The Category Theory Ontology

THE CATEGORY THEORY ONTOLOGY 1
The Namespace of Large Graphs *5*
Graphs 5
Multiplication 6
Unit 7
Graph Morphisms 7
Multiplication 9
Unit 10
2-Dimensional Category Structure 10
Coherence 12
Associative Law 12
Unit Laws 15
The Namespace of Large Categories *17*
Basics 17
Additional Categorical Structure 20
Examples 23
The Namespace of Large Functors *25*
Basics 25
Additional Functorial Structure 27
Quasi-Category Structure 29
Functor Theorems 30
Examples 30
The Namespace of Large Natural Transformations *32*
Natural Transformations 32
2-Dimensional Category Structure 33
Natural Transformation Theorems 36
The Namespace of Large Adjunctions *37*
Adjunctions 37
Adjunction Morphisms 41
Examples 42
The Namespace of Large Monads *43*
Monads and Monad Morphisms 43
Algebras and Freeness 45
The Namespace of Colimits/Limits *52*
Colimits 52
Finite Colimits 52
General Colimits 60
Examples 62
The Namespace of Large Kan Extensions *64*
The Namespace of Large Topoi *64*

As listed in Table 1, the namespaces in the Category Theory Ontology import and use terms from the core namespace of classes and their functions.

Table 1: Terms imported and used from other namespaces

`SET`	‘`conglomerate`’, ‘`class`’, ‘`subclass`’ ‘`function`’, ‘`signature`’, ‘`source`’, ‘`target`’ ‘`composition`’, ‘`identity`’, ‘`inclusion`’ ‘`terminal`’, ‘`unique`’ ‘`opspan`’, ‘`opvertex`’, ‘`opfirst`’, ‘`opsecond`’ ‘`pullback`’, ‘`pullback-projection1`’, ‘`pullback-projection2`’

Table 2 lists the terms defined and axiomatized in the namespaces of the Category Theory Ontology. The core terminology is listed in boldface. As illustrated in Diagram 1, conglomerates characterized the overall architecture for the large aspect of the Category Theory Ontology. Nodes in this diagram represent conglomerates and arrows represent conglomerate functions. The small oval on the right, containing the function and class conglomerates, represents the namespace (‘`SET`’) of large sets (classes) and their functions. The next large oval, containing the conglomerates of Graphs and their Morphisms, represents the large graph namespace (‘`GPH`’). Also indicated are namespaces for categories, functors, natural transformations and adjunctions.

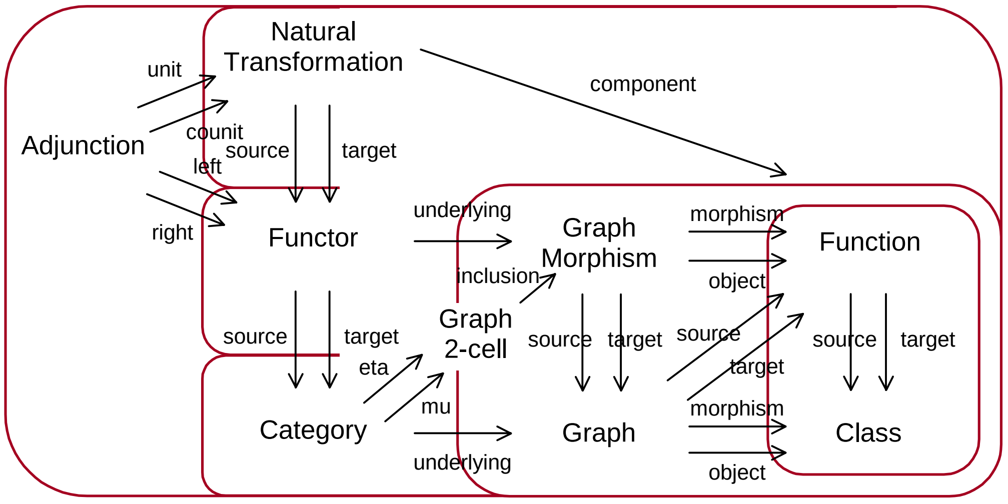


Diagram 1: Core Conglomerates and Functions

Table 2: Terms introduced in the Category Theory Ontology

	CNG$conglomerate	Unary CNG$function	Binary/Ternary CNG$function
GPH	‘**graph**’ ‘**small**’	‘**object**’, ‘**morphism**’, ‘**source**’, ‘**target**’ ‘**opposite**’ ‘**unit**’	‘multiplication-opspan’, ‘**multiplication**’
GPH.MOR	‘**graph-morphism**’ ‘**2-cell**’	‘**source**’, ‘**target**’, ‘**object**’, ‘**morphism**’ ‘**opposite**’ ‘**unit**’ ‘**identity**’ ‘**isomorphism**’, ‘**inverse**’,‘**isomorphic**’ ‘**tau**’ ‘**left**’, ‘**right**’	‘multiplication-cone’, ‘**multiplication**’ ‘**composition**’ ‘first-cone’, ‘opspan12-3’, ‘second-cone’ ‘**alpha**’, ‘**associativity**’
CAT	‘**category**’ ‘**small**’ ‘**monomorphism**’, ‘**epimorphism**’, ‘**isomorphism**’	‘**underlying**’, ‘**mu**’, ‘**eta**’ ‘**composition**’, ‘**identity**’ ‘**object**’, ‘**morphism**’, ‘**source**’, ‘**target**’ ‘composable-opspan’, ‘**composable**’, ‘first’, ‘second’, ‘**opposite**’ ‘object-pair’, ‘object-binary-product’ ‘source-target’,‘hom-object’,‘parallel-pair’ ‘left-composable’, ‘left-composition’ ‘right-composable’, ‘right-composition’	
FUNC	‘**functor**’	‘**source**’, ‘**target**’, ‘**underlying**’ ‘**object**’, ‘**morphism**’ ‘**unique**’,‘**element**’,‘**opposite**’ ‘**identity**’,‘**diagonal**’	‘**composition**’
NAT	‘**natural-transformation**’	‘**source-functor**’, ‘**target-functor**’, ‘**source-category**’, ‘**target-category**’, ‘**component**’ ‘**vertical-identity**’ ‘**horizontal-identity**’	‘**vertical-composition**’ ‘**horizontal-composition**’
ADJ	‘**adjunction**’	‘**underlying-category**’,‘**free-category**’,‘**left-adjoint**’,‘**right-adjoint**’,‘**unit**’,‘**counit**’ ‘**adjunction-monad**’ ‘**free**’,‘**eilenberg-moore-comparison**’, ‘**extension**’,‘**kliesli-comparison**’ ‘**reflection**’,‘**coreflection**’	
ADJ .MOR	‘**conjugate-pair**	‘**source**’,‘**target**’, ‘**left-conjugate**’,‘**right-conjugate**’	
MND	‘**monad**’	‘**underlying-category**’,‘**underlying-functor**’, ‘**unit**’,‘**multiplication**’	
	‘**monad-morphism**’	‘**source**’,‘**target**’, ‘**underlying-natural-transformation**’	
MON .ALG	‘**algebra**’	‘**underlying-object**’,‘**structure-map**’	
	‘**homomorphism**’	‘**source**’,‘**target**’,‘**underlying-morphism**’, ‘composable-opspan’,‘composable’, ‘**composition**’,‘**identity**’	

	‘**eilenberg-moore**’	‘**underlying-eilenberg-moore**’, ‘**free-eilenberg-moore**’, ‘**unit-eilenberg-moore**’, ‘**counit-eilenberg-moore**’, ‘**adjunction-eilenberg-moore**’	
	‘**kliesli**’	‘kliesli-morphism-opspan’, ‘kliesli-identity-cocone’, ‘kliesli-composable-opspan’, ‘extension’, ‘**underlying-kliesli**’, ‘embed’, ‘**free-kliesli**’, ‘**unit-kliesli**’, ‘**counit-kliesli**’, ‘**adjunction-kliesli**’	
COL		‘**initial**’, ‘**counique**’ ‘**diagram**’, ‘**shape** ‘**cocone**’, ‘**cocone-diagram**’, ‘**base**’, ‘**opvertex**’ ‘**colimiting-cocone**’, ‘**colimit**’ ‘**comediator**’ ‘**cocomplete**’, ‘**small-cocomplete**’	
COL .COPRD		‘**diagram**’, ‘**shape** ‘**cocone**’, ‘**cocone-diagram**’, ‘**base**’, ‘**opvertex**’ ‘**colimiting-cocone**’, ‘**binary-coproduct**’, ‘**injection1**’, ‘**injection2**’ ‘**comediator**’	
COL .PSH		‘**diagram**’, ‘**shape** ‘**cocone**’, ‘**cocone-diagram**’, ‘**base**’, ‘**opvertex**’ ‘**colimiting-cocone**’ ‘**pushout**’, ‘**injection1**’, ‘**injection2**’ ‘**comediator**’	

The Namespace of Large Graphs

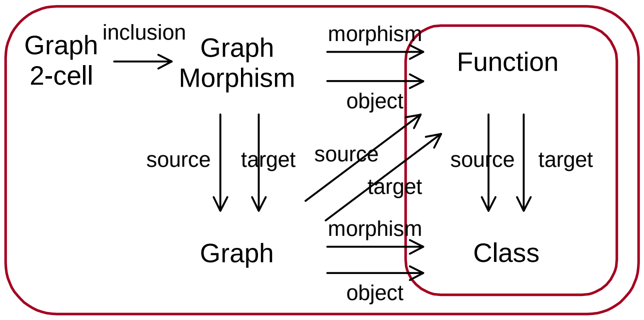
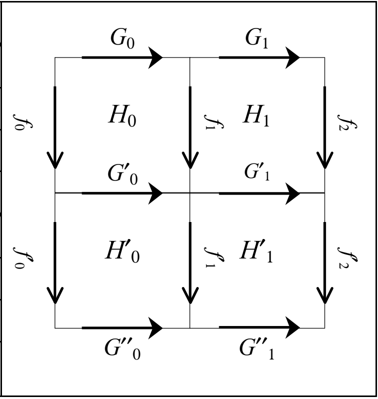


Diagram 1: Core Conglomerates and Functions

Table 1: Elements of the 2-dimensional category GRAPH

square	=	graph morphism	
vertical category	=	**GRAPH** (**Set**)	
vertical morphism	=	function	
vertical composition	=	graph morphism (function) composition	
vertical identity	=	graph morphism (function) identity	
horizontal pseudo-category			
horizontal morphism	=	graph	
horizontal composition	=	graph morphism (graph) multiplication	
horizontal identity	=	graph morphism (graph) unit	

Graphs

GPH

$$\begin{tikzcd} mor(G) \arrow[r, shift left, "src(G)"] \arrow[r, shift right, "tgt(G)"'] & obj(G) \end{tikzcd}$$

Figure 1: Graph

○ A (*large*) *graph* G (Figure 1) consists of a class $obj(G)$ of objects, a class $mor(G)$ of morphisms (arrows), a *source* (domain) function $src(G) : mor(G) \rightarrow obj(G)$ and a *target* (codomain) function $tgt(G) : mor(G) \rightarrow obj(G)$. A morphism $m \in mor(G)$, with source object $src(G)(m) = o_0 \in obj(G)$ and target object $tgt(G)(m) = o_1 \in obj(G)$, is usually represented graphically with the notation $m : o_0 \rightarrow o_1$.

The following is a IFF representation for the elements of a graph. Axiom (1) defines the graph conglomerate. Axioms (2–4) model the graph structure of Figure 7. In axiom (2) and axiom (3), the CNG functions '`object`' and '`morphism`' are used to specify the objects and morphisms of the graph. These are unary, since they take a graph as a parameter – the terms '`(object ?g)`' and '`(morphism ?g)`' are the actual classes. This parametric technique is used throughout the formulation of IFF. The CNG functions '`source`' and '`target`' in axioms (4) and (5) represent the source and target functions that assign objects to morphisms. Graphs are uniquely determined by their (object, morphism, source, target) quadruple.

```
(1) (CNG$conglomerate graph)

(2) (CNG$function object)
    (CNG$signature object graph SET$class)

(3) (CNG$function morphism)
```

```
        (CNG$signature morphism graph SET$class)

    (4) (CNG$function source)
        (CNG$signature source graph SET.FTN$function)
        (forall (?g (graph ?g))
            (and (= (SET.FTN$source (source ?g)) (morphism ?g))
                 (= (SET.FTN$target (source ?g)) (object ?g))))

    (5) (CNG$function target)
        (CNG$signature target graph SET.FTN$function)
        (forall (?g (graph ?g))
            (and (= (SET.FTN$source (target ?g)) (morphism ?g))
                 (= (SET.FTN$target (target ?g)) (object ?g))))

        (forall (?g1 (graph ?g1) ?g2 (graph ?g2))
            (=> (and (= (object ?g1) (object ?g2))
                     (= (morphism ?g1) (morphism ?g2))
                     (= (source ?g1) (source ?g2))
                     (= (target ?g1) (target ?g2)))
                (= ?g1 ?g2)))
```

- To each graph G, there is an *opposite graph* G^{op}. The opposite graph is also called the *dual graph*. The objects of G^{op} are the objects of G, and the morphisms of G^{op} are the morphisms of G. However, the source and target functions are reversed: $src(G^{op}) = tgt(G)$ and $tgt(G^{op}) = src(G)$. The type restriction axiom (6) specifies the notion of the opposite graph.

```
    (6) (CNG$function opposite)
        (CNG$signature opposite graph graph)
        (forall (?g (graph ?g))
            (and (= (object (opposite ?g)) (object ?g))
                 (= (morphism (opposite ?g)) (morphism ?g))
                 (= (source (opposite ?g)) (target ?g))
                 (= (target (opposite ?g)) (source ?g))))
```

An immediate theorem is that the opposite of the opposite is the original graph.

```
        (forall (?g (graph ?g))
            (= (opposite (opposite ?g)) ?g))
```

- A graph G is small when both the object and morphism classes are (small) sets.

```
    (7) (CNG$conglomerate small)
        (CNG$subconglomerate small graph)
        (forall (?g (graph ?g))
            (<=> (small ?g)
                 (and (set$set (object ?g))
                      (set$set (morphism ?g)))))
```

Multiplication

- Two graphs G_0 and G_1 are *composable* when they share a class of objects $obj(G_0) = obj = obj(G_1)$. For two composable graphs there is an associated *multiplication* graph $G_0 \otimes G_1$ (Figure 2), whose class of morphisms is the class of *composable pairs* of morphisms defined above. This is the pullback in foundations along the target function $tgt(G_0) : mor(G_0) \rightarrow obj$ and the source function $src(G_1) : mor(G_1) \rightarrow obj$.

$$\begin{tikzcd}
 & & mor(G_0)\times_{obj} mor(G_1) \arrow[dl, "1st"'] \arrow[dr, "2nd"] & & \\
 & mor(G_0) \arrow[dl, "src(G_0)"'] \arrow[dr, "tgt(G_0)"] & & mor(G_1) \arrow[dl, "src(G_1)"'] \arrow[dr, "tgt(G_1)"] & \\
obj & & obj & & obj
\end{tikzcd}$$

Figure 2: Multiplication Graph

$mor(G_0) \times_{Obj} mor(G_1) =$
$\{(m_0, m_1) \mid m_0 \in mor(G_0) \text{ and } m_1 \in mor(G_1) \text{ and } tgt(R_0)(m_0) = src(R_1)(m_1)\}$

Here is the KIF formalism. Each composable pair of graphs has an auxiliary foundational opspan represented as the IFF term '`(multiplication-opspan ?g0 ?g1)`' and axiomatized in (8). This opspan allows us to refer to the appropriate foundational pullback. The multiplication graph $G_0 \otimes G_1$ is represented as the IFF term '`(multiplication ?g0 ?g1)`' and axiomatized in (9).

```
(8) (CNG$function multiplication-opspan)
    (CNG$signature multiplication-opspan graph graph SET.LIM.PBK$opspan)
    (forall (?g0 (graph ?g0) ?g1 (graph ?g1))
          (<=> (exists (?s (SET.LIM.PBK$opspan ?s))
                   (= (multiplication-opspan ?g0 ?g1) ?s))
               (= (object ?g0) (object ?g1))))
    (forall (?g0 (graph ?g0) ?g1 (graph ?g1))
          (=> (= (object ?g0) (object ?g1))
              (and (= (SET.LIM.PBK$opvertex (multiplication-opspan ?g0 ?g1))
                      (object ?g0))
                   (= (SET.LIM.PBK$opfirst (multiplication-opspan ?g0 ?g1))
                      (target ?g0))
                   (= (SET.LIM.PBK$opsecond (multiplication-opspan ?g0 ?g1))
                      (source ?g1)))))

(9) (CNG$function multiplication)
    (CNG$signature multiplication graph graph graph)
    (forall (?g0 (graph ?g0) ?g1 (graph ?g1))
        (<=> (exists (?g (graph ?g))
                 (= (multiplication ?g0 ?g1) ?g))
             (= (object ?g0) (object ?g1))))
    (forall (?g0 (graph ?g0) ?g1 (graph ?g1))
        (=> (= (object ?g0) (object ?g1))
            (and (= (object (multiplication ?g0 ?g1))
                    (object ?g0))
                 (= (morphism (multiplication ?g0 ?g1))
                    (SET.LIM.PBK$pullback (multiplication-opspan ?g0 ?g1)))
                 (= (source (multiplication ?g0 ?g1))
                    (SET$composition
                        (SET.LIM.PBK$projection1 (multiplication-opspan ?g0 ?g1))
                        (source ?g0)))
                 (= (target (multiplication ?g0 ?g1))
                    (SET$composition
                        (SET.LIM.PBK$projection2 (multiplication-opspan ?g0 ?g1))
                        (target ?g1))))))
```

Unit

○ For any class (of objects) C there is a *unit* graph 1_C (Figure 3) whose classes of objects and morphisms are C, and whose source and target functions is the SET identity function on C. The unit graph has the following formalization.

```
(10) (CNG$function unit)
     (CNG$signature unit SET$class graph)
     (forall (?c (SET$class ?c))
         (and (= (object (unit ?c)) ?c)
              (= (morphism (unit ?c)) ?c)
              (= (source (unit ?c)) (SET.FTN$identity ?c))
              (= (target (unit ?c)) (SET.FTN$identity ?c))))
```

$$\begin{tikzcd} & C \arrow[dl, "id_C"'] \arrow[dr, "id_C"] & \\ C & & C \end{tikzcd}$$

$$\begin{tikzcd} C \arrow[r, shift left, "id_C"] \arrow[r, shift right, "id_C"'] & C \end{tikzcd}$$

Figure 3: Unit Graph

An immediate theorem is that the opposite of the unit graph is itself.

```
     (forall (?c (SET$class ?c))
         (= (opposite (unit ?c)) (unit ?c)))
```

Graph Morphisms

`GPH.MOR`

○ A *graph morphism* (Figure 4) $H : G \Rightarrow G'$ from graph G to graph G' consists of two functions, an *object function* and a *morphism function*, that preserve source and target (the diagram in Figure 4 is commutative). The object function $obj(H) : Obj(G) \rightarrow Obj(G')$ assigns to each object of G an object of G', and the morphism function $mor(H) : Mor(G) \rightarrow Mor(G')$ assigns to each morphism of G

$$\begin{tikzcd} Mor(G) \arrow[r, shift left, "src(G)"] \arrow[r, shift right, "tgt(G)"'] \arrow[d, "mor(H)"'] & Obj(G) \arrow[d, "obj(H)"] \\ Mor(G') \arrow[r, shift left, "src(G')"] \arrow[r, shift right, "tgt(G')"'] & Obj(G') \end{tikzcd}$$

Figure 4: Graph Morphism

a morphism of G'. In the graph morphism that is presented in Figure 4, the diagram is asserted to be commutative. What this means is that the component functions must preserve source and target in the sense that the following constraints must be satisfied.

$$mor(H) \cdot src(G') = src(G) \cdot obj(H) \text{ and } mor(H) \cdot tgt(G') = tgt(G) \cdot obj(H)$$

The following is a formalization of a graph morphism. The CNG functions '`source`' and '`target`' represent source (domain) and target (codomain) operations that assign graphs to graph morphisms. The SET function '`(object ?h)`' specifies the object function of a graph morphism '`?h`', and the SET function '`(morphism ?h)`' specifies the morphism function of the graph morphism. The CNG functions '`object`' and '`morphism`' have the graph morphism as a parameter. The last axiom represents preservation of source and target. Graph morphisms are uniquely determined by their (source, target, object, morphism) quadruple.

```
(1) (CNG$conglomerate graph-morphism)

(2) (CNG$function source)
    (CNG$signature source graph-morphism GPH$graph)

(3) (CNG$function target)
    (CNG$signature target graph-morphism GPH$graph)

(4) (CNG$function object)
    (CNG$signature object graph-morphism SET.FTN$function)
    (forall (?h (graph-morphism ?h))
        (and (= (SET.FTN$source (object ?h)) (GPH$object (source ?h)))
             (= (SET.FTN$target (object ?h)) (GPH$object (target ?h)))))

(5) (CNG$function morphism)
    (CNG$signature morphism graph-morphism SET.FTN$function)
    (forall (?h (graph-morphism ?h))
        (and (= (SET.FTN$source (morphism ?h)) (GPH$morphism (source ?h)))
             (= (SET.FTN$target (morphism ?h)) (GPH$morphism (target ?h)))))

    (forall (?h (graph-morphism ?h))
        (and (= (SET.FTN$composition (morphism ?h) (GPH$source (target ?h)))
                (SET.FTN$composition (GPH$source (source ?h)) (object ?h)))
             (= (SET.FTN$composition (morphism ?h) (GPH$target (target ?h)))
                (SET.FTN$composition (GPH$target (source ?h)) (object ?h)))))

    (forall (?h1 (graph-morphism ?h1) ?h2 (graph-morphism ?h2))
        (=> (and (= (source ?h1) (source ?h2))
                 (= (target ?h1) (target ?h2)))
                 (= (object ?h1) (object ?h2))
                 (= (morphism ?h1) (morphism ?h2))
            (= ?h1 ?h2)))
```

○ To each graph morphism $H : G \Rightarrow G'$, there is an *opposite graph morphism* $H^{op} : G^{op} \Rightarrow G'^{op}$. The opposite graph morphism is also called the *dual graph morphism*. The object function of H^{op} is the object function of H, and the morphism function of H^{op} is the morphism function of H. However, the source and target graphs are the opposite: $src(H^{op}) = src(H)^{op}$ and $tgt(H^{op}) = tgt(H)^{op}$. Axiom (6) specifies an opposite graph morphism.

```
(6) (CNG$function opposite)
    (CNG$signature opposite graph-morphism graph-morphism)
    (forall (?h (graph-morphism ?h))
        (and (= (source (opposite ?h)) (GPH$opposite (source ?h)))
             (= (target (opposite ?h)) (GPH$opposite (target ?h)))
             (= (object (opposite ?h)) (object ?h))
             (= (morphism (opposite ?h)) (morphism ?h))))
```

An immediate theorem is that the opposite of the opposite of a graph morphism is the original graph morphism.

```
    (forall (?h (graph-morphism ?h))
        (= (opposite (opposite ?h)) ?h))
```

Multiplication

○ The multiplication operation on graphs can be extended to graph morphisms. For any two graphs morphisms $H_0 : G_0 \Rightarrow G'_0$ and $H_1 : G_1 \Rightarrow G'_1$, which are horizontally composable, in that they share a common object function $obj(H_0) = obj = obj(H_1)$, there is a *multiplication* graph morphism $H_1 \otimes H_2 : G_0 \otimes G_1 \Rightarrow G'_0 \otimes G'_1$, (Figure 5) whose object function is the common object function and whose morphism function is determined by pullback. This is the SET pullback along the morphism functions of H_0 and H_1.

The multiplication graph morphism $H_0 \otimes H_1$ is represented as the SET term '`(multiplication ?h0 ?h1)`'. The formalism for the multiplication of graphs used an auxiliary associated pullback diagram (opspan). In comparison and contrast, the formalism for the multiplication of graph morphisms uses an auxiliary pullback cone represented as the term '`(multiplication-cone ?h0 ?h1)`' and axiomatized in (7). The multiplication span morphism $H_0 \otimes H_1$ is represented as the term '`(multiplication ?h0 ?h1)`' and axiomatized in (8).

$$\begin{tikzcd}
 & & Mor(G_0)\times_{Obj}Mor(G_1) \arrow[dl, "1st"'] \arrow[dr, "2nd"] \arrow[dd, "pair"] & & \\
 & Mor(G_0) \arrow[ddl, "src(G_0)"'] \arrow[ddr, "tgt(G_0)"] \arrow[ddd, "mor(H_0)"] & & Mor(G_1) \arrow[ddl, "src(G_1)"'] \arrow[ddr, "tgt(G_1)"] \arrow[ddd, "mor(H_1)"] & \\
 & & Mor(G'_0)\times_{Obj}Mor(G'_1) \arrow[ddl, "1st'"'] \arrow[ddr, "2nd'"] & & \\
Obj \arrow[dd, "obj"'] & & Obj \arrow[dd, "obj"] & & Obj \arrow[dd, "obj"] \\
 & Mor(G'_0) \arrow[dl, "src(G'_0)"'] \arrow[dr, "tgt(G'_0)"] & & Mor(G'_1) \arrow[dl, "src(G'_1)"'] \arrow[dr, "tgt(G'_1)"] & \\
Obj' & & Obj' & & Obj'
\end{tikzcd}$$

Figure 5: Multiplication Graph Morphism

```
(7) (CNG$function multiplication-cone)
    (CNG$signature multiplication-cone graph-morphism graph-morphism SET.LIM.PBK$cone)
    (forall (?h0 (graph-morphism ?h0) ?h1 (graph-morphism ?h1))
        (<=> (exists (?r (SET.LIM.PBK$cone ?r))
                 (= (multiplication-cone ?h0 ?h1) ?r))
             (= (object ?h0) (object ?h1))))
    (forall (?h0 (graph-morphism ?h0) ?h1 (graph-morphism ?h1))
        (=> (= (object ?h0) (object ?h1))
            (and (= (SET.LIM.PBK$cone-diagram (multiplication-cone ?h0 ?h1))
                    (GPH$multiplication-opspan (target ?h0) (target ?h1)))
                 (= (SET.LIM.PBK$vertex (multiplication-cone ?h0 ?h1))
                    (SET.LIM.PBK$pullback
                        (GPH$multiplication-opspan (source ?h0) (source ?h1))))
                 (= (SET.LIM.PBK$first (multiplication-cone ?h0 ?h1))
                    (SET.FTN$composition
                        (SET.LIM.PBK$projection1
                            (GPH$multiplication-opspan (source ?h0) (source ?h1)))
                        (morphism ?h0)))
                 (= (SET.LIM.PBK$second (multiplication-cone ?h0 ?h1))
                    (SET$composition
                        (SET.LIM.PBK$projection2
                            (GPH$multiplication-opspan (source ?h0) (source ?h1)))
                        (morphism ?h1))))))

(8) (CNG$function multiplication)
    (CNG$signature multiplication graph-morphism graph-morphism graph-morphism)
    (forall (?h0 (graph-morphism ?h0) ?h1 (graph-morphism ?h1))
        (<=> (exists (?h (graph-morphism ?h))
                 (= (multiplication ?h0 ?h1) ?h))
             (= (object ?h0) (object ?h1))))
    (forall (?h0 (graph-morphism ?h0) ?h1 (graph-morphism ?h1))
        (=> (= (object ?h0) (object ?h1))
            (and (= (source (multiplication ?h0 ?h1))
                    (GPH$multiplication (source ?h0) (source ?h1)))
                 (= (target (multiplication ?h0 ?h1))
                    (GPH$multiplication (target ?h0) (target ?h1)))
                 (= (object (multiplication ?h0 ?h1))
                    (object ?h0))
                 (= (morphism (multiplication ?h0 ?h1))
                    (SET.LIM.PBK$mediator (multiplication-cone ?h0 ?h1))))))
```

Unit

○ For any function (of objects) $f : A \rightarrow B$ there is a *unit* graph morphism (Figure 6) $1_f : 1_A \Rightarrow 1_B$, whose source and target graphs are the unit graphs for A and B, and whose object and morphism functions are f. The unit graph morphism has the following representation.

```
(9) (CNG$function unit)
    (CNG$signature unit SET.FTN$function graph-morphism)
    (forall (?f (SET.FTN$function ?f))
        (and (= (source (unit ?f)) (GPH$unit (SET.FTN$source ?f)))
             (= (target (unit ?f)) (GPH$unit (SET.FTN$target ?f)))
             (= (object (unit ?f)) ?f)
             (= (morphism (unit ?f)) ?f)))
```

$$\begin{tikzcd} A \arrow[r, shift left, "Id_A"] \arrow[r, shift right, "Id_A"'] \arrow[d, "f"'] & A \arrow[d, "f"] \\ B \arrow[r, shift left, "Id_B"] \arrow[r, shift right, "Id_B"'] & B \end{tikzcd}$$

Figure 6: Unit Graph Morphism

It is clear that the opposite of the unit graph morphism is itself.

```
    (forall (?f (SET.FTN$function ?f))
        (= (opposite (unit ?f)) (unit ?f)))
```

2-Dimensional Category Structure

○ A pair of graph morphisms H_1 and H_2 is *composable* when the target graph of H_1 is the source graph of H_2. For any composable pair of graph morphisms $H_1 : G_0 \Rightarrow G_1$ and $H_2 : G_1 \Rightarrow G_2$ there is a *composition graph morphism* $H_1 \circ H_2 : G_0 \Rightarrow G_2$. Its object and morphism functions are the compositions of the object and morphism functions of the component graph morphisms, respectively. For any graph G there is an *identity graph morphism* $id_G : G \Rightarrow G$ on that graph. Its object and morphism functions are the identity functions on the object and morphism classes of that graph, respectively. The following represents the declaration of composition and identity and the equivalence of composability.

```
(10) (CNG$function composition)
     (CNG$signature composition graph-morphism graph-morphism graph-morphism)
      (forall (?h1 (graph-morphism ?h1) ?h2 (graph-morphism ?h2))
         (<=> (exists (?h (graph-morphism ?h)) (= ?h (composition ?h1 ?h2)))
              (= (target ?h1) (source ?h2))))

(11) (CNG$function identity)
     (CNG$signature identity GPH$graph graph-morphism)
```

These graph morphism operations satisfy the following typing constraints.

$$src(id_G) = G = tgt(id_G),\ src(H_1 \circ H_2) = src(H_1),\ tgt(H_1 \circ H_2) = tgt(H_2)$$

for all graphs G and all composable pairs of graph morphisms H_1 and H_2. These theorems are expressed in KIF with the following formalism.

```
(12) (forall (?h1 (graph-morphism ?h1) ?h2 (graph-morphism ?h2))
         (=> (= (target ?h1) (source ?h2))
             (and (= (source (composition ?h1 ?h2)) (source ?h1))
                  (= (target (composition ?h1 ?h2)) (target ?h2)))))

(13) (forall (?g (GPH$graph ?g))
         (and (= (source (identity ?g)) ?g)
              (= (target (identity ?g)) ?g)))
```

○ Graph morphism composition and identity are defined componentwise in axioms (14–15). The preservation of source and target functions for composite and identity graph morphisms follows from this as theorems.

```
(14) (forall (?h1 (graph-morphism ?h1) ?h2 (graph-morphism ?h2))
         (=> (= (target ?h1) (source ?h2))
             (and (= (object (composition ?h1 ?h2))
                     (SET.FTN$composition (object ?h1) (object ?h2)))
                  (= (morphism (composition ?h1 ?h2))
                     (SET.FTN$composition (morphism ?h1) (morphism ?h2))))))

(15) (forall (?g (GPH$graph ?g))
         (and (= (object (identity ?g))
                 (SET.FTN$identity (GPH$object ?g)))
              (= (morphism (identity ?g))
                 (SET.FTN$identity (GPH$morphism ?g)))))
```

It can be shown that graph morphism composition satisfies the following associative law

$(H_1 \circ H_2) \circ H_3 = H_1 \circ (H_2 \circ H_3)$

for all composable pairs of graph morphisms (H_1, H_2) and (H_2, H_3), and graph morphism identity satisfies the following identity laws

$Id_{G0} \cdot H = H$ and $H = H \cdot Id_{G1}$

for any graph morphism $H : G_0 \Rightarrow G_1$ with source graph G_0 and target graph G_1. Graphs as objects and graph morphisms as morphisms form a quasi-category ("quasi" since this is at the level of conglomerates in foundations). This has the following expression in an external namespace.

```
     (forall (?h1 (GPH.MOR$graph-morphism ?h1)
              ?h2 (GPH.MOR$graph-morphism ?h2)
              ?h3 (GPH.MOR$graph-morphism ?h3))
         (=> (and (= (GPH.MOR$target ?h1) (GPH.MOR$source ?h2))
                  (= (GPH.MOR$target ?h2) (GPH.MOR$source ?h3)))
             (= (GPH.MOR$composition (GPH.MOR$composition ?h1 ?h2) ?h3)
                (GPH.MOR$composition ?h1 (GPH.MOR$composition ?h2 ?h3)))))

     (forall (?h (GPH.MOR$graph-morphism ?h))
         (and (= (GPH.MOR$composition (GPH.MOR$identity (GPH.MOR$source ?h)) ?h) ?h)
              (= (GPH.MOR$composition ?h (GPH.MOR$identity (GPH.MOR$target ?h))) ?h)))
```

○ Two oppositely directed graph morphisms $H : G_0 \rightarrow G_1$ and $H' : G_1 \rightarrow G_0$ are *inverses* of each other when $H \circ H' = id_{G0}$ and $H' \circ H = id_{G1}$. *A graph isomorphism* is a graph morphism that has an inverse. With these laws, we can prove the theorem that an inverse to a graph morphism is unique. The *inverse* function maps an isomorphism to its inverse (another isomorphism). This is a bijection. Two graphs are said to be *isomorphic* when there is a graph isomorphism between them.

```
(16) (CNG$conglomerate isomorphism)
     (CNG$subconglomerate isomorphism graph-morphism)
     (forall (?h (graph-morphism ?h))
         (<=> (isomorphism ?h)
              (exists (?h1 (graph-morphism ?h1))
                  (and (= (source ?h1) (target ?h))
                       (= (target ?h1) (source ?h))
                       (= (composition ?h ?h1) (identity (source ?h)))
                       (= (composition ?h1 ?h) (identity (target ?h)))))))

     (forall (?h (graph-morphism ?h)
              ?h1 (graph-morphism ?h1) ?h2 (graph-morphism ?h2))
         (=> (and (= (source ?h1) (target ?h)) (= (source ?h2) (target ?h))
                  (= (target ?h1) (source ?h)) (= (target ?h2) (source ?h))
                  (= (composition ?h ?h1) (identity (source ?h)))
                  (= (composition ?h ?h2) (identity (source ?h)))
                  (= (composition ?h1 ?h) (identity (target ?h)))
                  (= (composition ?h2 ?h) (identity (target ?h))))
             (= ?h1 ?h2)))

(17) (CNG$function inverse)
     (CNG$signature inverse isomorphism isomorphism)
     (forall (?h (isomorphism ?h))
         (= (inverse ?h)
            (the (?h1 (isomorphism ?h1))
                (and (= (source ?h1) (target ?h))
                     (= (target ?h1) (source ?h))
                     (= (composition ?h ?h1) (identity (source ?h)))
                     (= (composition ?h1 ?h) (identity (target ?h)))))))

     (forall (?h (isomorphism ?h))
         (= (inverse (inverse ?h)) ?h))
```

```
(18) (CNG$relation isomorphic)
     (CNG$signature isomorphic GPH$graph GPH$graph)
     (forall ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (<=> (isomorphic ?g1 ?g2)
              (exists (?h) (and (isomorphism ?h)
                                (= (source ?h) ?g1)
                                (= (target ?h) ?g2)))))
```

- A *graph 2-cell* $H : G \rightarrow G'$ from graph G to graph G' is a graph morphism whose object function is an identity. This means that the source and target graphs have the same object class $obj(G) = obj = obj(G')$.

```
(19) (CNG$conglomerate 2-cell)
     (CNG$subconglomerate 2-cell graph-morphism)

     (forall (?h (graph-morphism ?h))
         (<=> (2-cell ?h)
              (and (= (GPH$object (source ?h)) (GPH$object (target ?h)))
                   (= (object ?h) (SET.FTN$identity (GPH$object (source ?h)))))))
```

- The opposite of the multiplication of two graphs is isomorphic to the multiplication of the opposites of the component graphs. This isomorphism is mediated by the *tau* or *twist* graph morphism, which is both an isomorphism and a 2-cell.

  $$\tau_{G0,\,G1} : G_1^{\mathrm{op}} \otimes G_0^{\mathrm{op}} \rightarrow (G_0 \otimes G_1)^{\mathrm{op}}.$$

  The morphism function of tau is the SET tau function '`(tau ?s)`' for the multiplication pullback diagram (opspan) '`?s`'. Axiom (20) is the definition of the tau graph morphism.

```
(20) (CNG$function tau)
     (CNG$signature tau GPH$graph GPH$graph-morphism)
     (forall (?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (= (GPH$object ?g1) (GPH$object ?g2))
             (and (= (source (tau ?g1 ?g2))
                     (GPH$multiplication (GPH$opposite ?g2) (GPH$opposite ?g1)))
                  (= (target (tau ?g1 ?g2))
                     (GPH$opposite (GPH$multiplication ?g1 ?g2)))
                  (= (object (tau ?g1 ?g2))
                     (SET.FTN$identity (GPH$object ?g1)))
                  (= (morphism (tau ?g1 ?g2))
                     (SET.LIM.PBK$tau (GPH$multiplication-opspan ?g1 ?g2))))))

     (forall (?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (= (GPH$object ?g1) (GPH$object ?g2))
             (and (isomorphism (tau ?g1 ?g2))
                  (2-cell (tau ?g1 ?g2)))))
                  (isomorphic
                      (GPH$opposite (GPH$multiplication ?g1 ?g2))
                      (GPH$multiplication (GPH$opposite ?g2) (GPH$opposite ?g1))))))
```

Coherence

Associative Law

For any three graphs G_0, G_1 and G_2, where G_0 and G_1 are horizontally composable and G_1 and G_2 are horizontally composable – all three graphs share a common class of objects $Obj(G_0) = Obj(G_1) = Obj(G_2) = Obj$ – an associative law for graph multiplication would say that $G_0 \otimes (G_1 \otimes G_2) = (G_0 \otimes G_1) \otimes G_2$. However, this is too strong. What we can say is that the graph $G_0 \otimes (G_1 \otimes G_2)$ and the graph $(G_0 \otimes G_1) \otimes G_2$ are isomorphic. The definition for the appropriate associative graph isomorphic 2-cell

$$\alpha_{G0,\,G1,\,G2} : G_0 \otimes (G_1 \otimes G_2) \rightarrow (G_0 \otimes G_1) \otimes G_2$$

is illustrated in Figure 7.

$$\begin{tikzcd}
 & Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \arrow[rrrr, shift left, "mor(\alpha_{G0,\,G1,\,G2})"] \arrow[dd, "1st_{01}'"'] \arrow[dr, "mediator_{01}"] \arrow[drrr, "1st_{12}'"] & & & & (Mor(G_0)\times_{Obj}Mor(G_1))\times_{Obj}Mor(G_2) \arrow[llll, shift left, "mor(\alpha'_{G0,\,G1,\,G2})"] \arrow[dlll, "2nd_{01}'"'] \arrow[dd, "2nd_{12}'"] & \\
 & & Mor(G_0)\times_{Obj}Mor(G_1) \arrow[dl, "1st_{01}"] \arrow[dr, "2nd_{01}"] & & Mor(G_1)\times_{Obj}Mor(G_2) \arrow[dl, "1st_{12}"] \arrow[dr, "2nd_{12}"] & & \\
 & Mor(G_0) \arrow[dl, "src(G_0)"'] \arrow[dr, "tgt(G_0)"] & & Mor(G_1) \arrow[dl, "src(G_1)"'] \arrow[dr, "tgt(G_1)"] & & Mor(G_2) \arrow[dl, "src(G_2)"'] \arrow[dr, "tgt(G_2)"] & \\
Obj & & Obj & & Obj & & Obj
\end{tikzcd}$$

Figure 7: Associativity Graph Isomorphism

In order to define the morphism function

$$mor(\alpha_{G0,\,G1,\,G2}) : Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \rightarrow (Mor(G_0)\times_{Obj}Mor(G_1))\times_{Obj}Mor(G_2),$$

we need to specify the following auxiliary components for the associative law.

- The cone *first-cone* consists of
  - vertex: $Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2))$,
  - first function: $1st_{01}' : Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \rightarrow Mor(G_0)$,
  - second function: $2nd_{01}' \cdot 1st_{12} : Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \rightarrow Mor(G_1)$, and
  - opspan: opspan of the multiplication $G_0 \otimes G_1$.
- The opspan *opspan12-3* consists of
  - opvertex: *Obj*,
  - opfirst function: $2nd_{01} \cdot tgt(G_1) : Mor(G_0)\times_{Obj}(Mor(G_1) \rightarrow Obj$, and
  - opsecond function: $src(G_2) : Mor(G_2) \rightarrow Obj$.
- The cone *second-cone* consists of
  - vertex: $Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2))$,
  - first function: $mediator_{01} : Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \rightarrow Mor(G_0)\times_{Obj}(Mor(G_1)$ of *first-cone*,
  - second function: $2nd_{01}' \cdot 2nd_{12} : Mor(G_0)\times_{Obj}(Mor(G_1)\times_{Obj}Mor(G_2)) \rightarrow Mor(G_2)$, and
  - opspan: *opspan12-3*.

The morphism function $mor(\alpha_{G0,\,G1,\,G2})$ is the mediator function of the pullback cone *second-cone*. For convenience of reference, this morphism is called the *associativity* morphism. This is represented as the ternary CNG function '`(associativity ?g0 ?g1 ?g2)`'.

```
(21) (CNG$function first-cone)
     (CNG$signature first-cone GPH$graph GPH$graph GPH$graph SET.LIM.PBK$cone)
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (<=> (exists (?r (SET.LIM.PBK$cone ?r))
                  (= (first-cone ?g0 ?g1 ?g2) ?r))
              (and (= (GPH$object ?g0) (GPH$object ?g1))
                   (= (GPH$object ?g1) (GPH$object ?g2)))))
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (and (= (GPH$object ?g0) (GPH$object ?g1))
                  (= (GPH$object ?g1) (GPH$object ?g2)))
             (and (= (SET.LIM.PBK$vertex (first-cone ?g0 ?g1 ?g2))
                     (SET.LIM.PBK$pullback
                         (GPH$multiplication-opspan ?g0 (GPH$multiplication ?g1 ?g2))))
                  (= (SET.LIM.PBK$first (first-cone ?g0 ?g1 ?g2))
                     (SET.LIM.PBK$projection1
                         (GPH$multiplication-opspan ?g0 (GPH$multiplication ?g1 ?g2))))
                  (= (SET.LIM.PBK$second (first-cone ?g0 ?g1 ?g2))
                     (SET.FTN$composition
                         (SET.LIM.PBK$projection2
                            (GPH$multiplication-opspan ?g0 (GPH$multiplication ?g1 ?g2)))
                         (SET.LIM.PBK$projection1 (GPH$multiplication-opspan ?g1 ?g2))))
                  (= (SET.LIM.PBK$cone-diagram (first-cone ?g0 ?g1 ?g2))
                     (GPH$multiplication-opspan ?g0 ?g1)))))

(22) (CNG$function opspan12-3)
     (CNG$signature opspan12-3 GPH$graph GPH$graph GPH$graph SET.LIM.PBK$opspan)
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (<=> (exists (?s (SET.LIM.PBK$opspan ?s))
                  (= (opspan12-3 ?g0 ?g1 ?g2) ?s))
              (and (= (GPH$object ?g0) (GPH$object ?g1))
                   (= (GPH$object ?g1) (GPH$object ?g2)))))
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (and (= (GPH$object ?g0) (GPH$object ?g1))
                  (= (GPH$object ?g1) (GPH$object ?g2)))
             (and (= (SET$opvertex (opspan12-3 ?g0 ?g1 ?g2))
                     (GPH$object ?g1))
                  (= (SET$opfirst (opspan12-3 ?g0 ?g1 ?g2))
                     (SET.FTN$composition
                         (SET.LIM.PBK$projection2 (GPH$multiplication-opspan ?g0 ?g1))
                         (target ?g1)))
                  (= (SET$opsecond (opspan12-3 ?g0 ?g1 ?g2))
                     (source ?g2)))))

(23) (CNG$function second-cone)
     (CNG$signature second-cone GPH$graph GPH$graph GPH$graph SET.LIM.PBK$cone)
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (<=> (exists (?r (SET.LIM.PBK$cone ?r))
                  (= (second-cone ?g0 ?g1 ?g2) ?r))
              (and (= (GPH$object ?g0) (GPH$object ?g1))
                   (= (GPH$object ?g1) (GPH$object ?g2)))))
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (and (= (GPH$object ?g0) (GPH$object ?g1))
                  (= (GPH$object ?g1) (GPH$object ?g2)))
             (and (= (SET$vertex (second-cone ?g0 ?g1 ?g2))
                     (SET.LIM.PBK$pullback
                         (GPH$multiplication-opspan ?g0 (GPH$multiplication ?g1 ?g2))))
                  (= (SET.LIM.PBK$first (second-cone ?g0 ?g1 ?g2))
                     (SET.LIM.PBK$mediator (first-cone ?g0 ?g1 ?g2)))
                  (= (SET.LIM.PBK$second (second-cone ?g0 ?g1 ?g2))
                     (SET.FTN$composition
                         (SET.LIM.PBK$projection2
                            (GPH$multiplication-opspan ?g0 (GPH$multiplication ?g1 ?g2)))
                         (SET.LIM.PBK$projection2 (GPH$multiplication-opspan ?g1 ?g2))))
                  (= (SET.LIM.PBK$cone-diagram (second-cone ?g0 ?g1 ?g2))
                     (opspan12-3 ?g0 ?g1 ?g2)))))

(24) (CNG$function alpha)
     (CNG$signature alpha GPH$graph GPH$graph GPH$graph graph-morphism)
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (<=> (exists (?h (graph-morphism ?h))
                  (= (alpha ?g0 ?g1 ?g2) ?h))
              (and (= (GPH$object ?g0) (GPH$object ?g1))
                   (= (GPH$object ?g1) (GPH$object ?g2)))))
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (and (= (GPH$object ?g0) (GPH$object ?g1))
                  (= (GPH$object ?g1) (GPH$object ?g2)))
             (and (= (source (alpha ?g0 ?g1 ?g2))
                     (GPH$multiplication ?g0 (GPH$multiplication ?g1 ?g2)))
                  (= (target (alpha ?g0 ?g1 ?g2))
                     (GPH$multiplication (GPH$multiplication ?g1 ?g0) ?g2))
                  (= (object (alpha ?g0 ?g1 ?g2))
                     (SET.FTN$identity (GPH$object ?g0)))
                  (= (morphism (alpha ?g0 ?g1 ?g2))
                     (SET.LIM.PBK$mediator (second-cone ?g0 ?g1 ?g2))))))
```

```
(25) (CNG$function associativity)
     (CNG$signature associativity GPH$graph GPH$graph GPH$graph SET.FTN$function)
     (forall (?g0 (GPH$graph ?g0) ?g1 (GPH$graph ?g1) ?g2 (GPH$graph ?g2))
         (=> (and (= (GPH$object ?g0) (GPH$object ?g1))
                  (= (GPH$object ?g1) (GPH$object ?g2)))
             (= (associativity ?g0 ?g1 ?g2) (morphism (alpha ?g0 ?g1 ?g2)))))
```

The oppositely directed graph morphism $\alpha' : (G_0 \otimes G_1) \otimes G_2 \to G_0 \otimes (G_1 \otimes G_2)$ can be defined in a similar fashion, and, based upon uniqueness of the pullback mediator function, the two can be shown to be inverses. In addition, the associative coherence theorem in Diagram 3 can be proven.

$$\begin{tikzcd}
G_0 \otimes (G_1 \otimes (G_2 \otimes G_3)) \arrow[r, "\alpha"] \arrow[d, "G_0 \otimes \alpha"'] & (G_0 \otimes G_1) \otimes (G_2 \otimes G_3) \arrow[r, "\alpha"] & ((G_0 \otimes G_1) \otimes G_2) \otimes G_3 \arrow[d, "\alpha \otimes G_3"] \\
G_0 \otimes ((G_1 \otimes G_2) \otimes G_3) \arrow[rr, "\alpha"] & & (G_0 \otimes (G_1 \otimes G_2)) \otimes G_3
\end{tikzcd}$$

Diagram 3: Associativity Coherence

Unit Laws

For any graph G the unit laws for graph multiplication would say that $1_{Obj(G)} \otimes G = G = G \otimes 1_{Obj(G)}$. However, these are too strong. What we can say is that the graphs $1_{Obj(G)} \otimes G$ and G are isomorphic, and that the graphs $G \otimes 1_{Obj(G)}$ and G are isomorphic. The definitions for the appropriate graph isomorphic 2-cells, *left unit* $\lambda_G : 1_{Obj(G)} \otimes G \to G$ and *right unit* $\rho_G : G \otimes 1_{Obj(G)} \to G$, are illustrated in Figure 8.

$$\begin{tikzcd}
& & Obj(G) \times_{Obj(G)} Mor(G) \arrow[dl, "1st"'] \arrow[dr, shift left, "mor(\lambda_G) = 2nd"] & & \\
& Obj(G) \arrow[dl, "Id_{Obj(G)}"'] \arrow[dr, "Id_{Obj(G)}"] & & Mor(G) \arrow[ul, shift left, "mor(\lambda'_G)"] \arrow[dl, "src(G)"'] \arrow[dr, "tgt(G)"] & \\
Obj(G) & & Obj(G) & & Obj(G)
\end{tikzcd}
\qquad
\begin{tikzcd}
& & Mor(G) \times_{Obj(G)} Obj(G) \arrow[dl, shift right, "mor(\rho_G) = 1st"'] \arrow[dr, "2nd"] & & \\
& Mor(G) \arrow[ur, shift right, "mor(\rho'_G)"'] \arrow[dl, "src(G)"'] \arrow[dr, "tgt(G)"] & & Obj(G) \arrow[dl, "Id_{Obj(G)}"'] \arrow[dr, "Id_{Obj(G)}"] & \\
Obj(G) & & Obj(G) & & Obj(G)
\end{tikzcd}$$

Figure 8: Left Unit and Right Unit Graph Isomorphisms

Here is the KIF formulation for the unit isomorphisms.

```
(26) (CNG$function left)
     (CNG$signature left GPH$graph graph-morphism)
     (forall (?g (GPH$graph ?g))
         (and (= (source (left ?g))
                 (GPH$multiplication (GPH$unit (GPH$object ?g)) ?g))
              (= (target (left ?g)) ?g)
              (= (object (left ?g)) (SET.FTN$identity (GPH$object ?g)))
              (= (morphism (left ?g))
                 (SET.LIM.PBK$projection2
                     (GPH$multiplication-opspan (GPH$unit (GPH$object ?g)) ?g)))))

(27) (CNG$function right)
     (CNG$signature right GPH$graph graph-morphism)
     (forall (?g (GPH$graph ?g))
         (and (= (source (right ?g))
                 (GPH$multiplication ?g (GPH$unit (GPH$object ?g))))
              (= (target (right ?g)) ?g)
              (= (object (right ?g))
                 (SET.FTN$identity (GPH$object ?g)))
              (= (morphism (right ?g))
                 (SET.LIM.PBK$projection1
                (GPH$multiplication-opspan (GPH$unit (GPH$object ?g)) ?g)))))
```

An oppositely directed graph morphism $\lambda' : G \rightarrow 1_{Obj(G)} \otimes G$ can be defined, whose morphism function $mor(\lambda') : Mor(G) \rightarrow Obj(G) \times_{Obj(G)} Mor(G)$ is the mediator function for a pullback cone over the opspan associated with $1_{Obj(G)} \otimes G$, whose vertex is $Mor(G)$, whose first function is $src(G)$ and whose second function is $Id_{Mor(G)}$. Based upon uniqueness of the pullback mediator function, this can be shown to be inverse to λ. Similarly, an oppositely directed graph morphism $\rho' : G \rightarrow G \otimes 1_{Obj(G)}$ can be defined and shown to be the inverse to ρ. In addition, the unit coherence theorem and identity theorem in Diagram 4 can

$$\begin{tikzcd}
G_0 \otimes (1_C \otimes G_2)) \arrow[r, "\alpha"] \arrow[d, "G_0 \otimes \lambda"'] & (G_0 \otimes 1_C) \otimes G_2 \arrow[d, "\rho \otimes G_2"] \\
G_0 \otimes G_2 \arrow[r, phantom, "="] & G_0 \otimes G_2
\end{tikzcd}
\qquad
\lambda_1 = \rho_1 : 1_C \otimes 1_C \longrightarrow 1_C$$

Diagram 4: Unit Coherence

be proven.

The Namespace of Large Categories

CAT

Basics

○ A *category C* can be thought of as a special kind of graph $|C|$ – a graph with monoidal properties. More precisely, a category $C = \langle C, \mu_C, \eta_C\rangle$ is a monoid in the 2-dimensional quasi-category of (large) graphs and graph morphisms. This means that it consists of a graph $|C|$, a *composition* graph morphism $\mu_C : |C| \otimes |C| \rightarrow |C|$ and an *identity* graph morphism $\eta_C : 1_{obj(C)} \rightarrow |C|$, both with the identity object function $id_{obj(C)}$. Table 1 gives the notation for the composition function $\circ^C = mor(\mu_C)$ and the identity function $id^C = mor(\eta_C)$ – these are the morphism functions of the composition and identity graph morphisms.

$\circ^C : \mathrm{mor}(C) \times_{\mathrm{obj}(C)} \mathrm{mor}(C) \rightarrow \mathrm{mor}(C)$	$\mathrm{id}^C : \mathrm{obj}(C) \rightarrow \mathrm{mor}(C)$
$(m_1, m_2) \mapsto m_1 \circ m_2$	$o \mapsto \mathrm{id}_o$

Table 1: Elements of Monoidal Structure

Axioms (1–6) give the KIF representation for a category. The unary CNG function '`underlying`' in axiom (2) gives the *underlying* graph of a category. The SET function '`(composition ?c)`' of axiom (4) provides for an associative composition of morphisms in the category – it operates on any two morphisms that are composable, in the sense that the target object of the first is equal to the source object of the second, and returns a well-defined (composition) morphism. The SET function '`(identity ?c)`' in axiom (6) provides identities – it associates a well-defined (identity) morphism with each object in the category. The unary CNG functions '`composition`' and '`identity`' have the category as a parameter. Categories are determined by their (underlying, mu, eta) triples, and hence by their (underlying, composition, identity) triples.

```
(1) (CNG$conglomerate category)

(2) (CNG$function underlying)
    (CNG$signature underlying category GPH$graph)

(3) (CNG$function mu)
    (CNG$signature mu category GPH.MOR$2-cell)
    (forall (?c (category ?c))
        (and (= (GPH.MOR$source (mu ?c))
                (GPH$multiplication (underlying ?c) (underlying ?c)))
             (= (GPH.MOR$target (mu ?c))
                (underlying ?c))))

(4) (CNG$function composition)
    (CNG$signature composition category SET.FTN$function)
    (forall (?c (category ?c))
        (= (composition ?c)
           (GPH.MOR$morphism (mu ?c))))

(5) (CNG$function eta)
    (CNG$signature eta category GPH.MOR$2-cell)
    (forall (?c (category ?c))
        (and (= (GPH.MOR$source (eta ?c))
                (GPH$unit (GPH$object (underlying ?c))))
             (= (GPH.MOR$target (eta ?c))
                (underlying ?c))))

(6) (CNG$function identity)
    (CNG$signature identity category SET$function)
    (forall (?c (category ?c))
        (= (identity ?c)
           (GPH.MOR$morphism (eta ?c))))

    (forall (?c1 (category ?c1) ?c2 (category ?c2))
        (=> (and (= (underlying ?c1) (underlying ?c2))
                 (= (mu ?c1) (mu ?c2))
                 (= (eta ?c1) (eta ?c2)))
            (= ?c1 ?c2)))
```

○ For convenience in the language used for categories, in axioms (7–14) we rename the object and morphism classes, the source and target functions, the class of composable pairs of morphisms, and the first and second functions in the setting of categories.

```
(7) (CNG$function object)
    (CNG$signature object category SET$class)
    (forall (?c (category ?c))
        (= (object ?c)
           (GPH$object (underlying ?c))))

(8) (CNG$function morphism)
    (CNG$signature morphism category SET$class)
    (forall (?c (category ?c))
        (= (morphism ?c)
           (GPH$morphism (underlying ?c))))

(9) (CNG$function source)
    (CNG$signature source category SET.FTN$function)
    (forall (?c (category ?c))
        (= (source ?c)
           (GPH$source (underlying ?c))))

(10) (CNG$function target)
     (CNG$signature source category SET.FTN$function)
     (forall (?c (category ?c))
         (= (target ?c)
            (GPH$target (underlying ?c))))

(11) (CNG$function composable-opspan)
     (CNG$signature composable-opspan category SET.LIM.PBK$opspan)
     (forall (?c (category ?c))
         (= (composable-opspan ?c)
            (GPH$multiplication-opspan (underlying ?c) (underlying ?c))))

(12) (CNG$function composable)
     (CNG$signature composable category SET$class)
     (forall (?c (category ?c))
         (= (composable ?c)
            (GPH$morphism
                (GPH$multiplication (underlying ?c) (underlying ?c)))))

(13) (CNG$function first)
     (CNG$signature first category SET.FTN$function)
     (forall (?c (category ?c))
         (= (first ?c)
            (GPH$source
                (GPH$multiplication (underlying ?c) (underlying ?c)))))

(14) (CNG$function second)
     (CNG$signature second category SET$function)
     (forall (?c (category ?c))
         (= (second ?c)
            (GPH$target
                (GPH$multiplication (underlying ?c)(underlying ?c)))))
```

○ By the definitions of graph morphisms, graph multiplication and graph units, for any category *C* these operations satisfy the typing constraints listed in Table 2.

$\circ^{C} \cdot \mathrm{src}(C) = 1^{st}(C) \cdot \mathrm{src}(C)$
$\circ^{C} \cdot \mathrm{tgt}(C) = 2^{nd}(C) \cdot \mathrm{tgt}(C)$
$\mathrm{id}^{C} \cdot \mathrm{src}(C) = \mathrm{id}_{\mathrm{obj}(C)} = \mathrm{id}^{C} \cdot \mathrm{tgt}(C)$

Table 2: Preservation of Source and Target

○ Table 3 contains commutative diagrams involving the μ and η graph morphisms of categories and the coherence graph morphisms α, λ and ρ. The commutative diagram on the left represents the *associative law* for composition, and the commutative diagrams on the right represent the left and right *unit laws* for identity.

$$\begin{tikzcd} \lvert C\rvert\otimes(\lvert C\rvert\otimes\lvert C\rvert) \arrow[r, "\lvert C\rvert\otimes\mu_{\lvert C\rvert}"] \arrow[d, "\alpha_{\lvert C\rvert,\lvert C\rvert,\lvert C\rvert}"'] & \lvert C\rvert\otimes\lvert C\rvert \arrow[dd, "\mu_{\lvert C\rvert}"] \\ (\lvert C\rvert\otimes\lvert C\rvert)\otimes\lvert C\rvert \arrow[d, "\mu_{\lvert C\rvert}\otimes\lvert C\rvert"'] & \\ \lvert C\rvert\otimes\lvert C\rvert \arrow[r, "\mu_{\lvert C\rvert}"'] & \lvert C\rvert \end{tikzcd}$$	$$\begin{tikzcd} I\otimes\lvert C\rvert \arrow[r, "\eta_{\lvert C\rvert}\otimes\lvert C\rvert"] \arrow[dr, "\lambda_{\lvert C\rvert}"'] & \lvert C\rvert\otimes\lvert C\rvert \arrow[d, "\mu_{\lvert C\rvert}"] & \lvert C\rvert\otimes I \arrow[l, "\lvert C\rvert\otimes\eta_{\lvert C\rvert}"'] \arrow[dl, "\rho_{\lvert C\rvert}"] \\ & \lvert C\rvert & \end{tikzcd}$$
Associative Law	**Left/Right Unit Laws**

Table 3: Laws of Monoidal Structure

○ Axiom (15) represents the associative law in KIF. This is an important axiom, since the correct expression of (15) motivated the ontology for graphs and graph morphisms, the representation of categories as monoids in the 2-dimensional category of large graphs, and in particular the coherence axiomatization. Axiom (16) represents the unit laws in KIF. Both are expressed at the level of graph morphisms. Using composition and identity, these could also be expressed at the level of SET functions, as in Table 6.

```
(15) (forall (?c (category ?c))
         (= (GPH.MOR$composition
                (GPH.MOR$multiplication
                    (GPH.MOR$identity (underlying ?c))
                    (mu ?c))
                (mu ?c))
            (GPH.MOR$composition
                (GPH.MOR$composition
                    (GPH.MOR$alpha
                        (underlying ?c) (underlying ?c) (underlying ?c))
                    (GPH.MOR$multiplication
                        (mu ?c)
                        (GPH.MOR$identity (underlying ?c))))
                (mu ?c))))

(16) (forall (?c (category ?c))
         (and (= (GPH.MOR$composition
                     (GPH.MOR$multiplication
                         (eta ?c)
                         (GPH.MOR$identity (underlying ?c)))
                     (mu ?c))
                 (GPH.MOR$left (underlying ?c)))
              (= (GPH.MOR$composition
                     (GPH.MOR$multiplication
                         (GPH.MOR$identity (underlying ?c))
                         (eta ?c))
                     (mu ?c))
                 (GPH.MOR$right (underlying ?c)))))
```

○ Table 4 is derivative – it represents the associative and unit laws in terms of the composition and identity functions.

Associative law:	$(m_1 \circ^C m_2) \circ^C m_3 = m_1 \circ^C (m_2 \circ^C m_3)$
Identity laws:	$id^C_a \circ^C m = m = m \circ^C id^C_b$

Table 4: Laws of Monoidal Structure Redux

Additional Categorical Structure

Particular categories may have additional structure. This is true for the categories expressed by the IFF lower metalevel namespaces. Here is the KIF formalization for some of this additional structure.

- A category C is small when its underlying graph is small.

```
(17) (CNG$conglomerate small)
     (CNG$subconglomerate small category)
     (forall (?c (category ?c))
         (<=> (small ?c)
              (GPH$small (underlying ?c))))
```

- To each category C, there is an *opposite category* $C^{\mathrm{op}} = \langle C, \mu_C, \eta_C \rangle^{\mathrm{op}} = \langle C^{\mathrm{op}}, \tau_{C,\,C} \cdot \mu_C{}^{\mathrm{op}}, \eta_C{}^{\mathrm{op}} \rangle$. Since all categorical notions have their duals, the opposite category can be used to decrease the size of the axiom set. The objects of C^{op} are the objects of C, and the morphisms of C^{op} are the morphisms of C. However, the source and target of a morphism are reversed: $src(C^{\mathrm{op}})(m) = tgt(C)(m)$ and $tgt(C^{\mathrm{op}})(m) = src(C)(m)$. The composition is defined by $m_2 \circ^{\mathrm{op}} m_1 = m_1 \circ m_2$, and the identity is $id^{\mathrm{op}}_o = id_o$. The type restriction axioms in (18) specify the opposite operation on categories.

```
(18) (CNG$function opposite)
     (CNG$signature opposite category category)
     (forall (?c (category ?c))
         (and (= (underlying (opposite ?c))
                 (GPH$opposite (underlying ?c))
              (= (mu (opposite ?c))
                 (GPH.MOR$composition
                     (GPH.MOR$tau (underlying ?c) (underlying ?c))
                     (GPH.MOR$opposite (mu ?c))))
              (= (eta (opposite ?c))
                 (GPH.MOR$opposite (eta ?c)))))
```

- Part of the fact that opposite forms an involution is the theorem that $(C^{\mathrm{op}})^{\mathrm{op}} = C$.

```
     (forall (?c (category ?c))
         (= (opposite (opposite ?c)) ?c))
```

- It is sometime convenient to have a name for the pair of classes ⟨`(object ?c)`, `(object ?c)`⟩. This is called ‘`(object-pair ?c)`’.

```
(19) (CNG$function object-pair)
     (CNG$signature object-pair category SET.LIM.PRD$pair)
     (forall (?c (category ?c))
         (and (= (SET.LIM.PRD$class1 (object-pair ?c)) (object ?c))
              (= (SET.LIM.PRD$class2 (object-pair ?c)) (object ?c))))

(20) (CNG$function object-binary-product)
     (CNG$signature object-binary-product category SET.class)
     (forall (?c (category ?c))
         (= (object-binary-product ?c)
            (SET.LIM.PRD$binary-product (object-pair ?c))))

(21) (CNG$function morphism-pair)
     (CNG$signature morphism-pair category SET.LIM.PRD$pair)
     (forall (?c (category ?c))
         (and (= (SET.LIM.PRD$class1 (morphism-pair ?c)) (morphism ?c))
              (= (SET.LIM.PRD$class2 (morphism-pair ?c)) (morphism ?c))))

(22) (CNG$function morphism-binary-product)
     (CNG$signature morphism-binary-product category SET.class)
     (forall (?c (category ?c))
         (= (morphism-binary-product ?c)
            (SET.LIM.PRD$binary-product (morphism-pair ?c))))
```

- For any two objects o_1 and o_2 in a category C, the *hom-set* $C(o_1, o_2)$ consists of all morphisms with source o_1 and target o_2:

$$C(o_1, o_2) = \{m \mid m \in mor(C),\ src(C)(m) = o_1 \text{ and } tgt(C)(m) = o_2\} \subseteq mor(C).$$

```
(21) (CNG$function source-target)
```

```
     (CNG$signature source-target category SET.FTN$function)
     (forall (?c (category ?c))
         (and (= (SET.FTN$source (source-target ?c)) (morphism ?c))
              (= (SET.FTN$target (source-target ?c)) (object-binary-product ?c))
              (= (source-target ?c)
                 ((SET.LIM.PRD$pairing (object-pair ?c)) (source ?c) (target ?c)))))

(22) (CNG$function object-hom)
     (CNG$signature object-hom category SET.FTN$function)
     (forall (?c (category ?c))
         (and (= (SET.FTN$source (object-hom ?c)) (object-binary-product ?c))
              (= (SET.FTN$target (object-hom ?c)) (SET$power (morphism ?c)))
              (= (object-hom ?c)
                 (SET.FTN$fiber (source-target ?c)))))

(23) (CNG$function morphism-hom)
     (CNG$signature morphism-hom category CNG$function)
     (forall (?c (category ?c))
         (and (CNG$signature (morphism-hom ?c)
                  (morphism-binary-product ?c) SET.FTN$function)
              (forall (?m1 ((morphism c) ?m1) ?m2 ((morphism c) ?m2))
                  (and (= (SET.FTN$source ((morphism-hom ?c) [?m1 ?m2]))
                          ((object-hom ?c) [((target ?c) ?m1) ((source ?c) ?m2)]))
                       (= (SET.FTN$target ((morphism-hom ?c) [?m1 ?m2]))
                          ((object-hom ?c) [((souce ?c) ?m1) ((target ?c) ?m2)]))
                       (forall (?m (((object-hom ?c)
                                  [((target ?c) ?m1) ((source ?c) ?m2)]) ?m))
                           (= (((morphism-hom ?c) [?m1 ?m2])) ?m)
                              ((composition ?c)
                                  [((composition ?c) [?m1 ?m]) ?m2])))))))
```

○ A *parallel pair* of morphisms in a category *C* is a pair of morphisms with the same source and target objects. This equivalence relation is the kernel of the source-target function.

```
(24) (CNG$function parallel-pair)
     (CNG$signature parallel-pair category REL.ENDO$equivalence-relation)
     (forall (?c (category ?c))
         (= (parallel-pair ?c) (SET.LIM.EQU$kernel (source-target ?c))))
```

○ There are classes of left-composability and right-composability, and functions of left-composition and right-composition. Left-composition by morphism m_1 is the operation: $m_2 \mapsto m_1 \cdot m_2$.

```
(25) (CNG$function left-composable)
     (CNG$signature left-composable category SET.FTN$function)
     (forall (?c (category ?c))
         (and (= (SET.FTN$source (left-composable ?c)) (morphism ?c))
              (= (SET.FTN$target (left-composable ?c)) (SET$power (morphism ?c)))
              (= (left-composable ?c) (SET.LIM.PBK$fiber12 (composable-opspan ?c)))))

(26) (KIF$function left-composition)
     (KIF$signature left-composition category CNG$function)
     (forall (?c (category ?c))
       (and (CNG$signature (left-composition ?c) (morphism ?c) SET.FTN$function)
            (forall (?m1 ((morphism ?c) ?m1))
              (and (= (SET.FTN$source ((left-composition ?c) ?m1))
                      ((left-composable ?c) ?m1))
                   (= (SET.FTN$target ((left-composition ?c) ?m1))
                      (morphism ?c))
                   (= ((left-composition ?c) ?m1)
                      (SET.FTN$composition
                        (SET.FTN$composition
                          ((SET.LIM.PBK$fiber12-embedding (composable-opspan ?c)) ?m1)
                          ((SET.LIM.PBK$fiber-embedding (composable-opspan ?c))
                            ((source ?c) ?m1)))
                        (composition ?c)))))))

(27) (CNG$function right-composable)
     (CNG$signature right-composable category SET.FTN$function)
     (forall (?c (category ?c))
         (and (= (SET.FTN$source (right-composable ?c)) (morphism ?c))
              (= (SET.FTN$target (right-composable ?c)) (SET$power (morphism ?c)))
              (= (left-composable ?c) (SET.LIM.PBK$fiber21 (composable-opspan ?c)))))
```

```
(28) (KIF$function right-composition)
     (KIF$signature right-composition category CNG$function)
     (forall (?c (category ?c))
       (and (CNG$signature (right-composition ?c) (morphism ?c) SET.FTN$function)
            (forall (?m1 ((morphism ?c) ?m1))
              (and (= (SET.FTN$source ((right-composition ?c) ?m1))
                      ((right-composable ?c) ?m1))
                   (= (SET.FTN$target ((right-composition ?c) ?m1))
                      (morphism ?c))
                   (= ((right-composition ?c) ?m1)
                      (SET.FTN$composition
                        (SET.FTN$composition
                          ((SET.LIM.PBK$fiber12-embedding (composable-opspan ?c)) ?m1)
                          ((SET.LIM.PBK$fiber-embedding (composable-opspan ?c))
                            ((source ?c) ?m1)))
                        (composition ?c)))))))
```

○ A morphism $m_1 : o_0 \rightarrow o_1$ is an *epimorphism* (Axiom 19) in a category C when it is left-cancellable – for any two parallel morphisms $m_2, m_2' : o_1 \rightarrow o_2$, the equality $m_1 \circ^C m_2 = m_1 \circ^C m_2'$ implies $m_2 = m_2'$. Equivalently, a morphism $m_1 : o_0 \rightarrow o_1$ is an epimorphism (Axiom 19) in a category C when its left composition is an injection. Dually, a morphism $m_2 : o_1 \rightarrow o_2$ is a *monomorphism* in a category C when it is right-cancellable – that is, when it is an epimorphism in C^{op}. Axiom (20) uses the duality of the opposite category to express monomorphisms. A morphism is an *isomorphism* (Axiom 21) in a category C when it is both a monomorphism and an epimorphism.

```
(29) (CNG$function epimorphism)
     (CNG$signature epimorphism category SET$class)
     (forall (?c (category ?c))
         (and (SET$subclass (epimorphism ?c) (morphism ?c))
              (forall (?m1 ((morphism ?c) ?m1))
                  (<=> ((epimorphism ?c) ?m1)
                       (SET.FTN$injection ((left-composition ?c) ?m1))))))

(30) (CNG$function monomorphism)
     (CNG$signature monomorphism category SET$class)
     (forall (?c (category ?c))
         (and (SET$subclass (monomorphism ?c) (morphism ?c))
              (forall (?m2 ((morphism ?c) ?m2))
                  (<=> ((monomorphism ?c) ?m1)
                       ((epimorphism (opposite ?c)) ?m1)))))

(31) (CNG$function isomorphism)
     (CNG$signature isomorphism category SET$class)
     (forall (?c (category ?c))
         (and (SET$subclass (monomorphism ?c) (morphism ?c))
              (= (isomorphism ?c)
                 (SET$binary-intersection (epimorphism ?c) (monomorphism ?c)))))
```

○ Two objects $o_1, o_2 \in mor(C)$ are *isomorphic* when there is an isomorphism between them; we then use the notation $o_1 \cong_C o_2$.

```
(32) (CNG$function isomorphic)
     (CNG$signature isomorphic category REL.ENDO$endorelation)
     (forall (?c (category ?c))
         (and (REL.ENDO$class (isomorphic ?c)) (object ?c))
              (forall (?o1 ((object ?c) ?o1) ?o2 ((object ?c) ?o2))
                  (<=> ((REL.ENDO$extent (isomorphic ?c)) [?o1 ?o2])
                       (exists (?m ((isomorphism ?c) ?m))
                           (and (= ((source ?c) ?m) ?o1)
                                (= ((target ?c) ?m) ?o2)))))))
```

Examples

Here are some examples of small categories, which can be used as shapes for colimit/limit diagrams.

- The terminal category *set1* has one object 0 and one (identity) morphism 00. A *discrete category* is a category whose morphisms are all identity morphisms. So, essentially a discrete category is just a set (of objects). *set1* is clearly a discrete category.

```
(CAT$category terminal)
(CAT$category unit)
(CAT$category set1)
(= unit terminal)
(= set1 terminal)
(= (CAT$object terminal) SET.LIM$terminal)
(= (CAT$morphism terminal) SET.LIM$terminal)
(= ((CAT$source terminal) set1#00) set1#0)
(= ((CAT$target terminal) set1#00) set1#0)
```

- The discrete category *set2* = • • of two things, is the category, whose set of objects is {0, 1} , whose set of morphisms is {00, 11}, with 00 and 11 being the identity morphisms at objects 0 and 1, respectively. The following KIF represents this category.

```
(CAT$category set2)
((CAT$object set2) set2#0)
((CAT$object set2) set2#1)
((CAT$morphism set2) set2#00)
((CAT$morphism set2) set2#11)
(= ((CAT$identity set2) set2#0) set2#00)
(= ((CAT$identity set2) set2#1) set2#11)
```

- The ordinal category *ord3* (Figure 5) (Mac Lane 1971, p. 11) is the ordinal, whose set of objects is {0, 1, 2}, whose set of morphisms is {00, 11, 22, 01, 12, 02}, with 00, 11 and 22 being the identity morphisms at objects 0, 1 and 2, respectively, and possessing the one nontrivial composition 01 ∘ 12 = 02. The following KIF represents the category *ord3*.

```
(CAT$category ord3)
((CAT$object ord3) ord3#0)
((CAT$object ord3) ord3#1)
((CAT$object ord3) ord3#2)
((CAT$morphism ord3) ord3#00)
((CAT$morphism ord3) ord3#11)
((CAT$morphism ord3) ord3#22)
((CAT$morphism ord3) ord3#01)
((CAT$morphism ord3) ord3#12)
((CAT$morphism ord3) ord3#02)
(= ((CAT$source ord3) ord3#01) ord3#0)
(= ((CAT$target ord3) ord3#01) ord3#1)
(= ((CAT$source ord3) ord3#12) ord3#1)
(= ((CAT$target ord3) ord3#12) ord3#2)
(= ((CAT$source ord3) ord3#02) ord3#0)
(= ((CAT$target ord3) ord3#02) ord3#2)
(= ((CAT$identity ord3) ord3#0) ord3#00)
(= ((CAT$identity ord3) ord3#1) ord3#11)
(= ((CAT$identity ord3) ord3#2) ord3#22)
(= ((CAT$composition ord3) ord3#01 ord3#12) ord3#02)
```

$$\begin{tikzcd} & \overset{1}{\bullet} \arrow[dr] & \\ 0\ \bullet \arrow[ur] \arrow[rr] & & \bullet\ 2 \end{tikzcd}$$

Figure 5: ordinal 3

- The shape category *parpair* = $\bullet \rightrightarrows \bullet$ consists of a parallel pair of edges, whose set of objects is {0, 1}, whose set of morphisms is {00, 11, a_0, a_1}, with 00 and 11 being the identity morphisms at objects 0 and 1, respectively. The following KIF represents this category.

```
(CAT$category parpair)
((CAT$object parpair) parpair#0)
((CAT$object parpair) parpair#1)
((CAT$morphism parpair) parpair#00)
((CAT$morphism parpair) parpair#11)
((CAT$morphism parpair) parpair#a0)
((CAT$morphism parpair) parpair#a1)
(= ((CAT$source parpair) parpair#a0) parpair#0)
(= ((CAT$target parpair) parpair#a0) parpair#1)
(= ((CAT$source parpair) parpair#a1) parpair#0)
(= ((CAT$target parpair) parpair#a1) parpair#1)
(= ((CAT$identity parpair) parpair#0) parpair#00)
(= ((CAT$identity parpair) parpair#1) parpair#11)
```

○ The shape category *span* = • ← • → • consists of a pair of morphisms a_1 and a_2 with common source object 0 and target objects 1 and 2, respectively. The class of objects is *obj*(*J*) = {0, 1, 2}, and the class of morphisms is *mor*(*J*) = {00, 11, 22, a_1, a_2}, with 00, 11 and 22 being the identity morphisms at objects 0, 1 and 2 respectively. Here is the KIF representation of the *span* shape category.

```
(CAT$category span)
((CAT$object span) span#0)
((CAT$object span) span#1)
((CAT$object span) span#2)
((CAT$morphism span) span#00)
((CAT$morphism span) span#11)
((CAT$morphism span) span#22)
((CAT$morphism span) span#a1)
((CAT$morphism span) span#a2)
(= ((CAT$source span) span#a1) span#0)
(= ((CAT$target span) span#a1) span#1)
(= ((CAT$source span) span#a2) span#0)
(= ((CAT$target span) span#a2) span#2)
(= ((CAT$identity span) span#0) span#00)
(= ((CAT$identity span) span#1) span#11)
(= ((CAT$identity span) span#2) span#22)
```

Here are examples of categories defined elsewhere, but asserted to be categories here.

o Two important categories are implicitly defined within the classification namespace in the Model Ontology – Classification the category of classifications and infomorphisms, and Set the category of small sets and their functions. The assertion that "Classification and Set are categories" could not be made in the Model Theory Ontology (the lower metalevel of the IFF Foundation Ontology), since the appropriate functorial machinery is not present there. The Category Theory Ontology provides that machinery. To make that assertion requires that we also describe or identify the components of a category: the underlying graph (object and morphism sets, and source and target functions), and the composition and identity functions (or the mu and eta graph 2-cells). Here we make these assertions in an external namespace. Proofs of some categorical properties, such as associativity of composition, will involve getting further into the details of the specific category, in this case Classification; in particular, the associativity of '`set.ftn$composition`', etc.

```
(CAT$category Classification)
(= (CAT$underlying Classification)   cls.info$classification-graph)
    (= (CAT$object Classification)   cls$classification)
    (= (CAT$morphism Classification) cls.info$infomorphism)
    (= (CAT$source Classification)   cls.info$source)
    (= (CAT$target Classification)   cls.info$target)
(= (CAT$composable Classification)   cls.info$composable)
(= (CAT$first Classification)        cls.info$first)
(= (CAT$second Classification)       cls.info$second)
(= (CAT$composition Classification)  cls.info$composition)
(= (CAT$identity Classification)     cls.info$identity)

(CAT$category Set)
(= (CAT$underlying Set)   set.ftn$set-graph)
    (= (CAT$object Set)   set$set)
    (= (CAT$morphism Set) set.ftn$function)
    (= (CAT$source Set)   set.ftn$source)
    (= (CAT$target Set)   set.ftn$target)
(= (CAT$composable Set)   set.ftn$composable)
(= (CAT$first Set)        set.ftn$first)
(= (CAT$second Set)       set.ftn$second)
(= (CAT$composition Set)  set.ftn$composition)
(= (CAT$identity Set)     set.ftn$identity)
```

The Namespace of Large Functors

`FUNC`

Basics

○ A *functor* $F : C_0 \rightarrow C_1$ from category C_0 to category C_1 (Figure 1) is a morphism of categories. A functor is a special kind of graph morphism $|F| : |C_0| \rightarrow |C_1|$ – a graph morphism that preserves the monoidal properties of the categories. The underlying operator preserves source and target. These functions must preserve source and target in the sense that the diagram in Figure 1 is commutative. However, this follows from properties of graph morphisms. A functor is determined by its associated triple (source, target, underlying).

$$\begin{tikzcd}
mor(|C_0|) \arrow[r, shift left, "src(|C_0|)"] \arrow[r, shift right, "tgt(|C_0|)"'] \arrow[d, "mor(|F|)"'] & obj(|C_0|) \arrow[d, "obj(F)"] \\
mor(|C_1|) \arrow[r, shift left, "src(|C_1|)"] \arrow[r, shift right, "tgt(|C_1|)"'] & obj(|C_1|)
\end{tikzcd}$$

Figure 1: Functor

Axioms (1–4) give the KIF representation for a functor. The unary CNG function '`underlying`' in axiom (4) gives the *underlying* graph morphism of a functor. Functors are determined by their underlying graph morphism.

```
(1) (CNG$conglomerate functor)

(2) (CNG$function source)
    (CNG$signature source functor CAT$category)

(3) (CNG$function target)
    (CNG$signature target functor CAT$category)

(4) (CNG$function underlying)
    (CNG$signature underlying functor GPH.MOR$graph-morphism)

    (forall (?f (functor ?f))
        (and (= (CAT$underlying (source ?f))
                (GPH.MOR$source (underlying ?f)))
             (= (CAT$underlying (target ?f))
                (GPH.MOR$target (underlying ?f)))))

    (forall (?f1 (functor ?f1) ?f2 (functor ?f2))
        (=> (and (= (source ?f1) (source ?f2))
                 (= (target ?f1) (target ?f2))
                 (= (underlying ?f1) (underlying ?f2)))
            (= ?f1 ?f2)))
```

○ For convenience in the language used for functors, in axioms (5–6) we rename the object and morphism functions in the setting of functors.

```
(5) (CNG$function object)
    (CNG$signature object functor SET.FTN$function)
    (forall (?f (functor ?f))
        (= (object ?f)
           (GPH.MOR$object (underlying ?f))))

(6) (CNG$function morphism)
    (CNG$signature morphism function SET.FTN$function)
    (forall (?f (functor ?f))
        (= (morphism ?f)
           (GPH.MOR$morphism (underlying ?f))))
```

Table 1: Preservation of monoidal structure

$\begin{tikzcd} {|C_0| \otimes |C_0|} \arrow[r, "\mu_{|C0|}"] \arrow[d, "{|F| \otimes |F|}"'] & {|C_0|} \arrow[d, "{|F|}"] \\ {|C_1| \otimes |C_1|} \arrow[r, "\mu_{|C1|}"'] & {|C_1|} \end{tikzcd}$	$\begin{tikzcd} 1_{obj(|C0|)} \arrow[r, "\eta_{|C0|}"] \arrow[d, "1_{obj(|F|)}"'] & {|C_0|} \arrow[d, "{|F|}"] \\ 1_{obj(|C1|)} \arrow[r, "\eta_{|C1|}"'] & {|C_1|} \end{tikzcd}$
Preservation of composition	**Preservation of identity**

- ○ A functor must preserve monoidal properties – it must preserve identities and compositions in the sense of the commutative diagrams in Table 1. These are commutative diagrams of graph morphisms. The commutative diagram on the left represents preservation of composition, and the commutative diagram on the right represents preservation of identity.
- ○ Axiom (7) represents the preservation of composition law in KIF. Axiom (8) represents preservation of identity in KIF. Both are expressed at the level of graph morphisms.

```
(7) (forall (?f (functor ?f))
        (= (GPH.MOR$composition (mu (source ?f)) (underlying ?f))
           (GPH.MOR$composition
               (GPH.MOR$multiplication (underlying ?f) (underlying ?f))
               (mu (target ?f)))))

(8) (forall (?f (functor ?f))
        (= (GPH.MOR$composition (eta (source ?f)) (underlying ?f))
           (GPH.MOR$composition (unit (object ?f)) (eta (target ?f)))))
```

- ○ Using composition and identity, the associative and unit laws could also be expressed at the level of SET functions, as in Table 2, which is derivative – it represents the preservation of monoidal structure in terms of the composition and identity functions.

Associative law:	$mor(|F|)(m_1 \circ^0 m_2) = mor(|F|)(m_1) \circ^1 mor(|F|)(m_2)$ for all composable pairs of morphisms $m_1, m_2 \in Mor(C_0)$
Identity laws:	$mor(|F|)(id^0{}_o) = id^1{}_{obj(|F|)\,(o)}$ for all objects $o \in Obj(C_0)$

Table 2: Laws of Monoidal Structure Redux

```
(forall (?f (functor ?f))
    (= (SET.FTN$composition (CAT$composition (source ?f)) (morphism ?f))
       (SET.FTN$composition
           ((SET.LIM.PBK$pairing (CAT$composable-opspan (target ?f)))
               (SET.FTN$composition (CAT$first (source ?f)) (morphism ?f))
               (SET.FTN$composition (CAT$second (source ?f)) (morphism ?f)))
           (CAT$composition (target ?f)))))

(forall (?f (functor ?f))
    (= (SET.FTN$composition (CAT$identity (source ?f)) (morphism ?f))
       (SET.FTN$composition (object ?f) (CAT$identity (target ?f)))))
```

Additional Functorial Structure

- Given any category C, there is a *unique functor* $!_C : C \to 1$ to the terminal category – the object and morphism functions are the unique SET functions to the terminal class.

```
(9) (CNG$function unique)
    (CNG$signature unique CAT$category functor)
    (forall (?c (CAT$category ?c))
        (and (= (source (unique ?c)) ?c)
             (= (target (unique ?c)) (CAT$category terminal))
             (= (object (unique ?c)) (SET.FTN$unique (CAT$object ?c)))
             (= (morphism (unique ?c)) (SET.FTN$unique (CAT$morphism ?c)))))
```

- For each category C and each object $o \in C$ there is an *element functor* $elmt_C(o) : 1 \to C$.

```
(10) (CNG$function element)
     (CNG$signature element CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (element ?c) (CAT$object ?c) functor))
              (forall (?o ((CAT$object ?c) ?o))
                  (and (= (source ((element ?c) ?o)) CAT$terminal)
                       (= (target ((element ?c) ?o)) ?c)
                       (= ((object ((element ?c) ?o)) 0) ?o)
                       (= ((morphism ((element ?c) ?o)) 0) ((CAT$identity ?c) ?o))))))
```

- A category A is said to be a *subcategory* of category B when there is a functor $incl_{A,B} : A \to B$ whose object and morphism functions are injections. Clearly, this provides an partial order on categories.

```
(11) (CNG$relation subcategory)
     (CNG$signature subcategory CAT$category CAT$category)
     (forall (?a (CAT$category ?a) ?b (CAT$category ?b))
         (<=> (subcategory ?a ?b)
              (exists (?f (functor ?f))
                  (and ((source ?f) ?a)
                       ((target ?f) ?b)
                       (SET.FTN$injection (object ?f))
                       (SET.FTN$injection (morphism ?f))))))
```

- To each functor $F : C \to C'$, there is an *opposite functor* $F^{\mathrm{op}} : C^{\mathrm{op}} \Rightarrow C'^{\mathrm{op}}$. The underlying graph morphism of F^{op} is the opposite $|F^{\mathrm{op}}| = |F|^{\mathrm{op}} : |C_0|^{\mathrm{op}} \to |C_1|^{\mathrm{op}}$: the object function of F^{op} is the object function of F, and the morphism function of F^{op} is the morphism function of H. However, the source and target categories are the opposite: $src(F^{\mathrm{op}}) = src(F)^{\mathrm{op}}$ and $tgt(F^{\mathrm{op}}) = tgt(F)^{\mathrm{op}}$. Axiom (9) specifies an opposite functor.

```
(12) (CNG$function opposite)
     (CNG$signature opposite functor functor)
     (forall (?f (functor ?f))
         (and (= (source (opposite ?f)) (CAT$opposite (source ?f)))
              (= (target (opposite ?f)) (CAT$opposite (target ?f)))
              (= (underlying (opposite ?f)) (GPH.MOR$opposite (object ?h)))
              (= (object (opposite ?f)) (object ?f))
              (= (morphism (opposite ?h)) (morphism ?h))))
```

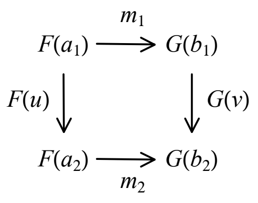
  An immediate theorem is that the opposite of the opposite of a functor is the original functor.

```
     (forall (?f (functor ?f))
         (= (opposite (opposite ?f)) ?f))
```

- An *opspan* of functors consists of two functors $F : A \to C$ and $G : B \to C$ with a common target category C. Each opspan of functors determines a *comma category* $(F \downarrow G)$, whose objects are triples $\langle a, m, b\rangle$ with $a \in obj(A)$, $b \in obj(B)$ and $m : F(a) \to G(b)$, and whose morphisms

  $\langle u, v\rangle : \langle a_1, m_1, b_1\rangle \to \langle a_2, m_2, b_2\rangle$

  are pairs of morphisms $u : a_1 \to a_2$ and $v : b_1 \to b_2$ from the source categories that satisfy the commutative Diagram 1.



$$\begin{tikzcd} F(a_1) \arrow[r, "m_1"] \arrow[d, "F(u)"'] & G(b_1) \arrow[d, "G(v)"] \\ F(a_2) \arrow[r, "m_2"'] & G(b_2) \end{tikzcd}$$

Diagram 1: comma category

Here is the KIF formalization of comma categories.

```
(13) (CNG$conglomerate opspan)

(14) (CNG$function category1)
     (CNG$signature category1 diagram CAT$category

(15) (CNG$function category2)
     (CNG$signature category2 diagram CAT$category)

(16) (CNG$function opvertex)
     (CNG$signature opvertex diagram CAT$category)

(17) (CNG$function opfirst)
     (CNG$signature opfirst diagram functor)

(18) (CNG$function opsecond)
     (CNG$signature opsecond diagram functor)

    (forall (?s (opspan ?s))
       (and (= (SET.FTN$source (opfirst ?s)) (category1 ?s))
            (= (SET.FTN$source (opsecond ?s)) (category2 ?s))
            (= (SET.FTN$target (opfirst ?s)) (opvertex ?s))
            (= (SET.FTN$target (opsecond ?s)) (opvertex ?s))))

    (forall (?s (opspan ?s) ?t (opspan ?t))
       (=> (and (= (opfirst ?s) (opfirst ?t))
                (= (opsecond ?s) (opsecond ?t)))
           (= ?s ?t)))

(19) (CNG$function comma-category)
     (CNG$signature comma-category opspan CAT$category)
     (forall (?s (opspan ?s))
         (and (forall (?o)
                  (<=> ((CAT$object (comma-category ?s)) ?o)
                       (and (KIF$triple ?o)
                            ((CAT$object (category1 ?s)) (?o 1))
                            ((CAT$morphism (opvertex ?s)) (?o 2))
                            ((CAT$object (category2 ?s)) (?o 3))
                            (= ((CAT$source (opvertex ?s)) (?o 2))
                               ((object (opfirst ?s)) (?o 1)))
                            (= ((CAT$target (opvertex ?s)) (?o 2))
                               ((object (opsecond ?s)) (?o 3))))))
              (forall (?m)
                  (<=> ((CAT$morphism (comma-category ?s)) ?m))
                       (and (KIF$pair ?m)
                            ((CAT$morphism (category1 ?s)) (?m 1))
                            (= ((CAT$source (category1 ?s)) (?m 1))
                               ((CAT$source ?m) 1))
                            (= ((CAT$target (category1 ?s)) (?m 1))
                               ((CAT$target ?m) 1))
                            ((CAT$morphism (category2 ?s)) (?m 2))
                            (= ((CAT$source (category2 ?s)) (?m 2))
                               ((CAT$source ?m) 3))
                            (= ((CAT$target (category2 ?s)) (?m 2))
                               ((CAT$target ?m) 3))
                            (= (((CAT$composition (opvertex ?s))
                                   [((CAT$source ?m) 2) (?m 2)])
                               (((CAT$composition (opvertex ?s))
                                   [(?m 1) ((CAT$target ?m) 2)])))))))

(20) (CNG$function objects-under-opspan)
     (CNG$signature objects-under-opspan functor CNG$function)
     (forall (?f (functor ?f))
         (and (CNG$signature (objects-under-opspan ?f)
                  (CAT$object (target ?f)) opspan)
              (forall (?a ((CAT$object (target ?f)) ?a))
                  (and (= (category1 ((objects-under-opspan ?f) ?a)) CAT$terminal)
                       (= (category2 ((objects-under-opspan ?f) ?a)) (source ?f))
                       (= (opvertex ((objects-under-opspan ?f) ?a)) (target ?f))
                       (= (opfirst ((objects-under-opspan ?f) ?a))
                          ((element (target ?f)) ?a))
                       (= (opsecond ((objects-under-opspan ?f) ?a)) ?f)))))
```

```
(21) (CNG$function objects-under)
     (CNG$signature objects-under functor CNG$function)
     (forall (?f (functor ?f))
         (and (CNG$signature (objects-under ?f)
                  (CAT$object (target ?f)) CAT.category)
              (forall (?a ((CAT$object (target ?f)) ?a))
                  (= ((objects-under ?f) ?a)
                     (comma-category ((objects-under-opspan ?f) ?a))))))
```

- For any functor $U : B \rightarrow A$ and any object $a \in obj(A)$, a *universal morphism* from a to U (Diagram 2) is a pair $\langle m_a, \tilde{a}\rangle$ consisting of an object $\tilde{a} \in obj(B)$ and a morphism $m_a : a \rightarrow U(\tilde{a})$, such that for every pair $\langle m, b\rangle$ consisting of an object $b \in obj(B)$ and a morphism $m : a \rightarrow U(b)$, there is a unique morphism $m' : \tilde{a} \rightarrow b$ with $m_a \cdot_A U(m') = m$. Equivalently, a universal morphism is an initial object in the comma category $(a \downarrow U)$.

$$\begin{tikzcd}
a \arrow[r, "m_a"] \arrow[dr, "m"'] & U(\tilde{a}) \arrow[d, "U(m')"] \\
& U(b)
\end{tikzcd}$$

Diagram 2: universal morphism

Here is the KIF formalism for universal morphisms. For an arbitrary pair $\langle U, a\rangle$ the universal morphism may not exist – in the code below the term '`COL$initial`' refers to a possibly empty class of objects.

```
(22) (CNG$function universal-morphism)
     (CNG$signature universal-morphism functor CNG$function)
     (forall (?f (functor ?f))
         (and (CNG$signature (universal-morphism ?f)
                  (CAT$object (target ?f)) SET.class)
              (forall (?a ((CAT$object (target ?f)) ?a))
                  (= ((universal-morphism ?f) ?a)
                     (COL$initial ((objects-under ?f) ?a))))))
```

Quasi-Category Structure

- A pair of functors F and G is composable when the target category of F is the source category of G. For any composable pair of functors $F : C_0 \rightarrow C_1$ and $G : C_1 \rightarrow C_2$ there is a *composition functor* $F \circ G : C_0 \rightarrow C_2$. It is defined in terms of the underlying graph morphisms – object and morphism functions are the composition functions of the object and morphism functions of the component functors, respectively. For any category C there is an *identity functor* $Id_C : C \rightarrow C$ on that category. Its underlying graph morphisms is the identity on the underlying graph – object and morphism functions are the identity functions on the object and morphism sets of that category, respectively.

Here is the KIF that formalizes the definitions of composition and identity.

```
(23) (CNG$function composition)
     (CNG$signature composition functor functor functor)
     (forall (?f1 (functor ?f1) ?f2 (functor ?f2))
         (<=> (exists (?f (functor ?f))
                  (= ?f (composition ?f1 ?f2)))
              (= (target ?f1) (source ?f2))))

(24) (forall (?f1 (functor ?f1) ?f2 (functor ?f2))
         (=> (= (target ?f1) (source ?f2))
             (and (= (source (composition ?f1 ?f2))
                     (source ?f1))
                  (= (target (composition ?f1 ?f2))
                     (target ?f2))
                  (= (underlying (composition ?f1 ?f2))
                     (GPH.MOR$composition (underlying ?f1) (underlying ?f2))))))

(25) (CNG$function identity)
     (CNG$signature identity CAT$category functor)

(26) (forall (?c (CAT$category ?c))
         (and (= (source (identity ?c)) ?c)
              (= (target (identity ?c)) ?c)
              (= (underlying (identity ?c))
                    (GPH.MOR$identity (underlying ?c)))))
```

○ Given any category C (to be used as a base category in the colimit namespace) and any category J (to be used as a shape category in the colimit namespace), the *diagonal functor* $\Delta_{J,\,C} : C \rightarrow C^J$ maps an object $o \in obj(C)$ to an associated constant functor $\Delta_{J,\,C}(o) : J \rightarrow C$, which is defined as the functor composition $\Delta_{J,\,C}(o) = !_J \circ elmt_C(o)$ – it maps each object $j \in obj(J)$ to the object $o \in obj(C)$ and maps each morphism $n \in mor(J)$ to the identity morphism at o.

```
(27) (KIF$function diagonal)
     (KIF$signature diagonal CAT$category CAT$category CNG$function)
     (forall (?j (CAT$category ?j) ?c (CAT$category ?c))
         (and (CNG$signature (diagonal ?j ?c) (CAT$object ?c) (diagram ?c))
              (forall (?o ((CAT$object ?c) ?o))
                  (and (= ((shape ?c) ((diagonal ?j ?c) ?o)) ?j)
                       (= ((diagonal ?j ?c) ?o)
                          (composition (unique ?j) ((element ?c) ?o)))))))
```

Functor Theorems

○ It can be shown that functor composition satisfies the following associative law

$$(F_1 \circ F_2) \circ F_3 = F_1 \circ (F_2 \circ F_3)$$

for all composable pairs of functors (F_1, F_2) and (F_2, F_3), and that graph morphism identity satisfies the following identity laws

$$Id_{C0} \circ F = F \text{ and } F = F \circ Id_{C1}$$

for any functor F: $C_0 \rightarrow C_1$ with source category C_0 and target category G_1. Categories as objects and functors as morphisms form a quasi-category ("quasi" since this is at the level of conglomerates in foundations). This has the following expression in an external namespace.

```
     (forall (?f1 (FUNC$functor ?f1)
              ?f2 (FUNC$functor ?f2)
              ?f3 (FUNC$functor ?f3))
         (=> (and (= (FUNC$target ?f1) (FUNC$source ?f2))
                  (= (FUNC$target ?f2) (FUNC$source ?f3)))
             (= (FUNC$composition (FUNC$composition ?f1 ?f2) ?f3)
                (FUNC$composition ?f1 (FUNC$composition ?f2 ?f3)))))

     (forall (?f (FUNC$functor ?f))
         (and (= (FUNC$composition (FUNC$identity (FUNC$source ?f)) ?f) ?f)
              (= (FUNC$composition ?f (FUNC$identity (FUNC$target ?f))) ?f)))
```

Examples

○ For any category C and any there is a counique functor from the initial category to

Here are examples of functors defined elsewhere, but asserted to be functors here.

o Three important functors are implicitly defined within the classification namespace in the Model Ontology. The SET functions '`cls$instance`' and '`cls.info$instance`' represent the object and morphism components of the *instance functor* *inst* : Classification $\rightarrow$ Set^{op} (the opposite of the category Set) – the object function takes a classification to its instance set and the morphism function takes an infomorphism to its instance function. Dually, the SET functions '`cls$type`' and '`cls.info$type`' represent the object and morphism components of the *type functor* *typ* : Classification $\rightarrow$ Set – the object function takes a classification to its type set and the morphism function takes an infomorphism to its type function. The SET functions '`cls$instance-power`' and '`cls.info$instance-power`' represent the object and morphism components of the contravariant *instance power functor* *pow* : Set^{op} $\rightarrow$ Classification. The assertion that "*inst*, *typ* and *pow* are functors" could not be made in the Model Theory Ontology (the lower metalevel of the IFF Foundation Ontology), since the appropriate functorial machinery is not present there. The Category Theory Ontology provides that machinery. To make that assertion requires that we also describe or identify the components of a functor: the source and target categories, and the underlying graph morphism (object and morphism functions). Here we make these assertions in an external namespace. Proofs of some functorial properties, such as preservation of associativity, will involve getting further into the details of the specific category, in this case Classification; in particular, the instance and type function components of an infomorphism, and the associativity of '`set.ftn$composition`'.

```
(FUNC$functor inst)
(= (FUNC$source inst)     Classification)
(= (FUNC$target inst)     (opposite Set))
(= (FUNC$underlying inst) cls.info$instance-graph-morphism)
    (= (FUNC$object inst)   cls$instance)
    (= (FUNC$morphism inst) cls.info$instance)

(FUNC$functor typ)
(= (FUNC$source typ)     Classification)
(= (FUNC$target typ)     Set)
(= (FUNC$underlying typ) cls.info$type-graph-morphism)
    (= (FUNC$object typ)   cls$type)
    (= (FUNC$morphism typ) cls.info$type)

(FUNC$functor instance-power)
(= (FUNC$source instance-power) (CAT$opposite set))
(= (FUNC$target instance-power) classification)
(= (FUNC$underlying instance-power) cls.info$instance-power-graph-morphism)
    (= (FUNC$object instance-power) cls$instance-power)
    (= (FUNC$morphism instance-power) cls.info$instance-power)
```

The Namespace of Large Natural Transformations

NAT

Natural Transformations

○ Suppose that two functors $F_0, F_1 : C_0 \rightarrow C_1$ share a common source category C_0 and a common target category C_1. A natural transformation τ from source functor F_0 to target functor F_1, written 1-dimensionally as $\tau : F_0 \Rightarrow F_1 : C_0 \rightarrow C_1$ or visualized 2-dimensionally in Figure 1, is a collection of morphisms in the target category parameterized by objects in the source category that link the functorial images:

$$\begin{tikzcd} C_0 \arrow[r, bend left, "F_0"] \arrow[r, bend right, "F_1"'] \arrow[r, phantom, "\Downarrow \tau"] & C_1 \end{tikzcd}$$

Figure 1: Natural Transformation

$\tau = \{\tau_o : F_0(o) \rightarrow F_1(o) \mid o \in Obj(C_0)\}$.

A natural transformation is determined by its (source-functor, target-functor, component) triple. The KIF encoding for the declaration of a natural transformation is as follows.

```
(1) (CNG$conglomerate natural-transformation)

(2) (CNG$function source-functor)
    (CNG$signature source-functor natural-transformation FUNC$functor)

(3) (CNG$function target-functor)
    (CNG$signature target-functor natural-transformation FUNC$functor)

(4) (CNG$function source-category)
    (CNG$signature source-category natural-transformation CAT$category)

(5) (CNG$function target-category)
    (CNG$signature target-category natural-transformation CAT$category)

    (forall (?tau (natural-transformation ?tau))
        (and (= (FUNC$source (source-functor ?tau))
                (source-category ?tau))
                (FUNC$source (target-functor ?tau))
                (source-category ?tau))
             (= (FUNC$target (source-functor ?tau))
                (target-category ?tau))
                (FUNC$target (target-functor ?tau))
                (target-category ?tau))))

(6) (CNG$function component)
    (CNG$signature component natural-transformation SET.FTN$function)

    (forall (?tau ?o ?m)
        (and (= (SET.FTN$source (component ?tau))
                (CAT$object (source-category ?tau)))
             (= (SET.FTN$target (component ?tau))
                (CAT$morphism (target-category ?tau)))))

    (forall (?n1 (natural-transformation ?n1) ?n2 (natural-transformation ?n2))
        (=> (and (= (source-functor ?n1) (source-functor ?n2))
                 (= (target-functor ?n1) (target-functor ?n2))
                 (= (component ?n1) (component ?n2)))
            (= ?n1 ?n2)))
```

The source and target objects of the components of a natural transformation are image objects of the source and target functors:

$$src(C_1)(\tau(o)) = obj(F_0)(o) \text{ and } tgt(C_1)(\tau(o)) = obj(F_1)(o)$$

for any object $o \in Obj(C_0)$.

Here is the KIF representation for component source and target.

```
    (forall (?tau (natural-transformation ?tau))
        (and (= (SET.FTN$composition
                    (component ?tau)
                    (CAT$source (target-category ?tau)))
                (FUNC$object (source-functor ?tau)))
             (= (SET.FTN$composition
                    (component ?tau)
                    (CAT$target (target-category ?tau)))
                (FUNC$object (target-functor ?tau)))))
```

The components of a natural transformation interact with the functorial images by satisfying the commutative Diagram 1:

$$mor(F_0)(m) \circ^1 \tau(\partial_1^{\,0}(m)) = \tau(\partial_0^{\,0}(m)) \circ^1 mor(F_1)\,(m)$$

for any morphism $m \in Mor(C_0)$. Here is the (somewhat complicated) KIF representation for this interaction, which uses pullback pairing with respect to the composition opspan of the target category of the natural transformation. Again, this is the logical KIF formalization of the commutative diagram for a natural transformation (Diagram 1).

$$\begin{tikzcd} o \arrow[d, "m"'] & F_0(o) \arrow[r, "\tau(o)"] \arrow[d, "F_0(m)"'] & F_1(o) \arrow[d, "F_1(m)"] \\ o' & F_0(o') \arrow[r, "\tau(o')"'] & F_1(o') \end{tikzcd}$$

Diagram 1: Natural Transformation

```
(forall (?tau (natural-transformation ?tau))
    (= (SET.FTN$composition
           ((SET.LIM.PBK$pairing (CAT$composable-opspan (target-category ?tau)))
               (SET.FTN$composition
                   (CAT$source (source-category ?tau))
                   (component ?tau)))
               (FUNC$morphism (target-functor ?tau))
           (CAT$composition (target-category ?tau)))
       (SET.FTN$composition
           ((SET.LIM.PBK$pairing (CAT$composable-opspan (target-category ?tau)))
               (FUNC$morphism (source-functor ?tau))
               (SET.FTN$composition
                   (CAT$target (source-category ?tau))
                   (component ?tau)))
           (CAT$composition (target-category ?tau)))))
```

2-Dimensional Category Structure

- A pair of natural transformations σ and τ is *vertically composable* when the target functor of the first is the source functor of the second, $\sigma : F_0 \Rightarrow F_1 : C_0 \to C_1$ and $\tau : F_1 \Rightarrow F_2 : C_0 \to C_1$. The *vertical composition* $\sigma \bullet \tau : F_0 \Rightarrow F_2 : C_0 \to C_1$ of two vertically composable natural transformations is defined as morphism composition in the target category: $\sigma \bullet \tau\ (o) = \sigma(o) \cdot \tau(o)$ for any object $o \in Obj(C_0)$. Diagram 2 illustrates vertical composition.

$$\begin{tikzcd} o \arrow[d, "m"'] & F_0(o) \arrow[r, "\sigma(o)"] \arrow[d, "F_0(m)"'] & F_1(o) \arrow[r, "\tau(o)"] \arrow[d, "F_1(m)"] & F_2(o) \arrow[d, "F_2(m)"] \\ o' & F_0(o') \arrow[r, "\sigma(o')"'] & F_1(o') \arrow[r, "\tau(o')"'] & F_2(o') \end{tikzcd}$$

$$\begin{tikzcd} C_0 \arrow[r, bend left=60, "F_0"] \arrow[r, "" description] \arrow[r, bend right=60, "F_2"'] & C_1 \end{tikzcd} \quad \Downarrow\sigma,\ \Downarrow\tau$$

Diagram 2: Vertical composition

- For any functor $F : C_0 \to C_1$ there is a *vertical identity* natural transformation $1_F : F \Rightarrow F : C_0 \to C_1$ defined in terms of identity morphisms in the target category: $1_F\ (o) = id_{obj(F)(o)}$ for any object $o \in Obj(C_0)$. Here is the KIF representation for vertical composition.

```
(7) (CNG$function vertical-composition)
    (CNG$signature vertical-composition
        natural-transformation natural-transformation natural-transformation)

    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (<=> (exists (?n (natural-transformation ?n))
                 (= ?n (vertical-composition ?sigma ?tau))))
             (= (target-functor ?sigma) (source-functor ?tau))))

    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (=> (= (target-functor ?sigma) (source-functor ?tau))
            (and (= (source-functor (vertical-composition ?sigma ?tau))
                    (source-functor ?sigma))
                 (= (target-functor (vertical-composition ?sigma ?tau))
                    (target-functor ?tau))
                 (= (source-category (vertical-composition ?sigma ?tau))
                    (source-category ?sigma))
                 (= (target-category (vertical-composition ?sigma ?tau))
                    (target-category ?sigma)))))

    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (=> (= (target-functor ?sigma) (source-functor ?tau))
            (= (component (vertical-composition ?sigma ?tau))
               (SET.FTN$composition
                   ((SET.LIM.PBK$pairing
                        (CAT$composable-opspan (target-category ?sigma)))
                       (component ?sigma))
                       (component ?tau)))
                   (CAT$composition (target-category ?sigma))))))

(8) (CNG$function vertical-identity)
    (CNG$signature vertical-identity FUNC$functor natural-transformation)

    (forall (?f (FUNC$functor ?f))
        (and (= (source-functor (vertical-identity ?f)) ?f)
             (= (target-functor (vertical-identity ?f)) ?f)))

    (forall (?f (FUNC$functor ?f) ?o ((CAT$object (FUNC$source ?f)) ?o))
        (= (component (vertical-identity ?f))
           (SET.FTN$composition (FUNC$object ?f) (CAT$identity (FUNC$target ?f)))))
```

- A pair of natural transformations σ and τ is *horizontally composable* when the target category of the first is the source category of the second, $\sigma : F_0 \Rightarrow F_1 : C_0 \rightarrow C_1$ and $\tau : G_0 \Rightarrow G_1 : C_1 \rightarrow C_2$. The *horizontal composition* $\sigma \circ \tau : F_0 \circ G_0 \Rightarrow F_1 \circ G_1 : C_0 \rightarrow C_2$ of two horizontally composable natural transformations is defined by pasting together naturality diagrams:

$$\begin{aligned} \sigma \circ \tau\,(o) &= \sigma{\circ}G_0(o) \cdot F_1{\circ}\tau(o) = \mathbf{G_0(\sigma(o)) \cdot \tau(F_1(o))} \\ &= F_0{\circ}\tau(o) \cdot \sigma{\circ}G_1(o) = \tau(F_0(o)) \cdot G_1(\sigma(o)) \end{aligned}$$

  for any object $o \in Obj(C_0)$. The alternate definitions are equal, due to the naturality of τ. Diagram 3 illustrates horizontal composition.

$$\begin{tikzcd} o \arrow[d, "m"'] & G_0(F_0(o)) \arrow[r, "G_0(\sigma(o))"] \arrow[d, "F_0(m)"'] & G_0(F_1(o)) \arrow[r, "\tau(F_1(o))"] \arrow[d, "F_1(m)"] & G_1(F_1(o)) \arrow[d, "F_2(m)"] \\ o' & G_0(F_0(o')) \arrow[r, "G_0(\sigma(o'))"'] & G_0(F_1(o')) \arrow[r, "\tau(F_1(o'))"'] & G_1(F_1(o')) \end{tikzcd}$$

$$\begin{tikzcd} C_0 \arrow[r, bend left, "F_0"] \arrow[r, bend right, "F_1"'] \arrow[r, phantom, "\Downarrow \sigma"] & C_1 \arrow[r, bend left, "G_0"] \arrow[r, bend right, "G_1"'] \arrow[r, phantom, "\Downarrow \tau"] & C_2 \end{tikzcd}$$

Diagram 3: Horizontal composition

- For any category C there is a *horizontal identity* natural transformation $1_C : id_C \Rightarrow id_C : C \rightarrow C$ defined in terms of identity morphisms in the category C: $1_C\,(o) = id^C{}_o$ for any object $o \in Obj(C)$. Here is the KIF representation for horizontal composition. We use the first alternative definition (in boldface).

```
(9) (CNG$function horizontal-composition)
    (CNG$signature horizontal-composition
        natural-transformation natural-transformation natural-transformation)
    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (<=> (exists (?n (natural-transformation ?n))
                 (= ?n (horizontal-composition ?sigma ?tau))))
             (= (target-category ?sigma) (source-category ?tau))))

    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (=> (= (target-category ?sigma) (source-category ?tau))
            (and (= (source-functor (horizontal-composition ?sigma ?tau))
                    (FUNC$composition (source-functor ?sigma) (source-functor ?tau)))
                 (= (target-functor (horizontal-composition ?sigma ?tau))
                    (FUNC$composition (target-functor ?sigma) (target-functor ?tau)))
                 (= (source-category (horizontal-composition ?sigma ?tau))
                    (source-category ?sigma))
                 (= (target-category (horizontal-composition ?sigma ?tau))
                    (target-category ?tau)))))

    (forall (?sigma (natural-transformation ?sigma)
             ?tau (natural-transformation ?tau))
        (=> (= (target-category ?sigma) (source-category ?tau))
            (= (component (horizontal-composition ?sigma ?tau))
               (SET.FTN$composition
                   ((SET.LIM.PBK$pairing
                        (CAT$composable-opspan (target-category ?sigma)))
                       (SET.FTN$composition
                           (component ?sigma)
                           (FUNC$morphism (source-functor ?tau)))
                       (SET.FTN$composition
                           (FUNC$object (target ?sigma))
                           (component ?tau))
                   (CAT$composition (target-category ?tau))))))

(10) (CNG$function horizontal-identity)
     (CNG$signature horizontal-identity CAT$category natural-transformation)

     (forall (?c (CAT$category ?c))
         (and (= (source-functor (horizontal-identity ?c))
                 (FUNC$identity ?c))
              (= (target-functor (horizontal-identity ?c))
                 (FUNC$identity ?c))))

     (forall (?c (CAT$Category ?c) ?o ((CAT$object ?c) ?o))
         (= (component (horizontal-identity ?c))
            (FUNC$identity ?c)))
```

○ Given any category C and any category J, the *diagonal natural transformation* $\Delta_{J,\,C}$ maps a morphism $m : o_0 \rightarrow o_1$ to an associated constant natural transformation $\Delta_{J,\,C}(m) : \Delta_{J,\,C}(o_0) \Rightarrow \Delta_{J,\,C}(o_1) : J \rightarrow C$ between constant functors – its component function maps each object $j \in$ *obj(J)* to the morphism $m \in$ *mor(C)*. This complete the definition of the diagonal functor $\Delta_{J,\,C} : C \rightarrow C^J$.

```
    (11) (KIF$function diagonal)
         (KIF$signature diagonal CAT$category CAT$category CNG$function)
         (forall (?j (CAT$category ?j) ?c (CAT$category ?c))
             (and (CNG$signature (diagonal ?j ?c)
                      (CAT$morphism ?c) natural-transformation)
                  (forall (?m ((CAT$morphism ?c) ?m))
                      (and (= (source-category ((diagonal ?j ?c) ?m)) ?j)
                           (= (target-category ((diagonal ?j ?c) ?m)) ?c)
                           (= (source-functor ((diagonal ?j ?c) ?m))
                              ((diagonal ?j ?c) ((CAT$source ?c) ?m)))
                           (= (target-functor ((diagonal ?j ?c) ?m))
                              ((diagonal ?j ?c) ((CAT$target ?c) ?m)))))))
```

Natural Transformation Theorems

- There is a quasi-category (a foundationally large category with object and morphism collections that are conglomerates), whose objects are functors, whose morphisms are natural transformation, whose source and target are the source and target functors for a natural transformation, whose composition is vertical composition, and whose identities are the vertical identities.
- There is a quasi-category, whose objects are categories, whose arrows are natural transformation, whose source and target are the source and target categories for a natural transformation, whose composition is horizontal composition, and whose identities are the horizontal identities.

We can prove the following theorems.

- The horizontal composition can be written in two different forms:

  $$\sigma \circ \tau = (1_{F0} \circ \tau) \cdot (\sigma \circ 1_{G1}) = (\sigma \circ 1_{G0}) \cdot (1_{F1} \circ \tau).$$

- Vertical and horizontal composition satisfy the *interchange law* (Figure 2):

  $$(\sigma_0 \cdot \sigma_1) \circ (\tau_0 \cdot \tau_1) = (\sigma_0 \circ \tau_0) \cdot (\sigma_1 \circ \tau_1)$$

  for any three categories, six functors and four natural transformations as in the diagram on the right. Here is the KIF formalization of the interchange law.

$$\begin{tikzcd}
C_0 \arrow[r, shift left=8, "F_0"] \arrow[r] \arrow[r, shift right=8, "F_2"'] \arrow[r, phantom, shift left=4, "\Downarrow \sigma_0"] \arrow[r, phantom, shift right=4, "\Downarrow \sigma_1"] & C_1 \arrow[r, shift left=8, "G_0"] \arrow[r] \arrow[r, shift right=8, "G_2"'] \arrow[r, phantom, shift left=4, "\Downarrow \tau_0"] \arrow[r, phantom, shift right=4, "\Downarrow \tau_1"] & C_2
\end{tikzcd}$$

Figure 2: Interchange Law

```
(forall (?sigma0 (natural-transformation ?sigma0)
         ?sigma1 (natural-transformation ?sigma1)
         ?tau0 (natural-transformation ?tau0)
         ?tau1 (natural-transformation ?tau1))
    (=> (and (= (target-functor ?sigma0) (source-functor ?sigma1))
             (= (target-functor ?tau0) (source-functor ?tau1))
             (= (target-category ?sigma0) (source-category ?tau0)))
        (= (horizontal-composition
               (vertical-composition ?sigma0 ?sigma1)
               (vertical-composition ?tau0 ?tau1))
           (vertical-composition
               (horizontal-composition ?sigma0 ?tau0)
               (horizontal-composition ?sigma1 ?tau1)))))
```

The Namespace of Large Adjunctions

Adjunctions

`ADJ`

Categories are not only related by functors but also related through adjunctions.

$$\begin{tikzcd} A \arrow[r, "\langle F,\, U,\, \eta,\, \varepsilon \rangle"] & B \end{tikzcd}$$

Figure 1: Adjunction

- An *adjunction* (Figure 1) $\langle F, U, \eta, \varepsilon\rangle : A \rightarrow B$ consists of a pair of natural transformations called the *unit* $\eta : Id_A \Rightarrow F \cdot U$ and *counit* $\varepsilon : U \cdot F \Rightarrow Id_B$ of the adjunction, a pair of functors called the *left adjoint* (or free functor) $F : A \rightarrow B$ and the *right adjoint* (or underlying functor) $U : B \rightarrow A$ of the adjunction, and a pair of categories called the *underlying category A* and *free category B* of the adjunction. An adjunction is governed by a pair of *triangle identities*,

  $$\eta F \bullet F\varepsilon = 1_F \qquad \text{and} \qquad \varepsilon U \bullet U\eta = 1_U,$$

  as illustrated in Diagram 1.

$$\begin{tikzcd} A \arrow[r, "F"] \arrow[dr, "Id_A"', ""{name=IA}] & B \arrow[d, "U"] \arrow[dr, "Id_B"] \arrow[dl, phantom, "\overset{\eta}{\Rightarrow}"] & & \\ & A \arrow[r, "F"', "\varepsilon"] \arrow[ur, phantom, "\Rightarrow"] & B & = 1_F \end{tikzcd} \qquad \begin{tikzcd} B \arrow[d, "U"'] \arrow[dr, "Id_B"] & & \\ A \arrow[r, "F"', "\overset{\varepsilon}{\Rightarrow}"] \arrow[dr, "Id_A"'] & B \arrow[d, "U"] \arrow[dl, phantom, "\overset{\Rightarrow}{\eta}"] & = 1_U \\ & A & \end{tikzcd}$$

Diagram 1: Triangle identities

Here is the KIF representation for adjunctions. The conglomerate '`adjunction`' declared in axiom (1) represents abstract adjunctions, allowing one to *declare* adjunctions themselves. The functions '`underlying-category`' and '`free-category`' declared in axioms (2–3), represent the category aspect of adjunctions, allowing one to *declare* the underlying and free categories of adjunctions. The terms of axioms (4–7), which represent the functorial aspect of adjunctions by the functions '`left-adjoint`' and '`right-adjoint`' and the natural transformation aspect by the functions '`unit`' and '`counit`', resolve adjunctions into their parts. Axioms (8–9) represent the left-hand triangle and right-hand triangle equality in Diagram 1. Axiom (10) represents the fact that adjunctions are determined by their (underlying-category, free-category, left-adjoint, right-adjoint, unit, counit) sextuples.

```
(1) (CNG$conglomeration adjunction)

(2) (CNG$function underlying-category)
    (CNG$signature underlying-category adjunction CAT$category)

(3) (CNG$function free-category)
    (CNG$signature free-category adjunction CAT$category)

(4) (CNG$function left-adjoint)
    (CNG$signature left-adjoint adjunction FUNC$functor)
    (forall (?a (adjunction ?a))
        (and (= (CAT$source (left-adjoint ?a)) (underlying-category ?a))
             (= (CAT$target (left-adjoint ?a)) (free-category ?a))))

(5) (CNG$function right-adjoint)
    (CNG$signature right-adjoint adjunction FUNC$functor)
    (forall (?a (adjunction ?a))
        (and (= (CAT$source (right-adjoint ?a)) (free-category ?a))
             (= (CAT$target (right-adjoint ?a)) (underlying-category ?a))))

(6) (CNG$function unit)
    (CNG$signature unit adjunction NAT$natural-transformation)
    (forall (?a (adjunction ?a))
        (and (= (NAT$source (unit ?a))
                (FUNC$identity (underlying-category ?a)))
             (= (NAT$target (unit ?a))
                (FUNC$composition (left-adjoint ?a) (right-adjoint ?a)))))

(7) (CNG$function counit)
    (CNG$signature counit adjunction NAT$natural-transformation)
    (forall (?a (adjunction ?a))
        (and (= (NAT$source (counit ?a))
                (FUNC$composition (right-adjoint ?a) (left-adjoint ?a)))
             (= (NAT$target (counit ?a))
                (FUNC$identity (free-category ?a)))))

(8) (forall (?a (adjunction ?a))
        (= (NAT$vertical-composition
               (NAT$horizontal-composition
                   (unit ?a) (NAT$vertical-identity (left-adjoint ?a)))
               (NAT$horizontal-composition
                   (NAT$vertical-identity (left-adjoint ?a)) (counit ?a)))
           (NAT$vertical-identity (left-adjoint ?a)))))

(9) (forall (?a (adjunction ?a))
        (= (NAT$vertical-composition
               (NAT$horizontal-composition
                   (counit ?a) (NAT$vertical-identity (right-adjoint ?a)))
               (NAT$horizontal-composition
                   (NAT$vertical-identity (right-adjoint ?a)) (unit ?a)))
           (NAT$vertical-identity (right-adjoint ?a)))))

(10) (forall (?a1 (adjunction ?a1) ?a2 (adjunction ?a2))
         (=> (and (= (underlying-category ?a1) (underlying-category ?a2))
                  (= (free-category ?a1) (free-category ?a2))
                  (= (left-adjoint ?a1) (left-adjoint ?a2))
                  (= (right-adjoint ?a1) (right-adjoint ?a2))
                  (= (unit ?a1) (unit ?a2))
                  (= (counit ?a1) (counit ?a2)))
             (= ?a1 ?a2)))
```

- Adjunctions and universal morphisms are closely related – we can prove the following theorem: if $U : B \rightarrow A$ is any functor, then U is the right adjoint functor in an adjunction $\langle F, U, \eta, \varepsilon\rangle : A \rightarrow B$ iff there is a universal morphism for every object $a \in obj(A)$. Given the adjunction, the universal morphism for $a \in obj(A)$ is given by $\langle \eta_a, F(a)\rangle$. Given the collection of universal morphisms $\{\langle m_a, \tilde{a}\rangle \mid a \in obj(A)\}$, define the object part of F by $F(a) = \tilde{a}$ and define the morphism part of F by $F(m : a \rightarrow a')$ is the unique morphism $\eta_a \cdot_A U(F(m)) = m \cdot \eta_{\hat{a}}$.

  Here is the KIF formalization of this theorem.

```
    (forall (?u (functor ?u))
        (<=> (exists (?adj (adjunction ?adj))
                 (= ?u (right-adjoint ?adj))
             (forall (?a ((CAT$object (target ?u)) ?a))
                 (exists (?x (((universal-morphism ?f) ?a) ?x)))))))
```

- It is a standard fact that every adjunction $\langle F, U, \eta, \varepsilon\rangle : A \rightarrow B$ gives rise to a monad $\langle T, \eta, \mu\rangle$ in a category A, where

$$\begin{aligned} T &= F \circ U \\ \eta &= \eta \\ \mu &= 1_F \circ \varepsilon \circ 1_U \end{aligned}$$

```
(11) (CNG$function adjunction-monad)
     (CNG$signature adjunction-monad adjunction MND$monad)
     (forall (?a (adjunction ?a))
         (and (= (MND$underlying-category (adjunction-monad ?a))
                 (underlying-category ?a))
```

```
                  (= (MND$endofunctor (adjunction-monad ?a))
                     (FUNC$composition (left-adjoint ?a) (right-adjoint ?a)))
                  (= (MND$unit (adjunction-monad ?a))
                     (unit ?a))
                  (= (MND$multiplication (adjunction-monad ?a))
                     (NAT$horizontal-composition
                         (NAT$vertical-identity (left-adjoint ?a))
                         (NAT$horizontal-composition
                             (counit ?a)
                             (NAT$vertical-identity (right-adjoint ?a)))))))
```

○ Consider the opposite direction. We know that every monad $\langle T, \eta, \mu\rangle$ gives rise to two distinguished adjunctions, the Eilenberg-Moore adjunction $\langle F^M, U^M, \eta^M, \varepsilon^M\rangle : A^M \rightarrow A$ and the Kliesli adjunction $\langle F_M, U_M, \eta_M, \varepsilon_M\rangle : A_M \rightarrow A$. If that monad is the one generated by an adjunction $\langle F, U, \eta, \varepsilon\rangle : A \rightarrow B$, then the three adjunctions are comparable: there exists two distinguished functors, the *Kliesli comparison functor* $K_M : A_M \rightarrow B$ and the *Eilenberg-Moore comparison functor* $K^M : B \rightarrow A^M$, that satisfy the following identities.

$$\begin{aligned} U &= K^M \circ U^M \\ F^M &= F \circ K^M \\ U_M &= K_M \circ U \\ F &= F_M \circ K_M \end{aligned}$$

The Eilenberg-Moore comparison functor $K^M : B \rightarrow A^M$ maps an object $b \in obj(B)$ to the *free* algebra $K^M(b) = \langle U(b), U(\varepsilon(b))\rangle$ and maps a B-morphism $h : b \rightarrow b'$ to the homomorphism $F^M(h) = U(h) : \langle U(b), U(\varepsilon(b))\rangle \rightarrow \langle U(b'), U(\varepsilon(b'))\rangle$.

The Kliesli comparison functor $K_M : A_M \rightarrow B$ maps an object $a \in obj(A_M)$ to the B-object algebra $K^M(b) = F(a)$ and maps an A_M-morphism $\langle h, a'\rangle : a \rightarrow a'$, where $h : a \rightarrow T(a')$ is an A-morphism, to the *extension* B-morphism $F(h') \cdot_B \varepsilon(F(a')) : F(a) \rightarrow F(a')$.

```
(12) (CNG$function free)
     (CNG$signature free adjunction SET.FTN$function)
     (forall (?a (adjunction ?a))
         (and (= (SET.FTN$source (free ?a))
                 (CAT$object (underlying-category ?a)))
              (= (SET.FTN$target (free ?a))
                 (MND$algebra (adjunction-monad ?a)))
              (= (SET.FTN$composition
                     (free ?a)
                     (MND.ALG$underlying-object (adjunction-monad ?a)))
                 (FUNC$object (right-adjoint ?a)))
              (= (SET.FTN$composition
                     (free ?a)
                     (MND.ALG$structure-map (adjunction-monad ?a)))
                 (SET.FTN$composition
                     (NAT$component (counit ?a))
                     (FUNC$morphism (right-adjoint ?a))))))

(13) (CNG$function eilenberg-moore-comparison)
     (CNG$signature eilenberg-moore-comparison adjunction FUNC$functor)
     (forall (?a (adjunction ?a))
         (and (= (FUNC$source (eilenberg-moore-comparison ?a))
                 (free-category ?a))
              (= (FUNC$target (eilenberg-moore-comparison ?a))
                 (MND.ALG$eilenberg-moore (adjunction-monad ?a)))
              (= (FUNC$object (eilenberg-moore-comparison ?a))
                 (free ?a))
              (= (SET.FTN$composition
                     (FUNC$morphism (eilenberg-moore-comparison ?a))
                     (MND.ALG$underlying-morphism (adjunction-monad ?a)))
                 (FUNC$morphism (right-adjoint ?a)))))

(14) (CNG$function extension)
```

```
     (CNG$signature extension adjunction SET.FTN$function)
     (forall (?a (adjunction ?a))
         (and (= (SET.FTN$source (extension ?a))
                 (CAT$morphism (MND.ALG$kliesli (adjunction-monad ?a))))
              (= (SET.FTN$target (extension ?a))
                 (CAT$morphism (free-category ?a)))
              (= (extension ?a)
                 (SET.FTN$composition
                     (SET.LIM.PBK$pairing
                         (CAT$composable-opspan (MND$free-category ?a)))
                         (SET.FTN$composition
                             (SET.LIM.PBK$projection1
                                 (MND.ALG$kliesli-morphism-opspan ?m))
                             (FUNC$morphism (left-adjoint ?a)))
                         (SET.FTN$composition
                             (SET.LIM.PBK$projection2
                                 (MND.ALG$kliesli-morphism-opspan ?m))
                             (SET.FTN$composition
                                 (FUNC$object (left-adjoint ?a))
                                 (NAT$component (counit ?a)))))
                     (CAT$composition (free-category ?a))))))

(15) (CNG$function kliesli-comparison)
     (CNG$signature kliesli-comparison adjunction FUNC$functor)
     (forall (?a (adjunction ?a))
         (and (= (FUNC$source (kliesli-comparison ?a))
                 (MND.ALG$kliesli (adjunction-monad ?a)))
              (= (FUNC$target (kliesli-comparison ?a))
                 (free-category ?a))
              (= (FUNC$object (kliesli-comparison ?a))
                 (FUNC$object (left-adjoint ?a)))
              (= (FUNC$morphism (kliesli-comparison ?a))
                 (extension ?a))))
```

○ Given any two categories A and B, a (*strong*) *reflection* of A into B is an adjunction $\langle F, U, \eta, 1_{IdB}\rangle : A \rightarrow B$ whose counit is the identity, $\varepsilon = 1_{IdB}$, with $U \cdot F = Id_B$, so that U has injective object and morphism functions and B is a subcategory of A via U. That is, a subcategory $B \subseteq A$ is a *reflection* of A when the injection functor has a left adjoint right inverse (lari). Dually, given any two categories A and B, a (*strong*) *coreflection* of A into B is an adjunction $\langle F, U, 1_{IdA}, \varepsilon\rangle : A \rightarrow B$ whose unit is the identity, $\eta = 1_{IdA}$, with $F \cdot U = Id_A$, so that F has injective object and morphism functions and A is a subcategory of B via F. That is, a subcategory $A \subseteq B$ is a *coreflection* of B when the injection functor has a right adjoint right inverse (rari).

Here is the KIF formalization of for reflections and coreflections.

```
(16) (CNG$conglomeration reflection)
     (CNG$subconglomerate reflection adjunction)
     (forall (?a (adjunction ?a))
         (<=> (reflection ?a)
              (and (= (FUNC$composition (right-adjoint ?a) (left-adjoint ?a))
                      (FUNC$identity (free-category ?a)))
                   (= (counit ?a)
                      (NAT$vertical-identity (FUNC$identity (free-category ?a)))))))

(17) (CNG$conglomeration coreflection)
     (CNG$subconglomerate coreflection adjunction)
     (forall (?a (adjunction ?a))
         (<=> (coreflection ?a)
              (and (= (FUNC$composition (left-adjoint ?a) (right-adjoint ?a))
                      (FUNC$identity (underlying-category ?a)))
                   (= (unit ?a)
                      (NAT$vertical-identity
                          (FUNC$identity (underlying-category ?a)))))))
```

Adjunction Morphisms

ADJ.MOR

Adjunctions are related (vertically) by conjugate pairs of natural transformations.

○ Suppose that two adjunctions $\langle F, U, \eta, \varepsilon\rangle, \langle F', U', \eta', \varepsilon'\rangle : A \rightarrow B$ share a common underlying (source) category A and a common free (target) category B. A *conjugate pair* of natural transformations $\langle\sigma, \tau\rangle : \langle F, U, \eta, \varepsilon\rangle \Rightarrow \langle F', U', \eta', \varepsilon'\rangle : A \rightarrow B$ from source adjunction $\langle F, U, \eta, \varepsilon\rangle$ to target functor $\langle F', U', \eta', \varepsilon'\rangle$, visualized 2-dimensionally in Figure 2, consists of a *left conjugate* natural transformation $\sigma : F \Rightarrow F'$ between the left adjoint (free) functors and a (contravariant) *right conjugate* natural transformation $\tau : U' \Rightarrow U$ between the right adjoint (underlying) functors, which satisfy either of the equivalent conditions in Table 1:

$$\begin{tikzcd} A \arrow[r, bend left, "\langle F, U, \eta, \varepsilon\rangle"] \arrow[r, bend right, "\langle F', U', \eta', \varepsilon'\rangle"'] \arrow[r, phantom, "\Downarrow \langle\sigma,\tau\rangle"] & B \end{tikzcd}$$

Figure 2: Conjugate Pair

Table 1: Equivalent Conditions for Conjugate Pairs

$$\tau = U'\eta \bullet U'\sigma U \bullet \varepsilon' U \qquad \sigma = \eta' F \bullet F'\tau F \bullet F'\varepsilon$$

$$\tau F \bullet \varepsilon = U'\sigma \bullet \varepsilon' \qquad \eta \bullet \sigma U = \eta' \bullet F'\tau$$

As these equivalents indicate, the natural transformation σ determines the natural transformation τ, and vice versa. Therefore, a conjugate pair is determined by either its left or right conjugate natural transformation. Here is the KIF formalization of conjugate pairs. Axiom (16) gives the left two of the above equivalent conditions.

```
(11) (CNG$conglomeration conjugate-pair)

(12) (CNG$function source)
     (CNG$signature source conjugate-pair ADJ$adjunction)

(13) (CNG$function target)
     (CNG$signature target conjugate-pair ADJ$adjunction)

     (forall (?p (conjugate-pair ?p))
         (and (= (ADJ$underlying-category (source ?p))
                 (ADJ$underlying-category (target ?p)))
              (= (ADJ$free-category (source ?p))
                 (ADJ$free-category (target ?p)))))

(14) (CNG$function left-conjugate)
     (CNG$signature left-conjugate conjugate-pair NAT$natural-transformation)

(15) (CNG$function right-conjugate)
     (CNG$signature right-conjugate conjugate-pair NAT$natural-transformation)

    (forall (?p (conjugate-pair ?p))
         (and (= (NAT$source (left-conjugate ?a))
                 (ADJ$left-adjoint (source ?p)))
              (= (NAT$target (left-conjugate ?a))
                 (ADJ$left-adjoint (target ?p)))
              (= (NAT$source (right-conjugate ?a))
                 (ADJ$right-adjoint (target ?p)))
              (= (NAT$target (right-conjugate ?a))
                 (ADJ$right-adjoint (source ?p)))))

(16) (forall (?p (conjugate-pair ?p))
         (and [τ = U′η • U′σU • ε′U]
              (= (left-conjugate ?a)
                 (ADJ$vertical-composition
                   (ADJ$vertical-composition
                     (ADJ$horizontal-composition
                       (NAT$vertical-identity (ADJ$right-adjoint (target ?p)))
                       (ADJ$unit (source ?p)))
                     (ADJ$horizontal-composition
                       (ADJ$horizontal-composition
                         (NAT$vertical-identity (ADJ$right-adjoint (target ?p)))
                         (left-conjugate ?p))
                       (NAT$vertical-identity (ADJ$right-adjoint (source ?p)))))
                   (ADJ$horizontal-composition
                     (ADJ$counit (target ?p))
                     (NAT$vertical-identity (ADJ$right-adjoint (source ?p))))))
```

$[\tau F \bullet \varepsilon = U'\sigma \bullet \varepsilon']$

```
           (= (ADJ$vertical-composition
                (ADJ$horizontal-composition
                  (right-conjugate ?p)
                  (NAT$vertical-identity (ADJ$left-adjoint (source ?p))))
                (ADJ$counit (source ?p)))
              (ADJ$vertical-composition
                (ADJ$horizontal-composition
                  (NAT$vertical-identity (ADJ$right-adjoint (target ?p)))
                  (left-conjugate ?p))
                (ADJ$counit (target ?p))))
  ))
```

Examples

Here are examples of adjunctions defined elsewhere, but asserted to be functors here.

○ There is a natural transformation η from the identity functor on Classification to the composition of the underlying instance and instance power functors, whose component at any classification is the extent infomorphism associated with that classification. The underlying instance functor

  inst : Classification → Set^{op}

  is left adjoint *inst* ⊣ *pow* to the instance power functor

  pow : Set^{op} → Classification,

  and η is the unit of this adjunction. Here is the KIF formalization for these facts, expressed in an external namespace.

```
(NAT$natural-transformation eta)
(= (NAT$source eta) (FUNC$identity Classification))
(= (NAT$target eta) (FUNC$composition [instance instance-power]))
(= (NAT$component eta) cls$extent)

(FUNC$composable [instance instance-power])
(= (FUNC$composition [instance-power instance]) (FUNC$identity set))

(ADJ$adjunction inst-pow)
(= (ADJ$underlying-category inst-pow) Classification)
(= (ADJ$free-category inst-pow)       Set)
(= (ADJ$left-adjoint inst-pow)        instance)
(= (ADJ$right-adjoint inst-pow)       instance-power)
(= (ADJ$unit inst-pow)                eta)
(= (ADJ$counit inst-pow)              (NAT$identity (FUNC$identity Set)))
```

The Namespace of Large Monads

Monads and Monad Morphisms

`MND`

In one sense, monads are universal algebra lifted to category theory. For any type of algebra, such as groups, complete semilattices, etcetera, there is its category of algebras Alg, its forgetful functor U to Set and its left adjoint free functor F in the reverse direction. The composite functor $T = F \circ U$ on Set comes equipped with two natural transformations that give it a monoid-like structure.

○ A *monad* $\langle T, \eta, \mu \rangle$ on a category A is a triple consisting of an underlying endofunctor $T : A \rightarrow A$ and two natural transformations

$$\eta : Id_A \Rightarrow T \text{ and } \mu : T \circ T \Rightarrow T$$

which satisfy the following commuting diagrams (Table 1) (where $T^2 = T \circ T$ and $T^3 = T \circ T \circ T$):

$$\begin{tikzcd} T^3 \arrow[r, "Id_T \circ \mu"] \arrow[d, "\mu \circ Id_T"'] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"'] & T \end{tikzcd} \qquad \begin{tikzcd} T = I \circ T \arrow[r, "\eta \circ Id_T"] \arrow[dr, "Id_T"'] & T^2 \arrow[d, "\mu"] & T \circ I = T \arrow[l, "Id_T \circ \eta"'] \arrow[dl, "Id_T"] \\ & T & \end{tikzcd}$$

Table 1: Monad

The natural transformation is $\eta : Id_A \Rightarrow T$ called the *unit* of the monad, and the natural transformation $\mu : T \circ T \Rightarrow T$ is called the *multiplication* of the monad. In Table 1, the axiom on the left is called the *associative law* for the monad and the axioms on the right are called the *left unit law* and the *right unit law*, respectively.

Here is the KIF representation for monads. The conglomerate '`monad`' declared in axiom (1) represents the collection of monads, allowing one to *declare* monads. The function '`underlying-category`' declared in axiom (2) represents the underlying or base category of adjunctions. The terms of axioms (3–5), which represent the functorial and natural transformation aspects of monads by the functions '`underlying-functor`', '`unit`' and '`multiplication`', resolve adjunctions into their parts. Axiom (6) represents the associative law for adjunctions, and axioms (7) represent the left and right unit laws for adjunctions (Table 1). Axiom (8) represents the fact that monads are determined by their (underlying-functor, unit, multiplication) triples.

```
(1) (CNG$conglomerate monad)

(2) (CNG$function underlying-category)
    (CNG$signature underlying-category monad CAT$category)

(3) (CNG$function underlying-functor)
    (CNG$signature underlying-functor monad FUNC$functor)
    (forall (?m (monad ?m))
        (and (= (FUNC$source (underlying-functor ?m)) (underlying ?m))
             (= (FUNC$target (underlying-functor ?m)) (underlying ?m))))

(4) (CNG$function unit)
    (CNG$signature unit monad NAT$natural-transformation)
    (forall (?m (monad ?m))
        (and (= (FUNC$source-functor (unit ?m))
                (FUNC$identity (underlying-category ?m)))
             (= (FUNC$target-functor (unit ?m))
                (underlying-functor ?m))))

(5) (CNG$function multiplication)
    (CNG$signature multiplication monad NAT$natural-transformation)
    (forall (?m (monad ?m))
        (and (= (FUNC$source-functor (multiplication ?m))
                (FUNC$composition (underlying-functor ?m) (underlying-functor ?m)))
             (= (FUNC$target-functor (multiplication ?m))
                (underlying-functor ?m))))

(6) (forall (?m (monad ?m))
        (= (NAT$vertical-identity
               (NAT$horizontal-composition
                   (NAT$vertical-identity (underlying-functor ?m))
                   (multiplication ?m))
               (multiplication ?m))
           (NAT$vertical-identity
               (NAT$horizontal-composition
                   (multiplication ?m)
                   (NAT$vertical-identity (underlying-functor ?m)))
               (multiplication ?m))))

(7) (forall (?m (monad ?m))
        (and (= (NAT$vertical-identity
                    (NAT$horizontal-composition
                        (unit ?m)
                        (NAT$vertical-identity (underlying-functor ?m)))
                    (multiplication ?m))
                (NAT$vertical-identity (underlying-functor ?m)))
             (= (NAT$vertical-identity
                    (NAT$horizontal-composition
                        (NAT$vertical-identity (underlying-functor ?m))
                        (unit ?m))
                    (multiplication ?m))
                (NAT$vertical-identity (underlying-functor ?m)))))

(8) (forall (?m1 (monad ?m1) ?m2 (monad ?m2))
         (=> (and (= (underlying-functor ?m1) (underlying-functor ?m2))
                  (= (unit ?m1) (unit ?m2))
                  (= (multiplication ?m1) (multiplication ?m2)))
             (= ?m1 ?m2)))
```

○ Monads are related by their morphisms. A *morphism of monads* $\tau : \langle T, \eta, \mu\rangle \Rightarrow \langle T', \eta', \mu'\rangle$ is a natural transformation $\tau : T \Rightarrow T'$ which preserves multiplication and unit in the sense that the diagrams in Table 2 commute. In Table 2, the axiom on the left represents *preservation of multiplication* and the axiom on the right represents *preservation of unit*.

$$\begin{tikzcd}
T^2 \arrow[r, "\mu"] \arrow[d, "\tau \circ \tau"'] & T \arrow[d, "\tau"] \\
T'^2 \arrow[r, "\mu'"'] & T'
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
T \arrow[d, "\tau"'] & \\
& I \arrow[ul, "\eta"'] \arrow[dl, "\eta'"] \\
T' &
\end{tikzcd}$$

Table 2: Monad Morphism

```
(9) (CNG$conglomerate monad-morphism)

(10) (CNG$function source)
     (CNG$signature source monad-morphism monad)

(11) (CNG$function target)
     (CNG$signature source monad-morphism monad)

(12) (CNG$function underlying-natural-transformation)
     (CNG$signature underlying-natural-transformation
        monad-morphism NAT$natural-transformation)

(13) (forall (?t (monad-morphism ?t))
         (and (= (underlying-category (source ?t))
                 (underlying-category (target ?t)))
```

```
              (= (NAT$source-functor (underlying-natural-transformation ?t))
                 (underlying-functor (source ?t)))
              (= (NAT$target-functor (underlying-natural-transformation ?t))
                 (underlying-functor (target ?t)))))
              (= (NAT$vertical-identity (multiplication (source ?t)) ?t)
                 (NAT$vertical-identity
                     (NAT$horizontal-composition ?t ?t)
                     (multiplication (target ?t))))
              (= (NAT$vertical-identity (unit (source ?t)) ?t)
                 (unit (target ?t)))))
```

Algebras and Freeness

MND.ALG

For any monad $M = \langle T, \eta, \mu\rangle$ the Eilenberg-Moore category of algebras represents universal algebras and their homomorphisms.

○ If $M = \langle T, \eta, \mu\rangle$ is a monad on category A, then an *M-algebra* $\langle a, \xi\rangle$ is a pair consisting of an object $a \in obj(A)$ (the *underlying object* of the algebra), and a morphism $\xi : T(a) \rightarrow a$ (called the *structure map* of the algebra) which makes the diagrams in Table 3 commute. The diagram on the left in Table 3 is called the *associative law* for the algebra and the diagram on the right is called the *unit law*.

$$\begin{tikzcd}
T^2(a) \arrow[r, "T(\xi)"] \arrow[d, "\mu(a)"'] & T(a) \arrow[d, "\xi"] \\
T(a) \arrow[r, "\xi"'] & a
\end{tikzcd}
\qquad
\begin{tikzcd}
 & T(a) \arrow[d, "\xi"] \\
a \arrow[ur, "\eta(a)"] \arrow[r, "id(a)"'] & a
\end{tikzcd}$$

Table 3: Algebra

```
(1) (CNG$function algebra)
    (CNG$signature algebra MND$monad SET.class)

(2) (CNG$function underlying-object)
    (CNG$signature underlying-object MND$monad SET.FTN.function)
    (forall (?m (MND$monad ?m))
        (and (= (SET.FTN$source (underlying-object ?m))
                (algebra ?m))
             (= (SET.FTN$target (underlying-object ?m))
                (CAT$object (MND$underlying-category ?m)))))

(3) (CNG$function structure-map)
    (CNG$signature structure-map MND$monad SET.FTN.function)
    (forall (?m (MND$monad ?m))
        (and (= (SET.FTN$source (structure-map ?m))
                (algebra ?m))
             (= (SET.FTN$target (structure-map ?m))
                (CAT$morphism (MND$underlying-category ?m)))
             (= (SET.FTN$composition
                    (structure-map ?m)
                    (CAT$source (MND$underlying-category ?m)))
                (SET.FTN$composition
                    (underlying-object ?m)
                    (FUNC$object (MND$underlying-functor ?m))))
             (= (SET.FTN$composition
                    (structure-map ?m)
                    (CAT$target (MND$underlying-category ?m)))
                (underlying-object ?m))
             (forall (?a ((algebra ?m) ?a))
                 (and (= ((CAT$composition (MND$underlying-category ?m))
                             [(FUNC$morphism (MND$underlying-functor ?m))
                                  ((structure-map ?m) ?a))
                              ((structure-map ?m) ?a)])
                         ((CAT$composition (MND$underlying-category ?m))
                             [((NAT$component (MND$multiplication ?m))
                                  ((underlying-object ?m) ?a))
                              ((structure-map ?m) ?a)]))
                      (= ((CAT$composition (MND$underlying-category ?m))
                             [((NAT$component (MND$unit ?m))
                                  ((underlying-object ?m) ?a))
                              ((structure-map ?m) ?a)]))
                         ((CAT$identity (MND$underlying-category ?m))
                             ((underlying-object ?m) ?a)))))))

(4) (CNG$function algebra-pair)
    (CNG$signature algebra-pair MND$monad SET.LIM.PRD$diagram)
    (forall (?m (MND$monad ?m))
        (and (= (SET.LIM.PRD$class1 (algebra-pair ?m)) (algebra ?m))
             (= (SET.LIM.PRD$class2 (algebra-pair ?m)) (algebra ?m))))
```

○ If $M = \langle T, \eta, \mu\rangle$ is a monad on category A, an *M-homomorphism* $h : \langle a, \xi\rangle \rightarrow \langle a', \xi'\rangle$ is an A-morphism $h : a \rightarrow a'$ between the underlying objects that preserves the algebraic structure by satisfying the commutative diagram in Table 4.

$$\begin{tikzcd}
T(a) \arrow[r, "\xi"] \arrow[d, "T(h)"'] & a \arrow[d, "h"] \\
T(a') \arrow[r, "\xi'"'] & a'
\end{tikzcd}$$

Table 4: Homomorphism

```
(5) (CNG$function homomorphism)
    (CNG$signature homomorphism MND$monad SET.class)

(6) (CNG$function source)
    (CNG$signature source MND$monad SET.FTN.function)
    (forall (?m (MND$monad ?m))
        (and (= (SET.FTN$source (source ?m))
                (homomorphism ?m))
             (= (SET.FTN$target (source ?m))
                (algebra ?m))))

(7) (CNG$function target)
    (CNG$signature target MND$monad SET.FTN.function)
    (forall (?m (MND$monad ?m))
        (and (= (SET.FTN$source (target ?m))
                (homomorphism ?m))
             (= (SET.FTN$target (target ?m))
                (algebra ?m))))

(8) (CNG$function underlying-morphism)
    (CNG$signature underlying-morphism MND$monad SET.FTN.function)
    (forall (?m (MND$monad ?m))
        (and (= (SET.FTN$source (underlying-morphism ?m))
                (homomorphism ?m))
             (= (SET.FTN$target (underlying-morphism ?m))
                (CAT$morphism (MND$underlying-category ?m)))
             (= (SET.FTN$composition
                    (underlying-morphism ?m)
                    (CAT$source (MND$underlying-category ?m)))
                (SET.FTN$composition (source ?m) (underlying-object ?m)))
             (= (SET.FTN$composition
                    (underlying-morphism ?m)
                    (CAT$target (MND$underlying-category ?m)))
                (SET.FTN$composition (target ?m) (underlying-object ?m)))
             (forall (?h ((homomorphism ?m) ?h))
                 (= ((CAT$composition (MND$underlying-category ?m))
                        [((structure-map ?m) (source ?h)) ?h])
                    ((CAT$composition (MND$underlying-category ?m))
                        [((FUNC$morphism (underlying-functor ?m)) ?h)
```

```
                              ((structure-map ?m) (target ?h))])))))

(9) (CNG$function composable-opspan)
    (CNG$signature composable-opspan MND$monad SET.LIM.PBK$diagram)
    (forall (?m (MND$monad ?m))
        (and (= (SET.LIM.PRD$class1 (composable-opspan ?m)) (homomorphism ?m))
             (= (SET.LIM.PRD$class2 (composable-opspan ?m)) (homomorphism ?m))
             (= (SET.LIM.PRD$opvertex (composable-opspan ?m)) (algebra ?m))
             (= (SET.LIM.PRD$opfirst (composable-opspan ?m)) (target ?m))
             (= (SET.LIM.PRD$opsecond (composable-opspan ?m)) (source ?m))))

(10) (CNG$function composable)
     (CNG$signature composable MND$monad SET$class)
     (forall (?m (MND$monad ?m))
         (= (composable ?m)
            (SET.LIM.PBK$pullback (composable-opspan ?m))))

(11) (CNG$function composition)
     (CNG$signature composition MND$monad SET.FTN.function)
     (forall (?m (MND$monad ?m))
         (and (= (SET.FTN$source (composition ?m))
                 (composable ?m))
              (= (SET.FTN$target (composition ?m))
                 (homomorphism ?m))
              (forall (?h1 ?h2 ((composable ?m) [?h1 ?h2]))
                  (= (underlying-morphism ((composition ?m) [?h1 ?h2]))
                     ((CAT$composition (MND$underlying-category ?m))
                         [(underlying-morphism ?h1) (underlying-morphism ?h2)])))))

(12) (CNG$function identity)
     (CNG$signature identity MND$monad SET.FTN.function)
     (forall (?m (MND$monad ?m))
         (and (= (SET.FTN$source (identity ?m))
                 (algebra ?m))
              (= (SET.FTN$target (identity ?m))
                 (homomorphism ?m))
              (forall (?a ((algebra ?m) ?a))
                  (= (underlying-morphism ((identity ?m) ?a))
                     ((CAT$identity (MND$underlying-category ?m))
                         ((underlying-object ?m) ?a))))))
```

- For any monad $M = \langle T, \eta, \mu \rangle$ on category A, the M-algebras and M-homomorphisms form the *Eilenberg-Moore category* A^M.

```
(13) (CNG$function eilenberg-moore)
     (CNG$signature eilenberg-moore MND$monad CAT$category)
     (forall (?m (monad ?m))
         (and (= (CAT$object (eilenberg-moore ?m))
                 (algebra ?m))
              (= (CAT$morphism (eilenberg-moore ?m))
                 (homomorphism ?m))
              (= (CAT$source (eilenberg-moore ?m))
                 (source ?m))
              (= (CAT$target (eilenberg-moore ?m))
                 (target ?m))
              (= (CAT$composition (eilenberg-moore ?m))
                 (composition ?m))
              (= (CAT$identity (eilenberg-moore ?m))
                 (identity ?m))))
```

- For any monad $M = \langle T, \eta, \mu \rangle$ on category A, there is an *underlying* functor $U^M : A^M \rightarrow A$, a *free* functor $F^M : A \rightarrow A^M$ that maps an object $a \in obj(A)$ to the "free" algebra $\langle a, \mu(a) \rangle$ and maps an A-morphism $h : a \rightarrow a'$ to the homomorphism $F^M(h) = T(h) : \langle a, \mu(a) \rangle \rightarrow \langle a', \mu(a') \rangle$, a *unit* natural transformation $\eta^M : Id_A \Rightarrow F^M \circ U^M$ with component $\eta^M(a) = \eta(a)$ at any object $a \in obj(A)$, and a *counit* natural transformation $\varepsilon^M : U^M \circ F^M \Rightarrow Id_A$ with component $\varepsilon^M(\langle a, \xi \rangle) = \xi$ at any algebra $\langle a, \xi \rangle$. This data forms an *adjunction* $\langle F^M, U^M, \eta^M, \varepsilon^M \rangle : A \rightarrow A^M$.

```
(14) (CNG$function underlying-eilenberg-moore)
     (CNG$signature underlying-eilenberg-moore MND$monad FUNC$functor)
     (forall (?m (MND$monad ?m))
         (and (= (FUNC$source (underlying-eilenberg-moore ?m))
                 (eilenberg-moore ?m))
              (= (FUNC$target (underlying-eilenberg-moore ?m))
                 (MND$underlying-category ?m))
              (= (FUNC$object (underlying-eilenberg-moore ?m))
                 (underlying-object ?m))
              (= (FUNC$morphism (underlying-eilenberg-moore ?m))
                 (underlying-morphism ?m))))

(15) (CNG$function free-eilenberg-moore)
     (CNG$signature free-eilenberg-moore MND$monad FUNC$functor)
     (forall (?m (MND$monad ?m))
         (and (= (FUNC$source (free-eilenberg-moore ?m))
                 (MND$underlying-category ?m))
              (= (FUNC$target (free-eilenberg-moore ?m))
                 (eilenberg-moore ?m))
              (= (SET.FTN$composition
                     (FUNC$object (free-eilenberg-moore ?m))
                     (underlying-object ?m))
                 (FUNC$object (MND$underlying-functor ?m)))
              (= (SET.FTN$composition
                     (FUNC$object (free-eilenberg-moore ?m))
                     (structure-map ?m))
                 (NAT$component (MND$multiplication ?m)))
              (= (SET.FTN$composition
                     (FUNC$morphism (free-eilenberg-moore ?m))
                     (underlying-morphism ?m))
                 (FUNC$morphism (MND$underlying-functor ?m)))))

(16) (CNG$function unit-eilenberg-moore)
     (CNG$signature unit-eilenberg-moore MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$source-functor (unit-eilenberg-moore ?m))
                 (FUNC$identity (MND$underlying-category ?m)))
              (= (NAT$target-functor (unit-eilenberg-moore ?m))
                 (FUNC$composition
                     (free-eilenberg-moore ?m)
                     (underlying-eilenberg-moore ?m)))
              (= (NAT$component (unit-eilenberg-moore ?m))
                 (NAT$component (MND$unit ?m)))))

(17) (CNG$function counit-eilenberg-moore)
     (CNG$signature counit-eilenberg-moore MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$source-functor (counit-eilenberg-moore ?m))
                 (FUNC$composition
                     (underlying-eilenberg-moore ?m)
                     (free-eilenberg-moore ?m)))
              (= (NAT$target-functor (counit-eilenberg-moore ?m))
                 (FUNC$identity (MND$underlying-category ?m)))
              (= (NAT$component (counit-eilenberg-moore ?m))
                 (structure-map ?m))))

(18) (CNG$function adjunction-eilenberg-moore)
     (CNG$signature adjunction-eilenberg-moore MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$underlying-functor (adjunction-eilenberg-moore ?m))
                 (underlying-eilenberg-moore ?m))
              (= (NAT$free-functor (adjunction-eilenberg-moore ?m))
                 (free-eilenberg-moore ?m))
              (= (NAT$unit (adjunction-eilenberg-moore ?m))
                 (unit-eilenberg-moore ?m))
              (= (NAT$counit (adjunction-eilenberg-moore ?m))
                 (counit-eilenberg-moore ?m))))
```

- We can then prove the theorem that the monad generated by the Eilenberg-Moore adjunction is the original monad.

```
     (forall (?m (MND$monad ?m))
         (= (ADJ$adjunction-monad (adjunction-eilenberg-moore ?m)) ?m))
```

For any monad $M = \langle T, \eta, \mu\rangle$ the *Kliesli category* represents the free part of universal algebras.

○ Let $M = \langle T, \eta, \mu\rangle$ be a monad on category A. Restrict attention to the morphisms in A of the form $h : a \rightarrow T(a')$. The Kliesli category A_M has this as a morphism with source object $a \in obj(A_M)$ and target object $a' \in obj(A_M)$. So $obj(A_M) = obj(A)$, $mor(A_M) \subseteq mor(A)$, composition of two morphisms $h : a \rightarrow T(a')$ and $h' : a' \rightarrow T(a'')$ is defined by $h \cdot_{AM} h' = h \cdot_A h'^{\#}$ where $h'^{\#} = T(h') \cdot_A \mu(a'')$ is the *extension* operator, and the identity at an object $a \in obj(A_M)$ is $\eta(a) : a \rightarrow T(a)$. More precisely, as can be seen below, a morphism is a pair algebra $\langle h, a'\rangle$, where $h : a \rightarrow T(a')$ is an A-morphism. Clearly, foundational pullback opspans, cocones and pairing play a key role in the definition of the Kliesli category.

```
(19) (CNG$function kliesli-morphism-opspan)
     (CNG$signature kliesli-morphism-opspan MND$monad SET.LIM.PBK$diagram)
     (forall (?m (MND$monad ?m))
         (and (= (SET.LIM.PBK$class1 (kliesli-morphism-opspan ?m))
                 (CAT$morphism (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$class2 (kliesli-morphism-opspan ?m))
                 (CAT$object (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$opvertex (kliesli-morphism-opspan ?m))
                 (CAT$object (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$opfirst (kliesli-morphism-opspan ?m))
                 (CAT$target (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$opsecond (kliesli-morphism-opspan ?m))
                 (FUNC$object (MND$underlying-functor ?m)))))

(20) (CNG$function kliesli-identity-cone)
     (CNG$signature kliesli-identity-cone MND$monad SET.LIM.PBK$cocone)
     (forall (?m (MND$monad ?m))
         (and (= (SET.LIM.PBK$cone-diagram (kliesli-identity-cone ?m))
                 (Kliesli-morphism-opspan ?m))
              (= (SET.LIM.PBK$vertex (kliesli-identity-cone ?m))
                 (CAT$object (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$first (kliesli-identity-cone ?m))
                 (NAT$component (MND$unit ?m)))
              (= (SET.LIM.PBK$second (kliesli-identity-cone ?m))
                 (SET.FTN$identity (MND$underlying-category ?m)))))

(21) (CNG$function kliesli-composable-opspan)
     (CNG$signature kliesli-composable-opspan MND$monad SET.LIM.PBK$diagram)
     (forall (?m (MND$monad ?m))
         (and (= (SET.LIM.PBK$class1 (kliesli-composable-opspan ?m))
                 (SET.LIM.PBK$pullback (kliesli-morphism-opspan ?m)))
              (= (SET.LIM.PBK$class2 (kliesli-composable-opspan ?m))
                 (SET.LIM.PBK$pullback (kliesli-morphism-opspan ?m)))
              (= (SET.LIM.PBK$opvertex (kliesli-composable-opspan ?m))
                 (CAT$object (MND$underlying-category ?m)))
              (= (SET.LIM.PBK$opfirst (kliesli-composable-opspan ?m))
                 (SET.LIM.PBK$projection2 (kliesli-morphism-opspan ?m)))
              (= (SET.LIM.PBK$opsecond (kliesli-composable-opspan ?m))
                 (SET.FTN$composition
                     (SET.LIM.PBK$projection1 (kliesli-morphism-opspan ?m))
                     (CAT$source (MND$underlying-category ?m))))))

(22) (CNG$function extension)
     (CNG$signature extension MND$monad SET.FTN$function)
     (forall (?m (MND$monad ?m))
         (and (= (SET.FTN$source (extension ?m))
                 (SET.LIM.PBK$pullback (kliesli-morphism-opspan ?m)))
              (= (SET.FTN$target (extension ?m))
                 (CAT$morphism (MND$underlying-category ?m)))
              (= (extension ?m)
                 (SET.FTN$composition
                     (SET.LIM.PBK$pairing
                         (CAT$composable-opspan (MND$underlying-category ?m)))
                         (SET.FTN$composition
                             (SET.LIM.PBK$projection1 (kliesli-morphism-opspan ?m))
                             (FUNC$morphism (MND$underlying-functor ?m)))
                         (SET.FTN$composition
                              (SET.LIM.PBK$projection2 (kliesli-morphism-opspan ?m))
                              (NAT$component (MND$multiplication ?m))))
                      (CAT$composition (MND$underlying-category ?m))))))

(23) (CNG$function kliesli)
     (CNG$signature kliesli MND$monad CAT$category)
     (forall (?m (monad ?m))
         (and (= (CAT$object (kliesli ?m))
                 (CAT$object (MND$underlying-category ?m)))
              (= ((CAT$morphism (kliesli ?m))
                 (SET.LIM.PBK$pullback (kliesli-morphism-opspan ?m)))
              (= (CAT$source (kliesli ?m))
                 (SET.FTN$composition
                     (SET.LIM.PBK$projection1 (kliesli-morphism-opspan ?m))
                     (CAT$source (MND$underlying-category ?m))))
              (= (CAT$target (kliesli ?m))
                 (SET.LIM.PBK$projection2 (kliesli-morphism-opspan ?m)))
              (= (CAT$composable (kliesli ?m))
                 (SET.LIM.PBK$pullback (kliesli-composable-opspan ?m)))
              (= (CAT$composition (kliesli ?m))
                 (SET.FTN$composition
                     (SET.LIM.PBK$pairing
                         (CAT$composable-opspan (MND$underlying-category ?m)))
                         (SET.FTN$composition
                             (SET.LIM.PBK$projection1 (kliesli-composable-opspan ?m))
                             (SET.LIM.PBK$projection1 (kliesli-morphism-opspan ?m)))
                         (SET.FTN$composition
                             (SET.LIM.PBK$projection2 (kliesli-composable-opspan ?m))
                             (SET.FTN$composition
                                 (SET.LIM.PBK$projection1
                                     (kliesli-morphism-opspan ?m)))
                                 (extension ?m))))
                     (CAT$composition (MND$underlying-category ?m))))
              (= (CAT$identity (kliesli ?m))
                 (SET.LIM.PBK$mediator (kliesli-identity-cone ?m)))))
```

- For any monad $M = \langle T, \eta, \mu \rangle$ on category A, there is an *underlying* functor $U_M : A_M \rightarrow A$ that maps an object $a \in obj(A_M)$ to the object $T(a) \in obj(A)$ and maps a morphism $\langle h, a' \rangle : a \rightarrow a'$ in A_M to the extension morphism $h^{\#} : T(a) \rightarrow T(a')$ in A, a *free* functor $F_M : A \rightarrow A_M$ that is the identity on objects and maps a A-morphism $h : a \rightarrow a'$ to the *embedded* morphism $\langle h \cdot \eta(a'), a' \rangle : a \rightarrow a'$ in A_M, a *unit* natural transformation $\eta_M : Id_A \Rightarrow F_M \circ U_M$ with component $\eta_M(a) = \eta(a)$ at any object $a \in obj(A)$, and a *counit* natural transformation $\varepsilon_M : U_M \circ F_M \Rightarrow Id_A$ with component $\varepsilon_M(a) = \langle id_A(T(a)), a \rangle : T(a) \rightarrow a$ at any $a \in obj(A_M)$. This data forms an *adjunction* $\langle F_M, U_M, \eta_M, \varepsilon_M \rangle : A \rightarrow A_M$.

```
(24) (CNG$function underlying-kliesli)
     (CNG$signature underlying-kliesli MND$monad FUNC$functor)
     (forall (?m (MND$monad ?m))
         (and (= (FUNC$source (underlying-kliesli ?m))
                 (kliesli ?m))
              (= (FUNC$target (underlying-kliesli ?m))
                 (MND$underlying-category ?m))
              (= (FUNC$object (underlying-kliesli ?m))
                 (FUNC$object (MND$underlying-functor ?m)))
              (= (FUNC$morphism (underlying-kliesli ?m))
                 (extension ?m))))

(25) (CNG$function embed)
     (CNG$signature embed MND$monad SET.FTN$function)
     (forall (?m (MND$monad ?m))
         (and (= (SET.FTN$source (embed ?m))
                 (CAT$morphism (MND$underlying-category ?m)))
              (= (SET.FTN$target (embed ?m))
                 (CAT$morphism (kliesli ?m)))
              (= (SET.FTN$composition
                     (embed ?m)
                     (SET.LIM.PBK$projection1 (kliesli-morphism-opspan ?m)))
                 (SET.FTN$composition
                     (SET.LIM.PBK$pairing
                         (CAT$composable-opspan (MND$underlying-category ?m)))
```

```
                        (SET.FTN$identity
                            (CAT$morphism (MND$underlying-category ?m)))
                        (SET.FTN$composition
                            (CAT$target (MND$underlying-category ?m))
                            (NAT$component (MND$unit ?m))))
                    (CAT$composition (MND$underlying-category ?m))))
             (= (SET.FTN$composition
                    (embed ?m)
                    (SET.LIM.PBK$projection2 (kliesli-morphism-opspan ?m)))
                (CAT$target (MND$underlying-category ?m)))))
```

```
(26) (CNG$function free-kliesli)
     (CNG$signature free-kliesli MND$monad FUNC$functor)
     (forall (?m (MND$monad ?m))
         (and (= (FUNC$source (free-kliesli ?m))
                 (MND$underlying-category ?m))
              (= (FUNC$target (free-kliesli ?m))
                 (kliesli ?m))
              (= (FUNC$object (free-kliesli ?m))
                 (SET.FTN$identity (CAT$object (MND$underlying-category ?m))))
              (= (FUNC$morphism (free-kliesli ?m))
                 (embed ?m))))
```

```
(27) (CNG$function unit-kliesli)
     (CNG$signature unit-kliesli MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$source-functor (unit-kliesli ?m))
                 (FUNC$identity (MND$underlying-category ?m)))
              (= (NAT$target-functor (unit-kliesli ?m))
                 (FUNC$composition
                     (free-kliesli ?m)
                     (underlying-kliesli ?m)))
              (= (NAT$component (unit-kliesli ?m))
                 (NAT$component (MND$unit ?m)))))
```

```
(28) (CNG$function counit-kliesli)
     (CNG$signature counit-kliesli MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$source-functor (counit-kliesli ?m))
                 (FUNC$composition (underlying-kliesli ?m) (free-kliesli ?m)))
              (= (NAT$target-functor (counit-kliesli ?m))
                 (FUNC$identity (MND$underlying-category ?m)))
              (= (NAT$component (counit-kliesli ?m))
                 (SET.FTN$composition
                     (FUNC$object (MND$underlying-functor ?m))
                     (CAT$identity (MND$underlying-category ?m))))))
```

```
(29) (CNG$function adjunction-kliesli)
     (CNG$signature adjunction-kliesli MND$monad NAT$natural-transformation)
     (forall (?m (MND$monad ?m))
         (and (= (NAT$underlying-functor (adjunction-kliesli ?m))
                 (underlying-kliesli ?m))
              (= (NAT$free-functor (adjunction-kliesli ?m))
                 (free-kliesli ?m))
              (= (NAT$unit (adjunction-kliesli ?m))
                 (unit-kliesli ?m))
              (= (NAT$counit (adjunction-kliesli ?m))
                 (counit-kliesli ?m))))
```

- We can then prove the theorem that the monad generated by the Kliesli adjunction is the original monad.

```
     (forall (?m (MND$monad ?m))
         (= (ADJ$adjunction-monad (adjunction-kliesli ?m)) ?m))
```

The Namespace of Colimits/Limits

COL

Colimits

The Information Flow Framework advances the following proposal: to relate ontologies via morphisms and to compose them using colimits. This section provides the foundation for that proposal. Colimits are important for manipulating and composing ontologies expressed in the object language. The use of colimits advocates a "building blocks approach" for ontology construction. Continuing the metaphor, this approach understands that the mortar between the ontological blocks must be strong and resilient in order to adequately support the ontological building, and requests that methods for composing component ontologies, such as merging, mapping and aligning ontologies, be made very explicit so that they can be analyzed. The Specware system of the Kestrel Institute, which is based on category theory, supports creation and combination of specifications (ontology analogs) using colimits. A compact but detailed discussion of classifications and infomorphisms with applications to this building blocks approach for ontology construction is given in the 6th ISKO paper The Information Flow Foundation for Conceptual Knowledge Organization.

Colimits form a separate namespace in the Category Theory Ontology. Within the colimit namespace are several subnamespaces concerned with binary coproducts, coequalizers and pushouts. To completely express colimits, we need elements from all the basic components of category theory – categories, functors, natural transformations and adjunctions. As such, colimits provide a glimpse of the other parts of the Category Theory Ontology – the other basic components used in colimits are indicated by the namespace prefixes in the KIF formalism.

Finite Colimits

For convenience of representation it is common to use special terminology for a few particular kinds of finite colimits: initial objects, binary coproducts, coequalizers and pushouts. Please note the very common formalisms that encode these. This commonality is expressed in the generalized colimits discussed after words.

Initial Objects

○ Given a category C, an *initial object* $0_C \in obj(C)$ is an object (Figure 1) such that for any object $o \in obj(C)$ there is a *counique morphism* $!_{C,o} : 0_C \rightarrow o$. In the following KIF the term '`initial`' of axiom (1) represents the class of initial objects (possibly empty), and the term '`counique`' of axiom (2) represents the counique function. From the more general theorem that "all colimits for a particular diagram in a category are isomorphic," we can derive the fact that all initial objects are isomorphic. We use a KIF definite description to define the counique function.

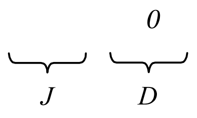


Figure 1: Initial object

```
(1)(CNG$function initial)
   (CNG$signature initial CAT$category SET.class)
   (forall (?c (CAT$category ?c))
       (SET$subclass (initial ?c) (CAT$object ?c))

(2)(CNG$function counique)
   (CNG$signature counique CAT$category SET.FTN$function)
   (forall (?c (CAT$category ?c))
       (= (counique ?c)
          (the (?f (SET.FTN$function ?f))
              (and (= (SET.FTN$source ?f) (CAT$object ?c))
                   (= (SET.FTN$target ?f) (CAT$morphism ?c))
                   (= (SET.FTN$composition ?f (CAT$source ?c))
                      ((SET.TOP$constant (CAT$object ?c) (CAT$object ?c))
                          (initial ?c)))
                   (= (SET.FTN$composition ?f (CAT$target ?c))
                      (SET.FTN$identity (CAT$object ?c)))))))
```

Binary Coproducts

COL.COPRD

A *binary coproduct* in a category *C* (Figure 2) is a finite colimit for a diagram of shape *set2* = • •. Such a diagram (of *C*-objects and *C*-morphisms) is called a *copair*.

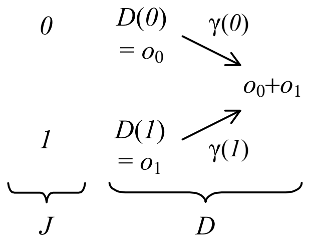


Figure 2: Binary coproduct

- A *copair* (of *C*-objects) is the appropriate base diagram for a binary coproduct in *C*. Each copair consists of a pair of *C*-objects called *object1* and *object2*. Let either '`diagram`' or '`copair`' be the COL.COPRD namespace term that denotes the *Copair* collection. Copairs are determined by their two component C-objects.

```
(1) (CNG$function diagram)
    (CNG$function copair)
    (= copair diagram)
    (CNG$signature diagram CAT$category SET$class)

(2) (CNG$function object1)
    (CNG$signature object1 CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (object1 ?c)) (diagram ?c))
             (= (SET.FTN$target (object1 ?c)) (CAT$object ?c))))

(3) (CNG$function object2)
    (CNG$signature object2 CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (object2 ?c)) (diagram ?c))
             (= (SET.FTN$target (object2 ?c)) (CAT$object ?c))))

    (forall (?c (CAT$category ?c)
             ?p1 ((diagram ?c) ?p1) ?p2 ((diagram ?c) ?p2))
        (=> (and (= ((object1 ?c) ?p1) ((object1 ?c) ?p2))
                 (= ((object2 ?c) ?p1) ((object2 ?c) ?p2)))
            (= ?p1 ?p2)))
```

- Every pair has an opposite.

```
(4) (CNG$function opposite)
    (CNG$signature opposite CAT$category SET.FTN$function)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (opposite ?c)) (diagram ?c))
             (= (SET.FTN$target (opposite ?c)) (diagram ?c))
             (= (SET.FTN$composition (opposite ?c) (object1 ?c))
                (object2 ?c))
             (= (SET.FTN$composition (opposite ?c) (object2 ?c))
                (object1 ?c))))
```

- The opposite of the opposite is the original pair – the following theorem can be proven.

```
    (forall (?c (CAT$category ?c))
        (= (SET.FTN$composition (opposite ?c) (opposite ?c))
           (SET.FTN$identity (diagram ?c))))
```

- *Coproduct cocones* are used to specify and axiomatize binary coproducts in a category *C*. Each coproduct cocone has an underlying coproduct *diagram* (copair), an *opvertex C*-object, and a pair of *C*-morphisms called *opfirst* and *opsecond*, whose common target *C*-object is the opvertex and whose source *C*-objects are the component *C*-objects in the underlying diagram. The opfirst and opsecond morphisms are the components of a natural transformation. A coproduct cocone is the very special case of a general colimit cocone over a coproduct diagram. Let '`cocone`' be the COL.PSH namespace term that denotes the *Coproduct Cocone* collection.

```
(5) (CNG$function cocone)
    (CNG$signature cocone CAT$category SET$class)

(6) (CNG$function cocone-diagram)
```

```
    (CNG$signature cocone-diagram CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (cocone-diagram ?c)) (cocone ?c))
             (= (SET.FTN$target (cocone-diagram ?c)) (diagram ?c))))

(7) (CNG$function opvertex)
    (CNG$signature opvertex CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (opvertex ?c)) (cocone ?c))
             (= (SET.FTN$target (opvertex ?c)) (CAT$object ?c))))

(8) (CNG$function opfirst)
    (CNG$signature opfirst CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (opfirst ?c)) (cocone ?c))
             (= (SET.FTN$target (opfirst ?c)) (CAT$morphism ?c))
             (= (SET.FTN$composition (opfirst ?c) (CAT$source ?c))
                (SET.FTN$composition (cocone-diagram ?c) (object1 ?c)))
             (= (SET.FTN$composition (opfirst ?c) (CAT$target ?c))
                (opvertex ?c))))

(9) (CNG$function opsecond)
    (CNG$signature opsecond CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (opsecond ?c)) (cocone ?c))
             (= (SET.FTN$target (opsecond ?c)) (CAT$morphism ?c))
             (= (SET.FTN$composition (opsecond ?c) (CAT$source ?c))
                (SET.FTN$composition (cocone-diagram ?c) (object2 ?c)))
             (= (SET.FTN$composition (opsecond ?c) (CAT$target ?c))
                (opvertex ?c))))
```

- There is a function '`colimiting-cocone`' that maps a copair in a category *C* to its collection of coproduct cocones (this may be empty). The opvertex *C*-object of a colimiting cocone (Figure 2) is given by the function '`binary-coproduct`'. It comes equipped with two injection *C*-morphisms '`injection1`' and '`injection2`' given by the opfirst and opsecond of the colimiting cocone.

```
(10) (KIF$function colimiting-cocone)
     (KIF$signature colimiting-cocone CAT$category CNGfunction)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (colimiting-cocone ?c) (diagram ?c) SET$class)
              (forall (?r ((diagram ?c) ?r))
                  (SET$subclass
                      ((colimiting-cocone ?c) ?r)
                      ((SET.FTN$fiber (cocone-diagram ?c)) ?r)))))

(11) (KIF$function binary-coproduct)
     (KIF$signature binary-coproduct CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (binary-coproduct ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((binary-coproduct ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((binary-coproduct ?c) ?r))
                          (CAT$object ?c))
                       (= ((binary-coproduct ?c) ?r)
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (opvertex ?c)))))))

(12) (KIF$function injection1)
     (KIF$signature injection1 CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (injection1 ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((injection1 ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((injection1 ?c) ?r))
                          (CAT$morphism ?c))
                       (= (SET.FTN$composition ((injection1 ?c) ?r) (CAT$source ?c))
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (SET.FTN$composition (cocone-diagram ?c) (object1 ?c))))
                       (= (SET.FTN$composition ((injection1 ?c) ?r) (CAT$target ?c))
                          ((binary-coproduct ?c) ?r))
                       (= ((injection1 ?c) ?r)
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (opfirst ?c)))))))

(13) (KIF$function injection2)
     (KIF$signature injection2 CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (injection2 ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((injection2 ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((injection2 ?c) ?r))
                          (CAT$morphism ?c))
                       (= (SET.FTN$composition ((injection2 ?c) ?r) (CAT$source ?c))
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (SET.FTN$composition (cocone-diagram ?c) (object2 ?c))))
                       (= (SET.FTN$composition ((injection2 ?c) ?r) (CAT$target ?c))
                          ((binary-coproduct ?c) ?r))
                       (= ((injection2 ?c) ?r)
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (opsecond ?c)))))))
```

o For any coproduct diagram in a category *C* and any colimiting-cocone with that diagram as its base, there is a function that maps any cocone with the same base diagram to a unique *comediator C*-morphism whose source is the binary coproduct and whose target is opvertex of the cocone. This is the unique morphism that commutes with opfirst and opsecond morphisms. We use a KIF definite description abbreviation to define this. Existence and uniqueness represents the universality of the coproduct operator. A derived theorem states that all binary coproducts are isomorphic in *C*.

```
(14) (KIF$function comediator)
     (KIF$signature comediator CAT$category KIF$function)
     (forall (?c (CAT$category ?c))
         (and (KIF$signature (comediator ?c) (diagram ?c) CNG$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (CNG$signature ((comediator ?c) ?r)
                           ((colimiting-cocone ?c) ?r) SET.FTN$function)
                       (forall (?s (((colimiting-cocone ?c) ?r) ?s))
                           (= (((comediator ?c) ?r) ?s)
                              (the (?f (SET.FTN$function ?f))
                                  (and (= (SET.FTN$source ?f)
                                          ((fiber (cocone-diagram ?c)) ?r))
                                       (= (SET.FTN$target ?f)
                                          (CAT$morphism ?c))
                                       (= (SET.FTN$composition ?f (CAT$source ?c))
                                          ((SET.TOP$diagonal
                                              ((fiber (cocone-diagram ?c)) ?r)
                                              (CAT$object ?c))
                                            (((binary-coproduct ?c) ?r) ?s)))
                                       (= (SET.FTN$composition ?f (CAT$target ?c))
                                          (SET.FTN$composition
                                              (SET.FTN$inclusion
                                                  ((fiber (cocone-diagram ?c)) ?r) (cocone ?c))
                                              (opvertex ?c))))))))))
```

Coequalizers

COL.COEQU

. . .

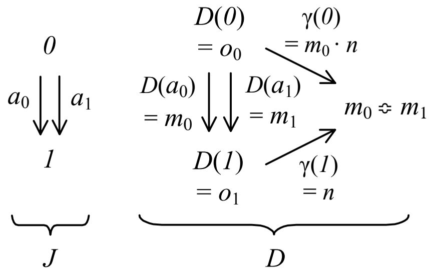


Figure 3: Coequalizer

Pushouts

COL.PSH

Given a category *C*, a *pushout* in *C* (Figure 4) is a finite colimit for a diagram of shape *J* = $\bullet \leftarrow \bullet \rightarrow \bullet$. Such a diagram (of *C*-objects and *C*-morphisms) is called an *span*.

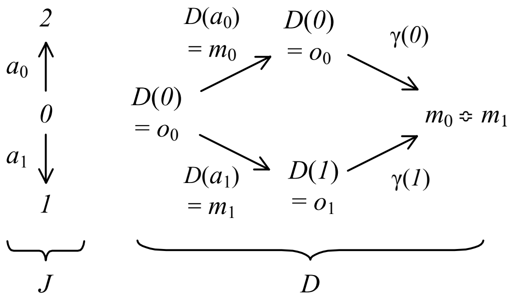


Figure 4: Pushout

- A *span* is the appropriate base diagram for a pushout. Each span consists of a pair of *C*-morphisms called *first* and *second*. These are required to have a common source *C*-object called the *vertex*. Let '`span`' be the COL.PSH namespace term that denotes the *Span* collection. This is synonymous with the phrase "pushout diagram". Spans are determined by their pair of component functions.

```
(1) (CNG$function diagram)
    (CNG$function span)
    (= span diagram)
    (CNG$signature diagram CAT$category SET$class)

(2) (CNG$function object1)
    (CNG$signature object1 CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (object1 ?c)) (diagram ?c))
             (= (SET.FTN$target (object1 ?c)) (CAT$object ?c))))

(3) (CNG$function object2)
    (CNG$signature object2 CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (object2 ?c)) (diagram ?c))
             (= (SET.FTN$target (object2 ?c)) (CAT$object ?c))))

(4) (CNG$function vertex)
    (CNG$signature vertex CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (vertex ?c)) (diagram ?c))
             (= (SET.FTN$target (vertex ?c)) (CAT$object ?c))))

(5) (CNG$function first)
    (CNG$signature first CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (first ?c)) (diagram ?c))
             (= (SET.FTN$target (first ?c)) (CAT$morphism ?c))
             (= (SET.FTN$composition (first ?c) (CAT$source ?c))
                (vertex ?c))
             (= (SET.FTN$composition (first ?c) (CAT$target ?c))
                (object1 ?c))))
```

```
(6) (CNG$function second)
    (CNG$signature second CAT$category SET.FTNfunction)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (second ?c)) (diagram ?c))
             (= (SET.FTN$target (second ?c)) (CAT$morphism ?c))
             (= (SET.FTN$composition (second ?c) (CAT$source ?c))
                (vertex ?c))
             (= (SET.FTN$composition (second ?c) (CAT$target ?c))
                (object2 ?c))))

    (forall (?c (CAT$category ?c)
             ?r1 ((diagram ?c) ?r1) ?r2 ((diagram ?c) ?r2))
        (=> (and (= ((first ?c) ?r1) ((first ?c) ?r2))
                 (= ((second ?c) ?r1) ((second ?c) ?r2)))
            (= ?r1 ?r2)))
```

o The *copair* of target C-objects (postfixing discrete diagram) of any span (pushout diagram) in a category *C* is named.

```
(7) (CNG$function copair)
    (CNG$signature copair CAT$category SET.FTN$function)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (copair ?c)) (diagram ?c))
             (= (SET.FTN$target (copair ?c)) (COL.COPRD$diagram))))
             (= (SET.FTN$composition (copair ?c) (COL.COPRD$object1 ?c))
                (object1 ?c))
             (= (SET.FTN$composition (copair ?c) (COL.COPRD$object2 ?c))
                (object2 ?c))))
```

o Every span in *C* has an opposite.

```
(8) (CNG$function opposite)
    (CNG$signature opposite CAT$category SET.FTN$function)
    (forall (?c (CAT$category ?c))
        (and (= (SET.FTN$source (opposite ?c)) (diagram ?c))
             (= (SET.FTN$target (opposite ?c)) (diagram ?c))
             (= (SET.FTN$composition (opposite ?c) (object1 ?c))
                (object2 ?c))
             (= (SET.FTN$composition (opposite ?c) (object2 ?c))
                (object1 ?c))
             (= (SET.FTN$composition (opposite ?c) (vertex ?c))
                (vertex ?c))
             (= (SET.FTN$composition (opposite ?c) (first ?c))
                (second ?c))
             (= (SET.FTN$composition (opposite ?c) (second ?c))
                (first ?c))))
```

o The opposite of the opposite is the original span – the following theorem can be proven.

```
    (forall (?c (CAT$category ?c))
        (= (SET.FTN$composition (opposite ?c) (opposite ?c))
           (SET.FTN$identity (diagram ?c))))
```

o *Pushout cocones* are used to specify and axiomatize pushouts in a category *C*. Each pushout cocone has an underlying pushout *diagram*, an *opvertex C*-object, and a pair of *C*-morphisms called *opfirst* and *opsecond*, whose common target *C*-object is the opvertex and whose source *C*-objects are the target *C*-objects of the *C*-morphisms in the underlying diagram. The opfirst and opsecond morphisms form a commutative diagram with the span, implicitly making the pushout cocone a natural transformation. A pushout cocone is the very special case of a general colimit cocone over a pushout diagram. Let '`cocone`' be the COL.PSH namespace term that denotes the *Pushout Cocone* collection.

```
(9) (CNG$function cocone)
    (CNG$signature cocone CAT$category SET$class)

(10) (CNG$function cocone-diagram)
     (CNG$signature cocone-diagram CAT$category SET.FTNfunction)
     (forall (?c (CAT$category ?c))
         (and (= (SET.FTN$source (cocone-diagram ?c)) (cocone ?c))
              (= (SET.FTN$target (cocone-diagram ?c)) (diagram ?c))))
```

```
(11) (CNG$function opvertex)
     (CNG$signature opvertex CAT$category SET.FTNfunction)
     (forall (?c (CAT$category ?c))
         (and (= (SET.FTN$source (opvertex ?c)) (cocone ?c))
              (= (SET.FTN$target (opvertex ?c)) (CAT$object ?c))))

(12) (CNG$function opfirst)
     (CNG$signature opfirst CAT$category SET.FTNfunction)
     (forall (?c (CAT$category ?c))
         (and (= (SET.FTN$source (opfirst ?c)) (cocone ?c))
              (= (SET.FTN$target (opfirst ?c)) (CAT$morphism ?c))
              (= (SET.FTN$composition (opfirst ?c) (CAT$source ?c))
                 (SET.FTN$composition (cocone-diagram ?c) (object1 ?c)))
              (= (SET.FTN$composition (opfirst ?c) (CAT$target ?c))
                 (opvertex ?c))))

(13) (CNG$function opsecond)
     (CNG$signature opsecond CAT$category SET.FTNfunction)
     (forall (?c (CAT$category ?c))
         (and (= (SET.FTN$source (opsecond ?c)) (cocone ?c))
              (= (SET.FTN$target (opsecond ?c)) (CAT$morphism ?c))
              (= (SET.FTN$composition (opsecond ?c) (CAT$source ?c))
                 (SET.FTN$composition (cocone-diagram ?c) (object2 ?c)))
              (= (SET.FTN$composition (opsecond ?c) (CAT$target ?c))
                 (opvertex ?c))))
```

o There is a function ‘`colimiting-cocone`’ that maps a span in a category *C* to its collection of pushout cocones (this may be empty). The opvertex *C*-object of a colimiting cocone (Figure 4) is given by the function ‘`pushout`’. It comes equipped with two `injection` *C*-morphisms ‘`injection1`’ and ‘`injection2`’ given by the opfirst and opsecond of the colimiting cocone.

```
(14) (KIF$function colimiting-cocone)
     (KIF$signature colimiting-cocone CAT$category CNGfunction)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (colimiting-cocone ?c) (diagram ?c) SET$class)
              (forall (?r ((diagram ?c) ?r))
                  (SET$subclass
                      ((colimiting-cocone ?c) ?r)
                      ((SET.FTN$fiber (cocone-diagram ?c)) ?r)))))

(15) (KIF$function pushout)
     (KIF$signature pushout CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (pushout ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((pushout ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((pushout ?c) ?r))
                          (CAT$object ?c))
                       (= ((pushout ?c) ?r)
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (opvertex ?c)))))))

(16) (KIF$function injection1)
     (KIF$signature injection1 CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (injection1 ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((injection1 ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((injection1 ?c) ?r))
                          (CAT$morphism ?c))
                       (= (SET.FTN$composition ((injection1 ?c) ?r) (CAT$source ?c))
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (SET.FTN$composition (cocone-diagram ?c) (object1 ?c))))
                          (= (SET.FTN$composition ((injection1 ?c) ?r) (CAT$target ?c))
                             ((pushout ?c) ?r))
                          (= ((injection1 ?c) ?r)
                             (SET.FTN$composition
                                 (SET.FTN$inclusion
                                     ((colimiting-cocone ?c) ?r) (cocone ?c))
                                 (opfirst ?c)))))))

(17) (KIF$function injection2)
     (KIF$signature injection2 CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (injection2 ?c) (diagram ?c) SET.FTN$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (= (SET.FTN$source ((injection2 ?c) ?r))
                          ((colimiting-cocone ?c) ?r))
                       (= (SET.FTN$target ((injection2 ?c) ?r))
                          (CAT$morphism ?c))
                       (= (SET.FTN$composition ((injection2 ?c) ?r) (CAT$source ?c))
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (SET.FTN$composition (cocone-diagram ?c) (object2 ?c))))
                       (= (SET.FTN$composition ((injection2 ?c) ?r) (CAT$target ?c))
                          ((pushout ?c) ?r))
                       (= ((injection2 ?c) ?r)
                          (SET.FTN$composition
                              (SET.FTN$inclusion
                                  ((colimiting-cocone ?c) ?r) (cocone ?c))
                              (opsecond ?c)))))))
```

- For any pushout diagram in a category *C* and any colimiting-cocone with that diagram as its base, there is a function that maps any cocone with the same base diagram to a unique *comediator C*-morphism whose source is the pushout and whose target is opvertex of the cocone. This is the unique morphism that commutes with opfirst and opsecond morphisms. We use a KIF definite description abbreviation to define this. Existence and uniqueness represents the universality of the pullback operator. A derived theorem states that all pushouts are isomorphic in *C*.

```
(18) (KIF$function comediator)
     (KIF$signature comediator CAT$category KIF$function)
     (forall (?c (CAT$category ?c))
         (and (KIF$signature (comediator ?c) (diagram ?c) CNG$function)
              (forall (?r ((diagram ?c) ?r))
                  (and (CNG$signature ((comediator ?c) ?r)
                           ((colimiting-cocone ?c) ?r) SET.FTN$function)
                       (forall (?s (((colimiting-cocone ?c) ?r) ?s))
                           (= (((comediator ?c) ?r) ?s)
                              (the (?f (SET.FTN$function ?f))
                                  (and (= (SET.FTN$source ?f)
                                          ((fiber (cocone-diagram ?c)) ?r))
                                       (= (SET.FTN$target ?f)
                                          (CAT$morphism ?c))
                                       (= (SET.FTN$composition ?f (CAT$source ?c))
                                          ((SET.TOP$diagonal
                                              ((fiber (cocone-diagram ?c)) ?r)
                                              (CAT$object ?c))
                                            (((pushout ?c) ?r) ?s)))
                                       (= (SET.FTN$composition ?f (CAT$target ?c))
                                          (SET.FTN$composition
                                              (SET.FTN$inclusion
                                                  ((fiber (cocone-diagram ?c)) ?r) (cocone ?c))
                                              (opvertex ?c))))))))))
```

General Colimits

- Given a category C, which serves as an environment within which colimits are to be constructed, a *diagram* D in C is a functor from some shape or indexing category J to the base category C. In the KIF formalism below the *shape* of a diagram in C is defined in terms of the '`FUNC$source`' predicate.

```
(1) (CNG$function diagram)
    (CNG$signature diagram CAT$category CNG$conglomeration)
    (forall (?c (CAT$category ?c))
        (and (CNG$subconglomeration (diagram ?c) FUNC$functor)
             (forall (?d (FUNC$functor ?d))
                 (<=> ((diagram ?c) ?d)
                      (= (FUNC$target ?d) ?c)))))

(2) (KIF$function shape)
    (KIF$signature shape CAT$category CNG$function)
    (forall (?c (CAT$category ?c))
        (and (CNG$signature (shape ?c) (diagram ?c) CAT$category)
             (forall (?d ((diagram ?c) ?d))
                 (= ((shape ?c) ?d) (FUNC$source ?d)))))
```

- Given a category C a *cocone* $\tau : D \Rightarrow \Delta_{J,\,C}(o) : J \rightarrow C$ (Figure 5) is a natural transformation from a diagram D in the category C of some shape $J = src(D)$ to the constant functor $\Delta_{J,\,C}(o)$ for some object $o \in obj(C)$. The source functor D is called the *base* of the cocone τ. The object $o \in obj(C)$ is called the *vertex* of the cocone τ. A cocone is analogous to the upper bound of a subset of a partial order – the order is analogous to the category C, the chosen subset is analogous to the base diagram D, and the upper bound element is analogous to the vertex C-object o. We use a KIF definite description to define the vertex. The vertex function restricts cocones to be natural transformations whose target is a constant function.

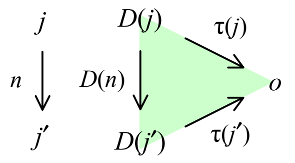

$$\begin{tikzcd} j \arrow[d, "n"'] & D(j) \arrow[d, "D(n)"'] \arrow[dr, "\tau(j)"] & \\ j' & D(j') \arrow[r, "\tau(j')"'] & o \end{tikzcd}$$

Figure 5: Cocone

```
(3) (KIF$function cocone)
    (KIF$signature cocone CAT$category CNG$conglomerate)
    (forall (?c (CAT$category ?c))
        (and (CNG$subconglomerate (cocone ?c) NAT$natural-transformation)
             (forall (?tau ((cocone ?c) ?tau))
                 (= (NAT$target-category ?tau) ?c))))

(4)  (KIF$function cocone-diagram)
     (KIF$function base)
     (= base cocone-diagram)
     (KIF$signature cocone-diagram CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (cocone-diagram ?c) (cocone ?c) (diagram ?c))
              (forall (?tau ((cocone ?c) ?tau))
                  (= ((cocone-diagram ?c) ?tau)
                     (NAT$source-functor ?tau)))))

(5)  (KIF$function opvertex)
     (KIF$signature opvertex CAT$category CNG$function)
     (forall (?c (CAT$category ?c))
         (and (CNG$signature (opvertex ?c) (cocone ?c) (CAT$object ?c))
              (forall (?tau ((cocone ?c) ?tau))
                  (= ((opvertex ?c) ?tau)
                     (the (?o ((CAT$object ?c) ?o))
                         (= (NAT$target-functor ?tau)
                            ((FUNC$diagonal (NAT$source-category ?tau) ?c) ?o)))))))
```

- Colimits are the vertices of special cocones. Given a category C a *colimiting-cocone* in C (Figure 6) is a universal cocone – it consists of a cocone from a diagram D in C of some shape $J = src(D)$ to a `opvertex` $\bar{o} \in obj(C)$

  $\gamma : D \Rightarrow \Delta_{C,\,J}(\bar{o}) : J \rightarrow C$

  that is *universal*: for any other cocone

  $\tau : D \Rightarrow \Delta_{C,\,J}(o) : J \rightarrow C$

with the same base diagram, there is a unique morphism $m : \bar{o} \rightarrow o$ with $\gamma(j) \cdot m = \tau(j)$ for any indexing object $j \in Obj(J)$, or equivalently as natural transformations, with $\gamma \bullet \Delta_{J,C}(m) = \tau$. The collection of colimiting cocones may be empty – it is empty if no colimits for the diagram *D* exist in the category *C*. If it is nonempty for any diagram of shape *J*, then C is said to have *J-colimits*. A category C is *cocomplete*, if it has *J*-colimits for any shape *J*. The `opvertex` $\bar{o}$ of the colimiting-cocone γ is called a *colimit*. Let $colim_C(D)$ denote the function that takes a colimiting cocone to its `opvertex`, an object of *C*. This will be the empty function, if no colimits exist for diagram D. In the same way that a cocone is analogous to the upper bound, a colimit is analogous to a least upper bound (or supremum) of a subset of a partial order.

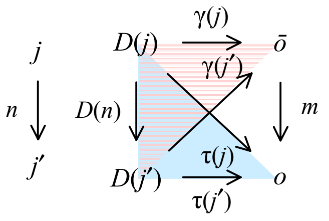

$$\begin{tikzcd} j \arrow[d, "n"'] & D(j) \arrow[r, "\gamma(j)"] \arrow[d, "D(n)"'] \arrow[dr] & \bar{o} \arrow[d, "m"] \\ j' & D(j') \arrow[r, "\tau(j')"'] \arrow[ur, "\gamma(j')"] & o \end{tikzcd}$$

Figure 6: Colimiting Cocone

Axioms (6–7) formalize colimits. Axiom (6) defines a unary KIF function ‘`(colimiting-cocone ?c)`’ that maps a diagram to its colimit conglomerate. The `opvertex` of a colimiting cocone is a particular colimit represented in axiom (7) by the CNG function ‘`((colimiting-cocone ?c) ?d)`’.

```
(6) (KIF$function colimiting-cocone)
    (KIF$signature colimiting-cocone CAT$category KIF$function)
    (forall (?c (CAT$category ?c))
        (and (KIF$signature (colimiting-cocone ?c) (diagram ?c) CNG$conglomerate)
             (forall (?d ((diagram ?c) ?d))
                 (and (CNG$subconglomerate
                          ((colimiting-cocone ?c) ?d)
                          (cocone ?c))
                      (forall (?g (((colimiting-cocone ?c) ?d) ?g))
                          (=> (((colimiting-cocone ?c) ?d) ?g)
                              (= ?d ((cocone-diagram ?c) ?g)))))))

(7)  (KIF$function colimit)
     (KIF$signature colimit CAT$category KIF$function)
     (forall (?c (CAT$category ?c))
         (and (KIF$signature (colimit ?c) (diagram ?c) CNG$function)
              (forall (?d ((diagram ?c) ?d))
                  (and (CNG$signature ((colimit ?c) ?d)
                           ((colimiting-cocone ?c) ?d) (CAT$object ?c))
                       (forall (?g (((colimiting-cocone ?c) ?d) ?g))
                           (= (((colimit ?c) ?d) ?g)
                              ((opvertex ?c) ?g)
```

For any diagram *D* and any colimiting-cocone γ with base *D*, let $m = comediator_{C,D}(\gamma)$ (Figure 6) denote the function that takes any cocone τ of base *D* and returns its unique comediating *C*-morphism whose source is the colimit and whose target is the opvertex of the cocone.

Axiom (8) formalizes comediators. There is a comediator function ‘`(((comediator ?c) ?d) ?g)`’ from the colimit of the diagram to the opvertex of a cocone over a diagram. This is the unique function that commutes with colimiting cocone and given cocone. We use a KIF definite description abbreviation to define this. Existence and uniqueness represents the universality of the colimit.

```
(8) (KIF$function comediator)
    (KIF$signature colimit (CAT$category KIF$function)
    (forall (?c (CAT$Category ?c))
      (and (KIF$signature (comediator ?c) (diagram ?c) KIF$function)))
           (forall (?d ((diagram ?c) ?d))
             (and (KIF$signature ((comediator ?c) ?d)
                    ((colimiting-cocone ?c) ?d) KIF$function)
                  (forall (?g (((colimiting-cocone ?c) ?d) ?g))
                    (and (KIF$signature (((comediator ?c) ?d) ?g)
                           (cocone ?c) (CAT$morphism ?c))
                         (forall (?t ((cocone ?c) ?t))
                           (= ((((comediator ?c) ?d) ?g) ?t)
                              (the (?m (CAT$morphism ?c) ?m))
                                (and (= ((CAT$source ?c) ?m) (((colimit ?c) ?d) ?g))
                                     (= ((CAT$target ?c) ?m) ((opvertex ?c) ?t))
                                     (= (NAT$vertical-composition
                                          ?g ((NAT$diagonal ((shape ?c) ?d) ?c) ?m))
                                        ?t)))))))))))
```

- If the colimiting cocone conglomerate is nonempty for any diagram of shape *J*, then C is said to have *J-colimits* and to be *J-cocomplete*. A category *C* is *small-cocomplete*, if it has colimits for any small diagram of *C*; that is, when it is *J*-cocomplete for all small shape categories J. Axiom (9) formalizes cocompleteness in terms of the colimiting cocone conglomerate '`((colimiting-cocone ?c) ?d)`'.

```
(9) (KIF$function cocomplete)
    (KIF$signature cocomplete CAT$category CNG$conglomerate)
    (forall (?j (CAT$category ?j))
        (and (CNG$subconglomerate (cocomplete ?j) CAT$category)
             (forall (?c (CAT$category ?c))
                 (<=> ((cocomplete ?j) ?c)
                      (forall (?d ((diagram ?c) ?d))
                          (=> (= ((shape ?c) ?d) ?j)
                              (exists (?g (((colimiting-cocone ?c) ?d) ?g))))))))

(10) (CNG$conglomerate small-cocomplete)
     (CNG$subconglomerate small-cocomplete CAT$category)
     (forall (?c (CAT$category ?c))
         (<=> (small-cocomplete ?c)
              (forall (?j (CAT$category ?j))
                  ((cocomplete ?j) ?c))))
```

- It is a standard theorem that given a category *C* and a diagram *D* in the category *C*, any two colimits are isomorphic. The following KIF expresses this in an external namespace.

```
     (forall (?c (CAT$category ?c) ?d ((diagram ?c) ?d)
              ?g1 (((colimiting-cocone ?c) ?d) ?g1)
              ?g2 (((colimiting-cocone ?c) ?d) ?g2))
         ((CAT$isomorphic ?c) (((colimit ?c) ?d) ?g1) (((colimit ?c) ?d) ?g2)))
```

Examples

The general KIF formulation for colimits can be related to several special kinds of finite colimits: initial objects, binary coproducts, coequalizers and pushouts.

- Suppose an initial object 0_C exists in a category *C*. Here we are interested in the initial (empty) shape category $J = \boldsymbol{0}$. If *D* is the empty diagram in the category *C* of shape $\boldsymbol{0}$, then the colimit object is the initial object 0_C and the colimiting cocone is the empty natural transformation (vertical identity at *D*) with target category *C*. Also, the mediator function to any object (cocone opvertex) is the *counique* function. (The initial object in Set is the empty set. The initial object in Classification is the classification with one instance, but no types.)

- Suppose binary coproducts exist in a category *C*. Consider the binary coproduct of any two objects o_0, $o_1 \in obj(C)$. Here we are interested in the shape category $J = set2$ of two discrete nodes *0* and *1*. If *D* is the diagram in the category *C* of shape *set2*, whose object function maps *0* and *1* to the *C*-objects o_0 and o_1, respectively, then the colimit object is the binary coproduct $o_0 + o_1$ and the colimiting cocone has as components the coproduct injections morphisms $i_0 : o_0 \rightarrow o_0+o_1$ and $i_1 : o_1 \rightarrow o_0+o_1$. This statement is represented as the following KIF theorem.

```
 (forall (?c (CAT$category ?c))
     (=> ((cocomplete set2) ?c)
       (forall ?d ((diagram ?c) ?d))
           (=> (= ((shape ?c) ?d) set2)
               (exists (?p (COL.COPRD$diagram ?p)
                        ?g (((colimiting-cocone ?c) ?d) ?g))
                   (and (= ((FUNC$object ?d) set2#0)
                           ((COL.COPRD$object1 ?c) ?p))
                        (= ((FUNC$object ?d) set2#1)
                           ((COL.COPRD$object2 ?c) ?p))
                        (= (((colimit ?c) ?d) ?g)
                           ((COL.COPRD$binary-coproduct ?c) ?p))
                        (= ((NAT$component ?g) set2#0)
                           ((COL.COPRD$injection1 ?c) ?p))
                        (= ((NAT$component ?g) set2#0)
                           ((COL.COPRD$injection2 ?c) ?p))))))))
```

- Suppose coequalizers exist in a category C. Consider the coequalizer of any two parallel C-morphisms $m_0, m_1 : o_0 \rightarrow o_1$. Here we are interested in the shape category $J = parpair$ of two parallel edges. If D is the diagram in the category C of shape *parpair*, whose object function maps *0* and *1* to the C-objects o_0 and o_1, respectively, and whose morphism function maps a_0 and a_1 to the C-morphisms m_0 and m_1, respectively, then the colimit object is the coequalizer $m_0 \Leftrightarrow m_1$ and the colimiting cocone has as its *1*-component the canonical morphism $n : o_1 \rightarrow m_0 \Leftrightarrow m_1$ and has as *0*-component is the C-composition $m_0 \cdot n = m_1 \cdot n : o_0 \rightarrow m_0 \Leftrightarrow m_1$. This statement is represented as the following KIF theorem.

```
(forall (?c (CAT$category ?c))
    (=> ((cocomplete parpair) ?c)
        (forall ?d ((diagram ?c) ?d))
            (=> (= ((shape ?c) ?d) parpair)
                (exists (?pp (COL.COEQU$diagram ?pp)
                         ?g (((colimiting-cocone ?c) ?d) ?g))
                    (and (= ((FUNC$object ?d) parpair#0)
                            ((COL.COEQU$object1 ?c) ?pp))
                         (= ((FUNC$object ?d) parpair#1)
                            ((COL.COEQU$object2 ?c) ?pp))
                         (= ((FUNC$morphism ?d) parpair#a0)
                            ((COL.COEQU$morphism1 ?c) ?pp))
                         (= ((FUNC$morphism ?d) parpair#a1)
                            ((COL.COEQU$morphism2 ?c) ?pp))
                         (= (((colimit ?c) ?d) ?g)
                            ((COL.COEQU$coequalizer ?c) ?pp))
                         (= ((NAT$component ?g) parpair#0)
                            ((CAT$composition ?c)
                                [((COL.COEQU$morphism1 ?c) ?pp)
                                 ((COL.COEQU$canon ?c) ?pp)]))
                         (= ((NAT$component ?g) set2#0)
                            ((COL.COEQU$canon ?c) ?pp))))))))
```

- Suppose pushouts exist in a category C. Consider the pushout of any span of C-morphisms $m_1 : o_0 \rightarrow o_1$ and $m_2 : o_0 \rightarrow o_2$. Here we are interested in the shape category $J = span$. If D is the diagram in the category C of shape *span*, whose object function maps *0, 1* and *2* to the C-objects o_0, o_1 and o_2, respectively, and whose morphism function maps a_1 and a_2 to the C-morphisms m_1 and m_2, respectively, then the colimit object is the pushout $o_1 +_o o_2$ and the colimiting cocone has as components the pushout injection morphisms $i_1 : o_1 \rightarrow o_1 \times_o o_2$ and $i_2 : o_2 \rightarrow o_1 \times_o o_2$. The analogous theorem can be proven.

The Namespace of Large Kan Extensions

The Namespace of Large Topoi