The IFF Top Core (meta) Ontology

IFF TCO

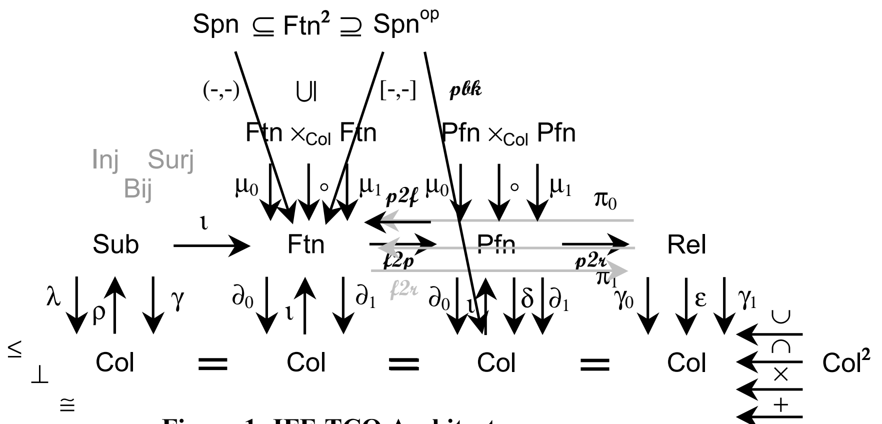


Figure 1: IFF-TCO Architecture

Key

λ = *lesser*	∂0 = *src*	p2f = *pfn2ftn*	γ0 = *col*0	∪ = *intersect*
γ = *greater*	∂1 = *tgt*	f2p = *ftn2pfn*	γ1 = *col*1	∩ = *union*
ρ = *reflex*	∘ = *comp*	p2r = *pfn2rel*	ε = *ext*	× = *prod*
ι = *incl*	ι = *id*		π0 = *proj*0	+ = *sum*
	μ0 = *mor*0		π1 = *proj*1	
	μ1 = *mor*1			

Figure 1 illustrates the architecture for the IFF-TCO. Diagram 1 illustrates the IFF Stack (the IFF core hierarchy), where the IFF Top Core (meta) Ontology (IFF-TCO) is colored in red. The IFF Top Core (meta) Ontology (previously known as the Basic KIF Ontology) is situated at the top metalevel – the highest level of the IFF Foundation Ontology other than Ur. Its purpose is to provide an interface between the KIF logical language and the SUO ontological structure. Principally, it does this by servicing the upper metalevel. It contains rudimentary (fundamental) namespaces for collections, functions, relations and finite limits. The IFF Top Core Ontology provides an adequate foundation for representing ontologies in general and for defining other metalevel ontologies in particular. The IFF Ur (meta) Ontology follows Mac Lane's beginning axiomatization for category theory (Mac Lane, 1971) in that it introduces terminology and provides an axiomatization for this terminology, but it does not give a formal interpretation using set theory – it only gives an informal, intuitive interpretation. The IFF Top Core (meta) Ontology initiates such a formal interpretation. The IFF Upper and Lower Core (meta) Ontologies complete such a formal interpretation.

Diagram 1: The IFF Stack

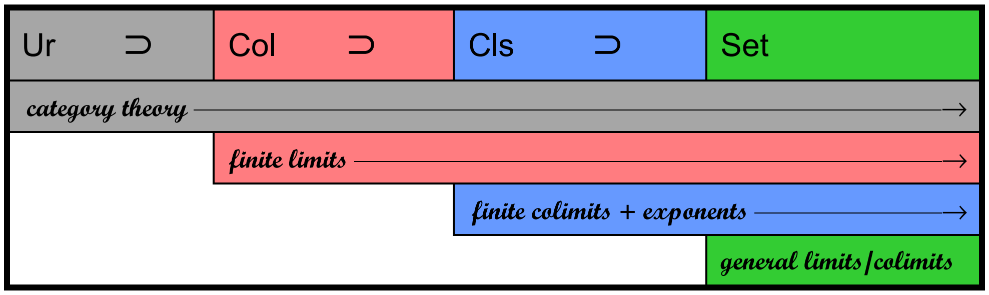


Table 1 lists all 315 terms (274 concepts or non-identical terms) in the Top Core Ontology, partitioned according to whether the term is a basic term (collections, relations or functions), a diagram term or a limit term. Ignoring the 10 terms used to designate the four indexing collections, there are a total of 305 terms partitioned into 58 basic terms designating 55 basic concepts dealing with collections, (partial) functions and relations, 103 terms designating 95 concepts of finite diagrams, and 144 terms designating 120 concepts of finite limits proper. Terminology has been placed in the IFF Top Core Ontology only when it is needed in the IFF upper metalevel[§]. All upper metalevel ontologies import and use, either directly or indirectly, the IFF Top Core Ontology. This includes the upper core, upper graph and upper category theory meta-ontologies.

Table 1: The KIF language – terms specified in the Top Core meta-ontology

	Ur Object	Ur Morphism	Ur Relation
`KIF`	`thing`		

collections

`KIF .COL`	`collection` `subordinate`	`lesser greater inclusion reflex`	`subcollection` `subcollection-parameterized`

	`pair triple`
	`collection-tuple`
		`binary-union binary-intersection`	`disjoint isomorphic`

	`zero = nothing = null = empty = initial one = unit = terminal two three`		

functions

`KIF .PFN`	`partial-function`
	`partial-function-`
	`function`
		`source target domain`
		`partial-function0 partial-function1`
		`composition identity`
		`pfn2ftn pfn2rel`
			`restriction`

`KIF .FTN`	`function = tuple`
	`function-function`
	`injection`
	`surjection`
	`bijection`
		`source = arity target = type`
		`function0 function1`
		`composition identity`
		`ftn2pfn ftn2rel`
			`restriction`
			`restriction-parameterize`

[§] The 'abridgment' term was placed here, in order to be able to precisely express the relationship between relations in the upper metalevel to their counterparts in this ontology ('subclass', 'disjoint' and 'restriction'). The 'pullback' term was placed here, since this is needed in the Upper Core Ontology to define the two conversions of functions to relations based upon a preorder (which in turn is used to define the instance and type embedding relations in the Upper Classification Ontology). The 'partial-function' was placed here, in order to be able to express (in a simple fashion) in the Upper Core Ontology the 'pairing' operator for pullbacks (which in turn is used in many different places in the Upper Category Theory Ontology; for example, with the composition opspan).

relations

KIF .REL	relation total functional total-functional	collection0 collection1 extent projection0 projection1	abridgment abridgment-parameterize subrelation

finite diagrams

KIF.DGM		
KIF.DGM.PR.OBJ	collection-pair	collection0 collection1 opspan
KIF.DGM.PR.MOR	function-pair composable-pair-function-pair	source target function0 function1 composition identity
KIF.DGM.TRP.OBJ	collection-triple	collection0 collection1 collection2
KIF.DGM.TRP.MOR	function-triple composable-pair-function-triple	source target function0 function1 function2 composition identity
KIF.DGM.PPR.OBJ	parallel-pair	collection-pair collection0 collection1 function0 function1 opspan
KIF.DGM.PPR.MOR	parallel-pair-morphism	source target function-pair function0 function1 composition identity
KIF.DGM.OSPN.OBJ	opspan	opvertex opzeroth opfirst collection-pair collection0 collection1 relation parallel-pair
	opspan-morphism composable-opspan-morphism	source target opvertex function-pair function0 function1 composition identity
KIF.DGM.MOSPN.OBJ	multi-opspan	opspan0 opvertex0 opfunction00 opfunction01 collection-pair0 collection00 collection01 opspan1 opvertex1 opfunction10 opfunction11 collection-pair1 collection10 collection11 collection-triple collection0 = collection00 collection = collection01 = collection10 collection1 = collection11
		source target opspan0 opvertex0 function-pair0 function00 function01 opspan1 opvertex1 function-pair1 function10 function11 function-triple function0 = function00 function = function 01 = function10 function1 = function11

finite limits

KIF.LIM		terminal unique
		global-element
KIF.LIM .PRD2.OBJ	cone = span	collection = vertex function-pair function0 function1
binary products	mediator	function collection-pair collection0 collection1
		constant delta-cone delta
		limit = binary-product projection-mediator projection projection0 projection1
		packing = pairing unpacking = components
KIF.LIM .PRD2.MOR		constant limit = binary-product
KIF.LIM .PRD3.OBJ	cone	collection = vertex function-triple function0 function1 function2
ternary products	mediator	function collection-triple collection0 collection1 collection2
		constant delta-cone delta
		limit = ternary-product projection-mediator projection projection0 projection1 projection2
		packing = tripling unpacking = components
KIF.LIM .PRD3.MOR		constant limit = ternary-product
KIF.LIM .EQU.OBJ	cone	collection = vertex parallel-pair-morphism function0
equalizers	mediator	function parallel-pair
		constant delta-cone delta
		limit = equalizer projection-mediator projection part

		packing unpacking
KIF.LIM .EQU.MOR		constant limit = equalizer
KIF.LIM .PBK.OBJ *pullbacks*	cone	collection = vertex opspan-morphism function-pair function0 function1
	mediator	function opspan collection-pair collection0 collection1
		constant delta-cone delta
		limit = pullback projection-mediator projection projection0 projection1
		packing = pairing unpacking = components
		kernel-pair-opspan kernel-pair
KIF.LIM .PBK.MOR		constant limit = pullback
KIF.LIM .MPBK.OBJ *multi-pullbacks*	cone	collection = vertex multi-opspan-morphism function-triple function0 function2 function1
	mediator	function multi-opspan collection-triple collection0 collection2 collection1
		constant delta-cone delta
		limit = multipullback projection-mediator projection0 projection2 projection1
		packing = tripling unpacking = components
KIF.LIM .PBK.MOR		constant limit = multipullback

Things

KIF

The central KIF Ur-objects are *thing*, *collection*, *relation*, *partial function* and *function*. Less central Ur objects will also be declared and axiomatized. Anything is a collection, a relation or a partial function; but not any two. That is, collections, relations and partial functions are pair-wise disjoint. Functions are special kinds of partial functions. Any KIF-thing is a UR-thing.

```
(1)  (UR$object thing)

(2)  (forall (?x (thing ?x))
         (UR$thing ?x))

(3)  (UR$subobject collection thing)
     (UR$subobject partial-function thing)
     (UR$subobject relation thing)

(4)  (UR$disjoint partial-function collection)
     (UR$disjoint collection relation)
     (UR$disjoint relation partial-function)
```

Collections

`KIF.COL`

The Ur-object *collection* represents the notion of a generic set. Any collection is a UR-object. The Ur-object of all collections is not itself a collection.

```
(1) (UR$object collection)

(2) (forall (?c (collection ?c))
        (UR$object ?c))

(3) (not (collection collection))
```

$$\begin{tikzcd}
 & \mathit{ext}(\text{subcol}) \arrow[dl, "\pi_0"'] \arrow[d] \arrow[r, "\pi_1"] \arrow[rrd] & & \\
\text{Col} \arrow[dr] & \text{Col} \times \text{Col} \arrow[l] \arrow[r] \arrow[dr] & \text{Col} \arrow[dr] & \mathit{ext}(\text{subobj}) \arrow[dll, "\pi_0"'] \arrow[dl] \arrow[d, "\pi_1"] \\
 & \text{Obj} & \text{Obj} \times \text{Obj} \arrow[l] \arrow[r] & \text{Obj}
\end{tikzcd}$$

Figure 2: Subcollection Relation

For any two collections C_0 and C_1, C_0 is a *subcollection* of C_1 (Figure 2), denoted by $C_0 \leq C_1$ or $C_0 \sqsubseteq C_1$, when every instance of C_0 is an instance of C_1. In that situation, C_0 is an Ur-subobject of C_1. For any two Ur-objects O_0 and O_1 where O_0 is an Ur-subobject of O_1, O_0 is a subcollection of O_1 iff O_0 and O_1 are both collections. The span of the (KIF) subcollection relation is the pullback of the span of the Ur-subobject relation along inclusions (but this cannot be explicitly expressed). The subobject relation is reflexive and transitive.

```
(4) (UR$relation subcollection)
    (= (UR$object0 subcollection) collection)
    (= (UR$object1 subcollection) collection)

(5) (forall (?c0 (collection ?c0) ?c1 (collection ?c1) (subcollection ?c0 ?c1))
        (UR$subobject ?c0 ?c1))

(6) (forall (?o0 (UR$object ?o0) ?o1 (UR$object ?o1) (UR$subobject ?o0 ?o1))
        (<=> (subcollection ?o0 ?o1)
             (and (collection ?o0) (collection ?o1))))

(7) (forall (?c0 (collection ?c0) ?c1 (collection ?c1))
        (<=> (subcollection ?c0 ?c1)
             (forall (?x (KIF$thing ?x)) (=> (?c0 ?x) (?c1 ?x)))))

(8) (forall (?x (object ?x))
        (subcollection ?x ?x))

(9) (forall (?x (object ?x) ?y (object ?y) ?z (object ?z))
        (=> (and (subcollection ?x ?y) (subcollection ?y ?z))
            (subcollection ?x ?z)))
```

The *subcollection* relation can be used in a *parametric* fashion with pairs of Ur-morphisms $m_0 : \mathsf{A} \rightarrow \mathsf{col}$ and $m_1 : \mathsf{A} \rightarrow \mathsf{col}$, which have a common source Ur-object A and each have the target Ur-object col of all collections. The subcollection-parametric relation holds for m_0 and m_1, when $m_0(x) \sqsubseteq m_1(x)$ for each element $x \in \mathsf{A}$. The parametric subcollection relation is reflexive and transitive.

```
(10) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1))
         (<=> (subcollection-parameterized ?m0 ?m1)
              (and (= (UR$source ?m0) (UR$source ?m1))
                   (= (UR$target ?m0) collection)
                   (= (UR$target ?m1) collection)
                   (forall (?x ((UR$source ?m0) ?x)
                       (subcollection (?m0 ?x) (?m1 ?x))))))
```

```
(11) (forall (?m (UR$morphism ?m))
         (subcollection-parameterized ?m ?m))

(12) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1) ?m2 (UR$morphism ?m2))
         (=> (and (subcollection-parameterized ?m0 ?m1) (subcollection-parameterized ?m1 ?m2))
             (subcollection-parameterized ?m0 ?m2)))
```

A *subordinate* pair of collections is a pair that stands in subcollection. Each subordinate pair has two projections, the *lesser* collection and the *greater* collection. The subordinate object is the Ur-extent of the subcollection Ur-relation, and the lesser and greater Ur-morphisms are the projections of the subcollection relation. Our purpose is the make explicit the very useful inclusion function (at least at the class level) between the extent of the subcollection relation and the binary cartesian product of collection with itself. Any collection is *reflex*ively subordinate to itself. For any two collections that are ordered by inclusion (subcollection) $C_0 \subseteq C_1$, there is an *inclusion* KIF function $C_0 \rightarrow C_1$. Any subordinate pair is an Ur subordinate pair and the lesser, greater, inclusion and reflex functions are restrictions of the analogous Ur functions. An Ur subordinate pair is a (KIF) subordinate pair iff the lesser and greater objects are collections.

```
(13) (UR$object subordinate)
     (= subordinate (UR$extent subcollection))

(14) (forall (?c0c1 (subordinate ?c0c1))
         (UR$subordinate ?c0c1))

(15) (forall (?o0o1 (UR$subordinate ?o0o1))
         (<=> (subordinate ?o0o1)
              (and (collection (UR$lesser ?o0o1))
                   (collection (UR$greater ?o0o1)))))

(16) (UR$morphism lesser)
     (= (UR$source lesser) subordinate)
     (= (UR$target lesser) collection)
     (= lesser (UR$projection0 subcollection))

(17) (UR$morphism greater)
     (= (UR$source greater) subordinate)
     (= (UR$target greater) collection)
     (= greater (UR$projection1 subcollection))

(18) (UR$morphism reflex)
     (= (UR$source reflex) collection)
     (= (UR$target reflex) subordinate)
     (= (UR$composition [reflex lesser]) (UR$identity collection))
     (= (UR$composition [reflex greater]) (UR$identity collection))

(19) (UR$morphism inclusion)
     (= (UR$source inclusion) subordinate)
     (= (UR$target inclusion) KIF.FTN$function)
     (= (UR$composition [inclusion KIF.FTN$source]) lesser)
     (= (UR$composition [inclusion KIF.FTN$target]) greater)

(20) (forall (?xy (UR$subordinate ?xy))
         (=> (subordinate ?xy)
             (and (= (UR$lesser ?xy) (lesser ?xy))
                  (= (UR$greater ?xy) (greater ?xy))
                  (= (UR$inclusion ?xy) (inclusion ?xy)))))

(21) (forall (?x (collection ?x))
         (= (UR$reflex ?x) (reflex ?x)))

(22) (forall (?c0 (collection ?c0) ?c1 (collection ?c1) (subcollection ?c0 ?c1))
         (and (= (lesser [?c0 ?c1]) ?c0)
              (= (greater [?c0 ?c1]) ?c1)))
              (forall (?x (?c0 ?x))
                  (= ((inclusion [?c0 ?c1]) ?x) ?x))))
```

For any two collections C_0 and C_1, C_0 is *disjoint* from C_1 (Figure 3), denoted $C_0 \perp C_1$, when there is no instance of both C_0 and C_1: $C_0 \perp C_1 = \emptyset$. In that situation, the Ur-object C_0 is Ur-disjoint from the Ur-object C_1. For any two Ur-objects O_0 and O_1 where O_0 is Ur-disjoint from O_1, O_0 is (KIF) disjoint from O_1 iff O_0

$$\begin{tikzcd}
 & \mathit{ext}(\mathsf{disj}) \arrow[dl, "\pi_0"'] \arrow[d] \arrow[dr, "\pi_1"] \arrow[drr] & & & \\
\mathsf{Col} \arrow[drr] & \mathsf{Col} \times \mathsf{Col} \arrow[l] \arrow[r] \arrow[drr] & \mathsf{Col} \arrow[drr] & \mathit{ext}(\mathsf{disj}) \arrow[dl, "\pi_0"'] \arrow[d] \arrow[dr, "\pi_1"] & \\
 & & \mathsf{Obj} & \mathsf{Obj} \times \mathsf{Obj} \arrow[l] \arrow[r] & \mathsf{Obj}
\end{tikzcd}$$

Figure 3: Disjointness Relation

and O_1 are both collections. The span of the disjointness relation is the pullback of the span of the Ur-disjointness relation along inclusions (but this cannot be explicitly expressed).

```
(23) (UR$relation disjoint)
     (= (UR$object0 disjoint) collection)
     (= (UR$object1 disjoint) collection)

(24) (forall (?c0 (collection ?c0) ?c1 (collection ?c1) (disjoint ?c0 ?c1))
         (UR$disjoint ?c0 ?c1))

(25) (forall (?o0 (UR$object ?o0) ?o1 (UR$object ?o1) (UR$disjoint ?o0 ?o1))
         (<=> (disjoint ?o0 ?o1)
              (and (collection ?o0) (collection ?o1))))

(26) (forall (?c0 (collection ?c0) ?c1 (collection ?c1))
         (and (<=> (disjoint ?c0 ?c1)
                   (not (exists (?x (KIF$thing ?x)) (and (?c0 ?x) (?c1 ?x)))))
              (=> (disjoint ?c0 ?c1) (UR$disjoint ?c0 ?c1))))
```

$$\begin{tikzcd}
 & \mathit{ext}(\mathsf{iso}) \arrow[dl, "\pi_0"'] \arrow[d] \arrow[dr, "\pi_1"] \arrow[drr] & & & \\
\mathsf{Col} \arrow[drr] & \mathsf{Col} \times \mathsf{Col} \arrow[l] \arrow[r] \arrow[drr] & \mathsf{Col} \arrow[drr] & \mathit{ext}(\mathsf{iso}) \arrow[dl, "\pi_0"'] \arrow[d] \arrow[dr, "\pi_1"] & \\
 & & \mathsf{Obj} & \mathsf{Obj} \times \mathsf{Obj} \arrow[l] \arrow[r] & \mathsf{Obj}
\end{tikzcd}$$

Figure 4: Isomorphic Relation

Two collections C_0 and C_1 are *isomorphic* (Figure 4) when there is a bijection between them. In that situation, the Ur-object C_0 is Ur-isomorphic to the Ur-object C_1. For any two Ur-objects O_0 and O_1 where O_0 is Ur-isomorphic to O_1, O_0 is KIF-isomorphic to O_1 iff O_0 and O_1 are both collections. The span of the isomorphism relation is the pullback of the span of the Ur-isomorphism relation along inclusions (but this cannot be explicitly expressed).

```
(27) (UR$relation isomorphic)
     (= (UR$object0 isomorphic) collection)
     (= (UR$object1 isomorphic) collection)

(28) (forall (?c0 (collection ?c0) ?c1 (collection ?c1) (isomorphic ?c0 ?c1))
         (UR$isomorphic ?c0 ?c1))

(29) (forall (?o0 (UR$object ?o0) ?o1 (UR$object ?o1) (UR$isomorphic ?o0 ?o1))
         (<=> (isomorphic ?o0 ?o1)
              (and (collection ?o0) (collection ?o1))))

(30) (forall (?c0 (collection ?c0) ?c1 (collection ?c1))
         (and (<=> (isomorphic ?c0 ?c1)
                   (exists (?f (KIF.FTN$bijection ?f))
                       (and (= (source ?f) ?c0) (= (target ?f) ?c1))))
              (=> (isomorphic ?c0 ?c1) (UR$isomorphic ?c0 ?c1))))
```

Here are some basic collections: *zero* = {} = ∅, *one* = {0}, *two* = {0, 1}, *three* = {0, 1, 2}. Zero and one have several synonyms. Two and three are often used for indexing. No thing is an instance of zero. Three canonical objects have been used in specifying these base collections: 0, 1 and 2. Clearly, we have the following inclusions (subcollection relationships): *zero* ⊆ *C* for any collection *C*, and *one* ⊆ *two* ⊆ *three*.

```
(31) (collection zero) (collection nothing) (collection null)
     (collection empty) (collection initial)
     (= zero nothing) (= nothing null) (= null empty) (= empty initial)
     (forall (?x (thing ?x)) (not (zero ?x)))

(32) (collection one) (collection unit) (collection terminal)
     (= one unit) (= unit terminal)
     (one 0)
     (forall (?x (one ?x)) (= ?x 0))

(33) (collection two)
     (two 0) (two 1)
     (forall (?x (two ?x)) (or (= ?x 0) (= ?x 1)))

(34) (collection three)
     (three 0) (three 1) (three 2)
     (forall (?x (three ?x)) (or (= ?x 0) (= ?x 1) (= ?x 2)))

(35) (forall (?c (collection ?c)) (subcollection zero ?c))
     (subcollection one two) (subcollection two three)
     (not (= one two)) (not (= one three)) (not (= two three))
```

The Ur object *pair* is defined to be the (implicit) Cartesian product of everything with itself (second Cartesian power of everything). The Ur object *triple* is defined to be the (implicit) third Cartesian power of everything. Pairs and triples are tuples. More explicitly, a pair is a tuple with arity two, and a triple is a tuple with arity three.

```
(36) (UR$object pair)
     (UR$subobject pair KIF.FTN$tuple)

(37) (forall (?x (KIF.FTN$tuple ?x))
         (<=> (pair ?x) (= (KIF.FTN$arity ?x) two)))

(38) (UR$object triple)
     (UR$subobject triple KIF.FTN$tuple)

(39) (forall (?x (KIF.FTN$tuple ?x))
         (<=> (triple ?x)  (= (KIF.FTN$arity ?x) three)))
```

A *tuple* of *collections* is a tuple (of collections).

```
(40) (UR$object collection-tuple}
     (UR$subobject collection-tuple KIF.FTN$tuple)

(41) (forall (?x (KIF.FTN$tuple ?x))
         (<=> (collection-tuple ?x) (= (KIF.FTN$type ?x) collection)))
```

For any pair of collections there is a *binary union* collection and a *binary intersection* collection.

```
(42) (UR$morphism binary-union)
     (= (UR$source binary-union) collection-pair)
     (= (UR$target binary-union) collection)

(43) (forall (?c0 (collection ?c0) ?c1 (collection ?c1)
              ?x (KIF$thing ?x))
         (<=> ((binary-union [?c0 ?c1]) ?x) (or (?c0 ?x) (?c1 ?x))))

(44) (UR$morphism binary-intersection)
     (= (UR$source binary-intersection) collection-pair)
     (= (UR$target binary-intersection) collection)

(45) (forall (?c0 (collection ?c0) ?c1 (collection ?c1)
              ?x (KIF$thing ?x))
         (<=> ((binary-intersection [?c0 ?c1]) ?x) (and (?c0 ?x) (?c1 ?x))))
```

Relations

KIF.REL

An IFF KIF relation $\boldsymbol{R} = \langle col_0(\boldsymbol{R}), col_1(\boldsymbol{R}), ext(\boldsymbol{R})\rangle$ is a Ur-object that consists of three collections: a domain or zeroth component collection $col_0(\boldsymbol{R})$, a codomain or first component collection $col_1(\boldsymbol{R})$, and an extent collection $ext(\boldsymbol{R}) \subseteq col_0(\boldsymbol{R}){\times}col_1(\boldsymbol{R})$. The extent is the collection of all pairs from the zeroth and first component collections that satisfy the relationship. A relation is determined by the triple of its zeroth, first and extent collections. Any relation is an UR-relation. A relation is an Ur-relation whose domain and codomain Ur-objects are collections. That is, an Ur-relation is a (KIF) relation iff its domain and codomain are collections. As a result, the domain (codomain) function for collections is the restriction of the Ur-domain (Ur-codomain) function for Ur-morphisms (but not explicitly expressible as such).

```
(1) (UR$object relation)

(2) (forall (?r (relation ?r))
        (UR$relation ?r))

(3) (forall (?r (UR$relation ?r))
        (<=> (relation ?r)
             (and (KIF.COL$collection (UR$object0 ?m))
                  (KIF.COL$collection (UR$object1 ?m)))))

(4) (UR$morphism collection0)
    (= (UR$source collection0) relation)
    (= (UR$target collection0) KIF.COL$collection)

(5) (UR$morphism collection1)
    (= (UR$source collection1) relation)
    (= (UR$target collection1) KIF.COL$collection)

(6) (UR$morphism extent)
    (= (UR$source extent) relation)
    (= (UR$target extent) KIF.COL$collection)

(7) (forall (?r (relation ?r))
        (and (= (UR$object0 ?r) (collection0 ?r))
             (= (UR$object1 ?r) (collection1 ?r))
             (= (UR$extent ?r) (extent ?r))))

(8) (forall (?r (relation ?r))
        (and (subcollection
                 (extent ?r)
                 (KIF.LIM.PRD2$binary-product [(collection0 ?r) (collection1 ?r)]))
             (forall (?x1 ((collection0 ?r) ?x1) ?x2 ((collection1 ?r) ?x2))
                 (<=> ((extent ?r) [?x1 ?x2]) (?r ?x1 ?x2)))))

(9) (forall (?r (relation ?r) ?s (relation ?s))
        (=> (and (= (collection0 ?r) (collection0 ?s))
                 (= (collection1 ?r) (collection1 ?s))
                 (= (extent ?r) (extent ?s)))
            (= r s)))
```

Each relation has a unique *projection*$_0$ function, whose source is the extent object and whose target is the domain *object*$_0$. Each relation has a unique *projection*$_1$, whose source is the extent object and whose target is the codomain *object*$_1$. The relation projection functions, which are restrictions of the analogous Ur functions, are the composition of the extent inclusion with the projection functions for the binary product of the component collections.

```
(10) (UR$morphism projection0)
     (= (UR$source projection0) relation)
     (= (UR$target projection0) KIF.FTN$function)
     (= (UR$composition [projection0 KIF.FTN$source]) extent)
     (= (UR$composition [projection0 KIF.FTN$target]) collection0)

(11) (UR$morphism projection1)
     (= (UR$source projection1) relation)
     (= (UR$target projection1) KIF.FTN$function)
     (= (UR$composition [projection1 KIF.FTN$source]) extent)
     (= (UR$composition [projection1 KIF.FTN$target]) collection1)

(12) (forall (?r (relation ?r))
         (and (= (UR$projection0 ?r) (projection0 ?r))
              (= (UR$projection1 ?r) (projection1 ?r))
              (= (projection0 ?r)
                 (KIF.FTN$composition
                     [(KIF.COL$inclusion
                          [(extent ?r)
                           (KIF.LIM.PRD2$binary-product [(collection0 ?r) (collection1 ?r)])])
                      (KIF.LIM.PRD2$binary-product-projection0 [(collection0 ?r) (collection1 ?r)])]))
              (= (projection1 ?r)
                 (KIF.FTN$composition
                     [(KIF.COL$inclusion
                           [(extent ?r)
                            (KIF.LIM.PRD2$binary-product [(collection0 ?r) (collection1 ?r)])])
                      (KIF.LIM.PRD2$binary-product-projection1 [(collection0 ?r) (collection1 ?r)])]))))
```

One relation *r* is a *subrelation* of another relation *s* when the first and second component of *r* and *s* are the same, and the extent collection of *r* is a subcollection of the extent collection of *s*.

```
(13) (UR$relation subrelation)
     (= (UR$object0 subrelation) relation)
     (= (UR$object1 subrelation) relation)

(14) (forall (?r (relation ?r) ?s (relation ?s))
         (<=> (subrelation ?r ?s)
              (and (= (collection0 ?r) (collection0 ?s))
                   (= (collection1 ?r) (collection1 ?s))
                   (subcollection (extent ?r) (extent ?s)))))
```

The notion of abridgment (Merriam-Webster) is that of "a shortened form of a work retaining the general sense and unity of the original". The abridgment relation is much more useful than the subrelation relation. For any two relations $R_0 \subseteq C_0 \times D_0$ and $R_1 \subseteq C_1 \times D_1$, R_0 is an *abridgment* of R_1, denoted by $R_0 \leq R_1$, when the domain (codomain) of R_0 is a subcollection of the domain (codomain) of R_1 and the extent of R_0 is the "restriction" of the extent of R_1 to the component collections of R_0. In that situation, R_0 is an Ur-abridgment of R_1. For any two Ur-relations R_0 and R_1 where R_0 is an Ur-abridgment of R_1, R_0 is a (KIF) -abridgment of R_1 iff R_0 and R_1 are both (KIF) relations. The latter states that the span of R_0 is the pullback of the span of R_1 along component object inclusions (but this cannot be explicitly expressed). If relation R_0 is an abridgment of relation R_1, then the extent of R_0 is a subcollection of the extent of R_1.

```
(15) (UR$relation abridgment)
     (= (UR$object0 abridgment) relation)
     (= (UR$object1 abridgment) relation)

(16) (forall (?r0 (relation ?r0) ?r1 (relation ?r1) (abridgment ?r0 ?r1))
         (UR$abridgment ?r0 ?r1))

(17) (forall (?r0 (UR$relation ?r0) ?r1 (UR$relation ?r1) (UR$abridgment ?r0 ?r1))
         (<=> (abridgment ?r0 ?r1)
              (and (relation ?r0) (relation ?r1))))

(18) (forall (?r0 (relation ?r0) ?r1 (relation ?r1))
         (<=> (abridgment ?r0 ?r1)
              (and (KIF.COL$subcollection (collection0 ?r0) (collection0 ?r1))
                   (KIF.COL$subcollection (collection1 ?r0) (collection1 ?r1))
                   (= (extent ?r0)
                      (KIF.COL2binary-intersection
                          [(extent ?r1)
                           (KIF.LIM.PRD2$binary-product [(collection0 ?r0) (collection1 ?r0)])])))))

(19) (forall (?r (relation ?r) ?s (relation ?s))
         (=> (abridgment ?r ?s)
             (KIF.COL$subcollection (extent ?r) (extent ?s))))
```

The *restriction* relation can be used in a *parametric* fashion with pairs of Ur-morphisms $m_0 : \mathsf{A} \rightarrow \mathsf{rel}$ and $m_1 : \mathsf{A} \rightarrow \mathsf{rel}$, which have a common source Ur-object A and each have the target Ur-object rel of all relations. The restriction-parametric relation holds for m_0 and m_1, when $m_0(x) \leq m_1(x)$ for each element $x \in A$. The parametric restriction relation is reflexive and transitive.

```
(20) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1))
         (<=> (abridgment-parameterized ?m0 ?m1)
              (and (= (UR$source ?m0) (UR$source ?m1))
                   (= (UR$target ?m0) relation)
                   (= (UR$target ?m1) relation)
                   (forall (?x ((UR$source ?m0) ?x)
                       (abridgment (?m0 ?x) (?m1 ?x))))))

(21) (forall (?m (UR$morphism ?m))
         (abridgment-parameterized ?m ?m))

(22) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1) ?m2 (UR$morphism ?m2))
         (=> (and (abridgment-parameterized ?m0 ?m1) (abridgment-parameterized ?m1 ?m2))
             (abridgment-parameterized ?m0 ?m2)))
```

A relation is *total* when it satisfies the condition: every object in the first component collection is the first component of some pair in the extent of the relation. A relation is *functional* when it satisfies the condition: if the relation holds for pairs p_1 and p_2 with the same first elements, then the second elements must be identical as well (that is, p_1 and p_2 must be the same pairs). A relation is *total functional* when it is both total and functional.

```
(23) (UR$object total)
     (UR$subobject total relation)

(24) (forall (?r (relation ?r))
         (<=> (total ?r)
              (forall (?x1 ((collection0 ?r) ?x1))
                  (exists ?x2 ((collection1 ?r) ?x2))
                      (?r ?x1 ?x2)))))

(25) (UR$object functional)
     (UR$subobject functional relation)

(26) (forall (?r (relation ?r))
         (<=> (functional ?r)
              (forall (?x ((collection0 ?r) ?x)
                       ?y1  ((collection1 ?r) ?y1)
                       ?y2  ((collection1 ?r) ?y2))
                  (=> (and (?r ?x ?y1) (?r ?x ?y2))
                      (= ?y1 ?y2)))))

(27) (UR$object total-functional)
     (UR$subobject total-functional relation)

(28) (forall (?r (relation ?r))
         (<=> (total-functional ?r)
              (and (total ?r) (functional ?r))))
```

Partial Functions

KIF.PFN

A KIF *partial function* represents the notion of a map, a so-called “black-box,” or an input-output device. A partial function has three component collections: *source*, *target* and *domain* (of definition). Each of these concepts is represented by a Ur morphism whose source is partial function and whose target is collection. We use the notation $f : \mathrm{X} \rightarrow \mathrm{Y}$ to indicate the source-target typing between collections of a KIF partial function. Most IFF KIF functions used in the IFF Foundation Ontology are total. A partial function should be functional (single valued); that is, it should satisfy the condition: if the function maps an element x in the source to two elements y_1 and y_2 in the target, then $y_1 = y_2$. In addition, a partial function should only be defined on its domain (of definition) and it should be total there.

```
 (1) (UR$object partial-function)

 (2) (UR$morphism source)
     (= (UR$source source) partial-function)
     (= (UR$target source) KIF.COL$collection)

 (3) (UR$morphism target)
     (= (UR$source target) partial-function)
     (= (UR$target target) KIF.COL$collection)

 (4) (UR$morphism domain)
     (= (UR$source domain) partial-function)
     (= (UR$target domain) KIF.COL$collection)

 (5) (forall (?f (partial-function ?f))
         (and (KIF.COL$subcollection (domain ?f) (source ?f))
              (forall (?x ((source ?f) ?x))
                  (and (<=> ((domain ?f) ?x)
                            (exists (?y ((target ?f) ?y)) (= (?f ?x) ?y)))
              (forall (?x ((source ?f) ?x)
                       ?y1 ((target ?f) ?y1) ?y2 ((target ?f) ?y2))
                  (=> (and (= (?f ?x) ?y1) (= (?f ?x) ?y2)) (= ?y1 ?y2)))))
```

Any partial function can be mapped to a functional relation. This *pfn2rel* Ur morphism is monomorphic (an injection). It can be restricted (at the target) to a bijection from partial-function to functional. As a result, partial functions and functional relations are isomorphic.

```
 (6) (UR$monomorphism pfn2rel)
     (= (UR$source pfn2rel) partial-function)
     (= (UR$target pfn2rel) KIF.REL$relation)
     (= (UR$composition [pfn2rel KIF.REL$collection0]) source)
     (= (UR$composition [pfn2rel KIF.REL$collection1]) target)

 (7) (forall (?f (partial-function ?f)
              ?x ((source ?f) ?x) ?y ((target ?f) ?y))
                  (<=> ((pfn2rel ?f) ?x ?y) (= (?f ?x) ?y)))

 (8) (UR$isomorphic partial-function KIF.REL$functional)
```

Since the Ur object of (total) functions is a subobject of the Ur-object of partial functions, there is an inclusion Ur morphism. When restricted to its domain, a partial function becomes a total function, defining a *pfn2ftn* Ur morphism. This is epimorphic (a surjection). It returns the total function “within” a partial function, which is a partial restriction of the partial function. The total function within a total function is itself.

```
 (9) (UR$epimorphism pfn2ftn)
     (= (UR$source pfn2ftn) partial-function)
     (= (UR$target pfn2ftn) KIF.FTN$function)
     (= (UR$composition [pfn2ftn KIF.FTN$source]) domain)
     (= (UR$composition [pfn2ftn KIF.FTN$target]) target)

(10) (forall (?f (partial-function ?f))
         (and (forall (?x ((domain ?f) ?x))
                  (= ((pfn2ftn ?f) ?x) (?f ?x)))
              (restriction (pfn2ftn ?f) ?f)))
```

A partial function $f_1 : C_1 \rightarrow D_1$ is a *restriction* of a partial function $f_2 : C_2 \rightarrow D_2$ when the source (target) of f_1 is a subcollection of the source (target) of f_2 and the functions agree (on domain elements of f_1); that is, the (total) function associated with f_1 is a restriction of the (total) function associated with f_2.

```
(11) (UR$relation restriction)
     (= (UR$object0 restriction) partial-function)
     (= (UR$object1 restriction) partial-function)

(12) (forall (?f1 (partial-function ?f1) ?f2 (partial-function ?f2))
         (<=> (restriction ?f1 ?f2)
              (and (KIF.COL$subcollection (source ?f1) (source ?f2))
                   (KIF.COL$subcollection (target ?f1) (target ?f2))
                   (KIF.FTN$restriction (pfn2ftn ?f1) (pfn2ftn ?f2)))))
```

A pair of partial-functions is a tuple whose arity is two and whose type is partial-function.

```
(13) (UR$object partial-function-pair)
     (UR$subobject partial-function-pair KIF.FTN$tuple)

(14) (forall (?fg (KIF.FTN$tuple ?fg))
         (<=> (partial-function-pair ?fg)
              (and (= (KIF.FTN$arity ?fg) KIF.COL$two)
                   (= (KIF.FTN$type ?fg) partial-function))))
```

Two partial functions are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable partial functions $f : A \rightarrow B$ with domain X and $g : B \rightarrow C$ with domain Y is the partial function $f \cdot g : A \rightarrow C$ with domain Z. The domain is defined by $x \in Z$ iff $x \in X$ and $f(x) \in Y$. The composed function is defined by $f \cdot g\,(x) = g(f(x))$ for any element $x \in Z$.

```
(15) (UR$object partial-function-function)
     (UR$subobject partial-function-function partial-function-pair)

(16) (forall (?f (partial-function ?f) ?g (partial-function ?g))
         (<=> (partial-function-function [?f ?g])
              (= (target ?f) (source ?g))))

(17) (UR$morphism partial-function0)
     (= (UR$source partial-function0) partial-function-function)
     (= (UR$target partial-function0) partial-function)

(18) (forall (?f (partial-function ?f) ?g (partial-function ?g)
             (partial-function-function [?f ?g]))
         (= (partial-function0 [?f ?g]) ?f))

(19) (UR$morphism partial-function1)
     (= (UR$source partial-function1) partial-function-function)
     (= (UR$target function-function1) partial-function)

(20) (forall (?f (partial-function ?f) ?g (partial-function ?g)
             (partial-function-function [?f ?g]))
         (= (partial-function1 [?f ?g]) ?g))

(21) (UR$morphism composition)
     (= (UR$source composition) partial-function-function)
     (= (UR$target composition) partial-function)
     (= (UR$composition [composition source])
        (UR$composition [partial-function0 source]))
     (= (UR$composition [composition target])
        (UR$composition [partial-function1 target]))

(22) (forall (?f1 (partial-function ?f1) ?f2 (partial-function ?f2)
             (partial-function-function [?f ?g]))
         (forall (?x ((source ?f1) ?x))
             (and (<=> ((domain (composition [?f1 ?f2])) ?x)
                       (and ((domain ?f1) ?x) ((domain ?f2) (?f1 ?x))))
                  (=> ((domain (composition [?f1 ?f2])) ?x)
                      (= ((composition [?f1 ?f2]) ?x) (?f2 (?f1 ?x)))))))
```

Composition satisfies the usual *associative law*.

```
(23) (forall (?f1 (partial-function ?f1) ?f2 (partial-function ?f2) ?f3 (partial-function ?f3)
              (partial-function-function [?f1 ?f2]) (partial-function-function [?f2 ?f3]))
         (= (composition [?f1 (composition [?f2 ?f3])])
            (composition [(composition [?f1 ?f2]) ?f3])))
```

For any collection *C*, there is an *identity* function.

```
(24) (UR$morphism identity)
     (= (UR$source identity) KIF.COL$collection)
     (= (UR$target identity) partial-function)
     (= (UR$composition [identity source]) (UR$identity KIF.COL$collection))
     (= (UR$composition [identity domain]) (UR$identity KIF.COL$collection))
     (= (UR$composition [identity target]) (UR$identity KIF.COL$collection))

(25) (forall (?c (KIF.COL$collection ?c))
         (forall (?x (?c ?x))
             (= ((identity ?c) ?x) ?x)))
```

The identity satisfies the usual *identity laws* with respect to composition.

```
(26) (forall (?f (partial-function ?f))
         (and (= (composition [(identity (source ?f)) ?f]) ?f)
              (= (composition [?f (identity (target ?f))]) ?f)))
```

Functions

KIF.FTN

A (total) KIF *function* is a partial function whose domain equals its source. For (total) functions, we can clearly ignore the domain. The function notation $f : C \rightarrow D$ shows a function f with source collection C and target collection D. This is an equalizer notion. Functions are also called tuples. For any tuple (function) $t : J \rightarrow A$ the tuple notation $t = \{j \in J \mid t_j \in A\}$ is useful. Any function is a UR-morphism. A function is an Ur-morphism whose source and target Ur-objects are KIF-collections. That is, an Ur-morphism is a function iff its Ur-source and Ur-target are collections.

```
(1) (UR$object function)
    (UR$object tuple)
    (= tuple function)
    (UR$subobject function KIF.PFN$partial-function)

(2) (forall (?f (KIF.PFN$partial-function ?f))
        (<=> (function ?f)
             (= (KIF.PFN$domain ?f) (KIF.PFN$source ?f))))

(3) (forall (?f (function ?f))
        (UR$morphism ?r))

(4) (forall (?m (UR$morphism ?m))
        (<=> (function ?m)
             (and (KIF.COL$collection (UR$source ?m))
                  (KIF.COL$collection (UR$target ?m)))))
```

Because total functions are more useful than partial functions, we provide special terms for their source and target. They are the restrictions of the partial source and target to the total functions. As a result, the source (target) function for KIF-functions is the restriction of the Ur-source (Ur-target) function for Ur-morphisms (but not explicitly expressible as such). The source (target) of KIF-functions is also called the arity (type) of tuples.

```
(5) (UR$morphism source)
    (UR$morphism arity)
    (= arity source)
    (= (UR$source source) function)
    (= (UR$target source) KIF.COL$collection)

(6) (UR$morphism target)
    (UR$morphism type)
    (= type target)
    (= (UR$source target) function)
    (= (UR$target target) KIF.COL$collection)

(7) (forall (?f (function ?f))
        (and (= (UR$source ?f) (source ?f))
             (= (UR$target ?f) (target ?f))))
```

For convenience, we name the map *ftn2pfn*, which is the inclusion of functions into partial functions.

```
(8) (UR$monomorphism ftn2pfn)
    (= (UR$source ftn2pfn) function)
    (= (UR$target ftn2pfn) KIF.PFN$partial-function)
    (= (UR$composition [ftn2pfn KIF.PFN$source]) source)
    (= (UR$composition [ftn2pfn KIF.PFN$domain]) source)
    (= (UR$composition [ftn2pfn KIF.PFN$target]) target)
    (= (UR$composition [ftn2pfn pfn2ftn]) (UR$identity function))
    (= ftn2pfn (UR$inclusion [function KIF.PFN$partial-function]))
```

Any (total) KIF function can be mapped by the *ftn2rel* UR morphism to a total functional relation. The domain (codomain) of the relation of a function is the source (target). This is the composition of the total function to partial function inclusion and the partial function to relation monomorphism. Hence, it is a monomorphism. Clearly, the pfn2rel function agrees with the ftn2rel function on (total) functions. The ftn2rel function can be restricted (at the target) to a bijection from the Ur-object of functions to the Ur-object of total-functional relations. As a result, functions and total functional relations are isomorphic. The function-to-relation map is the restriction of the UR-morphism-to-UR-relation map (but this cannot be explicitly expressed). The extent of the relation of a function is equal (not just isomorphic) to the pullback of the function with the target identity. The relation projections are the pullback projections. The source identity and the function form a span. The function is the composition of the mediating bijection and the target pullback projection.

```
 (9) (UR$monomorphism ftn2rel)
     (= (UR$source ftn2rel) function)
     (= (UR$target ftn2rel) KIF.REL$relation)
     (= (UR$composition [ftn2rel KIF.REL$collection0]) source)
     (= (UR$composition [ftn2rel KIF.REL$collection1]) target)
     (= ftn2rel (UR$composition [ftn2pfn KIF.PFN$pfn2rel])

(10) (UR$isomorphic function KIF.REL$total-functional)

(11) (forall (?f (function ?f))
         (= (UR$mor2rel ?f) (ftn2rel ?f)))

(12) (forall (?f (function ?f))
         (and (= (KIF.REL$extent (ftn2rel ?f))
                 (KIF.LIM.PBK$pullback [?f (identity (target ?f))]))
              (= (KIF.REL$projection0 (ftn2rel ?f))
                 (KIF.LIM.PBK$pullback-projection0 [?f (identity (target ?f))]))
              (= (KIF.REL$projection1 (ftn2rel ?f))
                 (KIF.LIM.PBK$pullback-projection1 [?f (identity (target ?f))]))
              (isomorphic (source ?f) (KIF.LIM.PBK$pullback [?f (identity (target ?f))]))
              (exists (?r (KIF.LIM.PBK$pullback-cone ?r))
                  (and (= (KIF.LIM.PBK$opspan ?r) [?f (identity (target ?f))])
                       (= (KIF.LIM.PBK$vertex ?r) (source ?f))
                       (= (KIF.LIM.PBK$function0 ?r) (identity (source ?f)))
                       (= (KIF.LIM.PBK$function1 ?r) ?f)
                       (bijection (KIF.LIM.PBK$pairing ?r))
                       (= ?f (composition
                                 [(KIF.LIM.PBK$pairing ?r)
                                  (KIF.LIM.PBK$projection1 [?f (identity (target ?f))])]))))))
```

For any two total functions $f_0 : C_0 \rightarrow D_0$ and $f_1 : C_1 \rightarrow D_1$ (Figure 5), f_0 is a *restriction* of f_1, denoted by $f_0 \leq f_1$, when the source (target) of f_0 is a subcollection of the source (target) of f_1 and the functions agree (on source elements of f_0); that is, the functions commute with the source/target inclusions. In that situation, f_0 is an Ur-restriction of f_1. For any two Ur-morphisms m_0 and m_1 where m_0 is an Ur-restriction of m_1, m_0 is a (KIF) restriction of m_1 iff both m_0 and m_1 are (KIF) functions. The span of the restriction relation is the pullback of the span of the Ur-restriction relation along inclusions (but this cannot be explicitly expressed). Restriction can be viewed as a constraint on the larger function – it says that the larger function maps the

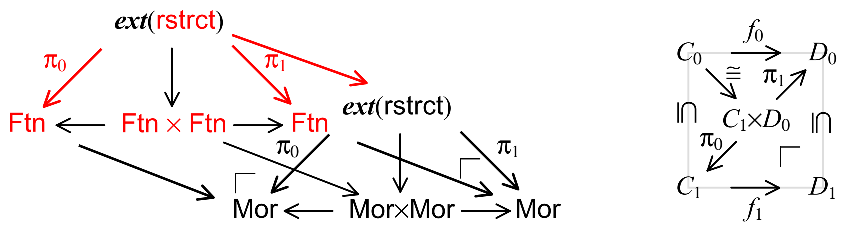


Figure 5: Restriction Relation

source collection of the smaller function into the target collection of the smaller function. One function is a restriction of another function iff the relation associated with the first function is an abridgment of the relation associated with the second functions. One function is a restriction of another function iff the source of the first function is isomorphic to the pullback of the second function with the target inclusion and the first function is the composition of a mediating bijection and the target pullback projection.

```
(13) (UR$relation restriction)
     (= (UR$object1 restriction) function)
     (= (UR$object2 restriction) function)

(14) (forall (?f0 (function ?f0) ?f1 (function ?f1) (restriction ?f0 ?f1))
         (UR$restriction ?f0 ?f1))

(15) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1) (UR$restriction ?m0 ?m1))
         (<=> (restriction ?m0 ?m1)
              (and (function ?m0) (function ?m1))))

(16) (forall (?f1 (function ?f1) ?f2 (function ?f2))
         (<=> ((UR$extent restriction) [?f1 ?f2])
              ((UR$extent KIF.REL$abridgment) [(ftn2rel ?f1) (ftn2rel ?f2)])))

(17) (forall (?f1 (function ?f1) ?f2 (function ?f2))
         (<=> (restriction ?f1 ?f2)
              (and (KIF.COL$subcollection (source ?f1) (source ?f2))
                   (KIF.COL$subcollection (target ?f1) (target ?f2))
                   (= (composition [(KIF.COL$inclusion [(source ?f1) (source ?f2)]) ?f2])
                      (composition [?f1 (KIF.COL$inclusion [(source ?f1) (source ?f2)])])))))

(18) (forall (?f1 (function ?f1) ?f2 (function ?f2) (restriction ?f1 ?f2))
         (and (isomorphic
                  (source ?f0)
                  (KIF.LIM.PBK$pullback [?f1 (KIF.COL$inclusion [(target ?f0) (target ?f1)])]))
              (exists (?r (KIF.LIM.PBK$pullback-cone ?r))
                  (and (= (KIF.LIM.PBK$opspan ?r)
                          [?f1 (KIF.COL$inclusion [(target ?f0) (target ?f1)])])
                       (= (KIF.LIM.PBK$vertex ?r) (source ?f0))
                       (= (KIF.LIM.PBK$function0 ?r) ?f0)
                       (= (KIF.LIM.PBK$function1 ?r)
                          (KIF.COL$inclusion [(source ?f0) (source ?f1)]))
                       (= ?f0 (composition
                                 [(KIF.LIM.PBK$pairing ?r)
                                  (KIF.LIM.PBK$projection1
                                      [?f1 (KIF.COL$inclusion [(target ?f0) (target ?f1)])])]))))))
```

The *restriction* relation can be used in a *parametric* fashion with pairs of Ur-morphisms $m_0 : \mathsf{A} \rightarrow \mathsf{ftn}$ and $m_1 : \mathsf{A} \rightarrow \mathsf{ftn}$, which have a common source Ur-object A and each have the target Ur-object ftn of all functions. The restriction-parametric relation holds for m_0 and m_1, when $m_0(x) \leq m_1(x)$ for each element $x \in \mathsf{A}$. The parametric restriction relation is reflexive and transitive.

```
(19) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1))
         (<=> (restriction-parameterized ?m0 ?m1)
              (and (= (UR$source ?m0) (UR$source ?m1))
                   (= (UR$target ?m0) function)
                   (= (UR$target ?m1) function)
                   (forall (?x ((UR$source ?m0) ?x)
                       (restriction (?m0 ?x) (?m1 ?x)))))))

(20) (forall (?m (UR$morphism ?m))
         (restriction-parameterized ?m ?m))

(21) (forall (?m0 (UR$morphism ?m0) ?m1 (UR$morphism ?m1) ?m2 (UR$morphism ?m2))
         (=> (and (restriction-parameterized ?m0 ?m1) (restriction-parameterized ?m1 ?m2))
             (restriction-parameterized ?m0 ?m2)))
```

Two functions are *composable* (Figure 6) when the target of the first is equal to the source of the second. Any composable pair of functions is a composable pair of UR-morphisms; and the component functions, composition and identity functions are restrictions of the analogous Ur functions. A composable pairs of Ur-morphisms is a composable pair of (KIF) functions iff the two component UR-morphisms are (KIF)

$$\begin{tikzcd}
 & & \mathsf{Ftn} \arrow[rr] & & \mathsf{Mor} \\
 & \mathsf{Ftn\text{-}Ftn} \arrow[ur, "\pi_1"] \arrow[rr] \arrow[dl, "\pi_0"'] \arrow[dd, "\circ"] & & \mathsf{Mor\text{-}Mor} \arrow[ur, "\pi_1"'] \arrow[dl, "\pi_0"] \arrow[dd, "\circ"] & \\
 \mathsf{Ftn} \arrow[rr] & & \mathsf{Mor} & & \\
 & \mathsf{Ftn} \arrow[rr] & & \mathsf{Mor} &
\end{tikzcd}$$

Figure 6: Composition Function

$$\begin{tikzcd}
\mathsf{Ftn} \arrow[r] & \mathsf{Mor} \\
\mathsf{Col} \arrow[u, "\iota"] \arrow[r] & \mathsf{Obj} \arrow[u, "\iota"']
\end{tikzcd}$$

Figure 7: Identity Function

functions. The *composition* of two composable functions $f : A \rightarrow B$ and $g : B \rightarrow C$ (Figure 6) is the function $f \cdot g : A \rightarrow C$ defined by $f \cdot g\,(x) = g(f(x))$ for any element $x \in A$. The composition of two composable functions $f : A \rightarrow B$ and $g : B \rightarrow C$ is equal to their UR-composition. For any collection C, there is an *identity* function (Figure 7). The identity of any collection is the UR-identity of that UR-object.

```
(22) (UR$object function-function)

(23) (forall (?fg (function-function ?fg))
         (UR$morphism-morphism ?fg))

(24) (forall (?fg (UR$morphism-morphism ?fg))
         (<=> (function-function ?fg)
              (and (function (UR$morphism0 ?fg))
                   (function (UR$morphism1 ?fg)))))

(25) (UR$morphism function0)
     (= (UR$source function0) function-function)
     (= (UR$target function0) function)

(26) (UR$morphism function1)
     (= (UR$source function1) function-function)
     (= (UR$target function1) function)

(27) (forall (?fg (function-function ?fg))
         (and (= (UR$morphism0 ?fg) (function0 ?fg))
              (= (UR$morphism1 ?fg) (function1 ?fg))))

(28) (forall (?f (function ?f) ?g (function ?g))
         (<=> (function-function [?f ?g])
              (= (target ?f) (source ?g))))

(29) (UR$morphism composition)
     (= (UR$source composition) function-function)
     (= (UR$target composition) function)
     (= (UR$composition [composition source])
        (UR$composition [function0 source]))
     (= (UR$composition [composition target])
        (UR$composition [function1 target]))

(30) (forall (?fg (function-function ?fg))
         (= (UR$composition ?fg) (composition ?fg)))

(31) (forall (?f (function ?f) ?g (function ?g) (function-function [?f ?g]))
         (and (= (function0 [?f ?g]) ?f))
              (= (function1 [?f ?g]) ?g))
              (forall (?x ((source ?f1) ?x))
                  (= ((composition [?f1 ?f2]) ?x) (?f2 (?f1 ?x)))))

(32) (UR$morphism identity)
     (= (UR$source identity) KIF.COL$collection)
     (= (UR$target identity) function)
     (= (UR$composition [identity source]) (UR$identity KIF.COL$collection))
     (= (UR$composition [identity target]) (UR$identity KIF.COL$collection))

(33) (forall (?c (KIF.COL$collection ?c))
         (= (identity ?c) (UR$identity ?c)))

(34) (forall (?c (KIF.COL$collection ?c))
         (forall (?x (?c ?x))
             (= ((identity ?c) ?x) ?x)))
```

Composition satisfies the usual *associative law*.

```
(35) (forall (?f1 (function ?f1) ?f2 (function ?f2) ?f3 (function ?f3)
            (function-function [?f1 ?f2]) (function-function [?f2 ?f3]))
         (= (composition [?f1 (composition [?f2 ?f3])])
            (composition [(composition [?f1 ?f2]) ?f3])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(36) (forall (?f (function ?f))
        (and (= (composition [(identity (source ?f)) ?f]) ?f)
             (= (composition [?f (identity (target ?f))]) ?f)))
```

A function is an *injection* when no distinct source elements have the same image. A function is a *surjection* when all elements of the target collection are images. A function is a *bijection* when it is both an injection and a surjection. Any injection (surjection, bijection) is an Ur-monomorphism (Ur-epimorphism, Ur-isomorphism). An Ur-monomorphism (Ur-epimorphism, Ur-isomorphism) is an injection (surjection, bijection) iff it is a KIF functions. A function is an injection (surjection, bijection) iff it is a Ur-monomorphism (Ur-epimorphism, Ur-isomorphism).

```
(37) (UR$object injection)
     (UR$subobject injection function)

(38) (forall (?f (injection ?f))
         (UR$monomorphism ?r))

(39) (forall (?m (UR$monomorphism ?m))
         (<=> (injection ?m)
              (function ?m)))

(40) (forall (?m (function ?m))
         (<=> (injection ?m)
              (UR$monomorphism ?m)))

(41) (forall (?f (function ?f))
         (<=> (injection ?f)
              (forall (?x1 ((source ?f) ?x1)
                       ?x2 ((source ?f) ?x2))
                  (=> (= (?f ?x1) (?f ?x2)) (= ?x1 ?x2)))))

(42) (UR$object surjection)
     (UR$subobject surjection function)

(43) (forall (?f (surjection ?f))
         (UR$epimorphism ?r))

(44) (forall (?m (UR$epimorphism ?m))
         (<=> (surjection ?m)
              (function ?m)))

(45) (forall (?m (function ?m))
         (<=> (surjection ?m)
              (UR$epimorphism ?m)))

(46) (forall (?f (function ?f))
         (<=> (surjection ?f)
              (forall (?y ((target ?f) ?y))
                  (exists (?x ((source ?f) ?x)) (= (?f ?x) ?y)))))

(47) (UR$object bijection)
     (UR$subobject bijection function)

(48) (forall (?f (bijection ?f))
         (UR$isomorphism ?r))

(49) (forall (?m (UR$isomorphism ?m))
         (<=> (bijection ?m)
              (function ?m)))

(50) (forall (?m (function ?m))
         (<=> (bijection ?m)
              (UR$isomorphism ?m)))

(51) (forall (?f (function ?f))
         (<=> (bijection ?f) (and (surjection ?f) (injection ?f))))
```

Finite Diagrams

KIF.DGM

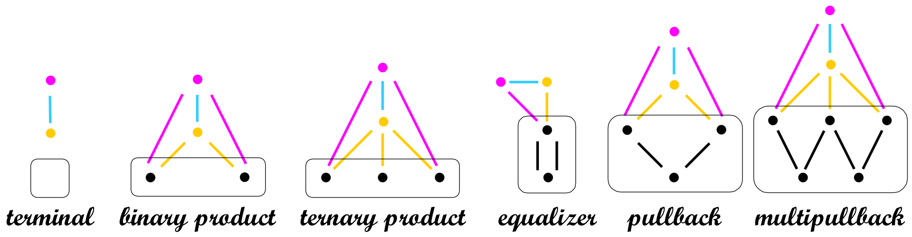


Figure 8: Finite Diagrams, Limits, Projections, Cones and Mediators

Abstractly, a finite diagram is a graph morphism from a finite (shape) graph into the quasi-graph of collections 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 for specifying finite limits. Diagrams are determined by their shape, collection and function components. These diagrams include (Figure 8) 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. In Figure 8, the diagrams are **black**, the limits and projection functions are **gold**, the cones over the diagrams are **magenta**, and the mediator functions are **teal**.

Collection Pairs

KIF.DGM.PR.OBJ

A *pair* (of *collections*) is the appropriate base diagram for a binary product. A collection pair (C_0, C_1) is both a pair and a tuple of collections. In particular, collection pairs are collection tuples. Intuitively, the UR object *collection-pair* = *collection*^{2} = *collection*×*collection* is the binary power of *collection*. Collection pairs are the diagrams used by binary products and coproducts. Pairs are determined by their two component collections.

```
(1)  (UR$object collection-pair}
     (UR$subobject collection-pair KIF.FTN$tuple)
     (UR$subobject collection-pair KIF.FTN$pair)

(2)  (forall (?c (KIF.FTN$tuple ?c))
         (<=> (collection-pair ?c)
              (and (= (KIF.FTN$arity ?c) KIF.COL$two)
                   (= (KIF.FTN$type ?c) KIF.COL$collection))))

(3)  (UR$morphism collection0)
     (= (UR$source collection0) collection-pair)
     (= (UR$target collection0) KIF.COL$collection)

(4)  (UR$morphism collection1)
     (= (UR$source collection1) collection-pair)
     (= (UR$target collection1) KIF.COL$collection)

(5)  (forall (?c (collection-pair ?c))
         (and (= (collection0 ?c) (?c 0))
              (= (collection1 ?c) (?c 1))))))
```

A *pair of collections* has an associated *opspan*. The opvertex is the terminal collection, and the component functions are the unique functions for the component collections in the pair.

```
(6)  (UR$morphism opspan)
     (= (UR$source opspan) collection-pair)
     (= (UR$target opspan) KIF.DGM.OSPN.OBJ$opspan)
     (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection0]) collection0)
     (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection1]) collection1)
     (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$function0])
        (UR$composition [collection0 KIF.LIM$unique]))
     (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$function1])
        (UR$composition [collection1 KIF.LIM$unique]))

 (7) (forall (?p (collection-pair ?p))
         (= (KIF.DGM.OSPN.OBJ$opvertex (opspan ?p)) KIF.LIM$terminal))
```

Pair Morphisms

KIF.DGM.PR.MOR

Collection pairs are connected by morphisms. A morphism of collection pairs consists of a pair of functions connecting the respective collection components. A *pair of functions* (f_0, f_1) is both a pair and a tuple of functions. In particular, function pairs are function tuples. One can prove that *function-pair* = *function*^{2} = *function*×*function* is the binary power of *function*. Any function pair has a source (target) pair of collections 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.

```
 (1) (UR$object function-pair}
     (UR$subobject function-pair KIF.FTN$tuple)

 (2) (forall (?f (KIF.FTN$tuple ?f))
         (<=> (function-pair ?f)
              (and (= (KIF.FTN$arity ?f) KIF.COL$two)
                   (= (KIF.FTN$type ?f) KIF.FTN$function))))

 (3) (UR$morphism function0)
     (= (UR$source function0) function-pair)
     (= (UR$target function0) KIF.FTN$function)

 (4) (UR$morphism function1)
     (= (UR$source function1) function-pair)
     (= (UR$target function1) KIF.FTN$function)

 (5) (UR$morphism source)
     (= (UR$source source) function-pair)
     (= (UR$target source) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$composition [source KIF.DGM.PR.OBJ$collection0)
        (UR$composition [function0 KIF.FTN$source]))
     (= (UR$composition [source KIF.DGM.PR.OBJ$collection1])
        (UR$composition [function1 KIF.FTN$source]))

 (6) (UR$morphism target)
     (= (UR$source target) function-pair)
     (= (UR$target target) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$composition [target KIF.DGM.PR.OBJ$collection0)
        (UR$composition [function0 KIF.FTN$target]))
     (= (UR$composition [target KIF.DGM.PR.OBJ$collection1])
        (UR$composition [function1 KIF.FTN$target]))

 (7) (forall (?f (function-pair ?f))
         (and (= (function0 ?f) (?f 0))
              (= (function1 ?f) (?f 1))))
```

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 collection pair $X = (X_0, X_1)$, there is an *identity* function pair.

```
 (8) (UR$object composable-pair-function-pair)

 (9) (forall (?z (composable-pair-function-pair ?z))
         (pair ?z))

(10) (forall (?f ?g (composable-pair-function-pair [?f ?g]))
         (and (function-pair ?f) (function-pair ?g)))

(11) (forall (?f (function-pair ?f) ?g (function-pair ?g))
         (<=> (composable-pair-function-pair [?f ?g])
              (= (target ?f) (source ?g))))
```

```
(12) (UR$morphism composition)
     (= (UR$source composition) composable-pair-function-pair)
     (= (UR$target composition) function-pair)

(13) (forall (?f (function-pair ?f) ?g (function-pair ?g) (= (target ?f) (source ?g)))
         (and (= (source (composition [?f ?g])) (source ?f))
              (= (target (composition [?f ?g])) (target ?g))
              (= (function0 (composition [?f ?g]))
                 (KIF.FTN$composition [(function0 ?f) (function0 ?g)]))
              (= (function1 (composition [?f ?g]))
                 (KIF.FTN$composition [(function1 ?f) (function1 ?g)]))))

(14) (UR$morphism identity)
     (= (UR$source identity) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target identity) function-pair)
     (= (UR$composition [identity source]) (UR$identity KIF.DGM.PR.OBJ$collection-pair))
     (= (UR$composition [identity target]) (UR$identity KIF.DGM.PR.OBJ$collection-pair))
     (= (UR$composition [identity function0])
        (UR$composition [KIF.DGM.PR.OBJ$collection0 KIF.FTN$identity]))
     (= (UR$composition [identity function1])
        (UR$composition [KIF.DGM.PR.OBJ$collection1 KIF.FTN$identity]))
```

Composition satisfies the usual *associative law*.

```
(15) (forall (?f (function-pair ?f) ?g (function-pair ?g) ?h (function-pair ?h)
                (= (target ?f) (source ?g)) (= (target ?g) (source ?h)))
         (= (composition [?f (composition [?g ?h])])
            (composition [(composition [?f ?g]) ?h])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(16) (forall (?f (function-pair ?f))
         (and (= (composition [(identity (source ?f)) ?f]) ?f)
              (= (composition [?f (identity (target ?f))]) ?f)))
```

Collection Triples

KIF.DGM.TRP.OBJ

A *triple* (of *collections*) is the appropriate base diagram for a ternary product. A collection triple (C_0, C_1, C_2) is both a triple and a tuple of collections. In particular, collection triples are collection tuples. Intuitively, the UR object *collection-triple* = *collection*^{3} = *collection*×*collection*×*collection* is the ternary power of *collection*. Collection triples are the diagrams used by ternary products and coproducts. Triples are determined by their three component collections.

```
 (1) (UR$object collection-triple}
     (UR$subobject collection-triple KIF.FTN$tuple)
     (UR$subobject collection-triple KIF.FTN$triple)

 (2) (forall (?c (KIF.FTN$tuple ?c))
         (<=> (collection-triple ?c)
              (and (= (KIF.FTN$arity ?c) KIF.COL$three)
                   (= (KIF.FTN$type ?c) KIF.COL$collection))))

 (3) (UR$morphism collection0)
     (= (UR$source collection0) collection-triple)
     (= (UR$target collection0) KIF.COL$collection)

 (4) (UR$morphism collection1)
     (= (UR$source collection1) collection-triple)
     (= (UR$target collection1) KIF.COL$collection)

 (5) (UR$morphism collection2)
     (= (UR$source collection2) collection-triple)
     (= (UR$target collection2) KIF.COL$collection)

 (6) (forall (?c (collection-triple ?c))
         (and (= (collection0 ?c) (?c 0))
              (= (collection1 ?c) (?c 1))
              (= (collection2 ?c) (?c 2))))
```

Triple Morphisms

KIF.DGM.TRP.MOR

Collection triples are connected by morphisms. A morphism of collection triples consists of a triple of functions connecting the respective collection components. A *function-triple* (f_0, f_1, f_2) is both a triple and a tuple of functions. In particular, function triples are function tuples. One can prove that *function-triple* = *function*^{3} = *function*×*function*×*function* is the ternary power of *function*. Any function triple has a source (target) collection-triple consisting of the sources (targets) of its function components. Special function triples are used as the cones for ternary products and multipullbacks.

```
 (1) (UR$object function-triple}
     (UR$subobject function-triple KIF.FTN$tuple)

 (2) (forall (?f (KIF.FTN$tuple ?f))
         (<=> (function-triple ?f)
              (and (= (KIF.FTN$arity ?f) KIF.COL$three)
                   (= (KIF.FTN$type ?f) KIF.FTN$function))))

 (3) (UR$morphism function0)
     (= (UR$source function0) function-triple)
     (= (UR$target function0) KIF.FTN$function)

 (4) (UR$morphism function1)
     (= (UR$source function1) function-triple)
     (= (UR$target function1) KIF.FTN$function)

 (5) (UR$morphism function2)
     (= (UR$source function2) function-triple)
     (= (UR$target function2) KIF.FTN$function)

 (6) (UR$morphism source)
     (= (UR$source source) function-triple)
     (= (UR$target source) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$composition [source KIF.DGM.TRP.OBJ$collection0)
        (UR$composition [function0 KIF.FTN$source]))
     (= (UR$composition [source KIF.DGM.TRP.OBJ$collection1])
        (UR$composition [function1 KIF.FTN$source]))
     (= (UR$composition [source KIF.DGM.TRP.OBJ$collection2])
        (UR$composition [function2 KIF.FTN$source]))

 (7) (UR$morphism target)
     (= (UR$source target) function-triple)
     (= (UR$target target) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$composition [target KIF.DGM.TRP.OBJ$collection0)
        (UR$composition [function0 KIF.FTN$target]))
     (= (UR$composition [target KIF.DGM.TRP.OBJ$collection1])
        (UR$composition [function1 KIF.FTN$target]))
     (= (UR$composition [target KIF.DGM.TRP.OBJ$collection2])
        (UR$composition [function2 KIF.FTN$target]))

 (8) (forall (?f (function-triple ?f))
         (and (= (function0 ?f) (?f 0))
              (= (function1 ?f) (?f 1))
              (= (function2 ?f) (?f 2))))
```

Function triples can be composed. Two function triples are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable function triples $f : (X_0, X_1, X_2) \rightarrow (Y_0, Y_1, Y_2)$ and $g : (Y_0, Y_1, Y_2) \rightarrow (Z_0, Z_1, Z_2)$ is the function triple $f \cdot g : (X_0, X_1, X_2) \rightarrow (Z_0, Z_1, Z_2)$ whose components are defined by $(f \cdot g)_0 = f_0 \cdot g_0$, $(f \cdot g)_1 = f_1 \cdot g_1$ and $(f \cdot g)_2 = f_2 \cdot g_2$. For any collection triple $X = (X_0, X_1, X_2)$, there is an *identity* function triple.

```
 (9) (UR$object composable-pair-function-triple)

(10) (forall (?z (composable-pair-function-triple ?z))
         (pair ?z))

(11) (forall (?f ?g (composable-pair-function-triple [?f ?g]))
         (and (function-triple ?f) (function-triple ?g)))
```

```
(12) (forall (?f (function-triple ?f) ?g (function-triple ?g))
         (<=> (composable-pair-function-triple [?f ?g])
              (= (target ?f) (source ?g))))

(13) (UR$morphism composition)
     (= (UR$source composition) composable-pair-function-triple)
     (= (UR$target composition) function-triple)

(14) (forall (?f (function-triple ?f) ?g (function-triple ?g) (= (target ?f) (source ?g)))
         (and (= (source (composition [?f ?g])) (source ?f))
              (= (target (composition [?f ?g])) (target ?g))
              (= (function0 (composition [?f ?g]))
                 (KIF.FTN$composition [(function0 ?f) (function0 ?g)]))
              (= (function1 (composition [?f ?g]))
                 (KIF.FTN$composition [(function1 ?f) (function1 ?g)]))
              (= (function2 (composition [?f ?g]))
                 (KIF.FTN$composition [(function2 ?f) (function2 ?g)]))))

(15) (UR$morphism identity)
     (= (UR$source identity) KIF.DGM.PR.OBJ$collection-triple)
     (= (UR$target identity) function-triple)
     (= (UR$composition [identity source]) (UR$identity KIF.DGM.TRP.OBJ$collection-triple))
     (= (UR$composition [identity target]) (UR$identity KIF.DGM.TRP.OBJ$collection-triple))
     (= (UR$composition [identity function0])
        (UR$composition [KIF.DGM.TRP.OBJ$collection0 KIF.FTN$identity]))
     (= (UR$composition [identity function1])
        (UR$composition [KIF.DGM.TRP.OBJ$collection1 KIF.FTN$identity]))
     (= (UR$composition [identity function2])
        (UR$composition [KIF.DGM.TRP.OBJ$collection2 KIF.FTN$identity]))
```

Composition satisfies the usual *associative law*.

```
(16) (forall (?f (function-triple ?f) ?g (function-triple ?g) ?h (function-triple ?h)
              (= (target ?f) (source ?g)) (= (target ?g) (source ?h)))
         (= (composition [?f (composition [?g ?h])])
            (composition [(composition [?f ?g]) ?h])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(17) (forall (?f (function-triple ?f))
         (and (= (composition [(identity (source ?f)) ?f]) ?f)
              (= (composition [?f (identity (target ?f))]) ?f)))
```

Parallel Pairs

KIF.DGM.PPR.OBJ

The *parallel pair* collection consists of pairs of functions $x_0 : X_0 \rightarrow X_1$ and $x_1 : X_0 \rightarrow X_1$ with common source and target collections. Parallel pairs are special function pairs that are used as the diagrams for equalizers. A parallel pair has an underlying collection pair.

```
 (1) (UR$object parallel-pair)
     (UR$subobject parallel-pair KIF.DGM.PR.MOR$function-pair)

 (2) (forall (?f (KIF.DGM.PR.MOR$function-pair ?f))
         (<=> (parallel-pair ?f)
              (and (= (KIF.DGM.PR.OBJ$collection0 (KIF.DGM.PR.MOR$source ?f))
                      (KIF.DGM.PR.OBJ$collection1 (KIF.DGM.PR.MOR$source ?f)))
                   (= (KIF.DGM.PR.OBJ$collection0 (KIF.DGM.PR.MOR$target ?f))
                      (KIF.DGM.PR.OBJ$collection1 (KIF.DGM.PR.MOR$target ?f))))))

 (3) (UR$morphism collection-pair)
     (= (UR$source collection-pair) parallel-pair)
     (= (UR$target collection-pair) KIF.DGM.PR.OBJ$collection-pair)

 (4) (UR$morphism collection0)
     (= (UR$source collection0) parallel-pair)
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection0]))
     (= collection0 (UR$composition
                        [(UR$inclusion [parallel-pair KIF.DGM.PR.MOR$function-pair])
                   (UR$composition [KIF.DGM.PR.MOR$source KIF.DGM.PR.OBJ$collection0])]))

(5) (UR$morphism collection1)
    (= (UR$source collection1) parallel-pair)
    (= (UR$target collection1) KIF.COL$collection)
    (= collection1 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection1]))
    (= collection1 (UR$composition
                  [(UR$inclusion [parallel-pair KIF.DGM.PR.MOR$function-pair])
                   (UR$composition [KIF.DGM.PR.MOR$target KIF.DGM.PR.OBJ$collection1])]))

(6) (UR$morphism function0)
    (= (UR$source function0) parallel-pair)
    (= (UR$target function0) KIF.FTN$function)
    (= (UR$composition [function0 KIF.FTN$source]) collection0)
    (= (UR$composition [function0 KIF.FTN$target]) collection1)
    (= function0 (UR$composition
                     [(UR$inclusion [parallel-pair KIF.DGM.PR.MOR$function-pair])
                      KIF.DGM.PR.MOR$function0]))

(7) (UR$morphism function1)
    (= (UR$source function1) parallel-pair)
    (= (UR$target function1) KIF.FTN$function)
    (= (UR$composition [function1 KIF.FTN$source]) collection0)
    (= (UR$composition [function1 KIF.FTN$target]) collection1)
    (= function1 (UR$composition
                     [(UR$inclusion [parallel-pair KIF.DGM.PR.MOR$function-pair])
                      KIF.DGM.PR.MOR$function1]))
```

A *parallel pair* of functions has an associated *opspan*. The opvertex is the target collection. , and the component functions are the unique functions for the component collections in the pair.

```
(8) (UR$morphism opspan)
    (= (UR$source opspan) parallel-pair)
    (= (UR$target opspan) KIF.DGM.OSPN.OBJ$opspan)
    (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection0]) collection0)
    (= (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection1]) collection0)
    (forall (?f (parallel-pair ?f))
        (and (= (KIF.DGM.OSPN.OBJ$opvertex (opspan ?f))
                (KIF.LIM.PRD2.OBJ$binary-product [(collection0 ?f) (collection1 ?f)]))
             (= (KIF.DGM.OSPN.OBJ$function0 (opspan ?f))
                (KIF.LIM.PRD2.OBJ$pairing [(KIF.FTN$identity (collection0 ?f)) (function0 ?f)]))
             (= (KIF.DGM.OSPN.OBJ$function1 (opspan ?f))
                (KIF.LIM.PRD2.OBJ$pairing [(KIF.FTN$identity (collection0 ?f)) (function1 ?f)]))
```

Parallel Pair Morphisms

KIF.DGM.PPR.MOR

Parallel pairs (of functions) are connected by morphisms. A *morphism* of *parallel pairs* $f = (f_0, f_1) : X \to Y$ connecting parallel pair $X = (x_0 : X_0 \to X_1, x_1 : X_0 \to X_1)$ to parallel pair $Y = (y_0 : Y_0 \to Y_1, y_1 : Y_0 \to Y_1)$ consists of a pair of functions $(f_0 : X_0 \to Y_0, f_1 : X_1 \to Y_1)$ connecting the respective collection components (Figure 9). Any parallel pair morphism has a source (target) parallel pair, whose collection components consist of the sources (targets) of its function components. The pair of functions forms a commuting diagram with the component functions of the source and target. Any parallel pair morphism has an underlying function pair (f_0, f_1). Parallel pair morphisms are used in defining the packing-unpacking isomorphism of equalizers.

$$\begin{tikzcd} X_0 \arrow[r, "f_0"] \arrow[d, shift right, "x_0"'] \arrow[d, shift left, "x_1"] & Y_0 \arrow[d, shift right, "y_0"'] \arrow[d, shift left, "y_1"] \\ X_1 \arrow[r, "f_1"] & Y_1 \end{tikzcd}$$

Figure 9: Parallel Pair Morphism

```
(1) (UR$object parallel-pair-morphism)
    (UR$subobject parallel-pair-morphism KIF.DGM.PR.MOR$function-pair)

(2) (UR$morphism source)
    (= (UR$source source) parallel-pair-morphism)
    (= (UR$target source) parallel-pair)

(3) (UR$morphism target)
    (= (UR$source target) parallel-pair-morphism)
    (= (UR$target target) parallel-pair)
```

```
 (4)  (UR$morphism function-pair)
      (= (UR$source function-pair) parallel-pair-morphism)
      (= (UR$target function-pair) KIF.DGM.PR.MOR$function-pair)
      (= (UR$composition [function-pair KIF.DGM.PR.MOR$source])
         (UR$composition [source KIF.DGM.PPR.OBJ$collection-pair]))
      (= (UR$composition [function-pair KIF.DGM.PR.MOR$target])
         (UR$composition [target KIF.DGM.PPR.OBJ$collection-pair]))
      (= function-pair (UR$inclusion [parallel-pair-morphism KIF.DGM.PR.MOR$function-pair]))

 (5)  (UR$morphism function0)
      (= (UR$source function0) parallel-pair-morphism)
      (= (UR$target function0) KIF.FTN$function)
      (= (UR$composition [function0 KIF.FTN$source])
         (UR$composition [source KIF.DGM.PPR.OBJ$collection0]))
      (= (UR$composition [function0 KIF.FTN$target])
         (UR$composition [target KIF.DGM.OSPN.OBJ$collection0]))
      (= function0 (UR$composition [function-pair KIF.DGM.PR.MOR$function0]))

 (6)  (UR$morphism function1)
      (= (UR$source function1) parallel-pair-morphism)
      (= (UR$target function1) KIF.FTN$function)
      (= (UR$composition [function1 KIF.FTN$source])
         (UR$composition [source KIF.DGM.PPR.OBJ$collection1]))
      (= (UR$composition [function1 KIF.FTN$target])
         (UR$composition [target KIF.DGM.PPR.OBJ$collection1]))
      (= function1 (UR$composition [function-pair KIF.DGM.PR.MOR$function1]))

 (7)  (forall ?f (parallel-pair-morphism ?f))
          (and (= (KIF.FTN$composition [(function0 ?f) (KIF.DGM.PPR.OBJ$function0 (target ?f))])
                  (KIF.FTN$composition [(KIF.DGM.PPR.OBJ$function0 (source ?f)) (function1 ?f)]))
               (= (KIF.FTN$composition [(function0 ?f) (KIF.DGM.PPR.OBJ$function1 (target ?f))])
                  (KIF.FTN$composition [(KIF.DGM.PPR.OBJ$function1 (source ?f)) (function1 ?f)]))))
```

Parallel pair morphisms can be composed. Two parallel pair morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable parallel pair morphisms $f = (f_0, f_1) : X \to Y$ and $g = (g_0, g_1) : Y \to Z$ is the parallel pair morphism $f \cdot g : X \to Z$ 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 parallel pair $X = (x_0 : X_0 \to X_1, x_1 : X_0 \to X_1)$, there is an *identity* parallel pair morphism.

```
 (8)  (UR$object composable-parallel-pair-morphism)

 (9)  (forall (?z (composable-parallel-pair-morphism ?z))
          (pair ?z))

(10)  (forall (?f ?g (composable-parallel-pair-morphism [?f ?g]))
          (and (parallel-pair-morphism-?f) (parallel pair-morphism ?g)))

(11)  (forall (?f (parallel-pair-morphism ?f) ?g (parallel-pair-morphism ?g))
          (<=> (composable-parallel-pair-morphism [?f ?g])
               (= (target ?f) (source ?g))))

(12)  (UR$morphism composition)
      (= (UR$source composition) composable-parallel-pair-morphism)
      (= (UR$target composition) parallel-pair-morphism)

(13)  (forall (?f (parallel-pair-morphism ?f) ?g (parallel-pair-morphism ?g)
                (= (target ?f) (source ?g)))
          (and (= (source (composition [?f ?g])) (source ?f))
               (= (target (composition [?f ?g])) (target ?g))
               (= (function-pair (composition [?f ?g]))
                  (KIF.DGM.PR.MOR$composition [(function-pair ?f) (function-pair ?g)]))
               (= (function0 (composition [?f ?g]))
                  (KIF.FTN$composition [(function0 ?f) (function0 ?g)]))
               (= (function1 (composition [?f ?g]))
                  (KIF.FTN$composition [(function1 ?f) (function1 ?g)]))))

(14)  (UR$morphism identity)
      (= (UR$source identity) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$target identity) parallel-pair-morphism)
```

```
     (= (UR$composition [identity source]) (UR$identity KIF.DGM.PPR.OBJ$parallel-pair))
     (= (UR$composition [identity target]) (UR$identity KIF.DGM.PPR.OBJ$parallel-pair))
     (= (UR$composition [identity function-pair])
        (UR$composition [KIF.DGM.PPR.OBJ$collection-pair KIF.DGM.PR.MOR$identity]))
     (= (UR$composition [identity function0])
        (UR$composition [KIF.DGM.PPR.OBJ$collection0 KIF.FTN$identity]))
     (= (UR$composition [identity function1])
        (UR$composition [KIF.DGM.PPR.OBJ$collection1 KIF.FTN$identity]))
```

Composition satisfies the usual *associative law*.

```
(15) (forall (?f (parallel-pair-morphism ?f)
              ?g (parallel-pair-morphism ?g) ?h (parallel-pair-morphism ?h)
             (= (target ?f) (source ?g)) (= (target ?g) (source ?h)))
         (= (composition [?f (composition [?g ?h])])
            (composition [(composition [?f ?g]) ?h])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(16) (forall (?f (parallel-pair-morphism ?f))
         (and (= (composition [(identity (source ?f)) ?f]) ?f)
              (= (composition [?f (identity (target ?f))]) ?f)))
```

Opspans

KIF.DGM.OSPN.OBJ

An *opspan* $X = (x_0 : X_0 \rightarrow X_\bullet \leftarrow X_1 : x_1)$ is a pair of functions called *opzeroth* = $x_0 : X_0 \rightarrow X_\bullet$ and *opfirst* = $x_1 : X_1 \rightarrow X_\bullet$ with common target collection called the *opvertex* $X_\bullet$. Hence, for each opspan there is an underlying pair (X_0, X_1) of source collections of its two component functions, and an underlying target collection $X_\bullet$. Opspans are special function pairs that are used as the diagrams for pullbacks.

```
 (1) (UR$object opspan)
     (UR$subobject opspan KIF.DGM.PR.MOR$function-pair)

 (2) (forall (?g0 (KIF.FTN$function ?g0) ?g1 (KIF.FTN$function ?g1))
         (<=> (opspan [?g0 ?g1])
              (= (KIF.FTN$target ?g0) (KIF.FTN$target ?g1))))

 (3) (UR$morphism opvertex)
     (= (UR$source opvertex) opspan)
     (= (UR$target opvertex) KIF.COL$collection)

 (4) (UR$morphism opzeroth)
     (= (UR$source opzeroth) opspan)
     (= (UR$target opzeroth) KIF.FTN$function)
     (= (UR$composition [opzeroth KIF.FTN$source]) collection0)
     (= (UR$composition [opzeroth KIF.FTN$target]) opvertex)
     (= opzeroth (UR$composition
                     [(UR$inclusion [opspan KIF.DGM.PR.MOR$function-pair])
                       KIF.DGM.PR.MOR$function0]))

 (5) (UR$morphism opfirst)
     (= (UR$source opfirst) opspan)
     (= (UR$target opfirst) KIF.FTN$function)
     (= (UR$composition [opfirst KIF.FTN$source]) collection1)
     (= (UR$composition [opfirst KIF.FTN$target]) opvertex)
     (= opfirst (UR$composition
                     [(UR$inclusion [opspan KIF.DGM.PR.MOR$function-pair])
                       KIF.DGM.PR.MOR$function1]))

 (6) (UR$morphism collection-pair)
     (= (UR$source collection-pair) opspan)
     (= (UR$target collection-pair) KIF.DGM.PR.OBJ$collection-pair)
     (= collection-pair (UR$composition
                [(UR$inclusion [opspan KIF.DGM.PR.MOR$function-pair])
                  KIF.DGM.PR.MOR$source]))

 (7) (UR$morphism collection0)
     (= (UR$source collection0) opspan)
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection0]))
 (8)  (UR$morphism collection1)
     (= (UR$source collection1) opspan)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection1]))
```

Associated with any opspan $X = (x_0 : X_0 \rightarrow X_\bullet,\ x_1 : X_1 \rightarrow X_\bullet)$ is a *relation* $rel(X) \subseteq X_0{\times}X_1$, whose extent is defined to be the pullback collection $\{(a_0, a_1) \mid x_0(a_0) = x_1(a_1)\}$.

```
 (9)  (UR$morphism relation)
      (= (UR$source relation) opspan)
      (= (UR$target relation) KIF.REL$relation)
      (= (UR$composition [relation KIF.REL$collection0]) collection0)
      (= (UR$composition [relation KIF.REL$collection1]) collection1)

(10)  (forall (?s (opspan ?s)
               ?x0 ((KIF.REL$collection0 ?s) ?x0) ?x1 ((KIF.REL$collection1 ?s) ?x1))
          (<=> ((relation ?s) ?x0 ?x1)
               (= ((opzeroth ?s) ?x0) ((opfirst ?s) ?x1))))
```

Any *opspan* of functions has an associated *parallel pair*, whose component functions are the composite of the product projections of the binary product of the pair of classes overlying the opspan with the component functions of the opspan. The equalizer and inclusion of this parallel pair can be used to define the pullback and pullback projections.

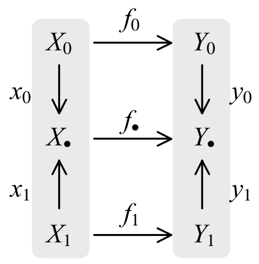
```
(11)  (UR$morphism parallel-pair)
      (= (UR$source parallel-pair) opspan)
      (= (UR$target parallel-pair) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$composition [parallel-pair KIF.DGM.PPR.OBJ$collection0])
         (UR$composition [pair KIF.LIM.PRD2$binary-product]))
      (= (UR$composition [parallel-pair KIF.DGM.PPR.OBJ$collection1]) opvertex)

(12)  (forall (?s (opspan ?s))
          (and (= (KIF.DGM.PPR.OBJ$function0 (parallel-pair ?s))
                  (KIF.FTN$composition [(KIF.LIM.PRD2projection0 (pair ?s)) (opzeroth ?s)]))
               (= (KIF.DGM.PPR.OBJ$function1 (parallel-pair ?s))
                  (KIF.FTN$composition [(KIF.LIM.PRD2projection1 (pair ?s)) (opfirst ?s)]))))
```

Opspan Morphisms

KIF.DGM.OSPN.MOR

Opspans are connected by morphisms. An *opspan morphism* $f = (f_0, f_\bullet, f_1) : X \rightarrow Y$ connecting opspan $X = (x_0 : X_0 \rightarrow X_\bullet,\ x_1 : X_1 \rightarrow X_\bullet)$ to opspan $Y = (y_0 : Y_0 \rightarrow Y_\bullet,\ y_1 : Y_1 \rightarrow Y_\bullet)$ consists of a triple of functions $(f_0 : X_0 \rightarrow Y_0,\ \ f_\bullet : X_\bullet \rightarrow Y_\bullet,\ \ f_1 : X_1 \rightarrow Y_1)$ connecting the respective collection components (Figure 10). Any opspan morphism has a source (target) opspan, whose components consist of the sources (targets) of its function components. The triple of functions forms commuting diagrams with the component functions of the source and target. Any opspan morphism has an underlying function pair (f_0, f_1). Opspan morphisms are used in defining the packing-unpacking isomorphism of pullbacks.



$$\begin{tikzcd}
X_0 \arrow[r, "f_0"] \arrow[d, "x_0"'] & Y_0 \arrow[d, "y_0"] \\
X_\bullet \arrow[r, "f_\bullet"] & Y_\bullet \\
X_1 \arrow[r, "f_1"] \arrow[u, "x_1"] & Y_1 \arrow[u, "y_1"']
\end{tikzcd}$$

Figure 10: Opspan Morphism

```
 (1)  (UR$object opspan-morphism)
      (UR$subobject opspan-morphism KIF.DGM.TRP.MOR$function-triple)

 (2)  (UR$morphism source)
      (= (UR$source source) opspan-morphism)
      (= (UR$target source) KIF.DGM.OSPN.OBJ$opspan)

 (3)  (UR$morphism target)
      (= (UR$source target) opspan-morphism)
      (= (UR$target target) KIF.DGM.OSPN.OBJ$opspan)

 (4)  (UR$morphism opvertex)
      (= (UR$source opvertex) opspan-morphism)
```

```
     (= (UR$target opvertex) KIF.FTN$function)
     (= (UR$composition [opvertex KIF.FTN$source])
        (UR$composition [source KIF.DGM.OSPN.OBJ$opvertex]))
     (= (UR$composition [opvertex KIF.FTN$target])
        (UR$composition [target KIF.DGM.OSPN.OBJ$opvertex]))

 (5) (UR$morphism function-pair)
     (= (UR$source function-pair) opspan-morphism)
     (= (UR$target function-pair) KIF.DGM.PR.MOR$function-pair)
     (= (UR$composition [function-pair KIF.DGM.PR.MOR$source])
        (UR$composition [source KIF.DGM.OSPN.OBJ$collection-pair]))
     (= (UR$composition [function-pair KIF.DGM.PR.MOR$target])
        (UR$composition [target KIF.DGM.OSPN.OBJ$collection-pair]))

 (6) (UR$morphism function0)
     (= (UR$source function0) opspan-morphism)
     (= (UR$target function0) KIF.FTN$function)
     (= (UR$composition [function0 KIF.FTN$source])
        (UR$composition [source KIF.DGM.OSPN.OBJ$collection0]))
     (= (UR$composition [function0 KIF.FTN$target])
        (UR$composition [target KIF.DGM.OSPN.OBJ$collection0]))
     (= function0 (UR$composition [function-pair KIF.DGM.PR.MOR$function0]))

 (7) (UR$morphism function1)
     (= (UR$source function1) opspan-morphism)
     (= (UR$target function1) KIF.FTN$function)
     (= (UR$composition [function1 KIF.FTN$source])
        (UR$composition [source KIF.DGM.OSPN.OBJ$collection1]))
     (= (UR$composition [function1 KIF.FTN$target])
        (UR$composition [target KIF.DGM.OSPN.OBJ$collection1]))
        (= function1 (UR$composition [function-pair KIF.DGM.PR.MOR$function1]))

 (8) (forall ?f (opspan-morphism ?f))
         (and (= (KIF.FTN$composition [(function0 ?f) (KIF.DGM.OSPN.OBJ$opzeroth (target ?f))])
                 (KIF.FTN$composition [(KIF.DGM.OSPN.OBJ$opzeroth (source ?f)) (opvertex ?f)]))
              (= (KIF.FTN$composition [(function1 ?f) (KIF.DGM.OSPN.OBJ$opfirst (target ?f))])
                 (KIF.FTN$composition [(KIF.DGM.OSPN.OBJ$opfirst (source ?f)) (opvertex ?f)]))))
```

Opspan morphisms can be composed. Two opspan morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable opspan morphisms $f = (f_0, f_\bullet, f_1) : X \rightarrow Y$ and $g = (g_0, g_\bullet, g_1) : Y \rightarrow Z$ is the opspan morphism $f \cdot g : X \rightarrow Z$ whose components are defined by $(f \cdot g)_0 = f_0 \cdot g_0$, $(f \cdot g)_\bullet = f_\bullet \cdot g_\bullet$ and $(f \cdot g)_1 = f_1 \cdot g_1$. For any opspan $X = (x_0 : X_0 \rightarrow X_\bullet, x_1 : X_1 \rightarrow X_\bullet)$, there is an *identity* opspan morphism.

```
 (9) (UR$object composable-opspan-morphism)

(10) (forall (?z (composable-opspan-morphism ?z))
         (pair ?z))

(11) (forall (?f ?g (composable-opspan-morphism [?f ?g]))
         (and (opspan-morphism ?f) (opspan-morphism ?g)))

(12) (forall (?f (opspan-morphism ?f) ?g (opspan-morphism ?g))
         (<=> (composable-opspan-morphism [?f ?g])
              (= (target ?f) (source ?g))))

(13) (UR$morphism composition)
     (= (UR$source composition) composable-opspan-morphism)
     (= (UR$target composition) opspan-morphism)

(14) (forall (?f (opspan-morphism ?f) ?g (opspan-morphism ?g)
                (= (target ?f) (source ?g)))
         (and (= (source (composition [?f ?g])) (source ?f))
              (= (target (composition [?f ?g])) (target ?g))
              (= (opvertex (composition [?f ?g]))
                 (KIF.FTN$composition [(opvertex ?f) (opvertex ?g)]))
              (= (function-pair (composition [?f ?g]))
                 (KIF.DGM.PR.MOR$composition [(function-pair ?f) (function-pair ?g)]))
              (= (function0 (composition [?f ?g]))
                 (KIF.FTN$composition [(function0 ?f) (function0 ?g)]))
```

```
                  (= (function1 (composition [?f ?g]))
                     (KIF.FTN$composition [(function1 ?f) (function1 ?g)]))))

(15)  (UR$morphism identity)
      (= (UR$source identity) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$target identity) opspan-morphism)
      (= (UR$composition [identity source]) (UR$identity KIF.DGM.OSPN.OBJ$opspan))
      (= (UR$composition [identity target]) (UR$identity KIF.DGM.OSPN.OBJ$opspan))
      (= (UR$composition [identity opvertex])
         (UR$composition [KIF.DGM.OSPN.OBJ$opvertex KIF.FTN$identity]))
      (= (UR$composition [identity function-pair])
         (UR$composition [KIF.DGM.OSPN.OBJ$collection-pair KIF.DGM.PR.MOR$identity]))
      (= (UR$composition [identity function0])
         (UR$composition [KIF.DGM.OSPN.OBJ$collection0 KIF.FTN$identity]))
      (= (UR$composition [identity function1])
         (UR$composition [KIF.DGM.OSPN.OBJ$collection1 KIF.FTN$identity]))
```

Composition satisfies the usual *associative law*.

```
(16)  (forall (?f (opspan-morphism ?f) ?g (opspan-morphism ?g) ?h (opspan-morphism ?h)
                (= (target ?f) (source ?g)) (= (target ?g) (source ?h)))
          (= (composition [?f (composition [?g ?h])])
             (composition [(composition [?f ?g]) ?h])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(17)  (forall (?f (opspan-morphism ?f))
          (and (= (composition [(identity (source ?f)) ?f]) ?f)
               (= (composition [?f (identity (target ?f))]) ?f)))
```

Multi-Opspans

`KIF.DGM.MOSPN.OBJ`

A *multi-opspan* $X = (S_0, S_1) = ((x_{00} : X_{00} \rightarrow X_{0\bullet} \leftarrow X_{01} : x_{01}), (x_{10} : X_{10} \rightarrow X_{1\bullet} \leftarrow X_{11} : x_{11}))$ is a pair of opspans connected by a common collection $X_{01} = X_{10}$. A multi-opspan is a diagram in the shape of a 'W'. A multi-opspan has the following components: there are four functions called *opfunction*$_{00} = x_{00} : X_{00} \rightarrow X_{0\bullet}$, *opfunction*$_{01} = x_{01} : X_{01} \rightarrow X_{0\bullet}$, *opfunction*$_{10} = x_{10} : X_{10} \rightarrow X_{1\bullet}$ and *opfunction*$_{11} = x_{11} : X_{11} \rightarrow X_{1\bullet}$; there are two vertices called *opvertex*$_0 = X_{0\bullet}$ and *opvertex*$_1 = X_{1\bullet}$; and there is a collection triple consisting of three collections called *collection*$_0 = X_0$, *collection*$_1 = X_1$ and *collection*$_2 = X_2$. Multi-opspans are special opspan pairs that are used as the diagrams for multi-pullbacks.

```
 (1)  (UR$object multi-opspan)
      (UR$subobject multi-opspan KIF.COL$pair)

 (2)  (forall (?S0 (KIF.DGM.OSPN.OBJ$opspan ?S0)
               ?S1 (KIF.DGM.OSPN.OBJ$opspan ?S1))
          (<=> (multi-opspan [?S0 ?S1])
               (= (KIF.DGM.OSPN.OBJ$collection1 ?S0)
                  (KIF.DGM.OSPN.OBJ$collection0 ?S1))))

 (3)  (UR$morphism opspan0)
      (= (UR$source opspan0) multi-opspan)
      (= (UR$target opspan0) KIF.DGM.OSPN.OBJ$opspan)

 (4)  (UR$morphism opvertex0)
      (= (UR$source opvertex0) multi-opspan)
      (= (UR$target opvertex0) KIF.COL$collection)
      (= opvertex0 (UR$composition [opspan0 KIF.DGM.OSPN.OBJ$opvertex]))

 (5)  (UR$morphism opfunction00)
      (= (UR$source opfunction00) multi-opspan)
      (= (UR$target opfunction00) KIF.FTN$function)
      (= (UR$composition [opfunction00 KIF.FTN$source]) collection0)
      (= (UR$composition [opfunction00 KIF.FTN$target]) opvertex0)
      (= opfunction00 (UR$composition [opspan0 KIF.DGM.OSPN.OBJ$opzeroth]))

 (6)  (UR$morphism opfunction01)
      (= (UR$source opfunction01) multi-opspan)
      (= (UR$target opfunction01) KIF.FTN$function)
      (= (UR$composition [opfunction01 KIF.FTN$source]) collection)
      (= (UR$composition [opfunction01 KIF.FTN$target]) opvertex0)
      (= opfunction01 (UR$composition [opspan0 KIF.DGM.OSPN.OBJ$opfirst]))

 (7)  (UR$morphism collection-pair0)
      (= (UR$source collection-pair0) multi-opspan)
      (= (UR$target collection-pair0) KIF.DGM.PR.OBJ$collection-pair)

 (8)  (UR$morphism collection00)
      (= (UR$source collection00) multi-opspan)
      (= (UR$target collection00) KIF.COL$collection)
      (= collection00 (UR$composition [collection-pair0 KIF.DGM.PR.OBJ$collection0]))

 (9)  (UR$morphism collection01)
      (= (UR$source collection01) multi-opspan)
      (= (UR$target collection01) KIF.COL$collection)
      (= collection01 (UR$composition [collection-pair0 KIF.DGM.PR.OBJ$collection1]))

(10)  (UR$morphism opspan1)
      (= (UR$source opspan1) multi-opspan)
      (= (UR$target opspan1) KIF.DGM.OSPN.OBJ$opspan)

(11)  (UR$morphism opvertex1)
      (= (UR$source opvertex1) multi-opspan)
      (= (UR$target opvertex1) KIF.COL$collection)
      (= opvertex1 (UR$composition [opspan1 KIF.DGM.OSPN.OBJ$opvertex]))

(12)  (UR$morphism opfunction10)
      (= (UR$source opfunction10) multi-opspan)
      (= (UR$target opfunction10) KIF.FTN$function)
      (= (UR$composition [opfunction10 KIF.FTN$source]) collection)
      (= (UR$composition [opfunction10 KIF.FTN$target]) opvertex1)
      (= opfunction10 (UR$composition [opspan1 KIF.DGM.OSPN.OBJ$opzeroth]))

(13)  (UR$morphism opfunction11)
      (= (UR$source opfunction11) multi-opspan)
      (= (UR$target opfunction11) KIF.FTN$function)
      (= (UR$composition [opfunction11 KIF.FTN$source]) collection1)
      (= (UR$composition [opfunction11 KIF.FTN$target]) opvertex1)
      (= opfunction11 (UR$composition [opspan1 KIF.DGM.OSPN.OBJ$opfirst]))

(14)  (UR$morphism collection-pair1)
      (= (UR$source collection-pair1) multi-opspan)
      (= (UR$target collection-pair1) KIF.DGM.PR.OBJ$collection-pair)

(15)  (UR$morphism collection10)
      (= (UR$source collection10) multi-opspan)
      (= (UR$target collection10) KIF.COL$collection)
      (= collection10 (UR$composition [collection-pair1 KIF.DGM.PR.OBJ$collection0]))

(16)  (UR$morphism collection11)
      (= (UR$source collection11) multi-opspan)
      (= (UR$target collection11) KIF.COL$collection)
      (= collection11 (UR$composition [collection-pair1 KIF.DGM.PR.OBJ$collection1]))

(17)  (UR$morphism collection-triple)
      (= (UR$source collection-triple) multi-opspan)
      (= (UR$target collection-triple) KIF.DGM.TRP.OBJ$collection-triple)

(18)  (UR$morphism collection0)
      (= (UR$source collection0) multi-opspan)
      (= (UR$target collection0) KIF.COL$collection)
      (= collection0 (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection0]))
      (= collection0 collection00)

(19)  (UR$morphism collection)
      (= (UR$source collection) multi-opspan)
      (= (UR$target collection) KIF.COL$collection)
      (= collection (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection1]))
      (= collection collection01)
      (= collection collection10)
```

```
(20) (UR$morphism collection1)
     (= (UR$source collection1) multi-opspan)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection2]))
     (= collection1 collection11)
```

Opspan Morphisms

KIF.DGM.OSPN.MOR

Multi-opspans are connected by morphisms. A *multi-opspan morphism* $f = (f_0, f_1) : X \rightarrow Y$ connecting multi-opspan $X = (R_0, R_1) = ((x_{00} : X_{00} \rightarrow X_{0\bullet} \leftarrow X_{01} : x_{01}), (x_{10} : X_{10} \rightarrow X_{1\bullet} \leftarrow X_{11} : x_{11}))$ to multi-opspan $Y = (S_0, S_1) = ((y_{00} : Y_{00} \rightarrow Y_{0\bullet} \leftarrow Y_{01} : y_{01}), (y_{10} : Y_{10} \rightarrow Y_{1\bullet} \leftarrow Y_{11} : y_{11}))$ is a pair of opspan morphisms connected by a common function $f_{01} = f_{10}$. A multi-opspan morphism is a quintuple of functions

$$(f_{00} : X_{00} \rightarrow Y_{00}, f_{0\bullet} : X_{0\bullet} \rightarrow Y_{0\bullet}, f_{01} = f_{10} : X_{01} = X_{10} \rightarrow Y_{01} = Y_{10}, f_{1\bullet} : X_{1\bullet} \rightarrow Y_{1\bullet}, f_{11} : X_{11} \rightarrow Y_{11})$$

connecting the corresponding vertices in the source/target 'W'-shaped diagrams (Figure 11). A multi-opspan morphism has the following components: there are two vertex functions called *opvertex*$_0$ = $f_{0\bullet} : X_{0\bullet} \rightarrow Y_{0\bullet}$ and *opvertex*$_1$ = $f_{1\bullet} : X_{1\bullet} \rightarrow Y_{1\bullet}$; and there is a function triple consisting of three functions called *function*$_0$ = f_0 = $f_{00} : X_{00} \rightarrow Y_{00}$, *function*$_1$ = $f = f_{01} = f_{10} : X_{01} = X_{10} \rightarrow Y_{01} = Y_{10}$ and *function*$_2$ = $f_1 = f_{11} : X_{11} \rightarrow Y_{11}$. Any multi-opspan morphism has a source (target) multi-opspan, whose components consist of the sources (targets) of its function components. The quintuple of functions forms commuting diagrams with the component functions of the source and target. Any multi-opspan morphism has an underlying function triple (f_0, f, f_1). Multi-opspan morphisms are used in defining the packing-unpacking isomorphism of multipullbacks.

$$\begin{tikzcd}
X_0 = X_{00} \arrow[r, "f_0", "=f_{00}"'] \arrow[d, "x_{00}"'] & Y_{00} = Y_0 \arrow[d, "y_{00}"] \\
X_{0\bullet} \arrow[r, "f_{0\bullet}"] & Y_{0\bullet} \\
X_{01} = X = X_{10} \arrow[u, "x_{01}"] \arrow[r, "f_{01}", "=f \; =f_{10}"'] \arrow[d, "x_{10}"'] & Y_{01} = Y = Y_{10} \arrow[u, "y_{01}"'] \arrow[d, "y_{10}"] \\
X_{1\bullet} \arrow[r, "f_{1\bullet}"] & Y_{1\bullet} \\
X_1 = X_{11} \arrow[u, "x_{11}"] \arrow[r, "f_{11}", "=f_1"'] & Y_{11} = Y_1 \arrow[u, "y_{11}"']
\end{tikzcd}$$

Figure 11: Multi-opspan Morphism

```
(1) (UR$object multi-opspan-morphism)
    (UR$subobject multi-opspan-morphism KIF.COL$pair)

(2) (forall (?f0 (KIF.DGM.OSPN.MOR$opspan-morphism ?f0)
             ?f1 (KIF.DGM.OSPN.MOR$opspan-morphism ?f1))
        (<=> (multi-opspan-morphism [?f0 ?f1])
             (= (KIF.DGM.OSPN.MOR$function1 ?f0)
                (KIF.DGM.OSPN.MOR$function0 ?f1))))

(3) (UR$morphism source)
    (= (UR$source source) multi-opspan-morphism)
    (= (UR$target source) KIF.DGM.MOSPN.OBJ$multi-opspan)

(4) (UR$morphism target)
    (= (UR$source target) multi-opspan-morphism)
    (= (UR$target target) KIF.DGM.MOSPN.OBJ$multi-opspan)

(5) (UR$morphism opspan-morphism0)
    (= (UR$source opspan-morphism0) multi-opspan-morphism)
    (= (UR$target opspan-morphism0) KIF.DGM.OSPN.MOR$opspan-morphism)
    (= (UR$composition [opspan-morphism0 KIF.DGM.OSPN.MOR$source])
       (UR$composition [source KIF.DGM.MOSPN.OBJ$opspan0]))
    (= (UR$composition [opspan-morphism0 KIF.DGM.OSPN.MOR$target])
       (UR$composition [target KIF.DGM.MOSPN.OBJ$opspan0]))

(6) (UR$morphism opvertex0)
    (= (UR$source opvertex0) multi-opspan-morphism)
    (= (UR$target opvertex0) KIF.FTN$function)
    (= (UR$composition [opvertex0 KIF.FTN$source])
       (UR$composition [source KIF.DGM.MOSPN.OBJ$opvertex0]))
    (= (UR$composition [opvertex0 KIF.FTN$target])
       (UR$composition [target KIF.DGM.MOSPN.OBJ$opvertex0]))
    (= opvertex0 (UR$composition [opspan-morphism0 KIF.DGM.OSPN.MOR$opvertex])

(7) (UR$morphism function-pair0)
    (= (UR$source function-pair0) multi-opspan-morphism)
    (= (UR$target function-pair0) KIF.DGM.PR.MOR$function-pair)
    (= (UR$composition [function-pair0 KIF.DGM.PR.MOR$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection-pair0]))
      (= (UR$composition [function-pair0 KIF.DGM.PR.MOR$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection-pair0]))
      (= function-pair0 (UR$composition [opspan-morphism0 KIF.DGM.OSPN.MOR$function-pair])

  (8) (UR$morphism function00)
      (= (UR$source function00) multi-opspan-morphism)
      (= (UR$target function00) KIF.FTN$function)
      (= (UR$composition [function00 KIF.FTN$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection00]))
      (= (UR$composition [function00 KIF.FTN$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection00]))
      (= function00 (UR$composition [function-pair0 KIF.DGM.PR.MOR$function0]))

  (9) (UR$morphism function01)
      (= (UR$source function01) multi-opspan-morphism)
      (= (UR$target function01) KIF.FTN$function)
      (= (UR$composition [function01 KIF.FTN$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection01]))
      (= (UR$composition [function01 KIF.FTN$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection01]))
      (= function01 (UR$composition [function-pair0 KIF.DGM.PR.MOR$function1]))

 (10) (UR$morphism opspan-morphism1)
      (= (UR$source opspan-morphism1) multi-opspan-morphism)
      (= (UR$target opspan-morphism1) KIF.DGM.OSPN.MOR$opspan-morphism)
      (= (UR$composition [opspan-morphism1 KIF.DGM.OSPN.MOR$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$opspan1]))
      (= (UR$composition [opspan-morphism1 KIF.DGM.OSPN.MOR$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$opspan1]))

 (11) (UR$morphism opvertex1)
      (= (UR$source opvertex1) multi-opspan-morphism)
      (= (UR$target opvertex1) KIF.FTN$function)
      (= (UR$composition [opvertex1 KIF.FTN$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$opvertex1]))
      (= (UR$composition [opvertex1 KIF.FTN$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$opvertex1]))
      (= opvertex1 (UR$composition [opspan-morphism1 KIF.DGM.OSPN.MOR$opvertex]))

 (12) (UR$morphism function-pair1)
      (= (UR$source function-pair1) multi-opspan-morphism)
      (= (UR$target function-pair1) KIF.DGM.PR.MOR$function-pair)
      (= (UR$composition [function-pair1 KIF.DGM.PR.MOR$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection-pair1]))
      (= (UR$composition [function-pair1 KIF.DGM.PR.MOR$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection-pair1]))
      (= function-pair1 (UR$composition [opspan-morphism1 KIF.DGM.OSPN.MOR$function-pair])

 (13) (UR$morphism function10)
      (= (UR$source function10) multi-opspan-morphism)
      (= (UR$target function10) KIF.FTN$function)
      (= (UR$composition [function10 KIF.FTN$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection10]))
      (= (UR$composition [function10 KIF.FTN$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection10]))
      (= function10 (UR$composition [function-pair1 KIF.DGM.PR.MOR$function0]))

 (14) (UR$morphism function11)
      (= (UR$source function11) multi-opspan-morphism)
      (= (UR$target function11) KIF.FTN$function)
      (= (UR$composition [function11 KIF.FTN$source])
         (UR$composition [source KIF.DGM.MOSPN.OBJ$collection11]))
      (= (UR$composition [function11 KIF.FTN$target])
         (UR$composition [target KIF.DGM.MOSPN.OBJ$collection11]))
      (= function11 (UR$composition [function-pair1 KIF.DGM.PR.MOR$function1]))

 (15) (UR$morphism function-triple)
      (= (UR$source function-triple) multi-opspan-morphism)
      (= (UR$target function-triple) KIF.DGM.TRP.MOR$function-triple)
      (= (UR$composition [function-triple KIF.DGM.TRP.MOR$source])
        (UR$composition [source KIF.DGM.MOSPN.OBJ$collection-triple]))
     (= (UR$composition [function-triple KIF.DGM.TRP.MOR$target])
        (UR$composition [target KIF.DGM.MOSPN.OBJ$collection-triple]))

(16) (UR$morphism function0)
     (= (UR$source function0) multi-opspan-morphism)
     (= (UR$target function0) KIF.FTN$function)
     (= (UR$composition [function0 KIF.FTN$source])
        (UR$composition [source KIF.DGM.MOSPN.OBJ$collection0]))
     (= (UR$composition [function0 KIF.FTN$target])
        (UR$composition [target KIF.DGM.MOSPN.OBJ$collection0]))
     (= function0 (UR$composition [function-triple KIF.DGM.TRP.MOR$function0]))
     (= function0 function00)

(17) (UR$morphism function)
     (= (UR$source function) multi-opspan-morphism)
     (= (UR$target function) KIF.FTN$function)
     (= (UR$composition [function KIF.FTN$source])
        (UR$composition [source KIF.DGM.MOSPN.OBJ$collection]))
     (= (UR$composition [function KIF.FTN$target])
        (UR$composition [target KIF.DGM.MOSPN.OBJ$collection]))
     (= function (UR$composition [function-triple KIF.DGM.TRP.OBJ$function1]))
     (= function function01) (= function function10)

(18) (UR$morphism function1)
     (= (UR$source function1) multi-opspan-morphism)
     (= (UR$target function1) KIF.FTN$function)
     (= (UR$composition [function1 KIF.FTN$source])
        (UR$composition [source KIF.DGM.MOSPN.OBJ$collection1]))
     (= (UR$composition [function1 KIF.FTN$target])
        (UR$composition [target KIF.DGM.MOSPN.OBJ$collection1]))
     (= function1 (UR$composition [function-triple KIF.DGM.TRP.OBJ$function2]))
     (= function1 function11)

(19) (forall ?f (multi-opspan-morphism ?f))
         (and (= (KIF.FTN$composition
                     [(function0 ?f) (KIF.DGM.MOSPN.OBJ$opfunction00 (target ?f))])
                 (KIF.FTN$composition
                     [(KIF.DGM.MOSPN.OBJ$opfunction00 (source ?f)) (opvertex0 ?f)]))
              (= (KIF.FTN$composition
                     [(function ?f) (KIF.DGM.MOSPN.OBJ$opfunction01 (target ?f))])
                 (KIF.FTN$composition
                     [(KIF.DGM.MOSPN.OBJ$opfunction01 (source ?f)) (opvertex0 ?f)]))
              (= (KIF.FTN$composition
                     [(function ?f) (KIF.DGM.MOSPN.OBJ$opfunction10 (target ?f))])
                 (KIF.FTN$composition
                     [(KIF.DGM.MOSPN.OBJ$opfunction10 (source ?f)) (opvertex1 ?f)]))
              (= (KIF.FTN$composition
                     [(function1 ?f) (KIF.DGM.MOSPN.OBJ$opfunction11 (target ?f))])
                 (KIF.FTN$composition
                     [(KIF.DGM.MOSPN.OBJ$opfunction11 (source ?f)) (opvertex1 ?f)]))))
```

Multi-opspan morphisms can be composed. Two multi-opspan morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable opspan morphisms $f = (f_0, f_{0\bullet}, f, f_{1\bullet}, f_1) : X \rightarrow Y$ and $g = (g_0, g_{0\bullet}, g, g_{1\bullet}, g_1) : Y \rightarrow Z$ is the multi-opspan morphism $f \cdot g : X \rightarrow Z$ whose components are defined by $(f \cdot g)_0 = f_0 \cdot g_0$, $(f \cdot g)_{0\bullet} = f_{0\bullet} \cdot g_{0\bullet}$, $(f \cdot g) = f \cdot g$, $(f \cdot g)_{1\bullet} = f_{1\bullet} \cdot g_{1\bullet}$ and $(f \cdot g)_1 = f_1 \cdot g_1$. For any multi-opspan $X = (R_0, R_1) = (x_{00}, x_{01}, x_{10}, x_{11})$, there is an *identity* opspan morphism.

```
(20) (UR$object composable-multi-opspan-morphism)

(21) (forall (?z (composable-multi-opspan-morphism ?z))
         (pair ?z))

(22) (forall (?f ?g (composable-multi-opspan-morphism [?f ?g]))
         (and (multi-opspan-morphism ?f) (multi-opspan-morphism ?g)))

(23) (forall (?f (multi-opspan-morphism ?f) ?g (multi-opspan-morphism ?g))
         (<=> (composable-multi-opspan-morphism [?f ?g])
              (= (target ?f) (source ?g))))
```

```
(24) (UR$morphism composition)
     (= (UR$source composition) composable-multi-opspan-morphism)
     (= (UR$target composition) multi-opspan-morphism)

(25) (forall (?f (multi-opspan-morphism ?f) ?g (multi-opspan-morphism ?g)
             (= (target ?f) (source ?g)))
         (and (= (source (composition [?f ?g])) (source ?f))
              (= (target (composition [?f ?g])) (target ?g))
              (= (opspan-morphism0 (composition [?f ?g]))
                 (KIF.DGM.OSPN.MOR$composition [(opspan-morphism0 ?f) (opspan-morphism0 ?g)]))
              (= (opspan-morphism1 (composition [?f ?g]))
                 (KIF.DGM.OSPN.MOR$composition [(opspan-morphism1 ?f) (opspan-morphism1 ?g)]))))

(26) (UR$morphism identity)
     (= (UR$source identity) KIF.DGM.MOSPN.OBJ$multi-opspan)
     (= (UR$target identity) multi-opspan-morphism)
     (= (UR$composition [identity source]) (UR$identity KIF.DGM.MOSPN.OBJ$multi-opspan))
     (= (UR$composition [identity target]) (UR$identity KIF.DGM.MOSPN.OBJ$multi-opspan))
     (= (UR$composition [identity opspan-morphism0])
        (UR$composition [KIF.DGM.MOSPN.OBJ$opspan0 KIF.FTN$identity]))
     (= (UR$composition [identity opspan-morphism1])
        (UR$composition [KIF.DGM.MOSPN.OBJ$opspan1 KIF.FTN$identity]))
```

Composition satisfies the usual *associative law*.

```
(27) (forall (?f (multi-opspan-morphism ?f)
              ?g (multi-opspan-morphism ?g) ?h (multi-opspan-morphism ?h)
             (= (target ?f) (source ?g)) (= (target ?g) (source ?h)))
         (= (composition [?f (composition [?g ?h])])
            (composition [(composition [?f ?g]) ?h])))
```

Identity satisfies the usual *identity laws* with respect to composition.

```
(28) (forall (?f (multi-opspan-morphism ?f))
         (and (= (composition [(identity (source ?f)) ?f]) ?f)
              (= (composition [?f (identity (target ?f))]) ?f)))
```

Finite Limits

KIF.LIM

$$\begin{tikzcd} X_0 \arrow[dr, "!_{X0}"'] & & X_1 \arrow[dl, "!_{X1}"] \\ & \mathbf{1} & \end{tikzcd}$$

Figure 12a: Binary Products in terms of Pullbacks and Terminal Object

$$\begin{tikzcd} X_0 \arrow[dr, "f_0"'] & X_0 \times X_1 \arrow[l, "\pi_0"'] \arrow[r, "\pi_1"] \arrow[d, shift left] \arrow[d, shift right] & X_1 \arrow[dl, "f_1"] \\ & X & \end{tikzcd}$$

Figure 12b: Pullbacks in terms of Binary Products and Equalizers

$$\begin{tikzcd} X \arrow[dr, "\langle 1_X, f_0 \rangle"'] & & X \arrow[dl, "\langle 1_X, f_1 \rangle"] \\ & X \times Y & \end{tikzcd}$$

Figure 12c: Equalizers in (isomorphic) terms of Binary Products and Pullbacks

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 are 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 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 collection 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 collection pair, any pullback cone has an underlying binary product cone, and the pullback of an opspan is a subcollection of the binary product of the collection pair underlying the opspan.

The Terminal Collection

$$\begin{tikzcd} \mathsf{Col} \arrow[d, shift right=2, "\Delta"'] \arrow[d, phantom, "\overset{\delta}{\underset{\pi}{\dashv}}"] \\ \mathsf{1} \arrow[u, shift right=2, "1"'] \end{tikzcd}$$

Figure 13a: Adjunction

$$\frac{X \longrightarrow 1}{\bullet \longrightarrow \bullet} \quad \downarrow \cong \uparrow$$

Figure 13b: Terminal Collection

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

```
(1) (KIF.COL$collection terminal)
    (= terminal KIF.COL$one)
```

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

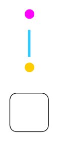


Figure 8.0: Terminal Collection

```
(2) (UR$morphism unique)
    (= (UR$source unique) KIF.COL$collection)
    (= (UR$target unique) KIF.FTN$function)

(3) (forall (?X (KIF.COL$collection ?X))
        (and (= (KIF.FTN$source (unique ?X)) ?X)
             (= (KIF.FTN$target (unique ?X)) terminal)
             (forall (?f (KIF.FTN$function ?f)
                         (= (KIF.FTN$source ?f) ?X)
                         (= (KIF.FTN$target ?f) terminal))
                 (= ?f (unique ?X)))))
```

For any collection C a *global element* (specializing a notion from category theory) is a function $x : 1 \rightarrow C$. The collection 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. At this metalevel, we cannot define the bijective function (part of the isomorphism) that maps the elements of a collection to the associated global elements. However, we do have such element functions at the Upper and Lower metalevels.

```
(4) (UR$morphism global-element)
    (= (UR$source global-element) KIF.COL$collection)
    (= (UR$target global-element) KIF.COL$collection)

(5) (forall (?c (KIF.COL$collection ?c))
        (and (KIF.COL$isomorphic ?c (global-element ?c))
             (forall ( ?x (KIF$thing ?x))
                 (<=> ((global-element ?c) ?x)
                      (and (KIF.FTN$function ?x)
                           (= (KIF.FTN$source ?x) terminal)
                           (= (KIF.FTN$target ?x) ?c))))))
```

Binary Products

KIF.LIM.PRD2.OBJ

$$\begin{tikzcd}
\mathrm{Col} \arrow[d, shift right=2, "\Delta"'] \arrow[d, phantom, "\overset{\delta}{\underset{\pi}{\dashv}}"] \\
\mathrm{Col}^2 \arrow[u, shift right=2, "\times"']
\end{tikzcd}$$

Figure 14a: Adjunction

$$\frac{X \longrightarrow Y_0 \times Y_1}{\Delta(X) \longrightarrow (Y_0, Y_1)} \;\downarrow\cong\uparrow$$

Figure 14b: Binary Product

The essential properties of the binary product operation are illustrated in Figure 14b. The horizontal bar designates a natural bijection between the mediator above the bar and the cone below the bar. A natural *components* (*unpacking*) process assigns a cone $\Delta(X) = (X, X) \rightarrow (Y_0, Y_1)$ below the bar to every mediator $X \rightarrow Y_0 \times Y_1$ above the bar. A natural *pairing* (*packing*) process assigns a mediator $X \rightarrow Y_0 \times Y_1$ above the bar to every cone $(X, X) \rightarrow (Y_0, Y_1)$ below the bar. These two processes are inverse to each other. 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 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).

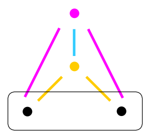


Figure 8.2: Binary Product

The cone Ur object consists of function pairs with constant source collection pair. Binary product cones are treated both abstractly and concretely. Abstractly, a binary product *cone* determines a constrained pair (X, g) consisting of a collection (or vertex) X and a function pair $g : \Delta(X) \rightarrow (Y_0, Y_1)$, where the source of g is the constant collection pair over the collection X. Concretely, a binary product *cone* is a function pair (g_0, g_1), which share a common source collection: $g_0 : X \rightarrow Y_0$ and $g_1 : X \rightarrow Y_1$. Hence, the binary product cone Ur object is a subobject of the function pairs Ur object with two related associated Ur morphisms mapping to the common source collection and representing the cone to function pair inclusion morphism. In applications, such as the IFF Upper Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since

since the pairing function (below) can be concretely referenced as '`(KIF.LIM.PRD2.OBJ$pairing [?g0 ?g1])`' rather than needing to specify a partial function from function pairs to mediators.

```
 (1) (UR$object cone) (UR$object span) (= span cone)
     (UR$subobject cone KIF.DGM.PR.MOR$function-pair)

 (2) (forall (?g (KIF.DGM.PR.MOR$function-pair ?g))
         (<=> (cone ?g)
              (= (KIF.FTN$source (KIF.DGM.PR.MOR$function0 ?g))
                 (KIF.FTN$source (KIF.DGM.PR.MOR$function1 ?g)))))

 (3) (UR$morphism collection) (UR$morphism vertex) (= vertex collection)
     (= (UR$source collection) cone)
     (= (UR$target collection) KIF.COL$collection)

 (4) (UR$morphism function-pair)
     (= (UR$source function-pair) cone)
     (= (UR$target function-pair) KIF.DGM.PR.MOR$function-pair)
     (= function-pair (UR$inclusion [cone KIF.DGM.PR.MOR$function-pair])

 (5) (= (UR$composition [function-pair KIF.DGM.PR.MOR$source])
        (UR$composition [collection constant]))

 (6) (UR$morphism function0)
     (= (UR$source function0) cone)
     (= (UR$target function0) KIF.FTN$function)
     (= function0 (UR$composition [function-pair KIF.DGM.PR.MOR$function0])

 (7) (UR$morphism function1)
     (= (UR$source function1) cone)
     (= (UR$target function1) KIF.FTN$function)
     (= function1 (UR$composition [function-pair KIF.DGM.PR.MOR$function1])
```

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

```
 (8) (UR$object mediator)

 (9) (UR$morphism function)
     (= (UR$source function) mediator)
     (= (UR$target function) KIF.FTN$function)

(10) (UR$morphism collection-pair)
     (= (UR$source collection-pair) mediator)
     (= (UR$target collection-pair) KIF.DGM.PR.OBJ$collection-pair)

(11) (= (UR$composition [collection-pair binary-product])
        (UR$composition [function KIF.FTN$target]))

(12) (UR$morphism collection0)
     (= (UR$source collection0) mediator)
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection0])

(13) (UR$morphism collection1)
     (= (UR$source collection1) mediator)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [collection-pair KIF.DGM.PR.OBJ$collection1])
```

$$\frac{X \xrightarrow{\delta_X} X \times X}{X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X}$$

Figure 15a: Delta Mediator

$$\frac{Y_0 \times Y_1 \xrightarrow{1} Y_0 \times Y_1}{Y_0 \times Y_1 \xrightarrow{\pi_0} Y_0,\ Y_0 \times Y_1 \xrightarrow{\pi_1} Y_1}$$

Figure 15b: Projection Cone

Right adjoint and unit. Given any collection X, the *constant* Ur-morphism constructs the collection pair $\Delta(X) = (X, X)$ that is constantly the given collection. Given any collection X, the *delta* Ur-morphism constructs the mediator (Figure 15a) with 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 collection – the delta mediator is the pairing of two copies of the identity function.

```
(14) (UR$morphism constant)
     (= (UR$source constant) KIF.COL$collection)
     (= (UR$target constant) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$composition [constant KIF.DGM.PR.OBJ$collection0])
        (UR$identity KIF.COL$collection))
     (= (UR$composition [constant KIF.DGM.PR.OBJ$collection1])
        (UR$identity KIF.COL$collection))

(15) (UR$morphism delta-cone)
     (= (UR$source delta-cone) KIF.COL$collection)
     (= (UR$target delta-cone) cone)
     (= (UR$composition [delta-cone collection]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta-cone function0]) KIF.FTN$identity)
     (= (UR$composition [delta-cone function1]) KIF.FTN$identity)

(16) (UR$morphism delta)
     (= (UR$source delta) KIF.COL$collection)
     (= (UR$target delta) mediator)
     (= (UR$composition [delta collection0]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta collection1]) (UR$identity KIF.COL$collection))
     (= delta (UR$composition [delta-cone packing]))
```

Left adjoint and counit. For any collection pair (X_0, X_1), the *binary* Cartesian *product* collection $X_0{\times}X_1$ 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 the quasi-category of collections and their (total) functions. These form a cone that is universal over the given collection pair. Given any collection pair (X_0, X_1), the *limit* Ur-morphism constructs the collection $X_0{\times}X_1$ that is the *binary product* of the given collection pair. And the *projection* Ur-morphism constructs the cone (Figure 15b), which is the components $\pi = 1_{01}$ of the mediator whose function $1_{X0{\times}X1} : X_0{\times}X_1 \rightarrow X_0{\times}X_1$ is the identity function for the binary product of the given collection pair – the projection cone is the components of the product identity. We introduce two convenience terms for the component projections. The last axiom expresses the concreteness of the binary-product and the binary-product projections.

```
(17) (UR$morphism limit) (UR$morphism binary-product) (= binary-product limit)
     (= (UR$source limit) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target limit) KIF.COL$collection)
     (= limit (UR$composition [projection collection]))

(18) (UR$morphism projection-mediator)
     (= (UR$source projection-mediator) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target projection-mediator) mediator)
     (= (UR$composition [projection-mediator function])
        (UR$composition [limit KIF.FTN$identity]))
     (= (UR$composition [projection-mediator collection-pair])
        (UR$identity KIF.DGM.PR.OBJ$collection-pair))

(19) (UR$morphism projection)
     (= (UR$source projection) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target projection) cone)
     (= (UR$composition [projection collection]) limit)
     (= projection (UR$composition [projection-mediator unpacking]))

(20) (UR$morphism projection0)
     (= (UR$source projection0) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target projection0) KIF.FTN$function)
     (= (UR$composition [projection0 KIF.FTN$source]) limit)
     (= (UR$composition [projection0 KIF.FTN$target]) KIF.DGM.PR.OBJ$collection0)
     (= projection0 (UR$composition [projection function0]))

(21) (UR$morphism projection1)
     (= (UR$source projection1) KIF.DGM.PR.OBJ$collection-pair)
     (= (UR$target projection1) KIF.FTN$function)
     (= (UR$composition [projection1 KIF.FTN$source]) limit)
     (= (UR$composition [projection1 KIF.FTN$target]) KIF.DGM.PR.OBJ$collection1)
     (= projection1 (UR$composition [projection function1]))

(22) (forall (?X (KIF.DGM.PR.OBJ$collection-pair ?X)
              ?x (KIF.COL$pair ?x))
         (and (<=> ((limit ?X) ?x)
                   (and (KIF.COL$pair ?x)
                        ((KIF.DGM.PR.OBJ$collection0 ?X) (?x 0))
                        ((KIF.DGM.PR.OBJ$collection1 ?X) (?x 1))))
              (= ((projection0 ?X) ?x) (?x 0))
              (= ((projection1 ?X) ?x) (?x 1))))
```

$$\frac{X \xrightarrow{f} Y_0 \times Y_1}{X \xrightarrow{f \cdot \pi_0} Y_0,\; X \xrightarrow{f \cdot \pi_1} Y_1} \downarrow$$

Figure 16b: unpacking or components

$$\frac{X \xrightarrow{\langle g_0,\, g_1 \rangle} Y_0 \times Y_1}{X \xrightarrow{g_0} Y_0,\; X \xrightarrow{g_1} Y_1} \uparrow$$

Figure 16b: packing or pairing

The components of a mediator unpack it into a cone. For any mediator f with collection 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 $f_{01} = (f_0, f_1) : X \rightarrow (Y_0, Y_1)$, 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 $\pi_{(Y0,\, Y1)}$ (a component of the counit π):

$$f_{01} = (f_0, f_1) = (f, f) \cdot \pi_{(Y0,\, Y1)} = (f \cdot \pi_{Y0}, f \cdot \pi_{Y1}) : (X, X) \rightarrow (Y_0{\times}Y_1,\, Y_0{\times}Y_1) \rightarrow (Y_0, Y_1).$$

The pairing of a cone packs it into a mediator. For any cone g with collection X and function pair $(g_0, g_1) : \Delta(X) = (X, X) \rightarrow (Y_0, Y_1)$, there is a *pairing* mediator $\langle g \rangle = \langle g_0, g_1 \rangle : X \rightarrow Y_0{\times}Y_1$, 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):

$$\langle g \rangle = \langle g_0, g_1 \rangle = \delta_X \cdot (g_0 \times g_1) : X \rightarrow (X \times X) \rightarrow (Y_0 \times Y_1).$$

These two processes are inverse to each other: $\langle g \rangle_{01} = g$ for each cone g and $\langle f_{01} \rangle = f$ for each mediator f.

```
(23) (UR$morphism packing) (UR$morphism pairing) (= pairing packing)
     (= (UR$source packing) cone)
     (= (UR$target packing) mediator)
     (forall (?g (cone ?g))
         (and (= (function (packing ?g))
                 (KIF.FTN$composition
                     [(delta (collection ?g))
                      (KIF.LIM.PRD2.MOR$binary-product (function-pair ?g))]))
              (= (collection-pair (packing ?g))
                 (KIF.DGM.PR.MOR$target (function-pair ?g)))))

(24) (UR$morphism unpacking) (UR$morphism components) (= components unpacking)
     (= (UR$source unpacking) mediator)
     (= (UR$target unpacking) cone)
     (forall (?f (mediator ?f))
         (and (= (collection (unpacking ?f))
                 (KIF.FTN$source (function ?f)))
              (= (function-pair (unpacking ?f))
                 (KIF.DGM.PR.MOR$composition
                     [(KIF.LIM.PRD2.MOR$constant (function ?f))
                      (function-pair (projection (collection-pair ?f)))]))))

(25) (= (UR$composition [packing unpacking]) (UR$identity cone))
     (= (UR$composition [unpacking packing]) (UR$identity mediator))
```

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

```
(26)  (= binary-product (UR$composition [KIF.DGM.PR.OBJ$opspan KIF.LIM.PBK.OBJ$pullback]))
      (= projection0 (UR$composition [KIF.DGM.PR.OBJ$opspan KIF.LIM.PBK.OBJ$projection0]))
      (= projection1 (UR$composition [KIF.DGM.PR.OBJ$opspan KIF.LIM.PBK.OBJ$projection1]))
      (= pairing (UR$composition [KIF.DGM.PR.OBJ$opspan KIF.LIM.PBK$.OBJpairing])
```

Binary Product Morphisms

`KIF.LIM.PRD2.MOR`

$$\begin{tikzcd}
X_0 \times X_1 \arrow[r, "g_0 \times g_1"] \arrow[d, "\pi_{Xk}"'] \arrow[dr, phantom, "k = 0{,}1"] & Y_0 \times Y_1 \arrow[d, "\pi_{Yk}"] \\
X_k \arrow[r, "g_k"'] & Y_k
\end{tikzcd}$$

Figure 17: Binary Product Function

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: $(g_0 \times g_1) \cdot \pi_{Y0} = \pi_{X0} \cdot g_0$ and $(g_0 \times g_1) \cdot \pi_{Y1} = \pi_{X1} \cdot g_1$ (Figure 17).

```
 (1)  (UR$morphism constant)
      (= (UR$source constant) KIF.FTN$function)
      (= (UR$target constant) KIF.DGM.PR.MOR$function-pair)
      (= (UR$composition [constant KIF.DGM.PR.MOR$source])
         (UR$composition [KIF.FTN$source KIF.LIM.PRD2.OBJ$constant]))
      (= (UR$composition [constant KIF.DGM.PR.MOR$target])
         (UR$composition [KIF.FTN$target KIF.LIM.PRD2.OBJ$constant]))
      (= (UR$composition [constant KIF.DGM.PR.MOR$function0])
         (UR$identity KIF.FTN$function))
      (= (UR$composition [constant KIF.DGM.PR.MOR$function1])
         (UR$identity KIF.FTN$function))

 (2)  (UR$morphism limit) (UR$morphism binary-product) (= binary-product limit)
      (= (UR$source limit) KIF.DGM.PR.MOR$function-pair)
      (= (UR$target limit) KIF.FTN$function)
      (= (UR$composition [limit KIF.FTN$source])
         (UR$composition [KIF.DGM.PR.MOR$source KIF.LIM.PRD2.OBJ$limit]))
      (= (UR$composition [limit KIF.FTN$target])
         (UR$composition [KIF.DGM.PR.MOR$target KIF.LIM.PRD2.OBJ$limit]))
      (forall (?g0 ?g1 (KIF.DGM.PR.MOR$function-pair [?g0 ?g1]))
          (and (= (KIF.FTN$composition
                      [(limit [?g0 ?g1])
                       (KIF.LIM.PRD2.OBJ$projection0 (KIF.DGM.PR.MOR$target [?g0 ?g1]))])
                  (KIF.FTN$composition
                      [(KIF.LIM.PRD2.OBJ$projection0 (KIF.DGM.PR.MOR$source [?g0 ?g1])) ?g0]))
               (= (KIF.FTN$composition
                      [(limit [?g0 ?g1])
                       (KIF.LIM.PRD2.OBJ$projection1 (KIF.DGM.PR.MOR$target [?g0 ?g1]))])
                  (KIF.FTN$composition
                      [(KIF.LIM.PRD2.OBJ$projection1 (KIF.DGM.PR.MOR$source [?g0 ?g1])) ?g0]))))
```

Ternary Products

`KIF.LIM.PRD3.OBJ`

$$\begin{tikzcd}
\mathrm{Col} \arrow[d, shift right=3, "\Delta"'] \arrow[d, phantom, "\overset{\delta}{\underset{\pi}{\dashv}}"] \\
\mathrm{Col}^3 \arrow[u, shift right=3, "\times"']
\end{tikzcd}$$

Figure 18a: Adjunction

$$\frac{X \longrightarrow Y_0 \times Y_1 \times Y_2}{\Delta(X) \longrightarrow (Y_0,\, Y_1,\, Y_2)} \;\; \downarrow \cong \uparrow$$

Figure 18b: Ternary Product

The essential properties of the ternary product operation are illustrated in Figure 18b. The horizontal bar designates a natural bijection between the mediator above the bar and the cone below the bar. A natural *components* (*unpacking*) process assigns a cone $\Delta(X) = (X, X) \rightarrow (Y_0, Y_1, Y_2)$ below the bar to every mediator $X \rightarrow Y_0 \times Y_1 \times Y_2$ above the bar. A natural *tripling* (*packing*) process assigns a mediator $X \rightarrow Y_0 \times Y_1 \times Y_2$ above the bar to every cone $(X, X) \rightarrow (Y_0, Y_1, Y_2)$ below the bar. These two processes are inverse to each other. The components process is realized by application of the constant (Δ) operation and then composition on the right with the projection function triple (a component of the counit π). The tripling 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).

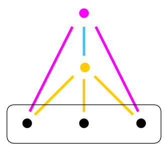


Figure 8.3: Ternary Product

The cone Ur object consists of function triples with constant source collection triple. Ternary product cones are treated both abstractly and concretely. Abstractly, a ternary product *cone* determines a constrained pair (X, g) consisting of a collection (or vertex) X and a function triple $g : \Delta(X) = (X, X, X) \rightarrow (Y_0, Y_1, Y_2)$, where the source of g is the constant collection triple over the collection X. Concretely, a ternary product *cone* is a function triple (g_0, g_1, g_2), which share a common source collection: $g_0 : X \rightarrow Y_0$, $g_1 : X \rightarrow Y_1$ and $g_2 : X \rightarrow Y_2$. Hence, the ternary product cone Ur object is a subobject of the function triples Ur object with two related associated morphisms mapping to the common source collection and representing the cone to function triple inclusion morphism. In applications, such as the IFF Upper Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since the tripling function (below) can be concretely referenced as '`(KIF.LIM.PRD3.OBJ$tripling [?g0 ?g1 ?g2])`' rather than needing to specify a partial function from function triples to mediators.

```
(1)  (UR$object cone)

(2)  (UR$morphism collection) (UR$morphism vertex) (= vertex collection)
     (= (UR$source collection) cone)
     (= (UR$target collection) KIF.COL$collection)

(3)  (UR$morphism function-triple)
     (= (UR$source function-triple) cone)
     (= (UR$target function-triple) KIF.DGM.TRP.MOR$function-triple)

(4)  (= (UR$composition [function-triple KIF.DGM.TRP.MOR$source])
        (UR$composition [collection constant]))

(5)  (UR$morphism function0)
     (= (UR$source function0) cone)
     (= (UR$target function0) KIF.FTN$function)
     (= function0 (UR$composition [function-triple KIF.DGM.TRP.MOR$function0]))

(6)  (UR$morphism function1)
     (= (UR$source function1) cone)
     (= (UR$target function1) KIF.FTN$function)
     (= function1 (UR$composition [function-triple KIF.DGM.TRP.MOR$function1]))

(7)  (UR$morphism function2)
     (= (UR$source function2) cone)
     (= (UR$target function2) KIF.FTN$function)
     (= function2 (UR$composition [function-triple KIF.DGM.TRP.MOR$function2]))
```

The *mediator* collection consists of functions with product target. A mediator consists of a function $f : X \rightarrow Y_0 \times Y_1 \times Y_2$ and a collection triple (Y_0, Y_1, Y_2) whose ternary product is the target of the function. We introduce three convenience terms for the component collections Y_0, Y_1 and Y_2.

```
(8)  (UR$object mediator)

(9)  (UR$morphism function)
     (= (UR$source function) mediator)
     (= (UR$target function) KIF.FTN$function)

(10) (UR$morphism collection-triple)
     (= (UR$source collection-triple) mediator)
     (= (UR$target collection-triple) KIF.DGM.TRP.OBJ$collection-triple)
```

```
(11) (= (UR$composition [collection-triple ternary-product])
        (UR$composition [function KIF.FTN$target]))

(12) (UR$morphism collection0)
     (= (UR$source collection0) mediator)
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection0]))

(13) (UR$morphism collection1)
     (= (UR$source collection1) mediator)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection1]))

(14) (UR$morphism collection2)
     (= (UR$source collection2) mediator)
     (= (UR$target collection2) KIF.COL$collection)
     (= collection2 (UR$composition [collection-triple KIF.DGM.TRP.OBJ$collection2]))
```

$$\frac{X \xrightarrow{\delta_X} X \times X \times X}{X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X}$$

Figure 19a: Delta Mediator

$$\frac{Y_0 \times Y_1 \times Y_2 \xrightarrow{1} Y_0 \times Y_1 \times Y_2}{Y_0 \times Y_1 \times Y_2 \xrightarrow{\pi_0} Y_0,\ Y_0 \times Y_1 \times Y_2 \xrightarrow{\pi_1} Y_1,\ Y_0 \times Y_1 \times Y_2 \xrightarrow{\pi_2} Y_2}$$

Figure 19b: Projection Cone

Right adjoint and unit. Given any collection X, the *constant* Ur-morphism constructs the collection triple $\Delta(X) = (X, X, X)$ that is constantly the given collection. Given any collection X, the *delta* Ur-morphism constructs the mediator (Figure 19a) with function $\delta_X : X \rightarrow (X \times X \times X)$, which is the tripling of the cone whose function triple $(1_X, 1_X, 1_X) : (X, X, X) \rightarrow (X, X, X)$ consists of the identity function for the given collection – the delta mediator is the tripling of three copies of the identity function.

```
(15) (UR$morphism constant)
     (= (UR$source constant) KIF.COL$collection)
     (= (UR$target constant) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$composition [constant KIF.DGM.TRP.OBJ$collection0])
        (UR$identity KIF.COL$collection))
     (= (UR$composition [constant KIF.DGM.TRP.OBJ$collection1])
        (UR$identity KIF.COL$collection))
     (= (UR$composition [constant KIF.DGM.TRP.OBJ$collection2])
        (UR$identity KIF.COL$collection))

(16) (UR$morphism delta-cone)
     (= (UR$source delta-cone) KIF.COL$collection)
     (= (UR$target delta-cone) cone)
     (= (UR$composition [delta-cone collection]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta-cone function0]) KIF.FTN$identity)
     (= (UR$composition [delta-cone function1]) KIF.FTN$identity)
     (= (UR$composition [delta-cone function2]) KIF.FTN$identity)

(17) (UR$morphism delta)
     (= (UR$source delta) KIF.COL$collection)
     (= (UR$target delta) mediator)
     (= (UR$composition [delta collection0]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta collection1]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta collection2]) (UR$identity KIF.COL$collection))
     (= delta (UR$composition [delta-cone packing]))
```

Left adjoint and counit. For any collection triple (X_0, X_1, X_2), the *ternary* Cartesian *product* collection $X_0{\times}X_1{\times}X_2$ and *ternary product projection* functions $\pi_0 : X_0{\times}X_1{\times}X_2 \rightarrow X_0$, $\pi_1 : X_0{\times}X_1{\times}X_2 \rightarrow X_1$ and $\pi_2 : X_0{\times}X_1{\times}X_2 \rightarrow X_2$ are defined. This is the ternary product in the quasi-category of collections and their (total) functions. These form a cone that is universal over the given collection triple. Given any collection triple (X_0, X_1, X_2), the *limit* Ur-morphism constructs the collection $X_0{\times}X_1{\times}X_2$ that is the *ternary product* of the given collection triple. And the *projection* Ur-morphism constructs the cone (Figure 10b), which is the components $\pi = 1_{012}$ of the mediator whose function $1_{X0{\times}X1{\times}X2} : X_0{\times}X_1{\times}X_2 \rightarrow X_0{\times}X_1{\times}X_2$ is the identity function for the ternary product of the given collection triple – the projection cone is the components of the product identity. We introduce three convenience terms for the component projections. The last axiom expresses the concreteness of the ternary-product and the ternary-product projections.

```
(18) (UR$morphism limit) (UR$morphism ternary-product) (= ternary-product limit)
     (= (UR$source limit) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target limit) KIF.COL$collection)
     (= limit (UR$composition [projection collection]))

(19) (UR$morphism projection-mediator)
     (= (UR$source projection-mediator) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target projection-mediator) mediator)
     (= (UR$composition [projection-mediator function])
        (UR$composition [limit KIF.FTN$identity]))
     (= (UR$composition [projection-mediator collection-triple])
        (UR$identity KIF.DGM.TRP.OBJ$collection-triple))

(20) (UR$morphism projection)
     (= (UR$source projection) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target projection) cone)
     (= (UR$composition [projection collection]) limit)
     (= projection (UR$composition [projection-mediator unpacking]))

(21) (UR$morphism projection0)
     (= (UR$source projection0) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target projection0) KIF.FTN$function)
     (= (UR$composition [projection0 KIF.FTN$source]) limit)
     (= (UR$composition [projection0 KIF.FTN$target]) KIF.DGM.TRP.OBJ$collection0)
     (= projection0 (UR$composition [projection function0]))

(22) (UR$morphism projection1)
     (= (UR$source projection1) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target projection1) KIF.FTN$function)
     (= (UR$composition [projection1 KIF.FTN$source]) limit)
     (= (UR$composition [projection1 KIF.FTN$target]) KIF.DGM.TRP.OBJ$collection1)
     (= projection1 (UR$composition [projection function1]))

(23) (UR$morphism projection2)
     (= (UR$source projection2) KIF.DGM.TRP.OBJ$collection-triple)
     (= (UR$target projection2) KIF.FTN$function)
     (= (UR$composition [projection2 KIF.FTN$source]) limit)
     (= (UR$composition [projection2 KIF.FTN$target]) KIF.DGM.TRP.OBJ$collection2)
     (= projection2 (UR$composition [projection function2]))

(24) (forall (?X (KIF.DGM.TRP.OBJ$collection-triple ?X)
              ?x (KIF.COL$triple ?x))
         (and (<=> ((limit ?X) ?x)
                   (and (KIF.COL$triple ?x)
                        ((KIF.DGM.TRP.OBJ$collection0 ?X) (?x 0))
                        ((KIF.DGM.TRP.OBJ$collection1 ?X) (?x 1))
                        ((KIF.DGM.TRP.OBJ$collection2 ?X) (?x 2))))
              (= ((projection0 ?X) ?x) (?x 0))
              (= ((projection1 ?X) ?x) (?x 1))
              (= ((projection2 ?X) ?x) (?x 2))))
```

$$\frac{X \xrightarrow{f} Y_0 \times Y_1 \times Y_2}{X \xrightarrow{f \cdot \pi_0} Y_0,\ X \xrightarrow{f \cdot \pi_1} Y_1,\ X \xrightarrow{f \cdot \pi_2} Y_2}\ \downarrow$$

Figure 20b: unpacking or components

$$\frac{X \xrightarrow{\langle g_0,\, g_1,\, g_2 \rangle} Y_0 \times Y_1 \times Y_2}{X \xrightarrow{g_0} Y_0,\ X \xrightarrow{g_1} Y_1,\ X \xrightarrow{g_2} Y_2}\ \uparrow$$

Figure 20b: packing or tripling

The components of a mediator unpack it into a cone. For any mediator f with collection triple (Y_0, Y_1, Y_2) and function $f : X \rightarrow Y_0{\times}Y_1{\times}Y_2 = {\times}(Y_0, Y_1, Y_2)$, there is a *components* cone $f_{012} = (f_0, f_1, f_2) : X \rightarrow (Y_0, Y_1, Y_2)$, which separates the mediator into three 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 triple of the projection cone $\pi_{(Y0,\ Y1,\ Y2)}$ (a component of the counit π):

$f_{012} = (f_0, f_1, f_2) = (f, f, f) \cdot \pi_{(Y0,\, Y1\, Y2)}$
$= (f \cdot \pi_{Y0}, f \cdot \pi_{Y1}, f \cdot \pi_{Y2}) : (X, X, X) \rightarrow (Y_0{\times}Y_1{\times}Y_2,\ Y_0{\times}Y_1{\times}Y_2,\ Y_0{\times}Y_1{\times}Y_2) \rightarrow (Y_0, Y_1, Y_2)$.

The tripling of a cone packs it into a mediator. For any cone g with collection X and function triple $(g_0, g_1, g_2) : \Delta(X) = (X, X, X) \rightarrow (Y_0, Y_1, Y_2)$, there is a *tripling* mediator $\langle g\rangle = \langle g_0, g_1, g_2\rangle : X \rightarrow Y_0{\times}Y_1{\times}Y_2$, which triples the images of the original three functions. The packing (tripling) process is realized by application of the product (×) operator to the function triple (g_0, g_1, g_2) and then composition on the left with the delta function δ_X (a component of the unit δ):

$\langle g\rangle = \langle g_0, g_1, g_2\rangle = \delta_X \cdot (g_0 \times g_1 \times g_2) : X \rightarrow (X \times X \times X) \rightarrow (Y_0 \times Y_1 \times Y_2)$.

These two processes are inverse to each other: $\langle g\rangle_{012} = g$ for each cone g and $\langle f_{012}\rangle = f$ for each mediator f.

```
(25) (UR$morphism packing) (UR$morphism tripling) (= tripling packing)
     (= (UR$source packing) cone)
     (= (UR$target packing) mediator)
     (forall (?g (cone ?g))
         (and (= (function (packing ?g))
                 (KIF.FTN$composition
                     [(delta (collection ?g))
                      (KIF.LIM.PRD3.MOR$ternary-product (function-triple ?g))]))
              (= (collection-triple (packing ?g))
                 (KIF.DGM.TRP.MOR$target (function-triple ?g)))))

(26) (UR$morphism unpacking) (UR$morphism components) (= components unpacking)
     (= (UR$source unpacking) mediator)
     (= (UR$target unpacking) cone)
     (forall (?f (mediator ?f))
         (and (= (collection (unpacking ?f))
                 (KIF.FTN$source (function ?f)))
              (= (function-triple (unpacking ?f))
                 (KIF.DGM.TRP.MOR$composition
                     [(KIF.LIM.PRD3.MOR$constant (function ?f))
                      (function-triple (projection (collection-triple ?f)))]))))

(27) (= (UR$composition [packing unpacking]) (UR$identity cone))
     (= (UR$composition [unpacking packing]) (UR$identity mediator))
```

Ternary Product Morphisms

KIF.LIM.PRD3.MOR

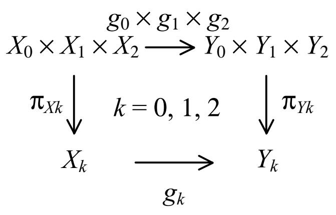

$$\begin{tikzcd} X_0 \times X_1 \times X_2 \arrow[r, "g_0 \times g_1 \times g_2"] \arrow[d, "\pi_{Xk}"'] \arrow[dr, phantom, "k = 0,\,1,\,2"] & Y_0 \times Y_1 \times Y_2 \arrow[d, "\pi_{Yk}"] \\ X_k \arrow[r, "g_k"'] & Y_k \end{tikzcd}$$

Figure 21: Ternary Product Function

The constant and ternary product operations are extended to morphisms. Given any function $f : X \rightarrow Y$, there is a constant function triple $(f, f, f) : (X, X, X) \rightarrow (Y, Y, Y)$, whose component functions are all f. Given any function triple $(g_0, g_1, g_2) : (X_0, X_1, X_2) \rightarrow (Y_0, Y_1, Y_2)$, there is a ternary product function $(g_0 \times g_1 \times g_2) : (X_0 \times X_1 \times X_2) \rightarrow (Y_0 \times Y_1 \times Y_2)$ defined using projections: $(g_0 \times g_1 \times g_2) \cdot \pi_{Y0} = \pi_{X0} \cdot g_0$, $(g_0 \times g_1 \times g_2) \cdot \pi_{Y1} = \pi_{X1} \cdot g_1$ and $(g_0 \times g_1 \times g_2) \cdot \pi_{Y2} = \pi_{X2} \cdot g_2$ (Figure 21).

```
 (1) (UR$morphism constant)
     (= (UR$source constant) KIF.FTN$function)
     (= (UR$target constant) KIF.DGM.TRP.MOR$function-triple)
     (= (UR$composition [constant KIF.DGM.TRP.MOR$source])
        (UR$composition [KIF.FTN$source KIF.LIM.PRD3.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.TRP.MOR$target])
        (UR$composition [KIF.FTN$target KIF.LIM.PRD3.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.TRP.MOR$function0])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [constant KIF.DGM.TRP.MOR$function1])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [constant KIF.DGM.TRP.MOR$function2])
        (UR$identity KIF.FTN$function))

 (2)  (UR$morphism limit) (UR$morphism ternary-product) (= ternary-product limit)
     (= (UR$source limit) KIF.DGM.TRP.MOR$function-triple)
     (= (UR$target limit) KIF.FTN$function)
     (= (UR$composition [limit KIF.FTN$source])
        (UR$composition [KIF.DGM.TRP.MOR$source KIF.LIM.PRD3.OBJ$limit]))
     (= (UR$composition [limit KIF.FTN$target])
        (UR$composition [KIF.DGM.TRP.MOR$target KIF.LIM.PRD3.OBJ$limit]))
     (forall (?g0 ?g1 ?g2 (KIF.DGM.TRP.MOR$function-triple [?g0 ?g1 ?g2]))
         (and (= (KIF.FTN$composition
                     [(limit [?g0 ?g1 ?g2])
                      (KIF.LIM.PRD3.OBJ$projection0 (KIF.DGM.TRP.MOR$target [?g0 ?g1 ?g2]))])
                 (KIF.FTN$composition
                     [(KIF.LIM.PRD3.OBJ$projection0 (KIF.DGM.TRP.MOR$source [?g0 ?g1 ?g2])) ?g0]))
              (= (KIF.FTN$composition
                     [(limit [?g0 ?g1 ?g2])
                      (KIF.LIM.PRD3.OBJ$projection1 (KIF.DGM.TRP.MOR$target [?g0 ?g1 ?g2]))])
                 (KIF.FTN$composition
                     [(KIF.LIM.PRD3.OBJ$projection1 (KIF.DGM.TRP.MOR$source [?g0 ?g1 ?g2])) ?g1]))
              (= (KIF.FTN$composition
                     [(limit [?g0 ?g1 ?g2])
                      (KIF.LIM.PRD3.OBJ$projection2 (KIF.DGM.TRP.MOR$target [?g0 ?g1 ?g2]))])
                 (KIF.FTN$composition
                     [(KIF.LIM.PRD3.OBJ$projection2 (KIF.DGM.TRP.MOR$source [?g0 ?g1 ?g2])) ?g2]))))
```

Equalizers

KIF.LIM.EQU.OBJ

$$\begin{tikzcd} \mathrm{Col} \arrow[d, shift right=3, "\Delta"'] \arrow[d, phantom, "\overset{\delta}{\underset{\pi}{\dashv}}"] \\ \mathrm{Ppr} \arrow[u, shift right=3, "="'] \end{tikzcd}$$

Figure 22a: Adjunction

$$\frac{X \longrightarrow (y_0 = y_1)}{\Delta(X) \longrightarrow (y_0, y_1)} \quad \downarrow\cong\uparrow$$

Figure 22b: Equalizer

The essential properties of the equalizer operation are illustrated in Figure 22b. The horizontal bar designates a natural bijection between the mediator above the bar and the cone below the bar. A natural *unpacking* process assigns a cone $\Delta(X) = (1_X, 1_X) \rightarrow (y_0, y_1)$ below the bar to every mediator $X \rightarrow (y_0 = y_1)$ above the bar. A natural *packing* process assigns a mediator $X \rightarrow (y_0 = y_1)$ above the bar to every cone $(1_X, 1_X) \rightarrow (y_0, y_1)$ below the bar. These two processes are inverse to each other. The unpacking process is realized by application of the constant (Δ) operation and then composition on the right with the projection parallel pair morphism (a component of the counit π). The packing 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).

$$\begin{tikzcd} X \arrow[r, "g_0"] \arrow[d, shift right, "1_X"'] \arrow[d, shift left, "1_X"] & Y_0 \arrow[d, shift right, "y_0"'] \arrow[d, shift left, "y_1"] \\ X \arrow[r, "g_1"'] & Y_1 \end{tikzcd}$$
$$= g_0 \cdot y_0$$
$$= g_0 \cdot y_1$$

Figure 23: Cone

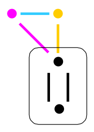


Figure 8.=: Equalizer

The cone Ur object consists of parallel pair morphisms with constant source parallel pairs. Equalizer cones are treated both abstractly and concretely. Abstractly, an equalizer *cone* determines a constrained pair (X, g) consisting of a collection (or vertex) X and a parallel pair morphism $g : \Delta(X) = (1_X, 1_X) \rightarrow (y_0, y_1)$, where the source of g is the constant parallel pair over the collection X. By definition of such a morphism, the first component function $g_1 : X \rightarrow Y_1$ is redundant – it is the composition of the zeroth component function $g_0 : X \rightarrow Y_0$ with either function component of the target parallel pair (Figure 23). Concretely, an equalizer *cone* is a function pair (g_0, g_1), which share a common source collection: $g_0 : X \rightarrow Y_0$ and $g_1 : X \rightarrow Y_1$ and commute with the function components of the target of the associated parallel pair morphism. Hence, the pullback cone Ur object is a subobject of the function pairs Ur object with two related associated Ur morphisms mapping to the common source collection and to a parallel pair morphism over that function pair. In applications, such as the IFF Upper Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since the packing function (below) can be concretely referenced as '`(KIF.LIM.EQU.OBJ$packing [?g0 ?g1])`' rather than needing to specify a partial function from function pairs to mediators.

```
 (1) (UR$object cone)
     (UR$subobject cone KIF.DGM.PR.MOR$function-pair)

 (2) (UR$morphism collection) (UR$morphism vertex) (= vertex collection)
     (= (UR$source collection) cone)
     (= (UR$target collection) KIF.COL$collection)

 (3) (UR$morphism parallel-pair-morphism)
     (= (UR$source parallel-pair-morphism) cone)
     (= (UR$target parallel-pair-morphism) KIF.DGM.PPR.MOR$parallel-pair-morphism)

 (4) (= (UR$composition [parallel-pair-morphism KIF.DGM.PR.MOR$source])
        (UR$composition [collection constant]))

 (5) (UR$morphism function-pair)
     (= (UR$source function-pair) cone)
     (= (UR$target function-pair) KIF.DGM.PR.MOR$function-pair)
     (= function-pair
        (UR$composition [parallel-pair-morphism KIF.DGM.PPR.MOR$function-pair])
     (= function-pair (UR$inclusion [cone KIF.DGM.PR.MOR$function-pair]))

 (6) (UR$morphism function0)
     (= (UR$source function0) cone)
     (= (UR$target function0) KIF.FTN$function)
     (= function0 (UR$composition [parallel-pair-morphism KIF.DGM.PPR.MOR$function0])
```

An equalizer *mediator* consists of a function with equalizer target. In more detail, a mediator consists of a function $f : X \rightarrow (y_0 = y_1)$ and a parallel pair (of functions) (y_0, y_1) whose equalizer is the target of the function.

```
 (7) (UR$object mediator)

 (8) (UR$morphism function)
     (= (UR$source function) mediator)
     (= (UR$target function) KIF.FTN$function)

 (9) (UR$morphism parallel-pair)
     (= (UR$source parallel-pair) mediator)
     (= (UR$target parallel-pair) KIF.DGM.PPR.OBJ$parallel-pair)

(10) (= (UR$composition [parallel-pair equalizer])
        (UR$composition [function KIF.FTN$target]))
```

$$\frac{X \xrightarrow{\delta_X = 1_X} (1_X = 1_X) = X}{X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X}$$

Figure 24a: Delta Mediator

$$\frac{(y_0 = y_1) \xrightarrow{1} (y_0 = y_1)}{(y_0 = y_1) \xrightarrow{\pi} Y_0,\ (y_0 = y_1) \xrightarrow{\pi \cdot y_0 = \pi \cdot y_1} Y_1}$$

Figure 24b: Projection Cone

Right adjoint and unit. Given any collection X, the *constant* Ur-morphism constructs the parallel pair $\Delta(X) = (1_X, 1_X)$ that is constantly the given collection. Given any collection X, the *delta* Ur-morphism constructs the mediator (Figure 24a) with function $\delta_X = 1_X : X \rightarrow (1_X = 1_X) = X$, which is the pairing of the cone whose parallel pair morphism $(1_X, 1_X) : (1_X, 1_X) \rightarrow (1_X, 1_X)$ consists of the identity function for the given collection – the delta mediator is the packing of two copies of the identity function.

```
(11) (UR$morphism constant)
     (= (UR$source constant) KIF.COL$collection)
     (= (UR$target constant) KIF.DGM.PPR.OBJ$parallel-pair)
     (= (UR$composition [constant KIF.DGM.PPR.OBJ$collection0])
        (UR$identity KIF.COL$collection))
     (= (UR$composition [constant KIF.DGM.PPR.OBJ$collection1])
        (UR$identity KIF.COL$collection))
     (= (UR$composition [constant KIF.DGM.PPR.OBJ$function0]) KIF.FTN$identity)
     (= (UR$composition [constant KIF.DGM.PPR.OBJ$function1]) KIF.FTN$identity)

(12)  (UR$morphism delta-cone)
     (= (UR$source delta-cone) KIF.COL$collection)
     (= (UR$target delta-cone) cone)
     (= (UR$composition [delta-cone collection]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta-cone function0]) KIF.FTN$identity)

(13)  (UR$morphism delta)
     (= (UR$source delta) KIF.COL$collection)
     (= (UR$target delta) mediator)
     (= (UR$composition [delta function]) KIF.FTN$identity)
     (= (UR$composition [delta parallel-pair]) constant)
     (= delta (UR$composition [delta-cone packing]))
```

Left adjoint and counit. For any parallel pair (x_0, x_1), the *equalizer* collection $(x_0 = x_1)$ and *equalizer part* $\pi : (x_0 = x_1) \rightarrow X_0$ are defined. This is the equalizer in the quasi-category of collections and their (total) functions. These form a cone that is universal over the given parallel pair. Given any parallel pair (x_0, x_1), the *limit* Ur-morphism constructs the collection $(x_0 = x_1)$ that is the *equalizer* of the given parallel pair. And the *projection* Ur-morphism constructs the cone (Figure 24b), which is the unpacking $\pi = 1_{01}$ of the mediator whose function $1 : (x_0 = x_1) \rightarrow (x_0 = x_1)$ is the identity function for the equalizer of the given parallel pair – the projection cone is the unpacking of the equalizer identity. We introduce the equalizer part as a convenience term for the zeroth component projection. The last axiom expresses the concreteness of the equalizer and the equalizer projections.

```
(14)  (UR$morphism limit) (UR$morphism equalizer) (= equalizer limit)
      (= (UR$source limit) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$target limit) KIF.COL$collection)
      (= limit (UR$composition [projection collection]))

(15)  (UR$morphism projection-mediator)
      (= (UR$source projection-mediator) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$target projection-mediator) mediator)
      (= (UR$composition [projection-mediator function])
         (UR$composition [limit KIF.FTN$identity]))
      (= (UR$composition [projection-mediator parallel-pair])
         (UR$identity KIF.DGM.PR.OBJ$collection-pair))

(16)  (UR$morphism projection)
      (= (UR$source projection) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$target projection) cone)
      (= (UR$composition [projection collection]) limit)
      (= projection (UR$composition [projection-mediator unpacking]))

(17)  (UR$morphism part)
      (= (UR$source part) KIF.DGM.PPR.OBJ$parallel-pair)
      (= (UR$target part) KIF.FTN$function)
      (= (UR$composition [part KIF.FTN$source]) limit)
      (= (UR$composition [part KIF.FTN$target]) KIF.DGM.PPR.OBJ$collection0)
      (= part (UR$composition [projection function0]))

(18)  (forall (?X (KIF.DGM.PPR.OBJ$parallel-pair ?X))
          (and (KIF.COL$subcollection (limit ?X) (KIF.DGM.PPR.OBJ$collection0 ?X))
               (= (part ?X)
                  (KIF.FTN$inclusion [(limit ?X) (KIF.DGM.PPR.OBJ$collection0 ?X)]))
               (forall (?x ((KIF.DGM.PPR.OBJ$collection0 ?X) ?x))
                   (and (<=> ((limit ?X) ?x)
                             (= ((KIF.DGM.PPR.OBJ$function0 ?X) ?x)
                                ((KIF.DGM.PPR.OBJ$function1 ?X) ?x)))
                        (= ((part ?X) ?x) ?x)))))
```

$$\begin{array}{c} X \xrightarrow{f} (y_0 = y_1) \\ \hline X \xrightarrow{f \cdot \pi} Y_0,\ X \xrightarrow{f \cdot \pi \cdot y_0 = f \cdot \pi \cdot y_1} Y_1 \end{array} \downarrow$$

Figure 25b: unpacking

$$\begin{array}{c} X \xrightarrow{\langle g_0, g_1 \rangle} (y_0 = y_1) \\ \hline X \xrightarrow{g_0} Y_0,\ X \xrightarrow{g_0 \cdot y_0 = g_0 \cdot y_1} Y_1 \end{array} \uparrow$$

Figure 25b: packing

The unpacking of a mediator unpacks it into a cone. For any mediator with parallel pair (y_0, y_1) and function $f : X \rightarrow (y_0 = y_1)$, there is an *unpacking* cone $f_{01} = (f_0, f_1) : X \rightarrow (y_0, y_1)$, which separates the mediator into two component functions. The unpacking process is realized by application of the constant (Δ) operator to the function f and then composition on the right with the parallel pair morphism of the projection cone $\pi_{(y0,\,y1)}$ (a component of the counit π):

$$f_{01} = (f_0, f_1) = (f, f) \cdot \pi_{(y0,\,y1)} = (f \cdot \pi, f \cdot \pi \cdot y_0 = f \cdot \pi \cdot y_1) : (1_X, 1_X) \rightarrow ((y_0 = y_1), (y_0 = y_1)) \rightarrow (y_0, y_1).$$

The packing of a cone packs it into a mediator. For any cone g with collection X and parallel pair morphism $(g_0, g_1) : \Delta(X) = (1_X, 1_X) \rightarrow (y_0, y_1)$, there is a *packing* mediator $\langle g \rangle = \langle g_0, g_1 \rangle : X \rightarrow (y_0 = y_1)$, which restricts to the part where the original two functions are equal. The packing process is realized by application of the equalizer (=) operator to the parallel pair morphism (g_0, g_1):

$$\langle g \rangle = \langle g_0, g_1 \rangle = \delta_X \cdot (g_0 = g_1) = (g_0 = g_1) : X \rightarrow X = (1_X = 1_X) \rightarrow (y_0 = y_1).$$

These two processes are inverse to each other: $\langle g \rangle_{01} = g$ for each cone g and $\langle f_{01} \rangle = f$ for each mediator f.

```
(19) (UR$morphism packing)
     (= (UR$source packing) cone)
     (= (UR$target packing) mediator)
     (forall (?g (cone ?g))
         (and (= (function (packing ?g))
                 (KIF.LIM.EQU.MOR$equalizer (parallel-pair-morphism ?g)))
              (= (parallel-pair (packing ?g))
                 (KIF.DGM.PPR.MOR$target (parallel-pair-morphism ?g)))))

(20) (UR$morphism unpacking)
     (= (UR$source unpacking) mediator)
     (= (UR$target unpacking) cone)
     (forall (?f (mediator ?f))
         (and (= (collection (unpacking ?f))
                 (KIF.FTN$source (function ?f)))
              (= (function0 (unpacking ?f))
                 (KIF.FTN$composition [(function ?f) (part (parallel-pair ?f))]))
              (= (parallel-pair-morphism (unpacking ?f))
                 (KIF.DGM.PPR.MOR$composition
                     [(KIF.LIM.EQU.MOR$constant (function ?f))
                      (projection (parallel-pair ?f))]))))

(21) (= (UR$composition [packing unpacking]) (UR$identity cone))
     (= (UR$composition [unpacking packing]) (UR$identity mediator))
```

Using the opspan of a parallel pair, we can show that the notion of an equalizer can be based (up to isomorphism) upon binary products and pullbacks. We do this by proving the theorem that the pullback of this opspan is isomorphic to the equalizer collection, and the zeroth pullback projection is isomorphic to the equalizer part.

```
(22) (forall (?X (parallel-pair ?X))
         (KIF.COL$isomorphic
             (equalizer ?X)
             (KIF.LIM.PBK.OBJ$pullback (KIF.DGM.PPR.OBJ$opspan ?X))))
```

Equalizer Morphisms

KIF.LIM.EQU.MOR

$$\begin{tikzcd}
X = (x_0 = x_1) \arrow[r, "(g_0{=}g_1)"] \arrow[d, "\pi_X"'] & (y_0 = y_1) = Y \arrow[d, "\pi_Y"] \\
X_0 \arrow[r, "g_0"'] & Y_0
\end{tikzcd}$$

Figure 26: Equalizer Function

The constant and equalizer operations are extended to morphisms. Given any function $f : X \rightarrow Y$, there is a constant parallel pair morphism $(f, f) : (1_X, 1_X) \rightarrow (1_Y, 1_Y)$, whose component functions are both f. Given any parallel pair morphism $(g_0, g_1) : (x_0, x_1) \rightarrow (y_0, y_1)$, there is an equalizer function $(g_0 = g_1) : (x_0 = x_1) \rightarrow (y_0 = y_1)$ defined using parts: $(g_0 = g_1) \cdot \pi_Y = \pi_X \cdot g_0$ (Figure 26).

```
(1) (UR$morphism constant)
    (= (UR$source constant) KIF.FTN$function)
    (= (UR$target constant) KIF.DGM.PPR.MOR$parallel-pair-morphism)
    (= (UR$composition [constant KIF.DGM.PPR.MOR$source])
       (UR$composition [KIF.FTN$source KIF.LIM.PPR.OBJ$constant]))
    (= (UR$composition [constant KIF.DGM.PPR.MOR$target])
       (UR$composition [KIF.FTN$target KIF.LIM.PPR.OBJ$constant]))
    (= (UR$composition [constant KIF.DGM.PPR.MOR$function0])
       (UR$identity KIF.FTN$function))
    (= (UR$composition [constant KIF.DGM.PPR.MOR$function1])
       (UR$identity KIF.FTN$function))

(2) (UR$morphism limit) (UR$morphism equalizer) (= equalizer limit)
    (= (UR$source limit) KIF.DGM.PPR.MOR$parallel-pair-morphism)
    (= (UR$target limit) KIF.FTN$function)
    (= (UR$composition [limit KIF.FTN$source])
       (UR$composition [KIF.DGM.PPR.MOR$source KIF.LIM.EQU.OBJ$limit]))
    (= (UR$composition [limit KIF.FTN$target])
       (UR$composition [KIF.DGM.PPR.MOR$target KIF.LIM.EQU.OBJ$limit]))
    (forall (?g (KIF.DGM.PPR.MOR$parallel-pair-morphism ?g))
        (KIF.FTN$restriction (limit ?g) (KIF.DGM.PPR.MOR$function0 ?g)))
```

Pullbacks

KIF.LIM.PBK.OBJ

$$\begin{tikzcd} \mathsf{Col} \arrow[d, shift right=2, "\Delta"'] \arrow[d, phantom, "\dashv"] \\ \mathsf{Ospn} \arrow[u, shift right=2, "\times_\bullet"'] \end{tikzcd}$$

Figure 27a: Adjunction

$$\frac{X \longrightarrow Y_0 \times_{Y_\bullet} Y_1}{\Delta(X) \longrightarrow (y_0 : Y_0 \rightarrow Y_\bullet,\ y_1 : Y_1 \rightarrow Y_\bullet)} \ \downarrow \cong \uparrow$$

Figure 27b: Pullback

The essential properties of the pullback operation are illustrated in Figure 27b. The horizontal bar designates a natural bijection between the mediator above the bar and the cone below the bar. A natural *components* (*unpacking*) process assigns a cone $\Delta(X) = (1_X : X \rightarrow X,\ 1_X : X \rightarrow X) \rightarrow (y_0 : Y_0 \rightarrow Y_\bullet,\ y_1 : Y_1 \rightarrow Y_\bullet)$ below the bar to every mediator $X \rightarrow Y_0 \times_{Y_\bullet} Y_1$ above the bar. A natural *pairing* (*packing*) process assigns a mediator $X \rightarrow Y_0 \times_{Y_\bullet} Y_1$ above the bar to every cone $\Delta(X) \rightarrow (y_0 : Y_0 \rightarrow Y_\bullet,\ y_1 : Y_1 \rightarrow Y_\bullet)$ below the bar. These two processes are inverse to each other. The components process is realized by application of the constant (Δ) operation and then composition on the right with the projection opspan morphism (a component of the counit π). The pairing process is realized by application of the pullback ($\times_\bullet$) operation and then composition on the left with the delta function (a component of the unit δ).

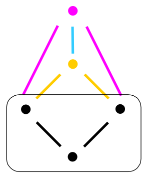


Figure 8.V: Pullback

The cone Ur object consists of opspan morphisms with constant source opspans. Pullback cones are treated both abstractly and concretely. Abstractly, a pullback *cone* determines a constrained pair (X, g) consisting of a collection (or vertex) X and an opspan morphism $g : \Delta(X) \rightarrow (y_0 : Y_0 \rightarrow Y_\bullet, y_1 : Y_1 \rightarrow Y_\bullet)$, where the source of g is the constant opspan over the collection X. Concretely, a pullback *cone* is a function pair (g_0, g_1), which share a common source collection: $g_0 : X \rightarrow Y_0$ and $g_1 : X \rightarrow Y_1$ and commute with the function components of the target of the associated opspan morphism. Hence, the pullback cone Ur object is a subobject of the function pairs Ur object with two related associated Ur morphisms mapping to the common source collection and to an opspan morphism over that function pair. In applications, such as the IFF Upper Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since the pairing function (below) can be concretely referenced as ‘`(KIF.LIM.PBK.OBJ$pairing [?g0 ?g1])`’ rather than needing to specify a partial function from function pairs to mediators.

```
 (1) (UR$object cone)
     (UR$subobject cone KIF.DGM.PR.MOR$function-pair)

 (2) (UR$morphism collection) (UR$morphism vertex) (= vertex collection)
     (= (UR$source collection) cone)
     (= (UR$target collection) KIF.COL$collection)

 (3) (UR$morphism opspan-morphism)
     (= (UR$source opspan-morphism) cone)
     (= (UR$target opspan-morphism) KIF.DGM.OSPN.MOR$opspan-morphism)

 (4) (= (UR$composition [opspan-morphism KIF.DGM.OSPN.MOR$source])
        (UR$composition [collection constant]))

 (5) (UR$morphism function-pair)
     (= (UR$source function-pair) cone)
     (= (UR$target function-pair) KIF.DGM.PR.MOR$function-pair)
     (= function-pair (UR$composition [opspan-morphism KIF.DGM.OSPN.MOR$function-pair])
     (= function-pair (UR$inclusion [cone KIF.DGM.PR.MOR$function-pair]))

 (6) (UR$morphism function0)
     (= (UR$source function0) cone)
     (= (UR$target function0) KIF.FTN$function)
     (= function0 (UR$composition [opspan-morphism KIF.DGM.OSPN.MOR$function0])

 (7) (UR$morphism function1)
     (= (UR$source function1) cone)
     (= (UR$target function1) KIF.FTN$function)
     (= function1 (UR$composition [opspan-morphism KIF.DGM.OSPN.MOR$function1])
```

The *mediator* collection consists of functions with pullback target. A mediator consists of a function $f : X \rightarrow Y_0 \times_{Y_\bullet} Y_1$ and an opspan $(y_0 : Y_0 \rightarrow Y_\bullet, y_1 : Y_1 \rightarrow Y_\bullet)$ whose pullback is the target of the function. We introduce convenience terms for the component collections Y_0 and Y_1.

```
 (8) (UR$object mediator)

 (9) (UR$morphism function)
     (= (UR$source function) mediator)
     (= (UR$target function) KIF.FTN$function)

(10) (UR$morphism opspan)
     (= (UR$source opspan) mediator)
     (= (UR$target opspan) KIF.DGM.OSPN.OBJ$opspan)

(11) (= (UR$composition [opspan pullback])
        (UR$composition [function KIF.FTN$target]))

(12) (UR$morphism collection-pair)
     (= (UR$source collection-pair) mediator)
     (= (UR$target collection-pair) KIF.DGM.PR.OBJ$collection-pair)
     (= collection-pair (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection-pair])

(13) (UR$morphism collection0)
     (= (UR$source collection0) mediator)
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection0])

(14) (UR$morphism collection1)
     (= (UR$source collection1) mediator)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [opspan KIF.DGM.OSPN.OBJ$collection1])
```

$$\frac{X \xrightarrow{\delta_X} X\times_X X}{X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X}$$

Figure 28a: Delta Mediator

$$\frac{Y_0\times_{Y_\bullet}Y_1 \xrightarrow{1} Y_0\times_\bullet Y_1}{Y_0\times_{Y_\bullet}Y_1 \xrightarrow{\pi_0} Y_0,\ Y_0\times_{Y_\bullet}Y_1 \xrightarrow{\pi_1} Y_1}$$

Figure 28b: Projection Cone

Right adjoint and unit. Given any collection X, the *constant* Ur-morphism constructs the opspan $\Delta(X) = (1_X : X \rightarrow X,\ 1_X : X \rightarrow X)$ that is constantly the given collection. Given any collection X, the *delta* Ur-morphism constructs the mediator (Figure 28a) with function $\delta_X : X \rightarrow (X \times_X X)$, which is the pairing of the cone whose opspan morphism $(id_X, id_X) : (1_X : X \rightarrow X,\ 1_X : X \rightarrow X) \rightarrow (1_X : X \rightarrow X,\ 1_X : X \rightarrow X)$ consists of the identity function for the given collection – the delta mediator is the pairing of two copies of the identity function. This delta function is a bijection, since $(X \times_X X) = \{(x, x) \mid x \in X\}$.

```
(15)  (UR$morphism constant)
      (= (UR$source constant) KIF.COL$collection)
      (= (UR$target constant) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$composition [constant KIF.DGM.OSPN.OBJ$opvertex])
         (UR$identity KIF.COL$collection))
      (= (UR$composition [constant KIF.DGM.OSPN.OBJ$function0]) KIF.FTN$identity)
      (= (UR$composition [constant KIF.DGM.OSPN.OBJ$function1]) KIF.FTN$identity)
      (= (UR$composition [constant KIF.DGM.OSPN.OBJ$collection0])
         (UR$identity KIF.COL$collection))
      (= (UR$composition [constant KIF.DGM.OSPN.OBJ$collection1])
         (UR$identity KIF.COL$collection))

(16)  (UR$morphism delta-cone)
      (= (UR$source delta-cone) KIF.COL$collection)
      (= (UR$target delta-cone) cone)
      (= (UR$composition [delta-cone collection]) (UR$identity KIF.COL$collection))
      (= (UR$composition [delta-cone function0]) KIF.FTN$identity)
      (= (UR$composition [delta-cone function1]) KIF.FTN$identity)

(17)  (UR$morphism delta)
      (= (UR$source delta) KIF.COL$collection)
      (= (UR$target delta) mediator)
      (= (UR$composition [delta collection0]) (UR$identity KIF.COL$collection))
      (= (UR$composition [delta collection1]) (UR$identity KIF.COL$collection))
      (= delta (UR$composition [delta-cone packing]))
```

Left adjoint and counit. For any opspan $(x_0 : X_0 \rightarrow X_\bullet,\ x_1 : X_1 \rightarrow X_\bullet)$, the *pullback* collection $X_0 \times_{X_\bullet} X_1$ and *pullback projection* functions $\pi_0 : X_0 \times_{X_\bullet} X_1 \rightarrow X_0$ and $\pi_1 : X_0 \times_{X_\bullet} X_1 \rightarrow X_1$ are defined. This is the pullback in the quasi-category of collections and their (total) functions. These form a cone that is universal over the given opspan. Given any opspan $(x_0 : X_0 \rightarrow X_\bullet,\ x_1 : X_1 \rightarrow X_\bullet)$, the *limit* Ur-morphism constructs the collection $X_0 \times_{X_\bullet} X_1$ that is the *pullback* of the given opspan. And the *projection* Ur-morphism constructs the cone (Figure 28b), which is the components $\pi = 1_{01}$ of the mediator whose function $1 : X_0 \times_{X_\bullet} X_1 \rightarrow X_0 \times_{X_\bullet} X_1$ is the identity function for the pullback of the given opspan – the projection cone is the components of the pullback identity. We introduce two convenience terms for the component projections. The last axiom expresses the concreteness of the pullback and the pullback projections.

```
(18)  (UR$morphism limit) (UR$morphism pullback) (= pullback limit)
      (= (UR$source limit) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$target limit) KIF.COL$collection)
      (= limit (UR$composition [projection collection]))

(19)  (UR$morphism projection-mediator)
      (= (UR$source projection-mediator) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$target projection-mediator) mediator)
      (= (UR$composition [projection-mediator function])
         (UR$composition [limit KIF.FTN$identity]))
      (= (UR$composition [projection-mediator opspan])
         (UR$identity KIF.DGM.OSPN.OBJ$opspan))

(20)  (UR$morphism projection)
      (= (UR$source projection) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$target projection) cone)
      (= (UR$composition [projection collection]) limit)
      (= projection (UR$composition [projection-mediator unpacking]))

(21)  (UR$morphism projection0)
      (= (UR$source projection0) KIF.DGM.OSPN.OBJ$opspan)
      (= (UR$target projection0) KIF.FTN$function)
      (= (UR$composition [projection0 KIF.FTN$source]) limit)
      (= (UR$composition [projection0 KIF.FTN$target]) KIF.DGM.OSPN.OBJ$collection0)
```

```
     (= projection0 (UR$composition [projection function0]))

(22) (UR$morphism projection1)
     (= (UR$source projection1) KIF.DGM.OSPN.OBJ$opspan)
     (= (UR$target projection1) KIF.FTN$function)
     (= (UR$composition [projection1 KIF.FTN$source]) limit)
     (= (UR$composition [projection1 KIF.FTN$target]) KIF.DGM.OSPN.OBJ$collection1)
     (= projection1 (UR$composition [projection function1]))

(23) (forall (?X (KIF.DGM.OSPN.OBJ$opspan ?X)
         (and (KIF.COL$subcollection
                  (limit ?X)
                  (KIF.LIM.PRD2.OBJ$binary-product
                      (KIF.DGM.OSPN.OBJ$collection-pair ?X)))
              (= (projection0 ?X)
                 (KIF.FTN$composition
                     [(KIF.COL$inclusion
                          [(limit ?X)
                           (KIF.LIM.PRD2.OBJ$binary-product
                               (KIF.DGM.OSPN.OBJ$collection-pair ?X))])
                      (KIF.LIM.PRD2.OBJ$projection0
                          (KIF.DGM.OSPN.OBJ$collection-pair ?X))]))
              (= (projection1 ?X)
                 (KIF.FTN$composition
                     [(KIF.COL$inclusion
                          [(limit ?X)
                           (KIF.LIM.PRD2.OBJ$binary-product
                               (KIF.DGM.OSPN.OBJ$collection-pair ?X))])
                      (KIF.LIM.PRD2.OBJ$projection1
                          (KIF.DGM.OSPN.OBJ$collection-pair ?X))]))
              (forall (?x (KIF.LIM.PRD2.OBJ$binary-product
                              (KIF.DGM.OSPN.OBJ$collection-pair ?X)) ?x))
                  (and (<=> ((limit ?X) ?x)
                            (= ((KIF.DGM.OSPN.OBJ$opfunction0 ?X) (?x 0))
                               ((KIF.DGM.OSPN.OBJ$opfunction1 ?X) (?x 1))))
                       (= ((projection0 ?X) ?x) (?x 0))
                       (= ((projection1 ?X) ?x) (?x 1))))))
```

$$\frac{X \xrightarrow{f} Y_0{\times_{Y\bullet}}Y_1}{X \xrightarrow{f\cdot\pi_0} Y_0,\ X \xrightarrow{f\cdot\pi_1} Y_1}\ \downarrow$$

Figure 29b: unpacking or components

$$\frac{X \xrightarrow{\langle g_0,\, g_1\rangle} Y_0{\times_{Y\bullet}}Y_1}{X \xrightarrow{g_0} Y_0,\ X \xrightarrow{g_1} Y_1}\ \uparrow$$

Figure 29b: packing or pairing

The components of a mediator unpack it into a cone. For any mediator f with opspan ($y_0 : Y_0 \rightarrow Y_\bullet$, $y_1 : Y_1 \rightarrow Y_\bullet$) and function $f : X \rightarrow Y_0 \times_{Y\bullet} Y_1$, there is a *components* cone $f_{01} = (f_0, f_1) : X \rightarrow (Y_0, Y_1)$, 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 opspan morphism of the projection cone $\pi_{(Y0,\ Y1)}$ (a component of the counit π):

$$f_{01} = (f_0, f_1) = (f, f) \cdot \pi_{(Y0,\ Y1)} = (f \cdot \pi_{Y0}, f \cdot \pi_{Y1}) : (X, X) \rightarrow (Y_0{\times}Y_1, Y_0{\times}Y_1) \rightarrow (Y_0, Y_1).$$

The pairing of a cone packs it into a mediator. For any cone g with collection X and opspan morphism $(g_0, g_1) : \Delta(X) \rightarrow (y_0 : Y_0 \rightarrow Y_\bullet,\ y_1 : Y_1 \rightarrow Y_\bullet)$, there is a *pairing* mediator $\langle g\rangle = \langle g_0, g_1\rangle : X \rightarrow Y_0 \times_{Y\bullet} Y_1$, which pairs the images of the two functions of the original opspan morphism. The packing (pairing) process is realized by application of the pullback ($\times_\bullet$) operator to the opspan morphism (g_0, g_1) and then composition on the left with the delta function δ_X (a component of the unit δ):

$$\langle g\rangle = \langle g_0, g_1\rangle = \delta_X \cdot \times_\bullet((g_0, g_1)) : X \rightarrow (X \times_X X) \rightarrow (Y_0 \times_{Y\bullet} Y_1).$$

These two processes are inverse to each other: $\langle g\rangle_{01} = g$ for each cone g and $\langle f_{01}\rangle = f$ for each mediator f.

```
(24) (UR$morphism packing) (UR$morphism pairing) (= pairing packing)
     (= (UR$source packing) cone)
     (= (UR$target packing) mediator)
     (forall (?g (cone ?g))
         (and (= (function (packing ?g))
                 (KIF.FTN$composition
                     [(delta (collection ?g))
                      (KIF.LIM.PBK.MOR$pullback (opspan-morphism ?g))]))
              (= (opspan (packing ?g))
                 (KIF.DGM.OSPN.MOR$target (opspan-morphism ?g)))))

(25) (UR$morphism unpacking) (UR$morphism components) (= components unpacking)
     (= (UR$source unpacking) mediator)
     (= (UR$target unpacking) cone)
     (forall (?f (mediator ?f))
         (and (= (collection (unpacking ?f))
                 (KIF.FTN$source (function ?f)))
              (= (opspan-morphism (unpacking ?f))
                 (KIF.DGM.OSPN.MOR$composition
                     [(KIF.LIM.PBK.MOR$constant (function ?f))
                      (opspan-morphism (projection (opspan ?f)))]))))

(26) (= (UR$composition [packing unpacking]) (UR$identity cone))
     (= (UR$composition [unpacking packing]) (UR$identity mediator))
```

For any collection function $f : A \rightarrow B$ there is a *kernel-pair* equivalence relation on the source collection A.

```
(27) (UR$morphism kernel-pair-opspan)
     (= (UR$source kernel-pair-opspan) KIF.FTN$function)
     (= (UR$target kernel-pair-opspan) KIF.DGM.OSPN.OBJ$opspan)
     (= (UR$composition [kernel-pair-opspan KIF.DGM.OSPN.OBJ$collection0]) KIF.FTN$source)
     (= (UR$composition [kernel-pair-opspan KIF.DGM.OSPN.OBJ$collection1]) KIF.FTN$source)
     (= (UR$composition [kernel-pair-opspan KIF.DGM.OSPN.OBJ$opvertex]) KIF.FTN$target)
     (= (UR$composition [kernel-pair-opspan KIF.DGM.OSPN.OBJ$function0])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [kernel-pair-opspan KIF.DGM.OSPN.OBJ$function1])
        (UR$identity KIF.FTN$function))

(28) (UR$morphism kernel-pair)
     (= (UR$source kernel-pair) KIF.FTN$function)
     (= (UR$target kernel-pair) KIF.REL$relation)
     (= (UR$composition [kernel-pair KIF.REL$collection0]) KIF.FTN$source)
     (= (UR$composition [kernel-pair KIF.REL$collection1]) KIF.FTN$source)
     (= (UR$composition [kernel-pair KIF.REL$extent])
        (UR$composition [kernel-pair-opspan pullback]))
     (= (UR$composition [kernel-pair KIF.REL$projection0])
        (UR$composition [kernel-pair-opspan projection0]))
     (= (UR$composition [kernel-pair KIF.REL$projection1])
        (UR$composition [kernel-pair-opspan projection1]))
```

Pullback Morphisms

KIF.LIM.PBK.MOR

$$\begin{tikzcd}
X_0 \times_{X_\bullet} X_1 \arrow[r, "\times_\bullet(g)"] \arrow[d, "\pi_{Xk}"'] \arrow[dr, phantom, "k = 0{,}1"] & Y_0 \times_{Y_\bullet} Y_1 \arrow[d, "\pi_{Yk}"] \\
X_k \arrow[r, "g_k"'] & Y_k
\end{tikzcd}$$

Figure 30: Pullback Function

The constant and pullback operations are extended to morphisms. Given any function $f : X \rightarrow Y$, there is a constant opspan morphism $(f, f) : \Delta(X) = (1_X : X \rightarrow X,\ 1_X : X \rightarrow X) \rightarrow (1_Y : Y \rightarrow Y,\ 1_Y : Y \rightarrow Y) = \Delta(Y)$, whose component functions are both f. Given any opspan morphism $g =$ $(g_0, g_1) : (x_0 : X_0 \rightarrow X_\bullet, x_1 : X_1 \rightarrow X_\bullet) \rightarrow (y_0 : Y_0 \rightarrow Y_\bullet, y_1 : Y_1 \rightarrow Y_\bullet)$, there is a pullback function $\times_\bullet(g) : (X_0 \times_{X_\bullet} X_1) \rightarrow (Y_0 \times_{Y_\bullet} Y_1)$ defined using projections: $\times_\bullet(g) \cdot \pi_{Y0} = \pi_{X0} \cdot g_0$ and $\times_\bullet(g) \cdot \pi_{Y1} = \pi_{X1} \cdot g_1$ (Figure 30).

```
 (1) (UR$morphism constant)
     (= (UR$source constant) KIF.FTN$function)
     (= (UR$target constant) KIF.DGM.OSPN.MOR$opspan-morphism)
     (= (UR$composition [constant KIF.DGM.OSPN.MOR$source])
        (UR$composition [KIF.FTN$source KIF.LIM.PBK.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.OSPN.MOR$target])
        (UR$composition [KIF.FTN$target KIF.LIM.PBK.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.OSPN.MOR$function0])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [constant KIF.DGM.OSPN.MOR$function1])
        (UR$identity KIF.FTN$function))

 (2) (UR$morphism limit) (UR$morphism pullback) (= pullback limit)
     (= (UR$source limit) KIF.DGM.OSPN.MOR$opspan-morphism)
     (= (UR$target limit) KIF.FTN$function)
     (= (UR$composition [limit KIF.FTN$source])
        (UR$composition [KIF.DGM.OSPN.MOR$source KIF.LIM.PBK.OBJ$limit]))
     (= (UR$composition [limit KIF.FTN$target])
        (UR$composition [KIF.DGM.OSPN.MOR$target KIF.LIM.PBK.OBJ$limit]))
     (forall (?g (KIF.DGM.OSPN.MOR$opspan-morphism ?g))
         (and (= (KIF.FTN$composition
                     [(limit ?g)
                      (KIF.LIM.PBK.OBJ$projection0 (KIF.DGM.OSPN.MOR$target ?g))])
                 (KIF.FTN$composition
                     [(KIF.LIM.PBK.OBJ$projection0 (KIF.DGM.OSPN.MOR$source ?g))
                      (KIF.DGM.OSPN.MOR$function0 ?g)]))
              (= (KIF.FTN$composition
                     [(limit ?g)
                      (KIF.LIM.PBK.OBJ$projection1 (KIF.DGM.OSPN.MOR$target ?g))])
                 (KIF.FTN$composition
                     [(KIF.LIM.PBK.OBJ$projection1 (KIF.DGM.OSPN.MOR$source ?g))
                      (KIF.DGM.OSPN.MOR$function1 ?g)])))))
```

Multipullbacks

KIF.LIM.MPBK.OBJ

$$\begin{tikzcd}
\text{Col} \arrow[d, shift right=2, "\Delta"'] \arrow[d, phantom, "\overset{\delta}{\underset{\pi}{\dashv}}"] \\
\text{Mospn} \arrow[u, shift right=2, "\prod"']
\end{tikzcd}$$

Figure 31a: Adjunction

$$\frac{X \longrightarrow \prod(Y)}{\Delta(X) \longrightarrow Y} \quad \downarrow\cong\uparrow$$

Figure 31b: Multipullback

The essential properties of the multipullback operation are illustrated in Figure 31b. The horizontal bar designates a natural bijection between the mediator above the bar and the cone below the bar. A natural *components* (*unpacking*) process assigns a cone $\Delta(X) \rightarrow Y$ below the bar to every mediator $X \rightarrow \prod(Y)$ above the bar, where

$\Delta(X) = ((1_X : X \rightarrow X \leftarrow X : 1_X), (1_X : X \rightarrow X \leftarrow X : 1_X))$ and

$Y = ((y_{00} : Y_{00} \rightarrow Y_{0\bullet} \leftarrow Y_{01} : y_{01}), (y_{10} : Y_{10} \rightarrow Y_{1\bullet} \leftarrow Y_{11} : y_{11}))$, where $Y_{01} = Y = Y_{10}$.

A natural *pairing* (*packing*) process assigns a mediator $X \rightarrow \prod(Y)$ above the bar to every cone $\Delta(X) \rightarrow Y$ below the bar. These two processes are inverse to each other. The components process is realized by application of the constant (Δ) operation and then composition on the right with the projection opspan morphism (a component of the counit π). The pairing process is realized by application of the multipullback ($\prod$) operation and then composition on the left with the delta function (a component of the unit δ).

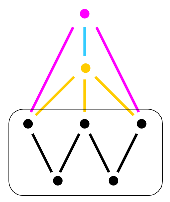


Figure 8.W: Multipullback

The cone Ur object consists of multi-opspan morphisms with constant source multi-opspans. Multipullback cones are treated both abstractly and concretely. Abstractly, a multipullback *cone* determines a constrained pair (X, g) consisting of a collection (or vertex) X and an multi-opspan morphism $g : \Delta(X) \rightarrow Y$, where the source of g is the constant multi-opspan over the collection X. Concretely, a multipullback *cone* is a function triple (g_0, g_2, g_1), which share a common source collection: $g_0 : X \rightarrow Y_{00}$, $g_2 : X \rightarrow Y$ and

$g_1 : X \rightarrow Y_1$ and commute with the function components of the target of the associated multi-opspan morphism. Hence, the multipullback cone Ur object is a subobject of the function triples Ur object with two related associated Ur morphisms mapping to the common source collection and to a multi-opspan morphism over that function triple. In applications, such as the IFF Upper Category Theory (meta) Ontology (IFF-CAT), the concrete representation is very useful and important, since the tripling function (below) can be concretely referenced as '`(KIF.LIM.MPBK.OBJ$tripling [?g0 ?g1 ?g2])`' rather than needing to specify a partial function from function triples to mediators.

```
 (1)  (UR$object cone)
      (UR$subobject cone KIF.DGM.TRP.MOR$function-triple)

 (2)  (UR$morphism collection) (UR$morphism vertex) (= vertex collection)
      (= (UR$source collection) cone)
      (= (UR$target collection) KIF.COL$collection)

 (3)  (UR$morphism multi-opspan-morphism)
      (= (UR$source multi-opspan-morphism) cone)
      (= (UR$target multi-opspan-morphism) KIF.DGM.MOSPN.MOR$multi-opspan-morphism)

 (4)  (= (UR$composition [multi-opspan-morphism KIF.DGM.MOSPN.MOR$source])
         (UR$composition [collection constant]))

 (5)  (UR$morphism function-triple)
      (= (UR$source function-triple) cone)
      (= (UR$target function-triple) KIF.DGM.TRP.MOR$function-triple)
      (= function-triple (UR$composition [multi-opspan-morphism KIF.DGM.MSPN.MOR$function-triple])
      (= function-triple (UR$inclusion [cone KIF.DGM.TRP.MOR$function-triple]))

 (6)  (UR$morphism function0)
      (= (UR$source function0) cone)
      (= (UR$target function0) KIF.FTN$function)
      (= function0 (UR$composition [multi-opspan-morphism KIF.DGM.MSPN.MOR$function0])

 (7)  (UR$morphism function2)
      (= (UR$source function2) cone)
      (= (UR$target function2) KIF.FTN$function)
      (= function2 (UR$composition [multi-opspan-morphism KIF.DGM.MSPN.MOR$function])

 (8)  (UR$morphism function1)
      (= (UR$source function1) cone)
      (= (UR$target function1) KIF.FTN$function)
      (= function1 (UR$composition [multi-opspan-morphism KIF.DGM.MOSPN.MOR$function1])
```

The *mediator* collection consists of functions with multipullback target. A mediator consists of a function $f : X \rightarrow \prod(Y)$ and a multi-opspan $Y = ((y_{00} : Y_{00} \rightarrow Y_{0\bullet} \leftarrow Y_{01} : y_{01}), (y_{10} : Y_{10} \rightarrow Y_{1\bullet} \leftarrow Y_{11} : y_{11}))$ with $Y_{01} = Y = Y_{10}$, whose multipullback is the target of the function. We introduce convenience terms for the component collections Y_0, Y_2 and Y_1.

```
 (9)  (UR$object mediator)

(10)  (UR$morphism function)
      (= (UR$source function) mediator)
      (= (UR$target function) KIF.FTN$function)

(11)  (UR$morphism multi-opspan)
      (= (UR$source multi-opspan) mediator)
      (= (UR$target multi-opspan) KIF.DGM.MOSPN.OBJ$multi-opspan)

(12)  (= (UR$composition [multi-opspan multipullback])
         (UR$composition [function KIF.FTN$target]))

(13)  (UR$morphism collection-triple)
      (= (UR$source collection-triple) mediator)
      (= (UR$target collection-triple) KIF.DGM.TRP.OBJ$collection-triple)
      (= collection-triple
         (UR$composition [multi-opspan KIF.DGM.MOSPN.OBJ$collection-triple])

(14)  (UR$morphism collection0)
      (= (UR$source collection0) mediator)
```

```
     (= (UR$target collection0) KIF.COL$collection)
     (= collection0 (UR$composition [multi-opspan KIF.DGM.MOSPN.OBJ$collection0]))

(15) (UR$morphism collection2)
     (= (UR$source collection2) mediator)
     (= (UR$target collection2) KIF.COL$collection)
     (= collection2 (UR$composition [multi-opspan KIF.DGM.MOSPN.OBJ$collection]))

(16) (UR$morphism collection1)
     (= (UR$source collection1) mediator)
     (= (UR$target collection1) KIF.COL$collection)
     (= collection1 (UR$composition [multi-opspan KIF.DGM.MOSPN.OBJ$collection1]))
```

$$\frac{X \xrightarrow{\delta_X} \prod(\Delta(X))}{X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X,\ X \xrightarrow{1_X} X}$$

Figure 32a: Delta Mediator

$$\frac{\prod(Y) \xrightarrow{1} \prod(Y)}{\prod(Y) \xrightarrow{\pi_0} Y_0,\ \prod(Y) \xrightarrow{\pi} Y,\ \prod(Y) \xrightarrow{\pi_1} Y_1}$$

Figure 32b: Projection Cone

Right adjoint and unit. Given any collection X, the *constant* Ur-morphism constructs the multi-opspan $\Delta(X) = ((1_X : X \rightarrow X \leftarrow X : 1_X), (1_X : X \rightarrow X \leftarrow X : 1_X))$ that is constantly the given collection. Given any collection X, the *delta* Ur-morphism constructs the mediator (Figure 32a) with function $\delta_X : X \rightarrow \prod(\Delta(X))$, which is the tripling of the cone whose opspan morphism $(id_X, id_X, id_X) : \Delta(X) \rightarrow \Delta(X)$ consists of the identity function for the given collection – the delta mediator is the tripling of three copies of the identity function. This delta function is a bijection, since $\prod(\Delta(X)) = \{(x, x, x) \mid x \in X\}$.

```
(17) (UR$morphism constant)
     (= (UR$source constant) KIF.COL$collection)
     (= (UR$target constant) KIF.DGM.MOSPN.OBJ$multi-opspan)
     (= (UR$composition [constant KIF.DGM.MOSPN.OBJ$opspan0]) KIF.LIM.OSPN.OBJ$constant)
     (= (UR$composition [constant KIF.DGM.MOSPN.OBJ$opspan1]) KIF.LIM.OSPN.OBJ$constant)

(18) (UR$morphism delta-cone)
     (= (UR$source delta-cone) KIF.COL$collection)
     (= (UR$target delta-cone) cone)
     (= (UR$composition [delta-cone collection]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta-cone function0]) KIF.FTN$identity)
     (= (UR$composition [delta-cone function2]) KIF.FTN$identity)
     (= (UR$composition [delta-cone function1]) KIF.FTN$identity)

(19) (UR$morphism delta)
     (= (UR$source delta) KIF.COL$collection)
     (= (UR$target delta) mediator)
     (= (UR$composition [delta collection0]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta collection2]) (UR$identity KIF.COL$collection))
     (= (UR$composition [delta collection1]) (UR$identity KIF.COL$collection))
     (= delta (UR$composition [delta-cone packing]))
```

Left adjoint and counit. For any multi-opspan

$$X = ((x_{00} : X_{00} \rightarrow X_{0\bullet} \leftarrow X_{01} : x_{01}), (x_{10} : X_{10} \rightarrow X_{1\bullet} \leftarrow X_{11} : x_{11})),$$

with $X_{01} = X_{10}$, the *multipullback* collection $\prod(X)$ and *pullback projection* functions $\pi_0 : \prod(X) \rightarrow X_0$, $\pi : \prod(X) \rightarrow X$ and $\pi_1 : \prod(X) \rightarrow X_1$ are defined. This is the multipullback in the quasi-category of collections and their (total) functions. These form a cone that is universal over the given multi-opspan. Given any multi-opspan X as above, the *limit* Ur-morphism constructs the collection $\prod(X)$ that is the *multipullback* of the given multi-opspan. And the *projection* Ur-morphism constructs the cone (Figure 32b), which is the components $\pi = 1_{012}$ of the mediator whose function $1 : \prod(X) \rightarrow \prod(X)$ is the identity function for the multipullback of the given multi-opspan – the projection cone is the components of the multipullback identity. We introduce three convenience terms for the component projections. The last axiom expresses the concreteness of the multi-pullback and the multi-pullback projections.

```
(20) (UR$morphism limit) (UR$morphism multipullback) (= multipullback limit)
     (= (UR$source limit) KIF.DGM.MOSPN.OBJ$multi-opspan)
     (= (UR$target limit) KIF.COL$collection)
```

```
      (= limit (UR$composition [projection collection]))

(21)  (UR$morphism projection-mediator)
      (= (UR$source projection-mediator) KIF.DGM.MOSPN.OBJ$multi-opspan)
      (= (UR$target projection-mediator) mediator)
      (= (UR$composition [projection-mediator function])
         (UR$composition [limit KIF.FTN$identity])
      (= (UR$composition [projection-mediator multi-opspan])
         (UR$identity KIF.DGM.MOSPN.OBJ$multi-opspan))

(22)  (UR$morphism projection)
      (= (UR$source projection) KIF.DGM.MOSPN.OBJ$multi-opspan)
      (= (UR$target projection) cone)
      (= (UR$composition [projection collection]) limit)
      (= projection (UR$composition [projection-mediator unpacking]))

(23)  (UR$morphism projection0)
      (= (UR$source projection0) KIF.DGM.MOSPN.OBJ$multi-opspan)
      (= (UR$target projection0) KIF.FTN$function)
      (= (UR$composition [projection0 KIF.FTN$source]) limit)
      (= (UR$composition [projection0 KIF.FTN$target]) KIF.DGM.MOSPN.OBJ$collection0)
      (= projection0 (UR$composition [projection function0]))

(24)  (UR$morphism projection2)
      (= (UR$source projection2) KIF.DGM.MOSPN.OBJ$multi-opspan)
      (= (UR$target projection2) KIF.FTN$function)
      (= (UR$composition [projection2 KIF.FTN$source]) limit)
      (= (UR$composition [projection2 KIF.FTN$target]) KIF.DGM.MOSPN.OBJ$collection)
      (= projection2 (UR$composition [projection function2]))

(25)  (UR$morphism projection1)
      (= (UR$source projection1) KIF.DGM.MOSPN.OBJ$multi-opspan)
      (= (UR$target projection1) KIF.FTN$function)
      (= (UR$composition [projection1 KIF.FTN$source]) limit)
      (= (UR$composition [projection1 KIF.FTN$target]) KIF.DGM.MOSPN.OBJ$collection1)
      (= projection1 (UR$composition [projection function1]))

(26)  (forall (?S0 ?S1 (KIF.DGM.OSPN.OBJ$multi-opspan [?S0 ?S1])
          (and (KIF.COL$subcollection
                   (limit [?S0 ?S1])
                   (KIF.LIM.PRD3$ternary-product
                       (KIF.DGM.MOSPN$collection-triple [?S0 ?S1])))
               (= (projection0 ?X)
                  (KIF.FTN$composition
                      [(KIF.COL$inclusion
                           [(limit [?S0 ?S1])
                            (KIF.LIM.PRD3.OBJ$ternary-product
                                (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))])
                       (KIF.LIM.PRD3.OBJ$projection0
                           (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))]))
               (= (projection2 ?X)
                  (KIF.FTN$composition
                      [(KIF.COL$inclusion
                           [(limit [?S0 ?S1])
                            (KIF.LIM.PRD3.OBJ$ternary-product
                                (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))])
                       (KIF.LIM.PRD3.OBJ$projection1
                           (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))]))
               (= (projection1 ?X)
                  (KIF.FTN$composition
                      [(KIF.COL$inclusion
                           [(limit [?S0 ?S1])
                            (KIF.LIM.PRD3.OBJ$ternary-product
                                (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))])
                       (KIF.LIM.PRD3.OBJ$projection2
                           (KIF.DGM.MOSPN.OBJ$collection-triple [?S0 ?S1]))]))
               (forall (?x ((KIF.LIM.PRD3.OBJ$ternary-product
                              (KIF.DGM.MOSPN$.OBJcollection-triple [?S0 ?S1])) ?x))
                   (and (<=> ((limit [?S0 ?S1]) ?x)
                             (and (= ((KIF.DGM.SPN$function0 ?S0) (?x 0))
                                     ((KIF.DGM.SPN$function1 ?S0) (?x 1)))
                                  (= ((KIF.DGM.SPN$function0 ?S1) (?x 1))
                                     ((KIF.DGM.SPN$function1 ?S1) (?x 2)))))
                        (=> ((limit [?S0 ?S1]) ?x)
                            (and (= ((projection0 [?S0 ?S1]) ?x) (?x 0))
                                 (= ((projection [?S0 ?S1]) ?x) (?x 1))
                                 (= ((projection1 [?S0 ?S1]) ?x) (?x 2))))))
```

$$\begin{tikzcd}[row sep=small] & X \arrow[r, "f"] & \prod(Y) & & \\ \hline X \arrow[r, "f \cdot \pi_0"] & Y_0, \; X \arrow[r, "f \cdot \pi"] & Y, \; X \arrow[r, "f \cdot \pi_1"] & Y_1 & \downarrow \end{tikzcd}$$

Figure 33b: unpacking or components

$$\begin{tikzcd}[row sep=small] & X \arrow[r, "\langle g_0, g, g_1 \rangle"] & \prod(Y) & & \\ \hline X \arrow[r, "g_0"] & Y_0, \; X \arrow[r, "g"] & Y, \; X \arrow[r, "g_1"] & Y_1 & \uparrow \end{tikzcd}$$

Figure 33b: packing or tripling

The components of a mediator unpack it into a cone. For any mediator f with multi-opspan $((y_{00} : Y_{00} \rightarrow Y_{0\bullet} \leftarrow Y_{01} : y_{01}), (y_{10} : Y_{10} \rightarrow Y_{1\bullet} \leftarrow Y_{11} : y_{11}))$ where $Y_{01} = Y_{10}$, and function $f : X \rightarrow \prod(Y)$, there is a *components* cone $f_{012} = (f_0, f, f_1) : X \rightarrow (Y_0, Y, Y_1)$, which separates the mediator into three 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 multi-opspan morphism of the projection cone $\pi_{(Y0,\, Y,\, Y1)}$ (a component of the counit π):

$$f_{012} = (f_0, f, f_1) = (f, f, f) \cdot \pi_{(Y0,\, Y,\, Y1)} = (f \cdot \pi_{Y0}, f \cdot \pi_Y, f \cdot \pi_{Y1}) : (X, X, X) \rightarrow (\prod(Y), \prod(Y), \prod(Y)) \rightarrow (Y_0, Y, Y_1).$$

The tripling of a cone packs it into a mediator. For any cone g with collection X and multi-opspan morphism $(g_0, g_1) : \Delta(X) \rightarrow Y$, there is a *tripling* mediator $\langle g \rangle = \langle g_0, g, g_1 \rangle : X \rightarrow \prod(Y)$, which triples the images of the three functions of the original multi-opspan morphism. The packing (tripling) process is realized by application of the multipullback ($\prod$) operator to the multi-opspan morphism (g_0, g, g_1) and then composition on the left with the delta function δ_X (a component of the unit δ):

$$\langle g \rangle = \langle g_0, g, g_1 \rangle = \delta_X \cdot \prod(g) : X \rightarrow \prod(X) \rightarrow \prod(Y).$$

These two processes are inverse to each other: $\langle g \rangle_{012} = g$ for each cone g and $\langle f_{012} \rangle = f$ for each mediator f.

```
(27) (UR$morphism packing) (UR$morphism tripling) (= tripling packing)
     (= (UR$source packing) cone)
     (= (UR$target packing) mediator)
     (forall (?g (cone ?g))
         (and (= (function (packing ?g))
                 (KIF.FTN$composition
                     [(delta (collection ?g))
                      (KIF.LIM.MPBK.MOR$multipullback (multi-opspan-morphism ?g))]))
              (= (multi-opspan (packing ?g))
                 (KIF.DGM.MOSPN.MOR$target (multi-opspan-morphism ?g)))))

(28) (UR$morphism unpacking) (UR$morphism components) (= components unpacking)
     (= (UR$source unpacking) mediator)
     (= (UR$target unpacking) cone)
     (forall (?f (mediator ?f))
         (and (= (collection (unpacking ?f))
                 (KIF.FTN$source (function ?f)))
              (= (multi-opspan-morphism (unpacking ?f))
                 (KIF.DGM.MOSPN.MOR$composition
                     [(KIF.LIM.MPBK.MOR$constant (function ?f))
                      (multi-opspan-morphism (projection (multi-opspan ?f)))]))))

(29) (= (UR$composition [packing unpacking]) (UR$identity cone))
     (= (UR$composition [unpacking packing]) (UR$identity mediator))
```

Multipullback Morphisms

`KIF.LIM.MPBK.MOR`

$$\begin{tikzcd}
\prod(X) \arrow[r, "\prod(g)"] \arrow[d, "\pi_{Xk}"'] \arrow[dr, phantom, "k = 0,1,2"] & \prod(Y) \arrow[d, "\pi_{Yk}"] \\
X_k \arrow[r, "g_k"'] & Y_k
\end{tikzcd}$$

Figure 34: Multipullback Function

The constant and multipullback operations are extended to morphisms. Given any function $f : X \rightarrow Y$, there is a constant multi-opspan morphism $(f, f, f) : \Delta(X) = \Delta(Y)$, whose component functions are all f. Given any opspan morphism $g = (g_0, g, g_1) : X \rightarrow Y$, there is a multipullback function $\prod(g) : \prod(X) \rightarrow \prod(Y)$ defined using projections: $\prod(g) \cdot \pi_{Y0} = \pi_{X0} \cdot g_0$, $\prod(g) \cdot \pi_Y = \pi_X \cdot g_2$ and $\prod(g) \cdot \pi_{Y1} = \pi_{X1} \cdot g_1$ (Figure 34).

```
(1)  (UR$morphism constant)
     (= (UR$source constant) KIF.FTN$function)
     (= (UR$target constant) KIF.DGM.MOSPN.MOR$multi-opspan-morphism)
     (= (UR$composition [constant KIF.DGM.MOSPN.MOR$source])
        (UR$composition [KIF.FTN$source KIF.LIM.MPBK.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.MOSPN.MOR$target])
        (UR$composition [KIF.FTN$target KIF.LIM.MPBK.OBJ$constant]))
     (= (UR$composition [constant KIF.DGM.MOSPN.MOR$function0])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [constant KIF.DGM.MOSPN.MOR$function])
        (UR$identity KIF.FTN$function))
     (= (UR$composition [constant KIF.DGM.MOSPN.MOR$function1])
        (UR$identity KIF.FTN$function))

(2)  (UR$morphism limit) (UR$morphism multi-pullback) (= multi-pullback limit)
     (= (UR$source limit) KIF.DGM.MOSPN.MOR$multi-opspan-morphism)
     (= (UR$target limit) KIF.FTN$function)
     (= (UR$composition [limit KIF.FTN$source])
        (UR$composition [KIF.DGM.MOSPN.MOR$source KIF.LIM.MPBK.OBJ$limit]))
     (= (UR$composition [limit KIF.FTN$target])
        (UR$composition [KIF.DGM.MOSPN.MOR$target KIF.LIM.MPBK.OBJ$limit]))
     (forall (?g (KIF.DGM.MOSPN.MOR$multi-opspan-morphism ?g))
         (and (= (KIF.FTN$composition
                     [(limit ?g)
                      (KIF.LIM.MPBK.OBJ$projection0 (KIF.DGM.MOSPN.MOR$target ?g))])
                 (KIF.FTN$composition
                     [(KIF.LIM.MPBK.OBJ$projection0 (KIF.DGM.MOSPN.MOR$source ?g))
                      (KIF.DGM.MOSPN.MOR$function0 ?g)]))
              (= (KIF.FTN$composition
                     [(limit ?g)
                      (KIF.LIM.MPBK.OBJ$projection2 (KIF.DGM.MOSPN.MOR$target ?g))])
                 (KIF.FTN$composition
                     [(KIF.LIM.MPBK.OBJ$projection2 (KIF.DGM.MOSPN.MOR$source ?g))
                      (KIF.DGM.MOSPN.MOR$function ?g)]))
              (= (KIF.FTN$composition
                     [(limit ?g)
                      (KIF.LIM.MPBK.OBJ$projection1 (KIF.DGM.MOSPN.MOR$target ?g))])
                 (KIF.FTN$composition
                     [(KIF.LIM.MPBK.OBJ$projection1 (KIF.DGM.MOSPN.MOR$source ?g))
                      (KIF.DGM.MOSPN.MOR$function1 ?g)]))))
```