
(meta:restriction projection1 ur.rel:projection1)
```

For any relation, the two projections are pairable. The pairing of the projections $(\pi_0, \pi_1) : \text{ext}(r) \rightarrow X_0 \times X_1$ is the *inclusion* of the extent into the product of the component sets. The level n inclusion function is the restriction of the level $n + 1$ inclusion function, and also is the restriction of the Ur inclusion function.

```
((#n+1).rel:parametric #n.prd2:pairable) projection0 projection1)
```

```
((#n+1).ftn:function inclusion)
(= ((#n+1).ftn:source inclusion) relation)
(= ((#n+1).ftn:target inclusion) #n.ftn:function)
((#n+2).ftn:restriction inclusion (#n+1).rel:inclusion)
(meta:restriction inclusion ur.rel:inclusion)
```

Any level n relation r can be used in a *parametric* fashion with a pairable pair of level n functions $f_0 : X \rightarrow \text{set}_0(r)$ and $f_1 : X \rightarrow \text{set}_1(r)$ that have a common level n source set X and the level n target sets $\text{set}_0(r)$ and $\text{set}_1(r)$. The parametric endorelation \hat{r} holds for f_0 and f_1 , $\hat{r}(f_0, f_1)$, when $r(f_0(x), f_1(x))$ for each element $x \in X$. Clearly, the extent of the parametric endorelation \hat{r} is contained in the function-cone level $n + 1$ set.

```
((#n+2).ftn:function parametric)
(= ((#n+2).ftn:source parametric) relation)
```

```

(= ((#n+2).ftn:target parametric) (#n+1).rel:endorelation)
((#n+3).ftn:restriction parametric (#n+1).rel:parametric)
(forall (?r (relation ?r))
  ((#n+1).set:subset (extent (parametric ?r)) #n.prd2:function-cone))
(forall (?r (relation ?r))
  ((#n+2).rel:abridgment (parametric ?r) ((#n+1).rel:parametric ?r)))
(forall (?r (relation ?r))
  (meta:abridgment (parametric ?r) (ur.rel:parametric ?r)))

```

There is a *pairing* or *mediator* function

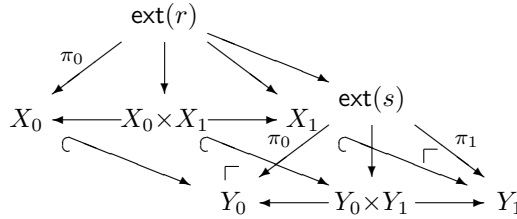
$$(-, -)_r : \text{ext}(\hat{r}) \rightarrow \text{ftn} : (f_0, f_1) \mapsto f : X \rightarrow \text{ext}(r) \subseteq \text{base}(r) \times \text{base}(r),$$

which gives the pairing of an r -parametric pair (f_0, f_1) restricted to $\text{ext}(r)$.

```

((#n+2).ftn:function pairing) ((#n+2).ftn:function mediator) (= mediator pairing)
(= ((#n+2).ftn:source pairing) endorelation)
(= ((#n+2).ftn:target pairing) (#n+1).ftn:function)
((#n+3).ftn:restriction pairing (#n+1).rel:pairing)
(forall (?r (endorelation ?r))
  (and (= ((#n+1).ftn:source (pairing ?r)) ((#n+1).rel:extent (parametric ?r)))
    (= ((#n+1).ftn:target (pairing ?r)) #n.ftn:funtion)
    (forall (?f0 (#n.ftn:funtion ?f0)
      ?f1 (#n.ftn:funtion ?f1)
      ((parametric ?r) ?f0 ?f1))
      (= (#n.ftn:source ((pairing ?r) [?f0 ?f1])) (#n.ftn:source ?f0))
      (= (#n.ftn:target ((pairing ?r) [?f0 ?f1])) (extent ?r))
      (= (#n.ftn:composition [((pairing ?r) [?f0 ?f1]) (inclusion ?r)])
        (#n.prd2:pairing [?f0 ?f1])))))

```



The notion of abridgment (Merriam-Webster) is that of “a shortened form of a work retaining the general sense and unity of the original”. One level n relation r is an abridgment of another level n relation s when the level n zeroth (domain) set X_0 and the level n first (codomain) set X_1 of r are subsets of the level n zeroth (domain) set Y_0 and the level n first (codomain) set Y_1 of s , respectively, and the extent of r is the “restriction” of the extent of s to the component sets of r . If relation r is an abridgment of relation s , then the extent of r is a subset of the extent of s . Abridgment can be expressed in terms of spans and pullbacks: the span of the relation r is the pullback of the span of the relation s along inclusions; or, the pairing of the relation r is the optimal restriction of the pairing of the relation s ; or, the extent of the smaller relation is the intersection of the extent of the larger relation with the product of the components of the smaller relation. This is a limit relationship between extents. There is a binary abridgment relationship between pairs of relations. The abridgment relation is a partial order (reflexive, antisymmetric and transitive). The level n abridgment endorelation is the abridgment of the level $n + 1$ abridgment endorelation; and the abridgment of the Ur abridgment endorelation.

```

((#n+1).rel:endorelation abridgment)
(= ((#n+1).rel:base abridgment) relation)
((#n+2).rel:abridgment abridgment (#n+1).rel:abridgment)
(meta:abridgment abridgment ur.rel:abridgment)
((#n+1).rel:partial-order abridgment)

```

Level n Endorelations. Level n *endorelations* are special level n relations, whose domain and codomain are identical. This set is called the underlying *base* set.

```

((#n+1).set:set endorelation)
((#n+1).set:subset endorelation relation)
(forall (?r (relation ?r))
  (<=> (endorelation r)
    (= (set0 ?r) (set1 ?r))))

((#n+1).ftn:function base)
(= ((#n+1).ftn:source base) endorelation)
(= ((#n+1).ftn:target base) #n.set:set)
(= base ((#n+1).ftn:composition
  [((#n+1).rel:inclusion [endorelation relation]) set0]))
(= base ((#n+1).ftn:composition
  [((#n+1).rel:inclusion [endorelation relation]) set1]))

```

We have unary predicates (level n sets) to express the fact that an level n endorelation is *reflexive*, *symmetric*, *antisymmetric* or *transitive*.

```

((#n+1).set:set reflexive)
((#n+1).set:subset reflexive endorelation)
(= reflexive ((#n+2).set:intersection [endorelation (#n+1).rel:reflexive])
(= reflexive (meta:intersection [endorelation ur.rel:reflexive])

((#n+1).set:set symmetric)
((#n+1).set:subset symmetric endorelation)
(= symmetric ((#n+2).set:intersection [endorelation (#n+1).rel:symmetric])
(= symmetric (meta:intersection [endorelation ur.rel:symmetric])

((#n+1).set:set antisymmetric)
((#n+1).set:subset antisymmetric endorelation)
(= antisymmetric ((#n+2).set:intersection [endorelation (#n+1).rel:antisymmetric])
(= antisymmetric (meta:intersection [endorelation ur.rel:antisymmetric])

((#n+1).set:set transitive)
((#n+1).set:subset transitive endorelation)
(= transitive ((#n+2).set:intersection [endorelation (#n+1).rel:transitive])
(= transitive (meta:intersection [endorelation ur.rel:transitive])

```

We can declare special level n endorelations called *preorders*, *partial-orders* and *equivalence relations*. Preorders are reflexive and transitive. Partial orders are preorders that are antisymmetric. Equivalence relations are reflexive, symmetric and transitive.

```

((#n+1).set:set preorder)
((#n+1).set:subset preorder endorelation)
(= preorder ((#n+1).set:intersection [reflexive transitive])
(= preorder ((#n+2).set:intersection [endorelation (#n+1).rel:preorder])
(= preorder (meta:intersection [endorelation ur.rel:preorder])

```

```

((#n+1).set:set partial-order)
((#n+1).set:subset partial-order endorelation)
(= partial-order ((#n+1).set:intersection [preorder antisymmetric])
(= partial-order ((#n+2).set:intersection [endorelation (#n+1).rel:partial-order])
(= partial-order (meta:intersection [endorelation ur.rel:partial-order])

((#n+1).set:set equivalence-relation)
((#n+1).set:subset equivalence-relation endorelation)
(= equivalence-relation ((#n+1).set:intersection [preorder symmetric])
(= equivalence-relation ((#n+2).set:intersection [endorelation (#n+1).rel:equivalence-relation])
(= equivalence-relation (meta:intersection [endorelation ur.rel:equivalence-relation])

```

Any reflexive level n endorelation has a *delta* function from its base to its extent. Delta is the optimal restriction of level $n + 1$ and Ur delta.

```

((#n+1).ftn:function delta)
(= ((#n+1).ftn:source delta) reflexive)
(= ((#n+1).ftn:target delta) #n.ftn:function)
(= ((#n+1).ftn:composition [delta #n.ftn:source])
    ((#n+1).ftn:composition [(#n+1).rel:inclusion [reflexive endorelation]) base]))
(= ((#n+1).ftn:composition [delta #n.ftn:target])
    ((#n+1).ftn:composition [(#n+1).rel:inclusion [reflexive relation]) extent]))
((#n+2).ftn:optimal-restriction delta (#n+1).set:delta)
(meta:optimal-restriction delta ur.set:delta)

```

If the endorelation r is reflexive, symmetric, antisymmetric, transitive, pre-ordered, partially-ordered or an equivalence relation, then so is the parametric endorelation \hat{r} .

```

(forall (?r (reflexive ?r))
  ((#n+1).rel:reflexive (parameteric ?r)))
(forall (?r (transitive ?r))
  ((#n+1).rel:transitive (parameteric ?r)))
(forall (?r (antisymmetric ?r))
  ((#n+1).rel:antisymmetric (parameteric ?r)))
(forall (?r (symmetric ?r))
  ((#n+1).rel:symmetric (parameteric ?r)))
(forall (?r (preorder ?r))
  ((#n+1).rel:preorder (parameteric ?r)))
(forall (?r (partial-order ?r))
  ((#n+1).rel:partial-order (parameteric ?r)))
(forall (?r (equivalence-relation ?r))
  ((#n+1).rel:equivalence-relation (parameteric ?r)))

```

2.1.4 Level n Binary Products

Namespace Prefix

Technical: #n.prd2
Recommended: #n.prd2
Categorical: #n.core.prd2.obj

Pairs of level n sets can be combined into a binary product level n set. The binary product of two level n sets X_0 and X_1 is the level n set of all pairs of elements $X_0 \times X_1 = \{(x_0, x_1) \mid x_0 \in X_0, x_1 \in X_1\}$. Binary product is the restriction of level $n + 1$ and Ur binary product.

```

((#n+1).ftn:function product)
(= ((#n+1).ftn:source product) ((n+1).prd2:product [#n.set:set #n.set:set]))
(= ((#n+1).ftn:target product) #n.set:set)
((#n+2).ftn:restriction product (#n+1).prd2:product)
(meta:restriction product ur.prd2:product)

```

This product comes equipped with two projection functions, $\pi_0 : X_0 \times X_1 \rightarrow X_0 : (x_0, x_1) \mapsto x_0$ and $\pi_1 : X_0 \times X_1 \rightarrow X_1 : (x_0, x_1) \mapsto x_1$. Binary product projection is the restriction of level $n + 1$ and Ur binary product projection.

```

((#n+1).ftn:function projection0)
(= ((#n+1).ftn:source projection0) ((#n+1).prd2:product [#n.set:set #n.set:set]))
(= ((#n+1).ftn:target projection0) #n.ftn:function)
((#n+2).ftn:restriction projection0 (#n+1).prd2:projection0)
(meta:restriction projection0 ur.prd2:projection0)

```

```

((#n+1).ftn:function projection1)
(= ((#n+1).ftn:source projection1) ((#n+1).prd2:product [#n.set:set #n.set:set]))
(= ((#n+1).ftn:target projection1) #n.ftn:function)
((#n+2).ftn:restriction projection1 (#n+1).prd2:projection1)
(meta:restriction projection1 ur.prd2:projection1)

```

Two functions are *pairable* when the source of the first is equal to the source of the second. There is a level n subset endorelation, which is the abridgment of the level $n + 1$ and Ur subset endorelation.

```

((#n+1).rel:endorelation pairable)
(= ((#n+1).rel:set0 pairable) #n.ftn:function)
(= ((#n+1).rel:set1 pairable) #n.ftn:function)
((#n+2).rel:abridgment pairable (#n+1).prd2:pairable)
(meta:abridgment pairable ur.prd2:pairable)

```

We name the extent and projections of the pairable endorelation. The extent, a subset of the second power $\text{ftn}_n \times \text{ftn}_n$ of the set of level n functions, is called the level n set of function cones. The projections are the zeroth and first functions. The level n function cone set is the intersection of $\text{ftn}_n \times \text{ftn}_n$ with the level n and Ur function cone sets. The projections are the restrictions of their level n and Ur counterparts.

```

((#n+1).set:set function-cone)
((#n+1).set:subset function-cone ((#n+1).prd2:product [function function]))
(= function-cone ((#n+2).set:intersection
  [((#n+1).prd2:product [function function])
  (#n+1).prd2:function-cone]))
(= function-cone (meta:intersection
  [((#n+1).prd2:product [function function])
  ur.prd2:function-cone]))

```

```

((#n+1).ftn:function function0)
(= ((#n+1).ftn:source function0) function-cone)
(= ((#n+1).ftn:target function0) #n.ftn:function)
((#n+2).ftn:restriction function0 (#n+1).prd2:function0)
(meta:restriction function0 ur.prd2:function0)

```

```

((#n+1).ftn:function function1)
(= ((#n+1).ftn:source function1) function-cone)
(= ((#n+1).ftn:target function1) #n.ftn:function)
((#n+2).ftn:restriction function1 (#n+1).prd2:function1)
(meta:restriction function1 ur.prd2:function1)

```

Pairs of functions with common source have a unique pairing function, whose source is their common source and whose target is the binary product of their target objects. Composition of pairing with product projections results in the original pair of functions. Pairing is the restriction of its level n and Ur counterparts.

```
((#n+1).ftn:function pairing)
(= ((#n+1).ftn:source pairing) function-cone)
(= ((#n+1).ftn:target pairing) #n.ftn:function)
((#n+2).ftn:restriction pairing (#n+1).prd2:pairing)
(meta:restriction pairing ur.prd2:pairing)
```

Given two level n set pairs (X_0, X_1) and (Y_0, Y_1) , if the components are subset ordered $X_0 \subseteq Y_0$ and $X_1 \subseteq Y_1$, then the level n products are subset ordered $X_0 \times X_1 \subseteq Y_0 \times Y_1$.

```
(forall (?X0 (#n.set:set ?X0) ?X1 (#n.set:set ?X1)
        ?Y0 (#n.set:set ?Y0) ?Y1 (#n.set:set ?Y1)
        (#n.set:subset ?X0 ?Y0) (#n.set:subset ?X1 ?Y1))
 (#n.set:subset (product [?X0 ?X1]) (product [?Y0 ?Y1])))
```

References

- [1] F. William Lawvere and Robert Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.