

The IFF Namespace of Hypergraphs

THE IFF NAMESPACE OF HYPERGRAPHS	1
THE CONTEXT OF HYPERGRAPHS	4
<i>The Contexts of Hypergraph Fibers</i>	11
HYPERGRAPHS.....	18
<i>Case and Spangraphs</i>	20
<i>Hypergraph Fibers</i>	23
HYPERGRAPH MORPHISMS	27
<i>Case and Spangraph Morphisms</i>	31
<i>Hypergraph Morphism Fibers</i>	34
COLIMITS FOR HYPERGRAPHS	39
<i>Initial Hypergraph</i>	39
<i>Binary Coproducts</i>	41
<i>Endorelations and Quotients</i>	47
<i>Coequalizers</i>	52
<i>Pushouts</i>	59
LIMITS FOR HYPERGRAPHS	67
<i>Final (Terminal) Hypergraph</i>	68
<i>Binary Products</i>	70

This namespace axiomatizes **Hypergraph** the category of hypergraphs and hypergraph morphisms. This is one of the basic categories of the IFF Ontology (meta) Ontology (IFF-OO).

Things still to do:

- The tables of terminology need to be updated.
- Move the diagram terminology from the colimit/limit namespaces to a namespace of its own.
- Some changes need to be made to colimits.
- The work on limits has just started. This effort has focused attention on the rather involved limit constructions in **Set[^]** the set semipair category and **Set** the basic category of sets and set functions.

The IFF Namespace of Hypergraphs

Robert E. Kent

Page 2

May 8, 2003

Table 1 lists the basic terminology in the namespace of hypergraphs.

Table 1: Basic Terminology for the Namespace of Hypergraphs

	Category	Functor	Natural Transformation	Adjunction
hgph	hypergraph	reference case edge node name spangraph	signature arity indication comediation projection eta	reflection
hgph.fbr	(hypergraph ?x)	(inclusion ?x) (reference ?x) (case ?x) (edge ?x) (node ?x) (name ?x) (spangraph ?x) (direct-image ?x) (inverse-image ?x)	(signature ?x) (arity ?x) (indication ?x) (comediation ?x) (projection ?x) (eta ?x)	(reflection ?x)

	Class/Collection	Function	Other
hgph.obj	object = hypergraph	reference signature edge-arity reference-arity edge node name tuple	
		arity case injection indication comediation projection arity-morphism case-signature spangraph	
hgph.fbr.obj	(object ?x) = (hypergraph ?x)	(inclusion ?x) (reference ?x) (signature ?x) (edge-arity ?x) (edge ?x) (node ?x) (name ?x)	
		(spangraph ?x) (direct-image ?x) (inverse-image ?x) (direct-connection ?x) (inverse-connection ?x)	
hgph.mor	morphism	source target reference signature edge-arity reference-arity edge node name composition identity eta tuple	composable- opspan composable
		arity-morphism case indication projection comediation spangraph	
hgph.fbr.mor	(morphism ?h) (isomorphism ?h)	(inclusion ?h) (source ?h) (target ?h) (reference ?h) (signature ?h) (edge-arity ?h) (edge ?h) (node ?h) (name ?h) (composition [?h1 ?h2]) (identity ?x)	composable- opspan composable
		(spangraph ?h) (direct-image ?h) (inverse-image ?h)	

The IFF Namespace of Hypergraphs

Robert E. Kent

Page 3

May 8, 2003

Table 2 lists the colimit terminology in the namespace of hypergraphs.

Table 2: Colimit Terminology for the Namespace of Hypergraphs

	Class/Collection	Function	Other
hgph.col			(initial ?x) (counique ?x)
hgph.col.copr2	(diagram ?x) = (pair ?x)	(hypergraph1 ?x) (hypergraph2 ?x) (edge-diagram ?x) (node-diagram ?x)	
	(cocone ?x)	(cocone-diagram ?x) (opvertex ?x) (opfirst ?x) (opsecond ?x) (edge-cocone ?x) (node-cocone ?x) (colimiting-cocone ?x) (colimit ?x) (injection1 ?x) (injection2 ?x) (comediator ?x)	
hgph.col.coeq	(diagram ?x) = (parallel-pair ?x)	(origin ?x) (destination ?x) (morphism1 ?x) (morphism2 ?x) (case-diagram ?x) (edge-diagram ?x) (node-diagram ?x) (indication ?x) (comediation ?x) (projection ?x) (endorelation ?x)	
	(cocone ?x)	(cocone-diagram ?x) (opvertex ?x) (morphism ?x) (case-cocone ?x) (edge-cocone ?x) (node-cocone ?x) (colimiting-cocone ?x) (colimit ?x) = (coequalizer ?x) (canon ?x) (comediator ?x)	
hgph.col.psh	(diagram ?x) = (span ?x)	(hypergraph1 ?x) (hypergraph2 ?x) (vertex ?x) (first ?x) (second ?x) (opposite ?x) (pair ?x) (case-diagram ?x) (edge-diagram ?x) (node-diagram ?x) (coequalizer-diagram ?x) = (parallel-pair ?x)	
	(cocone ?x)	(cocone-diagram ?x) (opvertex ?x) (opfirst ?x) (opsecond ?x) (binary-coproduct-cocone ?x) (coequalizer-cocone ?x) (case-cocone ?x) (edge-cocone ?x) (node-cocone ?x) (colimiting-cocone ?x) (colimit ?x) = (pushout ?x) (injection1 ?x) (injection2 ?x) (comediator ?x)	
hgph.col.endo	(endorelation ?x)	(hypergraph ?x) = (base ?x) (edge ?x) (node ?x)	(subrelation ?x)
	(cocone ?x)	(quotient ?x) (canon ?x) (kernel ?x) (comediator ?x)	(matches ?x) (respects ?x)

The Context of Hypergraphs

hgph

Categories

A useful category specified within the IFF Lower Core (meta) Ontology (IFF-LCO) is **Hypergraph**, the category of hypergraphs and their morphisms. This category is axiomatized in the hypergraph namespace of the IFF-LCO.

The notion of a *hypergraph* of sets is not new, but we use a specific formulation that may be new to the literature. The notion of a hypergraph is closely related to the notion of a spangraph. In fact, this document establishes the fact that the context of hypergraphs is categorically equivalent to the context of spangraphs. As a result, the terminology that we use for hypergraphs is similar to the terminology that we use for spangraphs.

```
(1) (CAT$category hypergraph)
    (= (CAT$object hypergraph) hgph.obj$object)
    (= (CAT$morphism hypergraph) hgph.mor$morphism)
    (= (CAT$source hypergraph) hgph.mor$source)
    (= (CAT$target hypergraph) hgph.mor$target)
    (= (CAT$composable hypergraph) hgph.mor$composable)
    (= (CAT$composition hypergraph) hgph.mor$composition)
    (= (CAT$identity hypergraph) hgph.mor$identity)
```

Functors

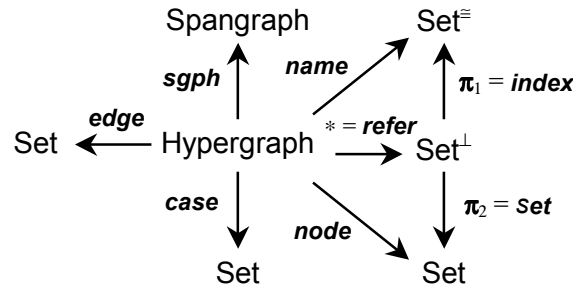


Figure 1: Hypergraph Category and Component Functors

Here is a list of the outer shell functors that are axiomatized in this namespace (Figure 1). The reference functor is basic. As derivative notions, composing this with the projection functors in the set semipair category \mathbf{Set}^\perp gives the component node and name functors to the category \mathbf{Set} and the set bijection subcategory $\mathbf{Set}^=$. There is no component vertex functor for hypergraphs, but in its place is the case functor. There is also a spangraph functor linking the context of hypergraphs with the categorically equivalent context of spangraphs.

- $* = \text{refer} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}^\perp$: The reference functor maps a hypergraph to its reference pair consisting of the name set and the node set, and maps a hypergraph morphism to its reference semi-quartet consisting of the name bijection and the node function.
- $\text{name} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}^=$: The *name* functor maps a hypergraph to its name set and maps a hypergraph morphism to its name bijection. This functor is not basic – it is the composition of the reference functor with the semipair index functor.
- $\text{node} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}$: The *node* functor maps a hypergraph to its node set and maps a hypergraph morphism to its node function. This functor is not basic – it is the composition of the reference functor with the semipair set functor.
- $\text{edge} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}$: The *edge* functor maps a hypergraph to its edge set and maps a hypergraph morphism to its edge function.
- $\text{case} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}$: The *case* functor maps a hypergraph to its case set defined as the coproduct of the edge arity, and maps a hypergraph morphism to its case function defined as the coproduct

of the edge arity morphism. The case functor is defined by (components of) the arity natural transformation.

- *sgph* : Hypergraph \rightarrow Spangraph : The spangraph functor connects the hypergraph category to the spangraph category. It maps a hypergraph to its associated spangraph that is calculated via the case construction. The edge, node and name components are the same. The vertex set is constructed as the case set of the edge arity function. It consists of edge-name pairs, where the name is in the arity of the edge. The indication, comediation and projection components for the spangraph are those of the case construction. The bijection is the identity function on the case set. Any spangraph has an associated hypergraph, which is axiomatized in the spangraph namespace. The hypergraph associated with the spangraph is the original hypergraph. The same is true for morphisms. Hence, the composition of the spangraph functor with the hypergraph functor (defined in the spangraph namespace) is the identity functor on the context of hypergraphs. This property is one half of the categorical equivalence between spangraphs and hypergraphs.

- ```
(2) (FUNC$functor reference)
 (= (FUNC$source reference) hypergraph)
 (= (FUNC$target reference) spr$semipair)
 (= (FUNC$object reference) hgph.obj$reference)
 (= (FUNC$morphism reference) hgph.mor$reference)

(3) (FUNC$functor name)
 (= (FUNC$source name) hypergraph)
 (= (FUNC$target name) set.semi$semi)
 (= (FUNC$object name) hgph.obj$name)
 (= (FUNC$morphism name) hgph.mor$name)
 (= name (FUNC$composition [reference spr$index]))

(4) (FUNC$functor node)
 (= (FUNC$source node) hypergraph)
 (= (FUNC$target node) set$set)
 (= (FUNC$object node) hgph.obj$node)
 (= (FUNC$morphism node) hgph.mor$node)
 (= node (FUNC$composition [reference spr$set]))

(5) (FUNC$functor edge)
 (= (FUNC$source edge) hypergraph)
 (= (FUNC$target edge) set$set)
 (= (FUNC$object edge) hgph.obj$edge)
 (= (FUNC$morphism edge) hgph.mor$edge)

(6) (FUNC$functor case)
 (= (FUNC$source case) hypergraph)
 (= (FUNC$target case) set$set)
 (= (FUNC$object case) hgph.obj$case)
 (= (FUNC$morphism case) hgph.mor$case)

(7) (FUNC$functor spangraph)
 (= (FUNC$source spangraph) hypergraph)
 (= (FUNC$target spangraph) sgph$spangraph)
 (= (FUNC$object spangraph) hgph.obj$spangraph)
 (= (FUNC$morphism spangraph) hgph.mor$spangraph)

(8) (= (FUNC$composition [spangraph sgph$hypergraph])
 (FUNC$identity hypergraph))
```

The spangraph functor is fully defined by four identities. These assert that the edge, node and name components are equivalently related through the spangraph functor, and that the spangraph vertex component is equivalently related to the hypergraph case component: *sgph*  $\circ$  *vert* = *case*, *sgph*  $\circ$  *edge* = *edge*, *sgph*  $\circ$  *node* = *node* and *sgph*  $\circ$  *name* = *name*.

- ```
(9) (= (FUNC$composition [spangraph sgph$vertex]) case)
    (= (FUNC$composition [spangraph sgph$edge]) edge)
    (= (FUNC$composition [spangraph sgph$node]) node)
    (= (FUNC$composition [spangraph sgph$name]) name)
```

Natural Transformations

In general, any natural transformation $\alpha : F_1 \Rightarrow F_2 : A \rightarrow B$ with B as its target category, is logically equivalent to a functor $F : A \rightarrow B^\downarrow$ with the arrow category B^\downarrow as its target category, where the source and target functors are defined as the composition with the (vertical) source and target projection functors $F_1 = F \cdot \pi_1$ and $F_2 = F \cdot \pi_2$. The naturality constraint is identified with the quartet constraint (Diagram 11). Originally, in the formalization there were functors corresponding to the signature, arity, indication, comediation and projection natural transformations. All of these natural transformations have **Set** as their target category. The corresponding functors had the arrow category **Set**[↓] as their target category. However, eventually these were deemed too confusing. The current representation corresponds more closely to the standard representation of category theory. However, for convenience of presentation, the arrow category **Set**[↓] in the guise of the horizontal category of the double category of set quartets **Set**[□], is still used as the target of the component aspects of all of these natural transformations.

Here is a list of the IFF-LCO outer shell natural transformations that are axiomatized in this namespace (Figure 2). The component signature natural transformation is basic – all the remaining natural transformations are derived from the reference functor and the signature natural transformation. The component arity natural transformation is derived as the composition of signature with reference arity. The arity natural transformation defines the case functor. The indication, comediation and projection component functions of hypergraphs (part of the case construction) define indication, comediation and projection component natural transformations. The source functor for all three is the case functor. The target of indication is the edge functor, the target of comediation is the node functor, and the target of projection is the name functor composed with the inclusion functor for the set bijection subcategory. Naturality is expressed by the defining conditions for the indication, comediation and projection quartets of hypergraph morphisms.

$$\begin{array}{ccc}
 A_1 & \xrightarrow{f} & A_2 \\
 F_1(A_1) & \xrightarrow{F_1(f)} & F_1(A_2) \\
 \alpha(A_1) \downarrow & \alpha & \downarrow \alpha(A_2) \\
 F_2(A_1) & \xrightarrow{F_2(f)} & F_2(A_2)
 \end{array}$$

Diagram 11: Naturality as Quartet Constraint

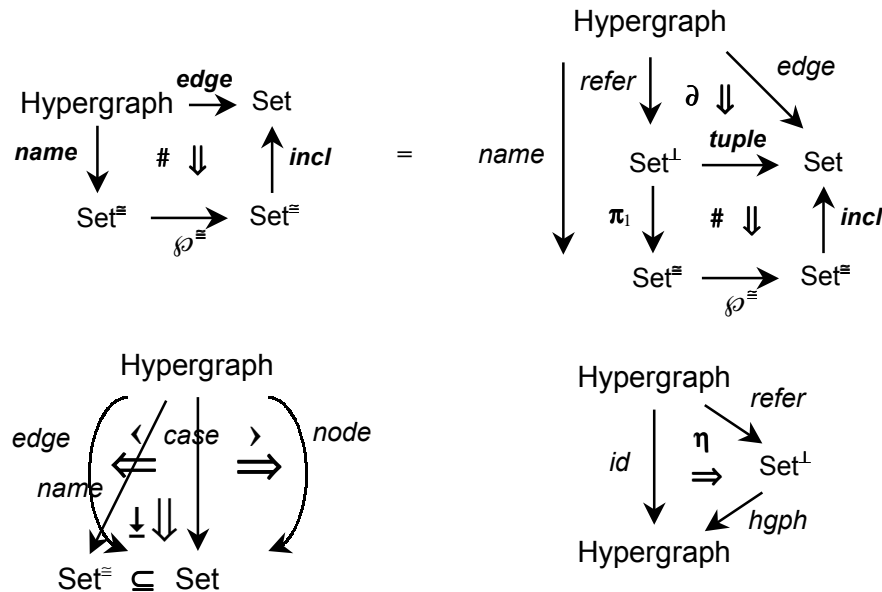


Figure 2: Hypergraph Category and Component Natural Transformations

- $\partial = \text{sign} : \text{edge} \Rightarrow \text{refer} \circ \text{tuple} : \text{Hypergraph} \rightarrow \text{Set}$: This is the *signature* natural transformation whose source is the edge functor, whose target is the composition of the reference and semipair tuple

functors, and whose component at any hypergraph is the signature function. The component signature function maps edges to their reference tuple. Its source is the edge set and its target is the tuple set of the reference set pair.

- $\# = \text{arity} : \text{edge} \Rightarrow \text{name} \circ \text{power} \circ \text{incl} : \text{Hypergraph} \rightarrow \text{Set}$: This is the *arity* natural transformation, whose source is the edge functor, whose target is the composition of the name functor, the set-semi power functor and the inclusion functor, and whose component at any hypergraph is the edge-arity function. The component edge arity function maps edges to their arity subset of names. Its source is the edge set and its target is the power of the name set. The arity natural transformation is not basic – it is definable as the vertical composition: $\# = \partial \bullet (1_{\text{refer}} \circ \#)$.
- $\lt = \text{indic} : \text{case} \Rightarrow \text{edge} : \text{Hypergraph} \rightarrow \text{Set}$: This is the *indication* natural transformation, whose source is the case functor, whose target is the edge functor, and whose component at any hypergraph is the indication function. The indication function maps cases to their (hyper) edges.
- $\gt = \text{comed} : \text{case} \Rightarrow \text{node} : \text{Hypergraph} \rightarrow \text{Set}$: This is the *comediation* natural transformation, whose source is the case functor, whose target is the node functor, and whose component at any hypergraph is the comediation function. The comediation function maps cases to their nodes via their signatures.
- $\perp = \text{proj} : \text{case} \Rightarrow \text{name} \circ \text{incl} : \text{Hypergraph} \rightarrow \text{Set}$: This is the *projection* natural transformation, whose source is the case functor, whose target is the composition of the name functor with the set-semi inclusion functor, and whose component at any hypergraph is the projection function. The projection function maps cases to their names (labels).
- $\eta : \text{id}_{\text{Hypergraph}} \Rightarrow \text{refer} \circ \text{hgph} : \text{Hypergraph} \rightarrow \text{Hypergraph}$: This is the eta natural transformation whose component at any hypergraph is the eta hypergraph morphism $\eta_G : G \rightarrow \text{hgph}(\text{refer}(G))$.

- ```
(10) (NAT$natural-transformation signature)
 (= (NAT$source-category signature) hypergraph)
 (= (NAT$target-category signature) set$set)
 (= (NAT$source-functor signature)
 (FUNC$composition [signature-functor arr$projection1]))
 (= (NAT$target-functor signature)
 (FUNC$composition [signature-functor arr$projection1]))
 (= (NAT$component signature) hgph.obj$signature)

(11) (NAT$natural-transformation arity)
 (= (NAT$source-category arity) hypergraph)
 (= (NAT$target-category arity) set$set)
 (= (NAT$source-functor arity)
 (FUNC$composition [arity-functor arr$projection1]))
 (= (NAT$target-functor arity)
 (FUNC$composition [arity-functor arr$projection2]))
 (= (NAT$component arity) hgph.obj$edge-arity)
 (= arity
 (NAT$vertical-composition
 [signature
 (NAT$horizontal-composition [(NAT$vertical-identity reference) spr$arity])]))

(12) (NAT$natural-transformation indication)
 (= (NAT$source-category indication) hypergraph)
 (= (NAT$target-category indication) set$set)
 (= (NAT$source-functor indication)
 (FUNC$composition [indication-functor arr$projection1]))
 (= (NAT$target-functor indication)
 (FUNC$composition [indication-functor arr$projection2]))
 (= (NAT$component indication) hgph.obj$indication)

(13) (NAT$natural-transformation comediation)
 (= (NAT$source-category comediation) hypergraph)
 (= (NAT$target-category comediation) set$set)
 (= (NAT$source-functor comediation)
 (FUNC$composition [comediation-functor arr$projection1]))
 (= (NAT$target-functor comediation)
 (FUNC$composition [comediation-functor arr$projection2]))
 (= (NAT$component comediation) hgph.obj$comediation)
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 8

May 8, 2003

```
(14) (NAT$natural-transformation projection)
 (= (NAT$source-category projection) hypergraph)
 (= (NAT$target-category projection) set$set)
 (= (NAT$source-functor projection)
 (FUNC$composition [projection-functor harr$projection1]))
 (= (NAT$target-functor projection)
 (FUNC$composition [name (FUNC$inclusion [set.semi$semi set.obj$object])]))
 (= (NAT$component projection) hgph.obj$projection)

(15) (NAT$natural-transformation eta)
 (= (NAT$source-category eta) hypergraph)
 (= (NAT$target-category eta) hypergraph)
 (= (NAT$source-functor eta) (FUNC$identity hypergraph))
 (= (NAT$target-functor eta) (FUNC$composition [reference spr$hypergraph]))
 (= (NAT$component eta) hgph.obj$eta)
```

These natural transformations satisfy several kinds of identities. The first set of identities assert that the indication, comediation and projection components are equivalently related through the spangraph functor:

$\mathbf{1}_{sgph} \circ \langle \prec, \mathbf{1}_{sgph} \circ \succ \rangle = \downarrow$ , and  $\mathbf{1}_{sgph} \circ \downarrow = \downarrow$ .

```
(16) (= (NAT$horizontal-composition
 [(NAT$vertical-identity spangraph) sgph$indication]) indication)

(17) (= (NAT$horizontal-composition
 [(NAT$vertical-identity spangraph) sgph$comediation])) comediation)

(18) (= (NAT$horizontal-composition
 [(NAT$vertical-identity spangraph) sgph$projection])) projection)
```

The second set of identities express the fact that bijection and realization are identity for the spangraph image:  $\mathbf{1}_{sgph} \circ \cong = \mathbf{1}_{case}$  and  $\mathbf{1}_{sgph} \circ \uparrow = \mathbf{1}_{sgph}$ .

```
(19) (= (NAT$horizontal-composition [(NAT$vertical-identity spangraph) sgph$bijection])
 (NAT$vertical-identity case))

(20) (= (NAT$horizontal-composition [(NAT$vertical-identity spangraph) sgph$realization])
 (NAT$vertical-identity spangraph))
```

The third set of identities express the triangle identities required for the adjunction below:  $\mathbf{1}_{hgph} \circ \eta = \mathbf{1}_{hgph}$  and  $\eta \circ \mathbf{1}_{refer} = \mathbf{1}_{refer}$ .

```
(21) (= (NAT$horizontal-composition [(NAT$vertical-identity spr$hypergraph) eta])
 (NAT$vertical-identity spr$hypergraph))

(22) (= (NAT$horizontal-composition [eta (NAT$vertical-identity reference)])
 (NAT$vertical-identity reference))
```

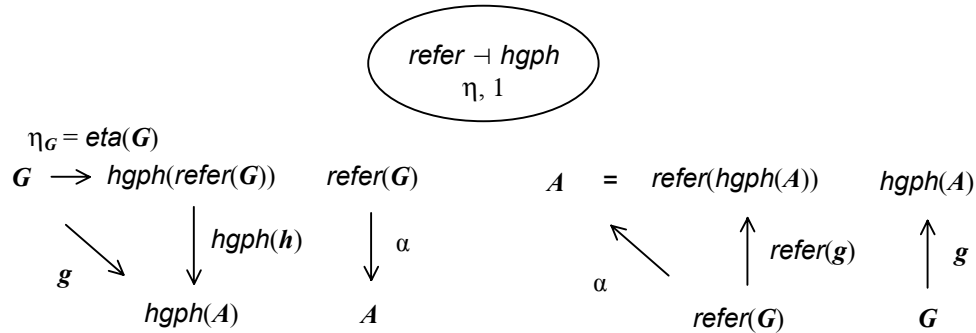


Diagram 1a: Universal Morphism

Diagram 1b: Couniversal Morphism

## Adjunctions

We want to verify that the reference adjunction  $\mathbf{p} = \langle refer, hgph, \eta, 1 \rangle : \text{Hypergraph} \rightarrow \text{Set}^1$  forms an adjunction (actually a reflection) (Diagrams 1a, 1b). This adjunction links the contexts of hypergraph and



semipairs by reference – the category  $\mathbf{Set}^\perp$  is a reflective subcategory of the category  $\mathbf{Hypergraph}$  with the reference functor  $\mathbf{refer} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}^\perp$  as the reflector. To do this we need to verify that the hypergraph inclusion functor

$$\mathbf{hgph} : \mathbf{Set}^\perp \rightarrow \mathbf{Hypergraph}$$

is the right adjoint left inverse (rali) of the reflection  $\rho = \langle \mathbf{refer}, \mathbf{hgph}, \eta, 1 \rangle : \mathbf{Hypergraph} \rightarrow \mathbf{Set}^\perp$ . In other words, given any hypergraph  $G$ , we want to verify that the reference set pair  $\mathbf{refer}(G)$  is the free pair over  $G$ . More abstractly, we want to verify that the reference and hypergraph functors participate in an adjunction where

- the reference functor  $\mathbf{refer} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}^\perp$  is the left adjoint,
- the hypergraph functor  $\mathbf{hgph} : \mathbf{Set}^\perp \rightarrow \mathbf{Hypergraph}$  is the right adjoint,
- for any hypergraph  $G$  the eta hypergraph morphism  $\eta_G : G \rightarrow \mathbf{hgph}(\mathbf{refer}(G))$  is the  $G^{\text{th}}$  component of the unit natural transformation  $\eta : \text{id}_{\mathbf{Hypergraph}} \Rightarrow \mathbf{refer} \circ \mathbf{hgph}$ , and
- the counit natural transformation is the identity  $1_{\mathbf{Set}^\perp} = \mathbf{hgph} \circ \mathbf{refer}$ .

$$\begin{array}{ccc} \text{refer}(G) \left\{ \begin{array}{ccc} \text{name}(G) & \xrightarrow[\cong]{\text{name}(g)} & \text{index}(A) \\ & \text{refer}(g) & \\ \text{node}(G) & \xrightarrow[\text{node}(g)]{} & \text{set}(A) \end{array} \right\} A & & \begin{array}{ccc} \text{edge}(G) & \xrightarrow{\text{edge}(g)} & \text{tuple}(A) \\ \text{sign}(G) \downarrow & \text{sign}(g) \downarrow & \text{id}_{\text{tuple}(A)} \\ \text{tuple}(\text{refer}(G)) & \xrightarrow{\text{tuple}(\text{refer}(g))} & \text{tuple}(A) \end{array} \end{array}$$

Figure 3: Hypergraph Morphism  $g$

$$\begin{array}{ccc} \text{refer}(G) \left\{ \begin{array}{ccc} \text{name}(G) & \xrightarrow[\cong]{\text{id}} & \text{name}(G) \\ & \text{refer}(\eta_G) & \\ \text{node}(G) & \xrightarrow[\text{id}]{} & \text{node}(G) \end{array} \right\} \text{refer}(G) & & \begin{array}{ccc} \text{edge}(G) & \xrightarrow{\text{sign}(G)} & \text{tuple}(\text{refer}(G)) \\ \text{sign}(G) \downarrow & \text{sign}(\eta_G) \downarrow & \text{id} \\ \text{tuple}(\text{refer}(G)) & \xrightarrow[\text{id}]{} & \text{tuple}(\text{refer}(G)) \end{array} \end{array}$$

Figure 3: Hypergraph Morphism  $\eta_G$

**[Universal morphism]** More concretely, we want to verify (Diagram 1a) that for any hypergraph  $G$ , the pair  $\langle \eta_G, \mathbf{refer}(G) \rangle$  consisting the reference set pair  $\mathbf{refer}(G)$  and the eta hypergraph morphism  $\eta_G = \mathbf{eta}(G) : G \rightarrow \mathbf{hgph}(\mathbf{refer}(G))$ , forms a universal morphism from  $G$  to  $\mathbf{hgph}$ . To verify this, we need to show that for every pair  $\langle g, A \rangle$  consisting of a set pair  $A$  and a hypergraph morphism  $g : G \rightarrow \mathbf{hgph}(A)$ , there is a unique semiquartet  $\alpha : \mathbf{refer}(G) \rightarrow A$  with  $\eta_G \cdot \mathbf{hgph}(\alpha) = g$ . Define the quartet  $\alpha = \mathbf{refer}(g)$ . This is easily seen to be unique – just look at the reference aspect of the equation:  $\mathbf{refer}(g) = \mathbf{refer}(\eta_G) \cdot \mathbf{refer}(\mathbf{hgph}(\alpha)) = \mathbf{refer}(\mathbf{hgph}(\alpha)) = \alpha$ .

**[Couniversal morphism]** Alternately, we want to verify (Diagram 1b) that for any set pair  $K$ , the pair  $\langle \mathbf{hgph}(A), \text{id}_A \rangle$  consisting of the pair  $A$  embedded as a hypergraph  $\mathbf{hgph}(A)$  and the identity semiquartet  $\text{id}_A : \mathbf{refer}(\mathbf{hgph}(A)) \rightarrow A$ , forms a couniversal morphism to  $A$  from  $\mathbf{refer}$ . To verify this, we need to show that for every pair  $\langle G, \alpha \rangle$  consisting of a hypergraph  $G$  and a semiquartet  $\alpha : \mathbf{refer}(G) \rightarrow A$ , there is a unique hypergraph morphism  $g : G \rightarrow \mathbf{hgph}(A)$  with  $\mathbf{refer}(g) = \alpha$ . Just define  $g$ , where  $\mathbf{refer}(g) = \alpha$  and  $\text{edge}(g) = \text{sign}(G) \cdot \text{tuple}(\alpha)$ .

```
(23) (ADJ$adjunction reflection)
 (= (ADJ$underlying-category reflection) hypergraph)
 (= (ADJ$free-category reflection) spr$semipair)
 (= (ADJ$left-adjoint reflection) reference)
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 10

May 8, 2003

```
(= (ADJ$right-adjoint reflection) spr$hypergraph)
(= (ADJ$unit reflection) eta)
(= (ADJ$counit reflection) (NAT$identity (FUNC$identity spr$semipair)))
```

## The Contexts of Hypergraph Fibers

**hgph.fbr**

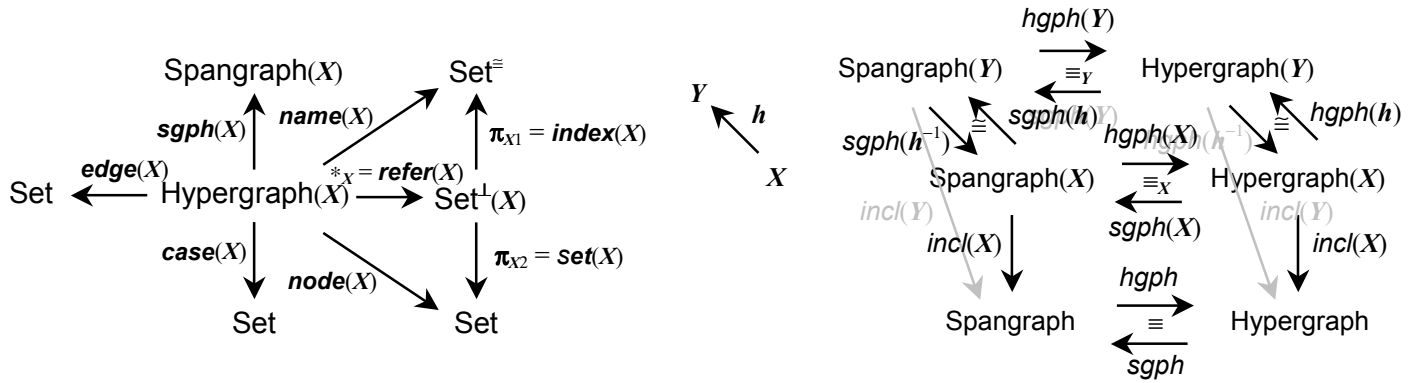
### Fiber Categories

For any set  $X$  representing a fixed set of names, there is a fiber *hypergraph* subcategory

$$\text{Hypergraph}(X) \subseteq \text{Hypergraph}$$

consisting of all hypergraphs whose name set is  $X$  and all hypergraph morphisms whose name bijection is  $\text{id}_X : X \rightarrow X$ . Since **Hypergraph** is the origin, all the functors and natural transformations for the full category of hypergraphs can be restricted to fibers.

```
(1) (KIF$function hypergraph)
 (= (KIF$source hypergraph) set.obj$object)
 (= (KIF$target hypergraph) CAT$category)
 (forall (?x (set.obj$object ?x))
 (and (CAT$subcategory (hypergraph ?x) hgph$hypergraph)
 (= (CAT$object (hypergraph ?x))
 (hgph.fbr.obj$object ?x))
 (= (CAT$morphism (hypergraph ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (CAT$source (hypergraph ?x))
 (hgph.fbr.mor$source (set.mor$identity ?x)))
 (= (CAT$target (hypergraph ?x))
 (hgph.fbr.mor$target (set.mor$identity ?x)))
 (= (CAT$composable (hypergraph ?x))
 (hgph.fbr.mor$composable [(set.mor$identity ?x) (set.mor$identity ?x)]))
 (= (CAT$composition (hypergraph ?x))
 (hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]))
 (= (CAT$identity (hypergraph ?x))
 (hgph.fbr.mor$identity ?x))))
```



**Diagram 2: Hypergraph Fiber Categories, Functors, Natural Transformations and Adjunctions**

### Fiber Functors

For any set  $X$  representing a fixed set of names, here is a list of the outer shell fiber functors that are axiomatized in this namespace (Diagram 2). The component fiber reference functor is basic. As derivative notions, composing this with the fiber projection functors in the fiber set semipair category  $\text{Set}^+(X)$  gives the component fiber node and name functors to the category **Set** and the set bijection subcategory  $\text{Set}^=$ . There is no component fiber vertex functor for hypergraphs, but in its place is the fiber case functor. There is also a fiber spangraph functor linking the fiber context of hypergraphs with the categorically equivalent fiber context of spangraphs.

- $\text{incl}(X) : \text{Hypergraph}(X) \rightarrow \text{Hypergraph}$  : The *inclusion* functor embeds the fiber category  $\text{Hypergraph}(X)$  into the full category of hypergraphs.

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 12

May 8, 2003

- $\ast = \text{refer}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}^+(X)$  : The fiber *reference* functor maps a hypergraph with name set  $X$  to its reference pair consisting of the name set  $X$  and the node set, and maps a hypergraph morphism with name bijection  $id_X$  to its reference semi-quartet consisting of the name bijection  $id_X$  and the node function.
- $\text{name}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}^{\pm}$  : The fiber *name* functor maps a hypergraph with name set  $X$  to that name set and maps a hypergraph morphism with name bijection  $id_X$  to that name bijection. This fiber functor is not basic – it is the composition of the fiber reference functor with the semipair fiber index functor.
- $\text{node}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : The fiber *node* functor maps a hypergraph with name set  $X$  to its node set and maps a hypergraph morphism with name bijection  $id_X$  to its node function. This functor is not basic – it is the composition of the fiber reference functor with the semipair fiber set functor.
- $\text{edge}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : The fiber *edge* functor maps a hypergraph with name set  $X$  to its edge set and maps a hypergraph morphism with name bijection  $id_X$  to its edge function.
- $\text{case}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : The fiber *case* functor maps a hypergraph with name set  $X$  to its case set defined as the coproduct of the edge arity, and maps a hypergraph morphism to its case function defined as the coproduct of the edge arity morphism. The fiber case functor is defined by (components of) the fiber arity natural transformation.
- $\text{sgph}(X) : \text{Hypergraph}(X) \rightarrow \text{Spangraph}(X)$  : The fiber spangraph functor connects the fiber hypergraph category to the fiber spangraph category. It maps a hypergraph with name set  $X$  to its associated spangraph that is calculated via the case construction. The edge, node and name components are the same. The vertex set is constructed as the case set of the edge arity function. It consists of edge-name pairs, where the name is in the arity of the edge. The indication, comediation and projection components for the spangraph are those of the case construction. The bijection is the identity function on the case set. Any spangraph with name set  $X$  has an associated hypergraph with name set  $X$ , which is axiomatized in the fiber spangraph namespace. The hypergraph associated with the spangraph is the original hypergraph. The same is true for morphisms. Hence, the composition of the fiber spangraph functor with the fiber hypergraph functor is the identity functor on the fiber context of hypergraphs. This property is one half of the fiber categorical equivalence between spangraphs and hypergraphs.

```
(2) (KIF$function inclusion)
(= (KIF$source inclusion) set.obj$object)
(= (KIF$target inclusion) FUNC$functor)
(forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (inclusion ?x)) (hypergraph ?x))
 (= (FUNC$target (inclusion ?x)) hgph$hypergraph)
 (= (FUNC$object (inclusion ?x))
 (SET.FTN$inclusion [(hgph.fbr.obj$object ?x) hgph.obj$object]))
 (= (FUNC$morphism (inclusion ?x))
 (SET.FTN$inclusion
 [(hgph.fbr.mor$morphism (set.mor$identity ?x)) hgph.mor$morphism]))))

(3) (KIF$function reference)
(= (KIF$source reference) set.obj$object)
(= (KIF$target reference) FUNC$functor)
(forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (reference ?x)) (hypergraph ?x))
 (= (FUNC$target (reference ?x)) (spr.fbr$semipair ?x))
 (= (FUNC$object (reference ?x)) (hgph.fbr.obj$reference ?x))
 (= (FUNC$morphism (reference ?x)) (hgph.fbr.mor$reference (set.mor$identity ?x)))
 (FUNC$restriction (reference ?x) hgph$reference)))

(4) (KIF$function name)
(= (KIF$source name) set.obj$object)
(= (KIF$target name) FUNC$functor)
(forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (name ?x)) (hypergraph ?x))
 (= (FUNC$target (name ?x)) set.semi$semi)
 (= (FUNC$object (name ?x)) (hgph.fbr.obj$name ?x))
 (= (FUNC$morphism (name ?x)) (hgph.fbr.mor$name (set.mor$identity ?x)))
 (= (name ?x) (FUNC$composition [(reference ?x) (spr.fbr$index ?x)]))
 (FUNC$restriction (name ?x) hgph$name)))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 13

May 8, 2003

```
(5) (KIF$function node)
 (= (KIF$source node) set.obj$object)
 (= (KIF$target node) FUNC$functor)
 (forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (node ?x)) (hypergraph ?x))
 (= (FUNC$target (node ?x)) set$set)
 (= (FUNC$object (node ?x)) (hgph.fbr.obj$node ?x))
 (= (FUNC$morphism (node ?x)) (hgph.fbr.mor$node (set.mor$identity ?x)))
 (= (node ?x) (FUNC$composition [(reference ?x) (spr.fbr$set ?x)]))
 (FUNC$restriction (node ?x) hgph$node)))

(6) (KIF$function edge)
 (= (KIF$source edge) set.obj$object)
 (= (KIF$target edge) FUNC$functor)
 (forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (edge ?x)) (hypergraph ?x))
 (= (FUNC$target (edge ?x)) set$set)
 (= (FUNC$object (edge ?x)) (hgph.fbr.obj$edge ?x))
 (= (FUNC$morphism (edge ?x)) (hgph.fbr.mor$edge (set.mor$identity ?x)))
 (FUNC$restriction (edge ?x) hgph$edge)))

(7) (KIF$function case)
 (= (KIF$source case) set.obj$object)
 (= (KIF$target case) FUNC$functor)
 (forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (case ?x)) (hypergraph ?x))
 (= (FUNC$target (case ?x)) set$set)
 (= (FUNC$object (case ?x)) (hgph.fbr.obj$case ?x))
 (= (FUNC$morphism (case ?x)) (hgph.fbr.mor$case (set.mor$identity ?x)))
 (FUNC$restriction (case ?x) hgph$case)))

(8) (KIF$function spangraph)
 (= (KIF$source spangraph) set.obj$object)
 (= (KIF$target spangraph) FUNC$functor)
 (forall (?x (set.obj$object ?x))
 (and (= (FUNC$source (spangraph ?x)) (hypergraph ?x))
 (= (FUNC$target (spangraph ?x)) (sgph.fbr$spangraph ?x))
 (= (FUNC$object (spangraph ?x)) (hgph.fbr.obj$spangraph ?x))
 (= (FUNC$morphism (spangraph ?x)) (hgph.fbr.mor$spangraph (set.mor$identity ?x)))
 (FUNC$restriction (spangraph ?x) hgph$spangraph)))

(9) (= (FUNC$composition [spangraph sgph$hypergraph])
 (FUNC$identity hypergraph))
```

The fiber spangraph functor is fully defined by four identities. These assert that the fiber edge, node and name components are equivalently related through the fiber spangraph functor, and that the fiber spangraph vertex component is equivalently related to the fiber hypergraph case component:  $sgph(X) \circ vert(X) = case(X)$ ,  $sgph(X) \circ edge(X) = edge(X)$ ,  $sgph(X) \circ node(X) = node(X)$  and  $sgph(X) \circ name(X) = name(X)$ .

```
(10) (forall (?x (set.obj$object ?x))
 (and (= (FUNC$composition [(spangraph ?x) (sgph.fbr$vertex ?x)]) (case ?x))
 (= (FUNC$composition [(spangraph ?x) (sgph.fbr$edge ?x)]) (edge ?x))
 (= (FUNC$composition [(spangraph ?x) (sgph.fbr$node ?x)]) (node ?x))
 (= (FUNC$composition [(spangraph ?x) (sgph.fbr$name ?x)]) (name ?x))))
```

For any set bijection  $h : X \rightarrow Y$  there is a *direct image* functor

$$h^{+1} : \text{Hypergraph}(X) \rightarrow \text{Hypergraph}(Y)$$

along  $h$  and an *inverse image* functor

$$h^{-1} : \text{Hypergraph}(Y) \rightarrow \text{Hypergraph}(X)$$

along  $h$ . These two functors are inverse to each other. Hence, the two fiber categories are isomorphic

$$\text{Hypergraph}(X) \cong \text{Hypergraph}(Y).$$

```
(11) (KIF$function direct-image)
 (= (KIF$source direct-image) set.mor$bijection)
 (= (KIF$target direct-image) FUNC$functor)
 (forall (?h (set.mor$bijection ?h))
```

```

 (and (= (FUNC$source (direct-image ?h)) (hypergraph (set.mor$source ?h)))
 (= (FUNC$target (direct-image ?h)) (hypergraph (set.mor$target ?h)))
 (= (FUNC$object (direct-image ?h)) (hgph.fbr.obj$direct-image ?h))
 (= (FUNC$morphism (direct-image ?h)) (hgph.fbr.mor$direct-image ?h)))

(12) (KIF$function inverse-image)
 (= (KIF$source inverse-image) set.mor$bijection)
 (= (KIF$target inverse-image) FUNC$functor)
 (forall (?h (set.mor$bijection ?h))
 (and (= (FUNC$source (inverse-image ?h)) (hypergraph (set.mor$target ?h)))
 (= (FUNC$target (inverse-image ?h)) (hypergraph (set.mor$source ?h)))
 (= (FUNC$object (inverse-image ?h)) (hgph.fbr.obj$inverse-image ?h))
 (= (FUNC$morphism (inverse-image ?h)) (hgph.fbr.mor$inverse-image ?h))))

(13) (forall (?h (set.mor$bijection ?h))
 (and (= (FUNC$composition [(direct-image ?h) (inverse-image ?h)])
 (FUNC$identity (hypergraph (set.mor$source ?h))))
 (= (FUNC$composition [(inverse-image ?h) (direct-image ?h)])
 (FUNC$identity (hypergraph (set.mor$target ?h))))
 (FUNC$isomorphic (hypergraph (set.mor$source ?h)) (hypergraph (set.mor$target ?h)))))

```

## Fiber Natural Transformations

For any set  $X$  representing a fixed set of names, here is a list of the IFF-LCO outer shell fiber natural transformations that are axiomatized in this namespace (Figure 4). The component fiber signature natural transformation is basic – all the remaining fiber natural transformations are derivable from the fiber reference functor and the fiber signature natural transformation. The component fiber arity natural transformation is derived as the composition of fiber signature with fiber reference arity. The fiber arity natural transformation defines the fiber case functor. The fiber indication, comediation and projection component functions of hypergraphs (part of the fiber case construction) define fiber indication, comediation and projection component natural transformations. The source functor for all three is the fiber case functor. The target of indication is the fiber edge functor, the target of comediation is the fiber node functor, and the target of projection is the fiber name functor composed with the inclusion functor for the set bijection subcategory.

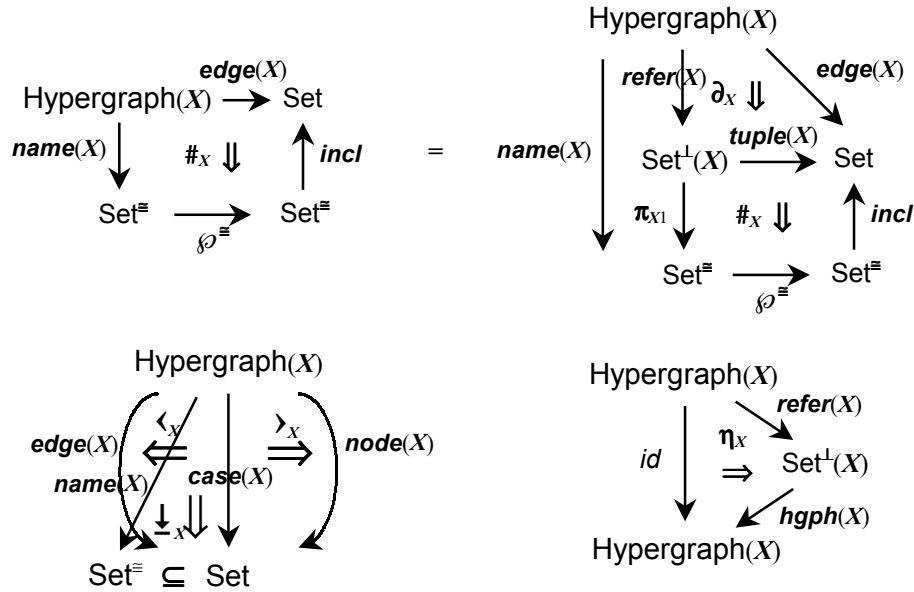


Figure 4: Hypergraph Category and Component Natural Transformations

- $\partial_X = \text{sign}(X) : \text{edge}(X) \Rightarrow \text{refer}(X) \circ \text{tuple}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : This is the fiber *signature* natural transformation whose source is the fiber edge functor, whose target is the composition of the

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 15

May 8, 2003

fiber reference and semipair fiber tuple functors, and whose component at any hypergraph with name set  $X$  is the fiber signature function.

- $\#_X = \text{arity}(X) : \text{edge}(X) \Rightarrow \text{name}(X) \circ \text{power} \circ \text{incl} : \text{Hypergraph}(X) \rightarrow \text{Set}$  : This is the fiber *arity* natural transformation, whose source is the fiber edge functor, whose target is the composition of the fiber name functor, the set-semi power functor and the inclusion functor, and whose component at any hypergraph with name set  $X$  is the fiber edge-arity function. The fiber arity natural transformation is not basic – it is definable as the vertical composition:  $\#_X = \partial_X \bullet (\mathbf{1}_{\text{refer}(X)} \circ \#_X)$ .
- $\llcorner_X = \text{indic}(X) : \text{case}(X) \Rightarrow \text{edge}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : This is the fiber *indication* natural transformation, whose source is the fiber case functor, whose target is the fiber edge functor, and whose component at any hypergraph with name set  $X$  is the fiber indication function.
- $\triangleright_X = \text{comed}(X) : \text{case}(X) \Rightarrow \text{node}(X) : \text{Hypergraph}(X) \rightarrow \text{Set}$  : This is the fiber *comediation* natural transformation, whose source is the fiber case functor, whose target is the fiber node functor, and whose component at any hypergraph with name set  $X$  is the fiber comediation function.
- $\perp_X = \text{proj}(X) : \text{case}(X) \Rightarrow \text{name}(X) \circ \text{incl} : \text{Hypergraph}(X) \rightarrow \text{Set}$  : This is the fiber *projection* natural transformation, whose source is the fiber case functor, whose target is the composition of the fiber name functor with the set-semi inclusion functor, and whose component at any hypergraph with name set  $X$  is the fiber projection function.
- $\eta_X : \text{id}_{\text{Hypergraph}(X)} \Rightarrow \text{refer}(X) \circ \text{hgph}(X) : \text{Hypergraph}(X) \rightarrow \text{Hypergraph}(X)$  : This is the fiber eta natural transformation whose component at any hypergraph  $G$  with name set  $X$  is the fiber eta hypergraph morphism  $\eta_{X,G} : G \rightarrow \text{hgph}(X)(\text{refer}(X)(G))$ .

- ```
(14) (KIF$function signature)
  (= (KIF$source signature) set.obj$object)
  (= (KIF$target signature) NAT$natural-transformation)
  (forall (?x (set.obj$object ?x))
    (and (= (NAT$source-category (signature ?x)) (hypergraph ?x))
          (= (NAT$target-category (signature ?x)) set$set)
          (= (NAT$source-functor (signature ?x)) (edge ?x))
          (= (NAT$target-functor (signature ?x))
              (FUNC$composition [(reference ?x) (spr.fbr$tuple ?x)]))
          (= (NAT$component (signature ?x)) (hgph.fbr.obj$signature ?x))
          (NAT$restriction (signature ?x) hgph$signature)))

(15) (KIF$function arity)
  (= (KIF$source arity) set.obj$object)
  (= (KIF$target arity) NAT$natural-transformation)
  (forall (?x (set.obj$object ?x))
    (and (= (NAT$source-category (arity ?x)) (hypergraph ?x))
          (= (NAT$target-category (arity ?x)) set$set)
          (= (NAT$source-functor (arity ?x)) (edge ?x))
          (= (NAT$target-functor (arity ?x))
              (FUNC$composition [(FUNC$composition
                                   [(name ?x) set.semi$power]
                                   (FUNC$inclusion [set.semi$semi set$set])])
                                [(name ?x) set.semi$power]
                                (FUNC$inclusion [set.semi$semi set$set]))))
          (= (NAT$component (arity ?x)) (hgph.fbr.obj$edge-arity ?x))
          (= (arity ?x)
              (NAT$vertical-composition
               [(signature ?x)
                (NAT$horizontal-composition
                 [(NAT$vertical-identity (reference ?x)) (spr.fbr$ararity ?x)])))))
  (NAT$restriction (arity ?x) hgph$ararity)))

(16) (KIF$function indication)
  (= (KIF$source indication) set.obj$object)
  (= (KIF$target indication) NAT$natural-transformation)
  (forall (?x (set.obj$object ?x))
    (and (= (NAT$source-category (indication ?x)) (hypergraph ?x))
          (= (NAT$target-category (indication ?x)) set$set)
          (= (NAT$source-functor (indication ?x)) (case ?x))
          (= (NAT$target-functor (indication ?x)) (edge ?x))
          (= (NAT$component (indication ?x)) (hgph.fbr.obj$indication ?x))))
  (NAT$restriction (indication ?x) hgph$indication)))

(17) (KIF$function comediation)
  (= (KIF$source comediation) set.obj$object)
```

The IFF Namespace of Hypergraphs

Robert E. Kent

Page 16

May 8, 2003

```
(= (KIF$target comediation) NAT$natural-transformation)
(forall (?x (set.obj$object ?x))
  (and (= (NAT$source-category (comediation ?x)) (hypergraph ?x))
    (= (NAT$target-category (comediation ?x)) set$set)
    (= (NAT$source-functor (comediation ?x)) (case ?x))
    (= (NAT$target-functor (comediation ?x)) (node ?x))
    (= (NAT$component (comediation ?x)) (hgph.fbr.obj$comediation ?x)))
    (NAT$restriction (comediation ?x) hgph$comediation)))

(18) (KIF$function projection)
(= (KIF$source projection) set.obj$object)
(= (KIF$target projection) NAT$natural-transformation)
(forall (?x (set.obj$object ?x))
  (and (= (NAT$source-category (projection ?x)) (hypergraph ?x))
    (= (NAT$target-category (projection ?x)) set$set)
    (= (NAT$source-functor (projection ?x)) (case ?x))
    (= (NAT$target-functor (projection ?x))
      (FUNC$composition [(name ?x) (FUNC$inclusion [set.semi$semi set$set])]))
    (= (NAT$component (projection ?x)) (hgph.fbr.obj$projection ?x)))
    (NAT$restriction (projection ?x) hgph$projection)))

(19) (KIF$function eta)
(= (KIF$source eta) set.obj$object)
(= (KIF$target eta) NAT$natural-transformation)
(forall (?x (set.obj$object ?x))
  (and (= (NAT$source-category (eta ?x)) (hypergraph ?x))
    (= (NAT$target-category (eta ?x)) (hypergraph ?x))
    (= (NAT$source-functor (eta ?x)) (FUNC$identity (hypergraph ?x)))
    (= (NAT$target-functor (eta ?x))
      (FUNC$composition [(reference ?x) (spr.fbr$hypergraph ?x)]))
    (= (NAT$component (eta ?x)) (hgph.fbr.obj$eta ?x)))
    (NAT$restriction (eta ?x) hgph$eta)))
```

These natural transformations satisfy several kinds of identities. The first set of identities assert that the fiber indication, comediation and projection components are equivalently related through the fiber spangraph functor: $\mathbf{1}_{sgph(X)} \circ \langle X = \langle X, \mathbf{1}_{sgph(X)} \rangle_X = \rangle_X$, and $\mathbf{1}_{sgph(X)} \circ \perp_X = \perp_X$.

```
(20) (forall (?x (set.obj$object ?x))
  (and (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spangraph ?x)) (sgph.fbr$indication ?x)])
    (indication ?x))
    (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spangraph ?x)) (sgph.fbr$comediation ?x)])
    (comediation ?x))
    (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spangraph ?x)) (sgph.fbr$projection ?x)])
    (projection ?x))))
```

The second set of identities express the fact that fiber bijection and realization are identity for the fiber spangraph image: $\mathbf{1}_{sgph(X)} \circ \cong_X = \mathbf{1}_{case(X)}$ and $\mathbf{1}_{sgph(X)} \circ \uparrow_X = \mathbf{1}_{sgph(X)}$.

```
(21) (forall (?x (set.obj$object ?x))
  (and (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spangraph ?x)) (sgph.fbr$bijection ?x)])
    (NAT$vertical-identity (case ?x)))
    (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spangraph ?x)) (sgph.fbr$realization ?x)])
    (NAT$vertical-identity (spangraph ?x)))))
```

The third set of identities express the triangle identities required for the fiber adjunction below: $\mathbf{1}_{hgph(X)} \circ \eta_X = \mathbf{1}_{hgph(X)}$ and $\eta_X \circ \mathbf{1}_{refer(X)} = \mathbf{1}_{refer(X)}$.

```
(22) (forall (?x (set.obj$object ?x))
  (and (= (NAT$horizontal-composition
    [(NAT$vertical-identity (spr.fbr$hypergraph ?x)) (eta ?x)])
    (NAT$vertical-identity (spr.fbr$hypergraph ?x)))
    (= (NAT$horizontal-composition
    [(eta ?x) (NAT$vertical-identity (reference ?x))])
    (NAT$vertical-identity (reference ?x)))))
```


Fiber Adjunctions

For any set X representing a fixed set of names, there is a fiber reference adjunction $\rho_X = \langle \mathbf{refer}(X), \mathbf{hgph}(X), \eta_X, 1 \rangle : \mathbf{Hypergraph}(X) \rightarrow \mathbf{Set}^\perp(X)$ (actually a reflection). This adjunction links the contexts of hypergraph fibers and semipair fibers by reference – the fiber category $\mathbf{Set}^\perp(X)$ is a reflective subcategory of the fiber category $\mathbf{Hypergraph}(X)$ with the fiber reference functor $\mathbf{refer}(X) : \mathbf{Hypergraph}(X) \rightarrow \mathbf{Set}^\perp(X)$ as the reflector. That is, the fiber reference and fiber hypergraph functors participate in an adjunction where

- the fiber reference functor $\mathbf{refer}(X) : \mathbf{Hypergraph}(X) \rightarrow \mathbf{Set}^\perp(X)$ is the left adjoint,
- the fiber hypergraph functor $\mathbf{hgph}(X) : \mathbf{Set}^\perp(X) \rightarrow \mathbf{Hypergraph}(X)$ is the right adjoint,
- for any hypergraph G the fiber eta hypergraph morphism $\eta_{X,G} : G \rightarrow \mathbf{hgph}(X)(\mathbf{refer}(X)(G))$ is the G^{th} component of the unit fiber natural transformation $\eta_X : \mathbf{id}_{\mathbf{Hypergraph}(X)} \Rightarrow \mathbf{refer}(X) \circ \mathbf{hgph}(X)$, and
- the counit natural transformation is the identity $1_{\mathbf{Set}^\perp(X)} = \mathbf{hgph}(X) \circ \mathbf{refer}(X)$.

```
(23) (KIF$function reflection)
      (= (KIF$source reflection) set.obj$object)
      (= (KIF$target reflection) ADJ$adjunction)
      (forall (?x (set.obj$object ?x))
        (and (= (ADJ$underlying-category (reference ?x)) (hypergraph ?x))
              (= (ADJ$free-category (reference ?x)) (spr.fbr$semipair ?x))
              (= (ADJ$left-adjoint (reference ?x)) (reference ?x))
              (= (ADJ$right-adjoint (reference ?x)) (spr.fbr$hypergraph ?x))
              (= (ADJ$unit (reference ?x)) (eta ?x))
              (= (ADJ$counit (reference ?x)) (NAT$identity (FUNC$identity (spr.fbr$semipair ?x))))))
```

Hypergraphs

hgph.obj

This section discusses hypergraphs. First, we give a concise mathematical definition (Figure 5), and then we discuss and formalize the various parts of this definition.

A hypergraph $G = \langle \text{refer}(G), \text{sign}(G) \rangle$ consists of

- a reference set pair $*_G = \text{refer}(G)$
- a signature function $\partial_G = \text{sign}(G)$

that satisfy the pullback constraint

$$\text{tgt}(\text{sign}(G)) = \text{tuple}(\text{refer}(G)).$$

- ```
(1) (SET$class object)
 (SET$class hypergraph)
 (= hypergraph object)

(2) (SET.FTN$function reference)
 (= (SET.FTN$source reference) object)
 (= (SET.FTN$target reference) set.pr$pair)

(3) (SET.FTN$function signature)
 (= (SET.FTN$source signature) object)
 (= (SET.FTN$target signature) set.mor$morphism)

(4) (= (SET.FTN$composition [signature set.mor$target])
 (SET.FTN$composition [reference set.pr$tuple]))
```

For convenience of theoretical presentation, we introduce additional hypergraph terminology for the composition between the reference function and the tuple arity function

- reference-arity function  $\text{refer-arity}(G) = \text{tuple-arity}(*_G) : \text{tuple}(\text{refer}(G)) \rightarrow \wp \text{name}(G)$ .

- ```
(5) (SET.FTN$function reference-arity)
    (= (SET.FTN$source reference-arity) object)
    (= (SET.FTN$target reference-arity) set.mor$morphism)
    (= reference-arity (SET.FTN$composition [reference set.pr$tuple-arity]))
```

The set function composition of the signature function with the reference arity function defines the

- edge-arity function $\#_G = \text{edge-arity}(G) = \text{sign}(G) \cdot \text{refer-arity}(G)$.

- ```
(6) (SET.FTN$function edge-arity)
 (= (SET.FTN$source edge-arity) object)
 (= (SET.FTN$target edge-arity) set.mor$morphism)
 (forall (?g (object ?g))
 (= (edge-arity ?g)
 (set.ftn$composition [(signature ?g) (reference-arity ?g)])))
```

For convenience of practical reference, we introduce additional hypergraph terminology for the source and target components of these functions. The source of the signature function is called the *edge* set of  $G$  and denoted  $\text{edge}(G)$ . The second component of the reference pair is called the *node* set of  $G$  and denoted  $\text{node}(G)$ . The first component of the reference pair is called the *name* set of  $G$  and denoted  $\text{name}(G)$ . In summary, we provide terminology for the following sets:

- the *edge* set  $\text{edge}(G) = \text{src}(\text{sign}(G))$ ,
- the *node* set  $\text{node}(G) = \text{set2}(\text{refer}(G))$ , and
- the *name* set  $\text{name}(G) = \text{set1}(\text{refer}(G))$ .

This results in the following presentations of the reference, signature and arity functions:

- reference pair  $\text{refer}(G) = \langle \text{name}(G), \text{node}(G) \rangle$ ,
- signature function  $\partial_G = \text{sign}(G) : \text{edge}(G) \rightarrow \text{tuple}(\text{refer}(G))$ , and

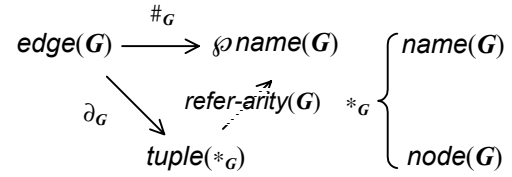


Figure 5: Hypergraph

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 19

May 8, 2003

– *edge-arity* function  $\#_G = \text{edge-arity}(G) : \text{edge}(G) \rightarrow \wp \text{name}(G)$ .

Names refer to nodes – a tuple in  $\text{tuple}(\text{refer}(G))$  maps a name to its referent node. We assume that each hypergraph has an adequate, possibly denumerable, set of names  $\text{name}(G)$ . This generalizes the usual case where sequences are used – the advantage for this generality is elimination of the dependency on sequences. One way to return to sequences is to define  $\text{name}(G) = \text{index} \times \text{node}(G)$  for some index set such as  $\text{index} = \text{Natno}$ .

The hyperedges in a hypergraph are many-sorted, which means that the component nodes in their signatures are named. For any hyperedge  $\varepsilon \in \text{edge}(G)$ , the *signature* of  $\varepsilon$  is a function  $\partial_G(\varepsilon) : \text{arity}(G)(\varepsilon) \rightarrow \text{node}(G)$ . The set of names  $\text{arity}(G)(\varepsilon)$  that reference the nodes in the *signature* of a hyperedge  $\varepsilon \in \text{edge}(G)$  is called its (edge) *arity*. The arity of hyperedges can overlap. Hence,  $\partial_G(\varepsilon)$  is a tuple of nodes with indexing set  $\text{arity}(G)(\varepsilon) \subseteq \text{name}(G)$ .

An external arity form for a hyperedge is  $(\varepsilon, x_1, x_2, \dots, x_n)$ , where  $\text{arity}(G)(\varepsilon) = \{x_1, x_2, \dots, x_n\}$ . An external signature form for a hyperedge is  $(\varepsilon, x_1 : v_1, x_2 : v_2, \dots, x_n : v_n)$ , where  $\partial_G(\varepsilon)(x_k) = v_k$  for all  $1 < k < n$ .

```
(7) (SET.FTN$function edge)
 (= (SET.FTN$source edge) object)
 (= (SET.FTN$target edge) set.obj$object)
 (= edge (SET.FTN$composition [signature set.mor$source]))

(8) (SET.FTN$function node)
 (= (SET.FTN$source node) object)
 (= (SET.FTN$target node) set.obj$object)
 (= node (SET.FTN$composition [reference set.pr$set2]))

(9) (SET.FTN$function name)
 (= (SET.FTN$source name) object)
 (= (SET.FTN$target name) set.obj$object)
 (= name (SET.FTN$composition [reference set.pr$set1]))
```

The composition of the reference operator and the hypergraph operator (in the semipair namespace) is called the *tuple* operator:  $\text{tuple} = \text{refer} \cdot \text{hgph}$ . The tuple operator is like a closer operator.

- (1) For any hypergraph  $G$ ,  $\text{tuple}(G)$  is “above”  $G$ , in the sense that there is a hypergraph morphism from  $G$  to  $\text{tuple}(G)$ , namely  $\eta_G : G \rightarrow \text{tuple}(G)$ ;
- (2) The tuple operator is idempotent:  $\text{tuple} \cdot \text{tuple} = \text{tuple}$ .

```
(10) (SET.FTN$function tuple)
 (= (SET.FTN$source eta) hgph.obj$object)
 (= (SET.FTN$target eta) hgph.obj$object)
 (= tuple (SET.FTN$composition [reference set.pr$hypergraph]))

(11) (forall (?g (hgph.obj$object ?g))
 (and (= (source (eta ?g)) ?g)
 (= (target (eta ?g)) (tuple ?g))))

(12) (= (SET.FTN$composition [tuple tuple]) tuple)
```

## Case and Spangraphs

In studies of natural language, the syntactic position of the argument of a verb correlates significantly with the meaning of the argument as a participant in either the action of the verb or the state indicated by the verb. These semantic relations between the verb and its arguments are called thematic roles or case relations. In frame systems, they are represented by slots in the frame for the corresponding verb. In the IFF verbs correspond to relations and hyperedges.

Any hypergraph  $G = \langle \text{refer}(G), \text{sign}(G) \rangle$  has a coproduct *arity*

$$\#_G = \text{arity}(G) : \text{edge}(G) \rightarrow \wp \text{name}(G).$$

```
(13) (SET.FTN$function arity)
 (= (SET.FTN$source arity) object)
 (= (SET.FTN$target arity) set.col.art$arity)
 (= (SET.FTN$composition [arity set.col.art$index]) edge)
 (= (SET.FTN$composition [arity set.col.art$base]) name)
 (= (SET.FTN$composition [arity set.col.art$function]) edge-arity)
```

Any hypergraph  $G$  has a set of *thematic roles* or *case relations* or *cases*

$$\begin{aligned} \text{case}(G) &= \sum \text{arity}(G) \\ &= \sum_{\varepsilon \in \text{edge}(G)} \text{arity}(G)(\varepsilon) \\ &= \{(\varepsilon, x) \mid \varepsilon \in \text{edge}(G), x \in \text{arity}(G)(\varepsilon)\}, \end{aligned}$$

that is the coproduct of its arity. In terms of frames, elements of  $\text{case}(A)$  can be regarded as frame-slot pairs. The set  $\text{case}(A)$  is rather large; this large cardinality will be handled properly with hypergraphs by adding a denotation or indexing function. In addition, for any hypergraph  $G$  and any hyperedge  $\varepsilon \in \text{edge}(G)$  there is a case *injection* function:

$$\text{inj}(G)(\varepsilon) : \#_G(\varepsilon) = \text{arity}_G(\varepsilon) \rightarrow \text{case}(G)$$

defined by  $\text{inj}(G)(\varepsilon)(x) = (\varepsilon, x)$  for all hyperedges  $\varepsilon \in \text{edge}(G)$  and all names  $x \in \text{arity}(G)(\varepsilon)$ . Obviously, the injections are injective. They commute (Diagram 3) with projection and inclusion.

```
(14) (SET.FTN$function case)
 (= (SET.FTN$source case) object)
 (= (SET.FTN$target case) set.obj$object)
 (= case (SET.FTN$composition [arity set.col.art$coproduct]))
```

```
(15) (KIF$function injection)
 (= (KIF$source injection) object)
 (= (KIF$target injection) SET.FTN$function)
 (= injection (SET.FTN$composition [arity set.col.art$injection]))
```

Any hypergraph  $G$  defines *indication* and *projection* functions (Diagram 4) based on its arity:

$$\text{indic}(G) : \text{case}(G) \rightarrow \text{edge}(G),$$

$$\text{proj}(G) : \text{case}(G) \rightarrow \text{name}(G).$$

These are defined by

$$\text{indic}(G)(\varepsilon, x) = \varepsilon \text{ and } \text{proj}(G)(\varepsilon, x) = x$$

for all hyperedges  $\varepsilon \in \text{edge}(G)$  and all names  $x \in \text{arity}(G)(\varepsilon)$ .

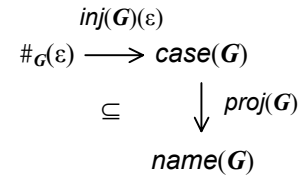


Diagram 3: Coproduct

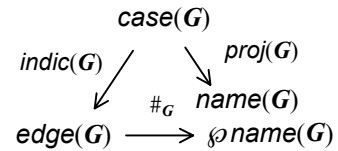


Diagram 4: Indication and Projection

```
(16) (SET.FTN$function indication)
 (= (SET.FTN$source indication) object)
 (= (SET.FTN$target indication) set.mor$morphism)
 (= indication (SET.FTN$composition [arity set.col.art$indication]))
```

```
(17) (SET.FTN$function projection)
 (= (SET.FTN$source projection) object)
 (= (SET.FTN$target projection) set.mor$morphism)
 (= projection (SET.FTN$composition [arity set.col.art$projection]))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 21

May 8, 2003

Any hypergraph  $G$  defines a morphism (Diagram 5) from its arity to the arity of its reference pair:

$$\text{arity-mor}(G) = \langle \text{sign}(G), \text{id} \rangle : \text{arity}(G) \rightarrow \text{arity}(\text{refer}(G)).$$

This arity morphism will be used to define a case function between the case set axiomatized here and the case set of the reference pair. Since both of these are coproducts of arities, this case function will be defined as the coproduct of the arity morphism. This case function is used to define comediation.

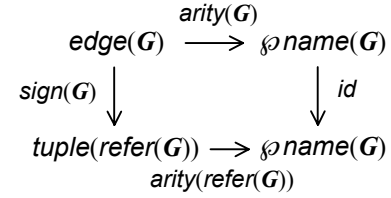


Diagram 5: Arity Morphism

```
(18) (SET.FTN$function arity-morphism)
 (= (SET.FTN$source arity-morphism) object)
 (= (SET.FTN$target arity-morphism) set.col.art.mor$arity-morphism)
 (= (SET.FTN$composition [arity-morphism set.col.art.mor$source]) arity)
 (= (SET.FTN$composition [arity-morphism set.col.art.mor$target])
 (SET.FTN$composition [refer set.pr$arity]))
 (= (SET.FTN$composition [arity-morphism set.col.art.mor$index]) signature)
 (= (SET.FTN$composition [arity-morphism set.col.art.mor$base])
 (SET.FTN$composition [(SET.FTN$composition [name set.$power]) set.mor$identity]))
```

Any hypergraph  $G$  has a function (Diagram 6)

$$\text{case-sign}(G) : \text{case}(G) \rightarrow \text{case}(\text{refer}(G)),$$

that satisfies the following constraints:

$$\text{case-sign}(G) \cdot \text{index}(\text{refer}(G)) = \text{index}(G) \cdot \text{sign}(G),$$

$$\text{case-sign}(G) \cdot \text{proj}(\text{refer}(G)) = \text{proj}(G).$$

Pointwise, this is defined by

$$\text{case-sign}(G)((\varepsilon, x)) = (\partial_G(\varepsilon), x)$$

for all hyperedges  $\varepsilon \in \text{edge}(G)$  and all names  $x \in \text{arity}(G)(\varepsilon)$ . Abstractly, this is defined to be the coproduct of the arity morphism of  $G$ .

It is the mediating function for the signature function  $\partial_G = \text{sign}(G) : \text{edge}(G) \rightarrow \text{tuple}(\text{refer}(G))$ , regarded as a coproduct cocone over the reference pair coproduct arity.

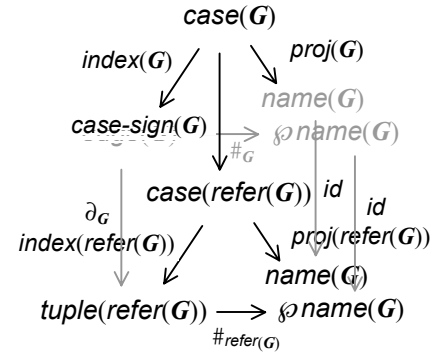


Diagram 6: Case-tuple function

```
(19) (SET.FTN$function case-signature)
 (= (SET.FTN$source case-signature) object)
 (= (SET.FTN$target case-signature) set.mor$morphism)
 (= (SET.FTN$composition [case-signature set.mor$source]) case)
 (= (SET.FTN$composition [case-signature set.mor$target])
 (SET.FTN$composition [reference set.pr$case]))
 (= case-signature
 (SET.FTN$composition [arity-morphism set.col.art.mor$coproduct]))
```

Any hypergraph  $G$  defines a comediation function:

$$\tilde{*}_G = \text{comed}(G) : \text{case}(G) \rightarrow \text{node}(G).$$

This function is the slot-filler function for frames. Pointwise, it is defined by

$$\text{comed}(G)((\varepsilon, x)) = \partial_G(\varepsilon)(x)$$

for all hyperedges  $\varepsilon \in \text{edge}(G)$  and all names  $x \in \text{arity}(G)(\varepsilon)$ . Abstractly, it is the comediation – hence the name – of the coproduct cotuple (cocone) consisting of the collection of edge signatures regarded as functions,

$$\{\partial_G(\varepsilon) : \text{arity}(G)(\varepsilon) \rightarrow \text{node}(G) \mid \varepsilon \in \text{edge}(G)\}.$$

However, a simpler definition is at hand: the hypergraph comediation can be defined (Diagram 7) as the composition of the case-signature function and the reference pair comediation. The comediation function commutes with the injection and signature functions of any hyperedge. If the indexed names in  $\text{case}(G)$  are viewed as roles (prehensions), the comediation is a reference function from roles to objects (actualities).

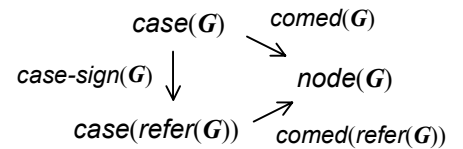
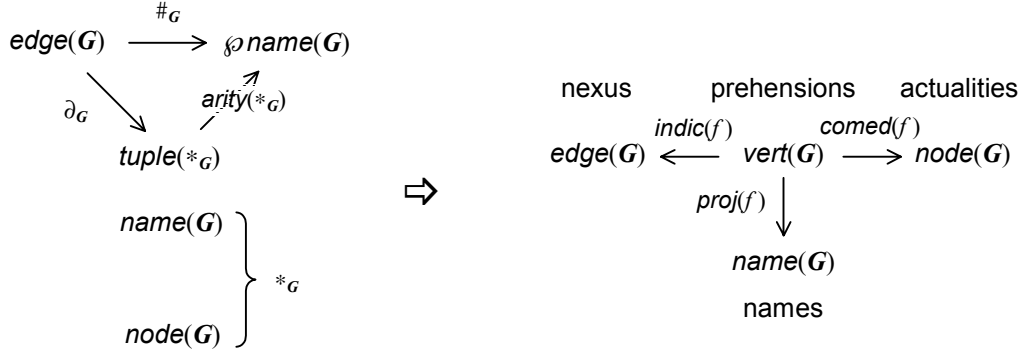


Diagram 7: Comediator

```
(20) (SET.FTN$function comediation)
```

```
(= (SET.FTN$source comediation) object)
(= (SET.FTN$target comediation) set.mor$morphism)
(= (SET.FTN$composition [comediation set.mor$source]) case)
(= (SET.FTN$composition [comediation set.mor$source]) node)
(forall (?g (object ?g))
 (= (comediation ?g)
 (set.ftn$composition [(case-signature ?g) (set.pr$comediation (reference ?g))])))
```



**Figure 6: From hypergraph to spangraph**

Associated with any hypergraph  $G$  is a spangraph (Figure 6)

$sgph(G)$   
 $= \langle vert(sgph(G)), \langle sgph(G), edge(sgph(G)), \rangle_{sgph(G)}, node(sgph(G)), \downarrow_{sgph(G)}, name(sgph(G)) \rangle,$

whose vertex set is the case set of  $G$ , whose source (nexus), target (actualities) and label (prehensions) functions are

$\langle_{sgph(G)} = indic(G) : case(G) \rightarrow edge(G),$   
 $\rangle_{sgph(G)} = comed(G) : case(G) \rightarrow node(G),$  and  
 $\downarrow_{sgph(G)} = proj(G) : case(G) \rightarrow name(G).$

```
(21) (SET.FTN$function spangraph)
(= (SET.FTN$source spangraph) object)
(= (SET.FTN$target spangraph) sgph.obj$object)
(= (SET.FTN$composition [spangraph sgph.obj$vertex]) case)
(= (SET.FTN$composition [spangraph sgph.obj$edge]) edge)
(= (SET.FTN$composition [spangraph sgph.obj$node]) node)
(= (SET.FTN$composition [spangraph sgph.obj$name]) name)
(= (SET.FTN$composition [spangraph sgph.obj$indication]) indication)
(= (SET.FTN$composition [spangraph sgph.obj$comediation]) comediation)
(= (SET.FTN$composition [spangraph sgph.obj$projection]) projection)
```

## Hypergraph Fibers

### hgph.fbr.obj

We define the fiber for the name class function, so that  $\text{hgph}(X) = \text{name}^{-1}(X) \subseteq \text{hgph} = \text{obj}(\text{Hypergraph})$  is the class of hypergraphs whose name set is  $X$ . Fibers are used when modeling the name bijection is onerous. They are needed when axiomatizing limits and colimits. All of the terminology for hypergraphs is used in the name fiber namespace.

```
(1) (SET.FTN$function object)
 (SET.FTN$function hypergraph)
 (= hypergraph object)
 (SET.FTN$source object) set.obj$object)
 (SET.FTN$target object) (SET$power hgph.obj$object))
 (= object (SET.FTN$fiber hgph.obj$name))
```

For any set  $X$  representing a fixed set of names, there is an inclusion fiber class function

$$\text{incl}(X) : \text{hgph}(X) \rightarrow \text{hgph}$$

that injects the hypergraph fiber class into the hypergraph class.

```
(2) (KIF$function inclusion)
 (= (KIF$source inclusion) set.obj$object)
 (= (KIF$target inclusion) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (inclusion ?x)) (object ?x))
 (= (SET.FTN$target (inclusion ?x)) hgph.obj$object)
 (= (inclusion ?x) (SET.FTN$inclusion [(object ?x) hgph.obj$object]))))
```

For any set  $X$  representing a fixed set of names, there are the following fiber class functions.

$$\text{refer}(X) = \text{incl}(X) \cdot \text{refer} : \text{hgph}(X) \rightarrow \text{hgph} \rightarrow \text{pr}$$

$$\text{sign}(X) = \text{incl}(X) \cdot \text{sign} : \text{hgph}(X) \rightarrow \text{hgph} \rightarrow \text{ftn}$$

$$\text{edge-arity}(X) = \text{incl}(X) \cdot \text{edge-arity} : \text{hgph}(X) \rightarrow \text{hgph} \rightarrow \text{ftn}$$

```
(3) (KIF$function reference)
 (= (KIF$source reference) set.obj$object)
 (= (KIF$target reference) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (reference ?x)) (object ?x))
 (= (SET.FTN$target (reference ?x)) set.pr$pair)
 (= (SET.FTN$composition [(reference ?x) set.pr$set1])
 ((set.ftn$constant [(object ?x) set.obj$object]) ?x))
 (SET.FTN$restriction (reference ?x) hgph.obj$reference)))

(4) (KIF$function signature)
 (= (KIF$source signature) set.obj$object)
 (= (KIF$target signature) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (signature ?x)) (object ?x))
 (= (SET.FTN$target (signature ?x)) set.mor$morphism)
 (SET.FTN$restriction (signature ?x) hgph.obj$signature)))

(5) (forall (?x (set.obj$object ?x))
 (= (SET.FTN$composition [(signature ?x) set.mor$target])
 (SET.FTN$composition [(reference ?x) set.pr$tuple])))

(6) (KIF$function edge-arity)
 (= (KIF$source edge-arity) set.obj$object)
 (= (KIF$target edge-arity) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-arity ?x)) (object ?x))
 (= (SET.FTN$target (edge-arity ?x)) set.mor$morphism)
 (= (SET.FTN$composition [(edge-arity ?x) set.mor$target])
 ((set.ftn$constant [(object ?x) set.obj$object]) (set$power ?x)))
 (SET.FTN$restriction (edge-arity ?x) hgph$edge-arity)))

(7) (forall (?x (set.obj$object ?x))
 (= (SET.FTN$composition [(edge-arity ?x) set.mor$target])
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 24

May 8, 2003

```
(SET.FTN$composition [(SET.FTN$composition
 [(reference ?x) set.pr$set1]) set$power]))
```

For any set  $X$  representing a fixed set of names, we define fiber terminology for the edge, node and name functions.

$$\text{edge}(X) = \text{incl}(X) \cdot \text{edge} : \text{sgph}(X) \rightarrow \text{sgph} \rightarrow \text{set}$$

$$\text{node}(X) = \text{incl}(X) \cdot \text{node} : \text{sgph}(X) \rightarrow \text{sgph} \rightarrow \text{set}$$

$$\text{name}(X) = \text{incl}(X) \cdot \text{name} : \text{sgph}(X) \rightarrow \text{sgph} \rightarrow \text{set}$$

```
(8) (KIF$function edge)
 (= (KIF$source edge) set.obj$object)
 (= (KIF$target edge) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge ?x)) (object ?x))
 (= (SET.FTN$target (edge ?x)) set.obj$object)
 (SET.FTN$restriction (edge ?x) hgph.obj$edge)))

(9) (KIF$function node)
 (= (KIF$source node) set.obj$object)
 (= (KIF$target node) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node ?x)) (object ?x))
 (= (SET.FTN$target (node ?x)) set.obj$object)
 (SET.FTN$restriction (node ?x) hgph.obj$node)))

(10) (KIF$function name)
 (= (KIF$source name) set.obj$object)
 (= (KIF$target name) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (name ?x)) (object ?x))
 (= (SET.FTN$target (name ?x)) set.obj$object)
 (SET.FTN$restriction (name ?x) hgph.obj$name)
 (= (name ?x)
 (SET.FTN$composition [(inclusion ?x) hgph.obj$name]))
 (= (name ?x)
 ((SET.FTN$constant [(object ?x) set.obj$object]) ?x))))
```

For any set  $X$  representing a fixed set of names, there is a fiber class function

$$\text{sgph}(X) : \text{hgph}(X) \rightarrow \text{sgph}(X)$$

that restricts the spangraph class function to hypergraph and spangraph fibers.

```
(11) (KIF$function spangraph)
 (= (KIF$source spangraph) set.obj$object)
 (= (KIF$target spangraph) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (spangraph ?x)) (object ?x))
 (= (SET.FTN$target (spangraph ?x)) (sgph.fbr.obj$object ?x))
 (SET.FTN$restriction (spangraph ?x) hgph.obj$spangraph)))
```

$$\text{refer}(G_X) \left\{ \begin{array}{ccc} X & \xrightarrow{h} & Y \\ & \text{ref}(\triangleright_h(G_X)) & \\ \text{node}(G_X) & \xrightarrow{id} & \text{node}(G_X) \end{array} \right\} \text{refer}(h^+(G_X))$$

$$\begin{array}{ccc} \text{edge}(G_X) & \xrightarrow{id} & \text{edge}(G_X) \\ \text{sign}(G_X) \downarrow & \text{sign}(\triangleright_h(G_X)) & \downarrow \text{sign}(h^+(G_X)) \\ \text{tuple}(\text{refer}(G_X)) & \xrightarrow{\text{tuple}(\text{refer}(\triangleright_h(G_X)))} & \text{tuple}(\text{refer}(h^+(G_X))) \end{array}$$

Diagram 8: Direct Image and Direct Connection



# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 25

May 8, 2003

For any set bijection  $h : X \rightarrow Y$  the *direct image*  $h^+ : \text{hgph}(X) \rightarrow \text{hgph}(Y)$  along  $h$  is a fiber operator that maps  $X$ -hypergraphs to  $Y$ -hypergraphs. For any hypergraph  $G_X$  with name set  $X$ , the direct image  $h^+(G_X)$  has the same edge and node sets as  $G_X$ , but has name set  $Y$ .

There is a *direct connection* hypergraph morphism  $\triangleright_h(G_X) : G_X \rightarrow h^+(G_X)$ .

```
(12) (KIF$function direct-image)
 (= (KIF$source direct-image) set.mor$bijection)
 (= (KIF$target direct-image) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (direct-image ?h)) (object (set.mor$source ?h)))
 (= (SET.FTN$target (direct-image ?h)) (object (set.mor$target ?h)))
 (= (SET.FTN$composition [(direct-image ?h) (edge (set.mor$target ?h))])
 (edge (set.mor$source ?h)))
 (= (SET.FTN$composition [(direct-image ?h) (node (set.mor$target ?h))])
 (node (set.mor$source ?h)))
 (= (SET.FTN$composition [(direct-image ?h) (name (set.mor$target ?h))])
 ((set.ftn$constant
 [(object (set.mor$source ?h)) set.obj$object]) (set.mor$target ?h)))
 (= (SET.FTN$composition [(direct-image ?h) (reference (set.mor$target ?h))])
 (SET.FTN$composition
 [(node (set.mor$source ?h)) (set.pr$couple (set.mor$target ?h))]))
 (forall (?g ((object (set.mor$source ?h)) ?g)
 (and (= (signature ((direct-image ?h) ?g))
 (set.ftn$composition
 [(signature ?g)
 (set.sqtt$tuple (hgph.mor$reference (direct-connection ?h)))])))
 (= (edge-arity ((direct-image ?h) ?g))
 (set.ftn$composition [(edge-arity ?g) (set.ftn$power ?h)]))))))

(13) (KIF$function direct-connection)
 (= (KIF$source direct-connection) set.mor$bijection)
 (= (KIF$target direct-connection) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (direct-connection ?h)) (object (set.mor$source ?h)))
 (= (SET.FTN$target (direct-connection ?h)) (hgph.fbr.mor$morphism ?h))
 (= (SET.FTN$composition [(direct-connection ?h) (hgph.fbr.mor$source ?h)])
 (SET.FTN$identity (object (set.mor$source ?h))))
 (= (SET.FTN$composition [(direct-connection ?h) (hgph.fbr.mor$target ?h)])
 (direct-image ?h))
 (= (SET.FTN$composition [(direct-connection ?h) (hgph.fbr.mor$edge ?h)])
 (SET.FTN$composition [(edge (set.mor$source ?h)) set.mor$identity]))
 (= (SET.FTN$composition [(direct-connection ?h) (hgph.fbr.mor$node ?h)])
 (SET.FTN$composition [(node (set.mor$source ?h)) set.mor$identity]))
 (= (SET.FTN$composition [(direct-connection ?h) (hgph.fbr.mor$name ?h)])
 ((set.ftn$constant [(object (set.mor$source ?h)) set.mor$bijection]) ?h))))
```

The direct image of a composition is the composition of the direct image operators of the components:

$$(f \cdot g)^+ = f^+ \cdot g^+ \text{ for any two composable bijections } f : X \rightarrow Y \text{ and } g : Y \rightarrow Z.$$

The direct image of an identity is the identity operator:

$$(id_X)^+ = id \text{ for any set } X.$$

```
(14) (forall (?f (set.mor$bijection ?f)
 ?g (set.mor$bijection ?g)
 (set.mor$composable ?f ?g))
 (= (direct-image (set.ftn$composition [?f ?g]))
 (SET.FTN$composition [(direct-image ?f) (direct-image ?g)])))

(15) (forall (?x (set.obj$object ?x))
 (= (direct-image (set.mor$identity ?x))
 (SET.FTN$identity (object ?x))))
```

The direct connection of a composition is the composition of the direct connection component operators:

$$\begin{aligned} \triangleright_{(f \cdot g)}(G_X) : G_X &\rightarrow (f \cdot g)^+(G_X) = (f^+ \cdot g^+)(G_X) = g^+(f^+(G_X)) \\ &= \triangleright_f(G_X) \cdot_{[f, g]} \triangleright_g(f^+(G_X)) : G_X \rightarrow f^+(G_X) \rightarrow g^+(f^+(G_X)) \end{aligned}$$

for any two composable bijections  $f : X \rightarrow Y$  and  $g : Y \rightarrow Z$ .

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 26

May 8, 2003

The direct connection of an identity is the identity operator:

$$\triangleright_{id_X}(G_X) = id(G_X) \text{ for any set } X.$$

In particular, for the a bijection  $h : X \rightarrow Y$  and its inverse image  $h^{-1} : Y \rightarrow X$ , we get

$$\triangleright_h(G_X) \cdot_{[h, h^{-1}]} \triangleright_{h^{-1}}(h^+(G_X)) = \triangleright_{(h \cdot h^{-1})}(G_X) = \triangleright_{id_X}(G_X) = id(G_X)$$

so that  $\triangleright_{h^{-1}}(h^+(G_X))$  is the inverse of  $\triangleright_h(G_X)$ .

```
(16) (forall (?f (set.mor$bijection ?f) ?g (set.mor$bijection ?g) (set.mor$composable ?f ?g)
 ?G ((object (set.mor$source ?f)) ?G))
 (= ((direct-connection (set.ftn$composition [?f ?g])) ?G)
 ((hgph.fbr.mor$composition [?f ?g])
 [((direct-connection ?f) ?G)
 ((direct-connection ?g) ((direct-image ?f) ?G))])))
```

```
(17) (forall (?x (set.obj$object ?x)
 ?G ((object ?x) ?G))
 (= ((direct-connection (set.mor$identity ?x)) ?G)
 ((hgph.fbr.mor$identity ?x) ?G)))
```

For any set bijection  $h : X \rightarrow Y$  the *inverse image*  $h^{-1} : \text{hgph}(Y) \rightarrow \text{hgph}(X)$  along  $h$  is a fiber operator that maps  $Y$ -hypergraphs to  $X$ -hypergraphs. The inverse image operator is defined to be the direct image along the inverse  $h^{-1} : Y \rightarrow X$ .

There is an *inverse connection* hypergraph morphism  $\triangleleft_h(G_Y) : G_Y \rightarrow h^{-1}(G_Y)$ , for any  $Y$ -hypergraph  $G_Y$ , so that  $\triangleleft_h(h^+(G_X)) = \triangleright_{h^{-1}}(h^+(G_X)) : h^+(G_X) \rightarrow h^{-1}(h^+(G_X)) = G_X$  for any  $X$ -hypergraph  $G_X$  is the inverse of the direct connection  $\triangleright_h(G_X) : G_X \rightarrow h^+(G_X)$ . Inverse connection is defined to be the direct connection along the inverse function  $\triangleleft_h(G_Y) = \triangleright_{h^{-1}}(G_Y) : G_Y \rightarrow (h^{-1})^+(G_Y)$ .

```
(18) (KIF$function inverse-image)
 (= (KIF$source inverse-image) set.mor$bijection)
 (= (KIF$target inverse-image) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (inverse-image ?h)) (object (set.mor$target ?h)))
 (= (SET.FTN$target (inverse-image ?h)) (object (set.mor$source ?h)))
 (= (inverse-image ?h) (direct-image (set.ftn$inverse ?h)))))

(19) (KIF$function inverse-connection)
 (= (KIF$source inverse-connection) set.mor$bijection)
 (= (KIF$target inverse-connection) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (inverse-connection ?h))
 (object (set.mor$target ?h)))
 (= (SET.FTN$target (inverse-connection ?h))
 (hgph.fbr.mor$morphism (set.ftn$inverse ?h)))
 (= (SET.FTN$composition [(inverse-connection ?h) (hgph.fbr.mor$target ?h)])
 (SET.FTN$identity (object (set.mor$target ?h))))
 (= (SET.FTN$composition [(inverse-connection ?h) (hgph.fbr.mor$source ?h)])
 (inverse-image ?h))
 (= (inverse-connection ?h) (direct-connection (set.ftn$inverse ?h)))))
```

The direct and inverse image operators are inverse to each other:  $id_{\text{hgph}(Y)} = h^+ \cdot h^{-1}$  and  $id_{\text{hgph}(X)} = h^{-1} \cdot h^+$  for any set bijection  $h : Y \rightarrow X$ .

The direct and inverse connection operators are inverses of each other.

```
(20) (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$composition [(direct-image ?h) (inverse-image ?h)])
 (SET.FTN$identity (object (set.mor$source ?h))))
 (= (SET.FTN$composition [(inverse-image ?h) (direct-image ?h)])
 (SET.FTN$identity (object (set.mor$target ?h)))))

(21) (forall (?h (set.mor$bijection ?h)
 ?G ((object (set.mor$source ?h)) ?G))
 (= ((hgph.fbr.mor$composition [?h (set.ftn$inverse ?h)])
 [((direct-connection ?h) ?G)
 ((inverse-connection ?h) ((direct-image ?h) ?G))])
 ((hgph.fbr.mor$identity (set.mor$source ?h)) ?G)))
```

## Hypergraph Morphisms

hgph.mor

Hypergraphs are connected by and comparable with hypergraph morphisms. This section discusses hypergraph morphisms. First, we give a concise mathematical definition, and then we discuss and formalize the various parts of this definition.

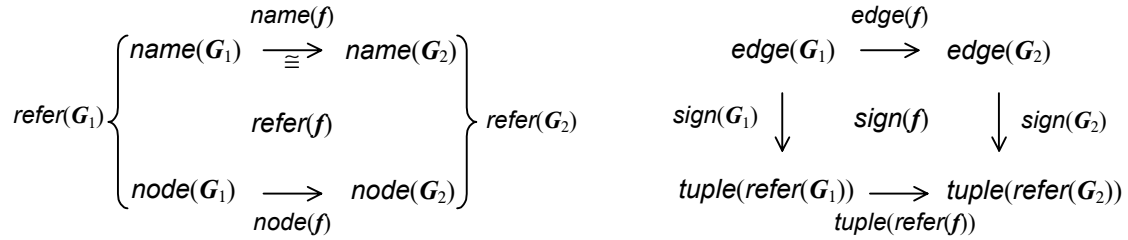


Figure 7: Hypergraph Morphism

A *hypergraph morphism*  $f = \langle \text{refer}(f), \text{sign}(f) \rangle : G_1 \rightarrow G_2$  from hypergraph  $G_1$  to hypergraph  $G_2$  (Figure 7) is a two dimensional construction consisting of

- a *reference* semiquartet  $*_f = \text{refer}(f)$  and
- a *signature* quartet  $\text{sign}(f)$ ,

where the tuple function of the reference semiquartet is the vertical target of the signature quartet

$$\text{tuple}(\text{refer}(f)) = \text{vert-tgt}(\text{sign}(f)).$$

```
(1) (SET$class morphism)

(2) (SET.FTN$function source)
 (= (SET.FTN$source source) morphism)
 (= (SET.FTN$target source) hgph.obj$object)

(3) (SET.FTN$function target)
 (= (SET.FTN$source target) morphism)
 (= (SET.FTN$target target) hgph.obj$object)

(4) (SET.FTN$function reference)
 (= (SET.FTN$source reference) morphism)
 (= (SET.FTN$target reference) set.sqtt$semiquartet)
 (= (SET.FTN$composition [reference set.sqtt$source])
 (SET.FTN$composition [source hgph.obj$reference]))
 (= (SET.FTN$composition [reference set.sqtt$target])
 (SET.FTN$composition [target hgph.obj$reference]))

(5) (SET.FTN$function signature)
 (= (SET.FTN$source signature) morphism)
 (= (SET.FTN$target signature) set.qtt$quartet)
 (= (SET.FTN$composition [signature set.qtt$horizontal-source])
 (SET.FTN$composition [source hgph.obj$signature]))
 (= (SET.FTN$composition [signature set.qtt$horizontal-target])
 (SET.FTN$composition [target hgph.obj$signature]))

(6) (= (SET.FTN$composition [signature set.qtt$vertical-target])
 (SET.FTN$composition [reference set.sqtt$tuple]))
```

$$\begin{array}{ccc}
 & \text{tuple}(\text{refer}(f)) & \\
 \text{tuple}(\text{refer}(G_1)) & \longrightarrow & \text{tuple}(\text{refer}(G_2)) \\
 \text{refer-arity}(G_1) \downarrow & \text{refer-arity}(f) = & \downarrow \text{refer-arity}(G_2) \\
 & \text{tuple-arity}(\text{refer}(f)) & \\
 \wp \text{name}(G_1) & \longrightarrow & \wp \text{name}(G_2) \\
 & \wp \text{name}(f) &
 \end{array}$$

**Figure 8: Reference-arity quartet**

For convenience of theoretical presentation, we introduce additional hypergraph terminology for the composition between the reference function to semiquartets and the tuple arity function from semiquartets to quartets. Associated with a hypergraph morphism  $f: G_1 \rightarrow G_2$  is a *reference-arity* quartet  $\text{refer-arity}(f)$  (Figure 8).

```

(7) (SET.FTN$function reference-arity)
 (= (SET.FTN$source reference-arity) morphism)
 (= (SET.FTN$target reference-arity) set.qtt$quartet)
 (= reference-arity (SET.FTN$composition [reference set.sqtt$tuple-arity]))

```

$$\begin{array}{ccc}
 & \text{edge}(f) & \\
 \text{edge}(G_1) & \longrightarrow & \text{edge}(G_2) \\
 \text{edge-arity}(G_1) \downarrow & \text{edge-arity}(f) & \downarrow \text{edge-arity}(G_2) \\
 \wp \text{name}(G_1) & \longrightarrow & \wp \text{name}(G_2) \\
 & \wp \text{name}(f) &
 \end{array}$$

**Figure 9: Edge-arity quartet**

The vertical composition of the signature quartet with the reference-arity quartet defines the *edge arity* quartet  $\#_f = \text{edge-arity}(f) = \text{sign}(f) \cdot \text{refer-arity}(f)$  (Figure 9).

```

(8) (SET.FTN$function edge-arity)
 (= (SET.FTN$source edge-arity) morphism)
 (= (SET.FTN$target edge-arity) set.qtt$quartet)
 (forall (?f (morphism ?f))
 (= (edge-arity ?f)
 (set.qtt$vertical-composition [(signature ?f) (reference-arity ?f)])))

```

For convenience of reference, we introduce additional hypergraph terminology for the vertical source (function<sub>1</sub>) and target components (function<sub>2</sub>) of these (semi)quartets. The vertical source of the signature quartet is called the *edge* function of  $f$  and denoted  $\text{edge}(f)$ . The second function of the reference semiquartet is called the *node* function of  $f$  and denoted  $\text{node}(f)$ . The first function (bijection) of the reference semiquartet is called the *name* bijection of  $f$  and denoted  $\text{name}(f)$ . In summary, a hypergraph morphism has the following component functions:

- the *edge* function  $\text{edge}(f) = \text{vert-src}(\text{sign}(f)) : \text{edge}(G_1) \rightarrow \text{edge}(G_2)$ ,
- the *node* function  $\text{node}(f) = \text{fn2}(\text{refer}(f)) : \text{node}(G_1) \rightarrow \text{node}(G_2)$ , and
- the *name* bijection  $\text{name}(f) = \text{fn1}(\text{refer}(f)) : \text{name}(G_1) \rightarrow \text{name}(G_2)$ .

This results in the following presentations of the reference, signature and arity (semi)quartets:

- the *reference* semiquartet  $\text{refer}(f) = \langle \text{name}(f), \text{node}(f) \rangle : \text{refer}(\text{src}(f)) \rightarrow \text{refer}(\text{tgt}(f))$ ,
- the *signature* quartet  $\text{sign}(f) = \langle \text{edge}(f), \text{tuple}(\text{refer}(f)) \rangle : \text{sign}(\text{src}(f)) \rightarrow \text{sign}(\text{tgt}(f))$ , and
- the *edge-arity* quartet

$$\text{edge-arity}(f) = \langle \text{edge}(f), \wp \text{name}(f) \rangle : \text{edge-arity}(\text{src}(f)) \rightarrow \text{edge-arity}(\text{tgt}(f)).$$

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 29

May 8, 2003

Therefore, a hypergraph morphism can be displayed as in Figure 7.

The edge-arity quartet asserts the preservation of hyperedge arity:  $\text{edge}(f) \cdot \#_{G_2} = \#_{G_1} \cdot \wp \text{name}(f)$ . This means that the  $G_2$ -arity of the image of any hyperedge  $\varepsilon \in \text{edge}(G_1)$  is the direct-image of the  $G_1$ -arity of the hyperedge:  $\#_{G_2}(\text{edge}(f)(\varepsilon)) = \wp \text{name}(f)(\#_{G_1}(\varepsilon))$ ; or symbolically,

if  $(\varepsilon, x_1, x_2, \dots, x_n)$  then  $(\text{edge}(f)(\varepsilon), \text{name}(f)(x_1), \text{name}(f)(x_2), \dots, \text{name}(f)(x_n))$ .

The signature quartet asserts the preservation of hyperedge signature:  $\text{edge}(f) \cdot \partial_{G_2} = \partial_{G_1} \cdot \text{tuple}(\text{refer}(f))$ . This means that the  $G_2$ -signature of the image of any hyperedge  $\varepsilon \in \text{edge}(G_1)$  is the image of the  $G_1$ -signature of the hyperedge:  $\partial_{G_2}(\text{edge}(f)(\varepsilon)) = \text{tuple}(\text{refer}(f))(\partial_{G_1}(\varepsilon))$ ; or,

if  $(\varepsilon, x_1 : v_1, x_2 : v_2, \dots, x_n : v_n)$

then  $(\text{edge}(f)(\varepsilon), \text{name}(f)(x_1) : \text{node}(f)(v_1), \dots, \text{name}(f)(x_n) : \text{node}(f)(v_n))$ .

Since the tuple function  $\text{tuple}(\text{refer}(f))$  preserves arity up to isomorphism, the edge function  $\text{edge}(f)$  also does this.

```
(9) (SET.FTN$function edge)
 (= (SET.FTN$source edge) morphism)
 (= (SET.FTN$target edge) set.mor$morphism)
 (= edge (SET.FTN$composition [signature set.qtt$vertical-source]))

(10) (SET.FTN$function node)
 (= (SET.FTN$source node) morphism)
 (= (SET.FTN$target node) set.mor$morphism)
 (= node (SET.FTN$composition [reference set.sqtt$function2]))

(11) (SET.FTN$function name)
 (= (SET.FTN$source name) morphism)
 (= (SET.FTN$target name) set.mor$bijection)
 (= name (SET.FTN$composition [reference set.sqtt$function1]))
```

Two hypergraph morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable hypergraph morphisms  $f_1 : G \rightarrow G'$  and  $f_2 : G' \rightarrow G''$  is defined in terms of the horizontal composition of their reference semiquartet and signature quartet.

```
(12) (SET.LIM.PBK$opspan composable-opspan)
 (= (SET.LIM.PBK$class1 composable-opspan) morphism)
 (= (SET.LIM.PBK$class2 composable-opspan) morphism)
 (= (SET.LIM.PBK$opvertex composable-opspan) hgph.obj$object)
 (= (SET.LIM.PBK$opfirst composable-opspan) target)
 (= (SET.LIM.PBK$opsecond composable-opspan) source)

(13) (REL$relation composable)
 (= (REL$class1 composable) morphism)
 (= (REL$class2 composable) morphism)
 (= (REL$extent composable) (SET.LIM.PBK$pullback composable-opspan))

(14) (SET.FTN$function composition)
 (= (SET.FTN$source composition) (SET.LIM.PBK$pullback composable-opspan))
 (= (SET.FTN$target composition) morphism)
 (forall (?f1 (morphism ?f1) ?f2 (morphism ?f2))
 (composable ?f1 ?f2))
 (and (= (source (composition [?f1 ?f2])) (source ?f1))
 (= (target (composition [?f1 ?f2])) (target ?f2))
 (= (reference (composition [?f1 ?f2]))
 (set.sqtt$composition [(reference ?f1) (reference ?f2)])))
 (= (signature (composition [?f1 ?f2]))
 (set.qtt$horizontal-composition [(signature ?f1) (signature ?f2)]))))
```

Composition satisfies the usual *associative law*.

```
(forall (?f1 (morphism ?f1)
 ?f2 (morphism ?f2)
 ?f3 (morphism ?f3))
 (composable ?f1 ?f2) (composable ?f2 ?f3))
 (= (composition [?f1 (composition [?f2 ?f3])])
 (composition [(composition [?f1 ?f2]) ?f3])))
```

For any hypergraph  $G$ , there is an *identity* hypergraph morphism.

```
(15) (SET.FTN$function identity)
 (= (SET.FTN$source identity) hgph.obj$object)
 (= (SET.FTN$target identity) morphism)
 (forall (?g (hgph.obj$object ?g))
 (and (= (source (identity ?g)) ?g)
 (= (target (identity ?g)) ?g)
 (= (reference (identity ?g))
 (set.sqtt$identity (hgph.obj$reference ?g)))
 (= (signature (identity ?g))
 (set.qtt$horizontal-identity (hgph.obj$signature ?g)))))
```

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

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

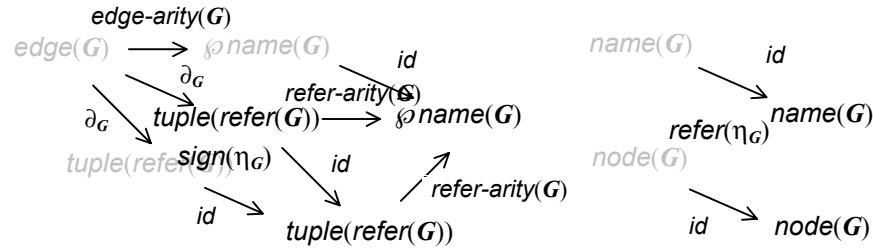


Diagram 9: Eta Hypergraph Morphism

For every hypergraph  $G$ , there is a special hypergraph morphism (Diagram 9)

$$\eta_G : G \rightarrow hgph(refer(G))$$

from  $G$  to the hypergraph of its reference pair, where

$$refer(\eta_G) = id_{refer(G)}, \text{ vert-src}(edge(\eta_G)) = sign(G),$$

$$\text{and } vert-tgt(edge(\eta_G)) = id_{tuple(refer(G))}.$$

```
(16) (SET.FTN$function eta)
 (= (SET.FTN$source eta) hgph.obj$object)
 (= (SET.FTN$target eta) morphism)
 (forall (?g (hgph.obj$object ?g))
 (and (= (source (eta ?g)) ?g)
 (= (target (eta ?g)) (set.pr$hypergraph (hgph.obj$reference ?g)))
 (= (reference (eta ?g))
 (set.sqtt$identity (hgph.obj$reference ?g)))
 (= (set.qtt$vertical-source (signature (eta ?g))
 (hgph.obj$signature ?g))
 (= (set.qtt$vertical-target (signature (eta ?g))
 (set.mor$identity (set.pr$tuple (hgph.obj$reference ?g))))))
```

The composition of the reference operator and the hypergraph operator (in the semiquartet namespace) is called the *tuple* operator:  $tuple = refer \cdot hgph$ . The tuple operator is idempotent:  $tuple \cdot tuple = tuple$ .

```
(17) (SET.FTN$function tuple)
 (= (SET.FTN$source eta) hgph.mor$morphism)
 (= (SET.FTN$target eta) hgph.mor$morphism)
 (= tuple (SET.FTN$composition [reference set.sqtt$hypergraph]))
```

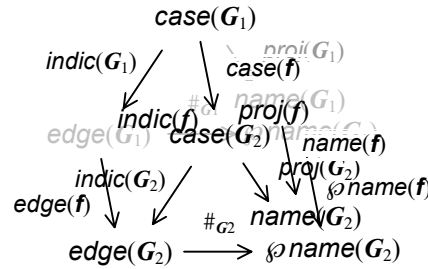
```
(18) (= (SET.FTN$composition [tuple tuple]) tuple)
```

## Case and Spangraph Morphisms

Any hypergraph morphism  $f = \langle \text{refer}(f), \text{sign}(f) \rangle : G_1 \rightarrow G_2$  defines a coproduct *arity morphism*, whose set quartet is edge arity

$$\text{arity}(f) = \langle \text{edge}(f), \wp \text{name}(f) \rangle : \text{arity}(G_1) \rightarrow \text{arity}(G_2).$$

```
(19) (SET.FTN$function arity-morphism)
 (= (SET.FTN$source arity-morphism) morphism)
 (= (SET.FTN$target arity-morphism) set.col.art.mor$arity-morphism)
 (forall (?f (morphism ?f))
 (and (= (set.col.art.mor$source (arity-morphism ?f)) (hgph.obj$arity (source ?f)))
 (= (set.col.art.mor$target (arity-morphism ?f)) (hgph.obj$arity (target ?f)))
 (= (set.col.art.mor$index (arity-morphism ?f)) (edge ?f))
 (= (set.col.art.mor$base (arity-morphism ?f)) (name ?f))))
```



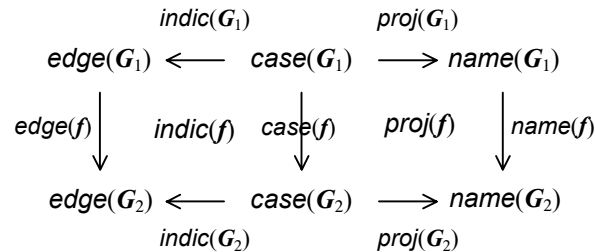
**Diagram 10: Case function = Coproduct of the Arity of a Hypergraph Morphism**

Any hypergraph morphism  $f = \langle \text{refer}(f), \text{sign}(f) \rangle : G_1 \rightarrow G_2$  defines a *case* function  $\text{case}(f) = \sum \text{arity}(f) : \text{case}(G_1) = \sum \text{arity}(G_1) \rightarrow \sum \text{arity}(G_2) = \text{case}(G_2)$ . The pointwise definition is:

$$\text{case}(f)((\varepsilon, x)) = (\text{edge}(f)(\varepsilon), \text{name}(f)(x))$$

for any hyperedge  $\varepsilon \in \text{edge}(G_1)$  and any name  $x \in \text{arity}(G_1)(\varepsilon)$ . This is well defined since  $f$  preserves hyperedge arity. The abstract definition (Diagram 10) is in terms of the coproduct of the arity morphism.

```
(20) (SET.FTN$function case)
 (= (SET.FTN$source case) morphism)
 (= (SET.FTN$target case) set.mor$morphism)
 (forall (?f (morphism ?f))
 (and (= (set.mor$source (case ?f)) (hgph.obj$case (source ?f)))
 (= (set.mor$target (case ?f)) (hgph.obj$case (target ?f)))
 (= (case ?f)
 (set.col.art.mor$coproduct (arity-morphism ?f)))))
```



**Diagram 11: Indication and Projection Quartets**

The case function is the vertical source for two quartets (Diagram 11): an *indication* quartet  $\text{indic}(f)$  and a *projection* quartet  $\text{proj}(f)$ .

- The commutativity  $\text{case}(f) \cdot \text{indic}(G_2) = \text{indic}(G_1) \cdot \text{edge}(f)$ , a property of the coproduct of arities (preservation of index), is obvious from the pointwise definition of the case function.

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 32

May 8, 2003

- The commutativity  $\text{case}(f) \cdot \text{proj}(G_2) = \text{proj}(G_1) \cdot \text{name}(f)$ , a property of the coproduct of arities (preservation of projection), is also obvious from the pointwise definition of the case function.

Even though the name function  $\text{name}(f) : \text{name}(G_1) \rightarrow \text{name}(G_2)$  and its power  $\wp \text{name}(f)$  are bijections, the case function  $\text{case}(f) : \text{case}(G_1) \rightarrow \text{case}(G_2)$  is not necessarily a bijection since the edge function  $\text{edge}(f) : \text{edge}(G_1) \rightarrow \text{edge}(G_2)$  need not be bijective. By the preservation of hyperedge arity, the index quartet is a fibration: for any hyperedge  $\varepsilon_1 \in \text{edge}(G_1)$  and any name  $x_2 \in \text{arity}(G_2)(\text{edge}(f)(\varepsilon_1))$  there is a name  $x_1 \in \text{arity}(G_1)(\varepsilon_1)$  such that  $\text{name}(f)(x_1) = x_2$ .

```
(21) (SET.FTN$function indication)
 (= (SET.FTN$source indication) morphism)
 (= (SET.FTN$target indication) set.qtt$fibration)
 (forall (?f (morphism ?f))
 (and (= (set.qtt$horizontal-source (indication ?f)) (hgph.obj$indication (source ?f)))
 (= (set.qtt$horizontal-target (indication ?f)) (hgph.obj$indication (target ?f)))
 (= (set.qtt$vertical-source (indication ?f)) (case ?f))
 (= (set.qtt$vertical-target (indication ?f)) (edge ?f))))

(22) (SET.FTN$function projection)
 (= (SET.FTN$source projection) morphism)
 (= (SET.FTN$target projection) set.qtt$quartet)
 (forall (?f (morphism ?f))
 (and (= (set.qtt$horizontal-source (projection ?f)) (hgph.obj$projection (source ?f)))
 (= (set.qtt$horizontal-target (projection ?f)) (hgph.obj$projection (target ?f)))
 (= (set.qtt$vertical-source (projection ?f)) (case ?f))
 (= (set.qtt$vertical-target (projection ?f)) (name ?f))))
```

$$\begin{array}{ccc}
 \tilde{*}_{G_1} = \text{comed}(G_1) & & \\
 \text{case}(G_1) \longrightarrow \text{node}(G_1) & & \\
 \text{case}(f) \downarrow & \text{comed}(f) & \downarrow \text{node}(f) \\
 \text{case}(G_2) \longrightarrow \text{node}(G_2) & & \\
 \tilde{*}_{G_2} = \text{comed}(G_2) & & 
 \end{array}$$

**Diagram 12: Comediator Quartet**

Any hypergraph morphism  $f = \langle \text{refer}(f), \text{sign}(f) \rangle : G_1 \rightarrow G_2$  defines a *comediation* quartet  $\tilde{*}_f = \text{comed}(f)$  (Diagram 12). The commutativity  $\text{case}(f) \cdot \text{comed}(G_2) = \text{comed}(G_1) \cdot \text{node}(f)$  holds by a property of the coproduct of arities (preservation of cotupling). Commutativity states that

$$\partial_{G_2}(\text{edge}(f)(\varepsilon))(\text{name}(f)(x)) = \text{node}(f)(\partial_{G_1}(\varepsilon)(x))$$

for any hyperedge  $\varepsilon \in \text{edge}(G_1)$  and any name  $x \in \text{name}(G_1)$ , which is true by preservation of hyperedge signature.

```
(23) (SET.FTN$function comediation)
 (= (SET.FTN$source comediation) morphism)
 (= (SET.FTN$target comediation) set.qtt$quartet)
 (forall (?f (morphism ?f))
 (and (= (set.qtt$horizontal-source (comediation ?f)) (hgph.obj$comediation (source ?f)))
 (= (set.qtt$horizontal-target (comediation ?f)) (hgph.obj$comediation (target ?f)))
 (= (set.qtt$vertical-source (comediation ?f)) (case ?f))
 (= (set.qtt$vertical-target (comediation ?f)) (node ?f)))))
```



# The IFF Namespace of Hypergraphs

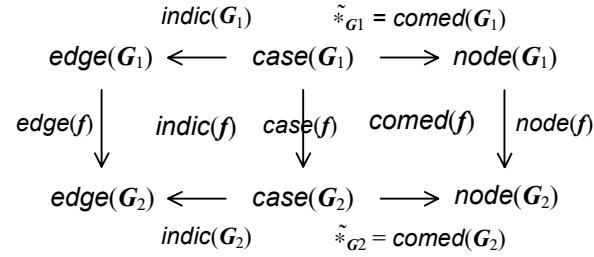
Robert E. Kent

Page 33

May 8, 2003

Associated with any hypergraph morphism  $f = \langle refer(f), sign(f) \rangle : G_1 \rightarrow G_2$  is a spangraph morphism

$sgph(f) = \langle \downarrow_{sgph(f)}(f), \uparrow_{sgph(f)}(f), \perp_{sgph(f)}(f) \rangle : sgph(G_1) \rightarrow sgph(G_2)$  (Diagram 13),



**Diagram 13: Spangraph Morphism  
of a Hypergraph Morphism**

whose vertex function is the case function, and whose three quartets (the 3<sup>rd</sup> is a fibration) are

$\downarrow_{sgph(f)}(f) = indic(f)$ ,

$\uparrow_{sgph(f)}(f) = comed(f)$ , and

$\perp_{sgph(f)}(f) = proj(f)$ .

```

(24) (SET.FTN$function spangraph)
 (= (SET.FTN$source spangraph) morphism)
 (= (SET.FTN$target spangraph) sgph.mor$morphism)
 (forall (?f (morphism ?f))
 (and (= (sgph.mor$source (spangraph ?f)) (hgph.obj$object (source ?f)))
 (= (sgph.mor$target (spangraph ?f)) (hgph.obj$object (target ?f)))
 (= (sgph.mor$indication (spangraph ?f)) (indication ?f))
 (= (sgph.mor$comediation (spangraph ?f)) (comediation ?f))
 (= (sgph.mor$projection (spangraph ?f)) (projection ?f))))

```

## Hypergraph Morphism Fibers

### hgph.fbr.mor

For any set bijection  $h : X \rightarrow Y$  representing a fixed function of names, we define the fiber for the name class function, so that  $\text{hgphmor}(h) = \text{name}^{-1}(h) \subseteq \text{hgphmor} = \text{mor}(\text{Hypergraph})$  is the class of hypergraph morphisms whose name function is  $h$ . Fibers over identities are used when modeling the name bijection is onerous. They are needed when axiomatizing limits and colimits. All of the terminology for hypergraph morphisms is used in the name fiber namespace.

```
(1) (SET.FTN$function morphism)
 (SET.FTN$source morphism) set.mor$bijection)
 (SET.FTN$target morphism) (SET$power hgph.mor$morphism))
 (= morphism (SET.FTN$fiber sgph.mor$name))
```

For any set bijection  $h : X \rightarrow Y$  representing a fixed function of names, there is an inclusion function

$$\text{incl}(h) : \text{hgphmor}(h) \rightarrow \text{hgphmor}$$

that injects the hypergraph morphism fiber class into the hypergraph morphism class.

```
(2) (KIF$function inclusion)
 (= (KIF$source inclusion) set.mor$bijection)
 (= (KIF$target inclusion) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (inclusion ?h)) (morphism ?h))
 (= (SET.FTN$target (inclusion ?h)) hgph.mor$morphism)
 (= (inclusion ?h) (SET.FTN$inclusion [(morphism ?h) hgph.mor$morphism]))))
```

For any set bijection  $h : X \rightarrow Y$  representing a fixed name function, there are source and target functions

$$\text{src}(h) : \text{hgphmor}(h) \rightarrow \text{hgph}(\text{src}(h))$$

$$\text{tgt}(h) : \text{hgphmor}(h) \rightarrow \text{hgph}(\text{tgt}(h)),$$

and reference, signature and edge arity functions

$$\text{refer}(h) = \text{incl}(h) \cdot \text{refer} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{sqtt}$$

$$\text{sign}(h) = \text{incl}(h) \cdot \text{sign} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{qtt}$$

$$\text{edge-arity}(h) = \text{incl}(h) \cdot \text{edge-arity} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{bqtt}.$$

```
(3) (KIF$function source)
 (= (KIF$source source) set.mor$bijection)
 (= (KIF$target source) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (source ?h)) (morphism ?h))
 (= (SET.FTN$target (source ?h)) (hgph.fbr.obj$object (set.mor$source ?h))
 (SET.FTN$restriction (source ?h) hgph.mor$source))))

(4) (KIF$function target)
 (= (KIF$source target) set.mor$bijection)
 (= (KIF$target target) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (target ?h)) (morphism ?h))
 (= (SET.FTN$target (target ?h)) (hgph.fbr.obj$object (set.mor$target ?h))
 (SET.FTN$restriction (target ?h) hgph.mor$target))))

(5) (KIF$function reference)
 (= (KIF$source reference) set.mor$bijection)
 (= (KIF$target reference) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (reference ?h)) (morphism ?h))
 (= (SET.FTN$target (reference ?h)) set.sqtt$semiquartet)
 (= (SET.FTN$composition [(reference ?h) set.sqtt$source]
 (SET.FTN$composition [(source ?h) (hgph.fbr.obj$reference (set.mor$source ?h))]))
 (= (SET.FTN$composition [(reference ?h) set.sqtt$target]
 (SET.FTN$composition [(target ?h) (hgph.fbr.obj$reference (set.mor$target ?h))]))
 (= (reference ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$reference]))
 (SET.FTN$restriction (reference ?h) hgph.mor$reference))))

(6) (KIF$function signature)
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 35

May 8, 2003

```
(= (KIF$source signature) set.mor$bijection)
(= (KIF$target signature) SET.FTN$function)
(forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (signature ?h)) (morphism ?h))
 (= (SET.FTN$target (signature ?h)) set.qtt$quartet)
 (= (SET.FTN$composition [(signature ?h) set.qtt$horizontal-source])
 (SET.FTN$composition [(source ?h) (hgph.fbr.obj$signature (set.mor$source ?h))]))
 (= (SET.FTN$composition [(signature ?h) set.qtt$horizontal-target])
 (SET.FTN$composition [(target ?h) (hgph.fbr.obj$signature (set.mor$target ?h))]))
 (= (signature ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$signature]))
 (SET.FTN$restriction (signature ?h) hgph.mor$signature)))

(7) (forall (?h (set.mor$bijection ?h))
 (= (SET.FTN$composition [(signature ?h) set.qtt$vertical-target])
 (SET.FTN$composition [(reference ?h) set.sqtt$tuple])))

(8) (KIF$function edge-arity)
(= (KIF$source edge-arity) set.mor$bijection)
(= (KIF$target edge-arity) SET.FTN$function)
(forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (edge-arity ?h)) (morphism ?h))
 (= (SET.FTN$target (edge-arity ?h)) set.qtt$quartet)
 (= (SET.FTN$composition [(edge-arity ?h) set.qtt$horizontal-source])
 (SET.FTN$composition [(source ?h) (hgph.fbr.obj$edge-arity (set.mor$source ?h))]))
 (= (SET.FTN$composition [(edge-arity ?h) set.qtt$horizontal-target])
 (SET.FTN$composition [(target ?h) (hgph.fbr.obj$edge-arity (set.mor$target ?h))]))
 (= (edge-arity ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$edge-arity]))
 (SET.FTN$restriction (edge-arity ?h) hgph.mor$edge-arity)))

(9) (forall (?h (set.mor$bijection ?h))
 (= (SET.FTN$composition [(edge-arity ?h) set.qtt$vertical-target])
 (SET.FTN$composition [(SET.FTN$composition
 [(reference ?h) set.sqtt$function1]) set$power]))))
```

We define terminology for the edge, node and name functions for hypergraph morphisms in the fiber of  $h$ :

$$\text{edge}(h) = \text{incl}(h) \cdot \text{edge} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{ftn}$$

$$\text{node}(h) = \text{incl}(h) \cdot \text{node} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{ftn}$$

$$\text{name}(h) = \text{incl}(h) \cdot \text{name} : \text{hgphmor}(h) \rightarrow \text{hgphmor} \rightarrow \text{ftn}.$$

```
(10) (KIF$function edge)
(= (KIF$source edge) set.mor$bijection)
(= (KIF$target edge) SET.FTN$function)
(forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (edge ?h)) (morphism ?h))
 (= (SET.FTN$target (edge ?h)) set.mor$morphism)
 (= (edge ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$edge]))
 (SET.FTN$restriction (edge ?h) hgph.mor$edge)))

(11) (KIF$function node)
(= (KIF$source node) set.mor$bijection)
(= (KIF$target node) SET.FTN$function)
(forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (node ?h)) (morphism ?h))
 (= (SET.FTN$target (node ?h)) set.mor$morphism)
 (= (node ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$node]))
 (SET.FTN$restriction (node ?h) hgph.mor$node)))

(12) (KIF$function name)
(= (KIF$source name) set.mor$bijection)
(= (KIF$target name) SET.FTN$function)
(forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (name ?h)) (morphism ?h))
 (= (SET.FTN$target (name ?h)) set.mor$bijection)
 (= (name ?h) (SET.FTN$composition [(inclusion ?h) hgph.mor$name]))
 (= (name ?h) ((SET.FTN$constant [(morphism ?h) set.mor$bijection]) ?h))
 (SET.FTN$restriction (name ?h) hgph.mor$name)))
```

For any set bijection  $h : X \rightarrow Y$  representing a fixed name function, there is a fiber class function

$$\text{sgph}(h) : \text{hgphmor}(h) \rightarrow \text{sgphmor}(h)$$

that restricts the spangraph morphism class function to hypergraph and spangraph morphism fibers.

```
(13) (KIF$function spangraph)
 (= (KIF$source spangraph) set.mor$bijection)
 (= (KIF$target spangraph) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (spangraph ?h)) (morphism ?h))
 (= (SET.FTN$target (spangraph ?h)) (sgph.fbr.mor$morphism ?h))
 (SET.FTN$restriction (spangraph ?h) hgph.mor$spangraph)))
```

For any set bijection  $h : X \rightarrow Y$  representing a fixed name function, a hypergraph morphism  $f : G \rightarrow G'$  in the fiber of  $h$  is an *isomorphism* when all component functions are bijections.

```
(14) (KIF$function isomorphism)
 (= (KIF$source isomorphism) set.mor$bijection)
 (= (KIF$target isomorphism) SET$class)
 (forall (?h (set.mor$bijection ?h))
 (= (isomorphism ?h)
 (SET$intersection (morphism ?h) hgph.mor$isomorphism)))
```

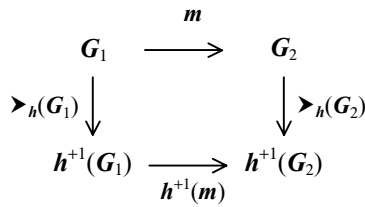


Diagram 14a: Direct Image - abstract

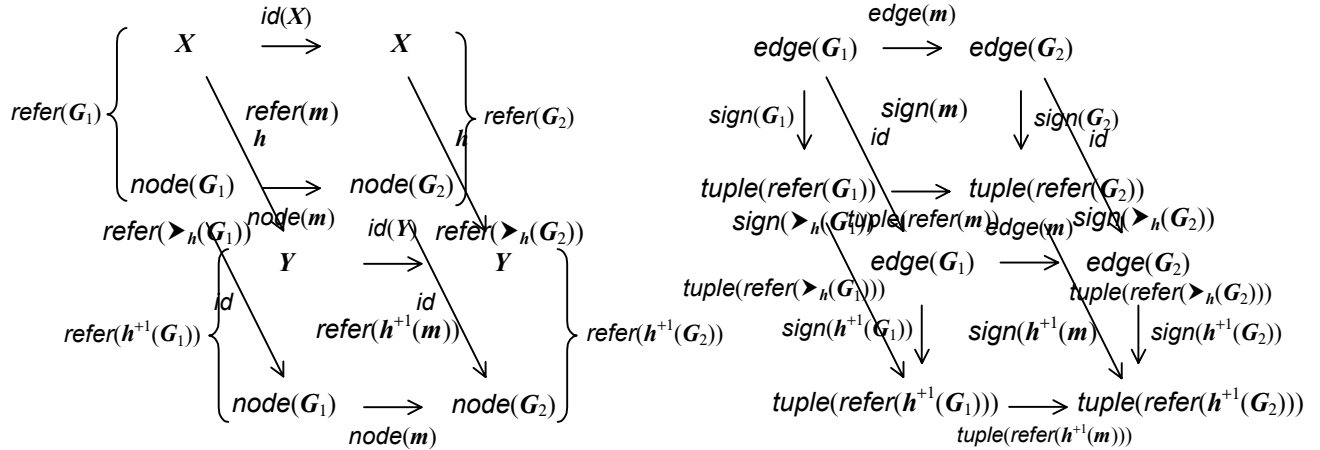


Diagram 14b: Direct Image - details

For any set bijection  $h : X \rightarrow Y$  representing a fixed name function, the *direct image* fiber operator  $h^1 : \text{hgphmor}(id_X) \rightarrow \text{hgphmor}(id_Y)$  along  $h$  maps  $id_X$ -hypergraph morphisms to  $id_Y$ -hypergraph morphisms (Diagram 14). For any hypergraph morphism  $m : G_1 \rightarrow G_2$  with name function  $id(X)$ , the direct image  $h^1(m)$  has the same edge and node function as  $m$ , but has the name function  $id(Y)$ . We use the direct connection operator to define this via the abstract commuting Diagram 14. This is well defined, since the direct connection is an isomorphism.

```
(15) (KIF$function direct-image)
 (= (KIF$source direct-image) set.mor$bijection)
 (= (KIF$target direct-image) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (direct-image ?h))
 (morphism (set.mor$identity (set.mor$source ?h))))
 (= (SET.FTN$target (direct-image ?h))
```

```
(morphism (set.mor$identity (set.mor$target ?h)))
(forall ?m ((morphism (set.mor$identity (set.mor$source ?h))) ?m))
(= (hgph.mor$composition
 [(direct-connection ?h)
 ((source (set.mor$identity (set.mor$source ?h))) ?m))
 ((direct-image ?h) ?m)])
(hgph.mor$composition
 [?m
 ((direct-connection ?h)
 ((target (set.mor$identity (set.mor$source ?h))) ?m))]))))
```

The reference operator commutes with the direct image operator. For any set bijection  $h : X \rightarrow Y$  representing a fixed name function and any hypergraph morphism  $m : G_1 \rightarrow G_2$  with name function  $id(X)$ , the reference semiquartet of the direct image hypergraph morphism is the direct image semiquartet of the reference semiquartet:

$$refer(h^{+1}(m)) = h^{+1}(refer(m)).$$

The commuting diagram used in the definition of direct image, defines a *tuple reference* quartet of a hypergraph morphism  $m$  along  $h$ , whose horizontal source is the tuple reference of the direct connection of the source hypergraph  $\triangleright_h(G_1)$ , whose horizontal target is the tuple reference of the direct connection of the target hypergraph  $\triangleright_h(G_2)$ , whose vertical source is the tuple reference of  $m$ , and whose vertical target is the tuple reference of the direct image  $h^{+1}(m)$ . The signature quartet of the direct image hypergraph morphism is the vertical composition of the signature quartet with the tuple reference quartet.

```
(16) (forall (?h (set.mor$bijection ?h))
 (= (SET.FTN$composition
 [(direct-image ?h)
 (reference (set.mor$identity (set.mor$target ?h)))]))
 (SET.FTN$composition
 [(reference (set.mor$identity (set.mor$source ?h))
 (spr.fbr.mor$direct-image ?h))]))))
```

The direct image of a composition is the composition of the direct image operators of the components:

$$(f \cdot g)^{+1} = f^{+1} \cdot g^{+1} \text{ for any two composable bijections } f : X \rightarrow Y \text{ and } g : Y \rightarrow Z.$$

The direct image of an identity is the identity operator:

$$(id_X)^{+1} = id \text{ for any set } X.$$

```
(17) (forall (?f (set.mor$bijection ?f) ?g (set.mor$bijection ?g) (set.mor$composable ?f ?g))
 (= (direct-image (set.ftn$composition [?f ?g]))
 (SET.FTN$composition [(direct-image ?f) (direct-image ?g)])))

(18) (forall (?x (set.obj$object ?x))
 (= (direct-image (set.mor$identity ?x))
 (SET.FTN$identity (morphism (set.mor$identity ?x)))))
```

For any set bijection  $h : X \rightarrow Y$  representing a fixed name function, the *inverse image* fiber operator  $h^{-1} : hgphmor(id_Y) \rightarrow hgphmor(id_X)$  along  $h$  maps  $id_Y$ -hypergraph morphisms to  $id_X$ -hypergraph morphisms. The inverse image operator is defined to the direct image along the inverse of the bijection.

```
(19) (KIF$function inverse-image)
 (= (KIF$source inverse-image) set.mor$bijection)
 (= (KIF$target inverse-image) SET.FTN$function)
 (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$source (inverse-image ?h))
 (morphism (set.mor$identity (set.mor$target ?h))))
 (= (SET.FTN$target (inverse-image ?h))
 (morphism (set.mor$identity (set.mor$source ?h))))
 (= (inverse-image ?h) (direct-image (set.ftn$inverse ?h)))))
```

The direct and inverse image operators are inverse to each other:  $id_{hgph(X)} = h^{+1} \cdot h^{-1}$  and  $id_{hgph(Y)} = h^{-1} \cdot h^{+1}$  for any set bijection  $h : X \rightarrow Y$ .

```
(20) (forall (?h (set.mor$bijection ?h))
 (and (= (SET.FTN$composition [(direct-image ?h) (inverse-image ?h)])
 (SET.FTN$identity (morphism (set.mor$identity (set.mor$source ?h)))))
 (= (SET.FTN$composition [(inverse-image ?h) (direct-image ?h)])
 (SET.FTN$identity (morphism (set.mor$identity (set.mor$target ?h)))))
```

## The IFF Namespace of Hypergraphs

Robert E. Kent

Page 38

May 8, 2003

For any two composable bijections  $f: X \rightarrow Y$  and  $g: Y \rightarrow Z$  representing two fixed name functions, there are composition fiber class components:

$$\text{comp-opspn}([f, g]) \subseteq \text{comp-opspn}$$

$$\text{cmpbl}([f, g]) \subseteq \text{hgphmor}(f) \times \text{hgphmor}(g)$$

$$\text{comp}([f, g]) : \text{plbk}([f, g]) = \text{hgphmor}(f) \times_{\text{hgph}(Y)} \text{hgphmor}(g) \rightarrow \text{hgphmor}(f \cdot g).$$

```
(21) (KIF$function composable-opsan)
 (= (KIF$source composable-opsan) (REL$extent set.mor$composable))
 (= (KIF$target composable-opsan) SET.LIM.PBK$opspan)
 (forall (?f (set.mor$bijection ?f) ?g (set.mor$bijection ?g) (set.mor$composable ?f ?g))
 (and (= (SET.LIM.PBK$class1 (composable-opsan [?f ?g])) (morphism ?f))
 (= (SET.LIM.PBK$class2 (composable-opsan [?f ?g])) (morphism ?g))
 (= (SET.LIM.PBK$opvertex (composable-opsan [?f ?g]))
 (hgph.fbr.obj$object (set.mor$target ?f)))
 (= (SET.LIM.PBK$opfirst (composable-opsan [?f ?g])) (target ?f))
 (= (SET.LIM.PBK$opsecond (composable-opsan [?f ?g])) (source ?g))))

(22) (KIF$function composable)
 (= (KIF$source composable) (REL$extent set.mor$composable))
 (= (KIF$target composable) REL$relation)
 (forall (?f (set.mor$morphism ?f) ?g (set.mor$morphism ?g) (set.mor$composable ?f ?g))
 (and (= (REL$class1 (composable [?f ?g])) (morphism ?f))
 (= (REL$class2 (composable [?f ?g])) (morphism ?g))
 (= (REL$extent (composable [?f ?g]))
 (SET.LIM.PBK$pullback (composable-opsan [?f ?g])))
 (REL$abridgement (composable [?f ?g]) hgph.mor$composable)))

(23) (KIF$function composition)
 (= (KIF$source composition) (REL$extent set.mor$composable))
 (= (KIF$target composition) SET.FTN$function)
 (forall (?f (set.mor$bijection ?f) ?g (set.mor$bijection ?g) (set.mor$composable ?f ?g))
 (and (= (SET.FTN$source (composition [?f ?g])) (REL$extent (composable [?f ?g])))
 (= (SET.FTN$target (composition [?f ?g]))
 (morphism (set.ftn$composition [?f ?g])))
 (SET.FTN$restriction (composition [?f ?g]) hgph.mor$composition)))
```

For any set  $X$  representing a fixed set of names, there is the following fiber class function.

$$\text{idnode}(X) : \text{hgph}(X) \rightarrow \text{hgphmor}(\text{id}_X)$$


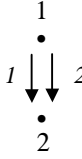
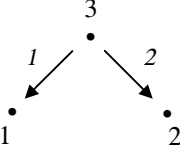
```
(24) (KIF$function identity)
 (= (KIF$source identity) set.obj$object)
 (= (KIF$target identity) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (identity ?x)) (hgph.fbr.obj$object ?x))
 (= (SET.FTN$target (identity ?x)) (morphism (set.mor$identity ?x)))
 (SET.FTN$restriction (identity ?x) hgph.mor$identity)))
```

## Colimits for Hypergraphs

hgph.col

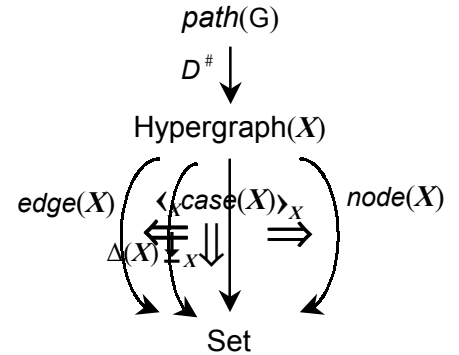
Colimits of hypergraphs are basic. They underlie the colimit construction for languages, theories, models and logics. Thus, they are used for constructing the *type pole* of object-level ontologies. As demonstrated below, since  $\text{Hypergraph}(X)$  has all sums and coequalizers, it has all colimits – it is cocomplete. For any set bijection  $h : X \rightarrow Y$  representing a fixed function of names, the categories  $\text{Hypergraph}(X)$  and  $\text{Hypergraph}(Y)$  are isomorphic, and hence have isomorphic colimits. For any set  $X$  representing a fixed set of names, the category  $\text{Spangraph}(X)$  is categorically equivalent to the category  $\text{Hypergraph}(X)$ , and hence is also cocomplete.

**Table 2: Shapes for Diagrams**

|                           |                                                                                   |                                                                                   |                                                                                    |
|---------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
|                           |  |  |  |
| <i>empty</i><br>(initial) | <i>two</i><br>(binary coproduct)                                                  | <i>parallel pair</i><br>(coequalizer)                                             | <i>span</i><br>(pushout)                                                           |

We axiomatize colimits for finite diagrams  $D : G \rightarrow |\text{Hypergraph}(X)|$  of the following diagram shape graphs  $G$  (Table 2): (i) *empty*, (ii) *two*, (iii) *parallel-pair* and (iv) *span*. These colimits are called (i) the initial hypergraph, (ii) a binary coproduct (sum), (iii) a coequalizer, and (iv) a pushout.

The extension functor of a diagram  $D$  of a particular shape  $G$  in the fiber category  $\text{Hypergraph}(X)$  can be composed with the case, edge and node component functors, and also with the indication, comediation and projection component natural transformations (Figure 10). These compositions result in corresponding case, edge and node component diagrams in  $\text{Set}$ , and indication, comediation and projection component diagram morphisms. Composition with the name functor results in the constant functor  $\Delta(X) : \text{path}(G) \rightarrow \text{Set}$ . These components will be axiomatized in the namespaces for each shape. The colimits of various shapes are axiomatized both abstractly and concretely. The concrete axiomatization is this component axiomatization. That is, colimits for hypergraphs are concrete, and defined in terms of colimits for their component diagrams and diagram morphisms.



**Figure 10: Component Diagrams and Diagram Morphisms**

### Initial Hypergraph

For any set  $X$  representing a fixed set of variables, there is a special hypergraph  $0_X$  (see Figure 11, where arrows denote functions) called the *initial hypergraph*, which has the required set of variables  $X$ , but no

$$\begin{array}{ccccc}
 \emptyset X & \xleftarrow{! \emptyset X} & \emptyset & \xrightarrow{! \text{tuple}(\text{refer}(0_X))} & \text{tuple}(\text{refer}_X(0_X)) & \xleftarrow{\emptyset} & X \\
 \text{id} \downarrow & & \downarrow ! \text{edge}(G) & & \downarrow \text{tuple}(\text{refer}(!_X G)) & \downarrow ! \text{node}(G) & \downarrow \text{id} \\
 \emptyset X & \xleftarrow{\#_{X,G}} & \text{edge}_X(G) & \xrightarrow{\partial_{X,G}} & \text{tuple}(\text{refer}_X(G)) & \xrightarrow{\text{node}_X(G)} & X
 \end{array}$$

**Figure 11: Initial Hypergraph and Counique Hypergraph Morphism**

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 40

May 8, 2003

edges and no nodes:

- the set of *edges*  $\mathbf{edge}(0_X) = \emptyset$ ;
- the set of *nodes*  $\mathbf{node}(0_X) = \emptyset$ ; and
- the set of *names* is  $\mathbf{name}(0_X) = X$ .

It has the following reference set pair, and signature and edge arity functions:

- the *reference* set pair  $*_{0_X} = \langle X, \emptyset \rangle$ ;
- the *signature* function  $\partial_{0_X} = !_{\text{tuple}(\text{refer}(0_X))} : \emptyset \rightarrow \text{tuple}(\text{refer}(0_X)) \cong 1$ ; and
- the *edge-arity* function  $\#_{0_X} = !_{\emptyset X} : \emptyset \rightarrow \emptyset X$ .

The initial hypergraph  $0_X$  in the fibered category  $\mathbf{Hypergraph}(X)$  is the “first” hypergraph in the following sense: for any hypergraph  $G$  in  $\mathbf{Hypergraph}(X)$  there is a (co) *unique* hypergraph morphism  $!_{X,G} : 0_X \rightarrow G$  in  $\mathbf{Hypergraph}(X)$ , (see Figure 1, where arrows denote functions) the unique hypergraph morphism from  $0_X$  to  $G$  with identity name function on  $X$ . To express universality we need to know what a cocone is for the empty graph shape. This is just an object in  $\mathbf{Hypergraph}(X)$ ; that is, a hypergraph  $G$  with name set  $X$ . To express universality, we need to axiomatize that  $!_{X,G} : 0_X \rightarrow G$  is the unique hypergraph morphism in  $\mathbf{Hypergraph}(X)$  with these source and target hypergraphs. We use a definite description to express this.

The edge and node functions of this hypergraph morphism are the counique functions (the empty functions) from the initial set  $\emptyset$  to the respective set  $\mathbf{edge}(G)$  or  $\mathbf{node}(G)$ .

- The counique (empty) *edge* function is  $\mathbf{edge}(!_{X,G}) = !_{\mathbf{edge}(G)} : \emptyset \rightarrow \mathbf{edge}(G)$ ; and
- The counique (empty) *node* function is  $\mathbf{node}(!_{X,G}) = !_{\mathbf{node}(G)} : \emptyset \rightarrow \mathbf{node}(G)$ .

The function  $\text{tuple}(\text{refer}(!_{X,G})) : \text{tuple}(\text{refer}(0_X)) \cong 1 \rightarrow \text{tuple}(\text{refer}(G))$  maps the single (empty) tuple in  $\text{tuple}(\text{refer}(0_X))$  to the empty tuple in  $\text{tuple}(\text{refer}(G))$ . The reference and signature quartets for the counique hypergraph morphism are vacuous.

```
(1) (SET.FTN$function initial)
 (= (SET.FTN$source initial) set.obj$object)
 (= (SET.FTN$target initial) hgph.obj$object)
 (forall (?x (set.obj$object ?x))
 (and ((hgph.fbr.obj$object ?x) (initial ?x))
 (= (hgph.obj$edge (initial ?x)) set.col$initial)
 (= (hgph.obj$node (initial ?x)) set.col$initial)
 (= (hgph.obj$name (initial ?x)) ?x)
 (= (set.pr$set1 (hgph.obj$reference (initial ?x))) ?x)
 (= (set.pr$set2 (hgph.obj$reference (initial ?x))) set.col$initial)
 (= (hgph.obj$signature (initial ?x))
 (set.col$counique (set.pr$tuple (hgph.obj$reference (initial ?x)))))
 (= (hgph.obj$edge-arity (initial ?x))
 (set.col$counique (set$power ?x)))))

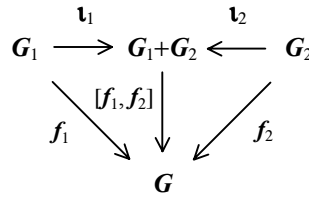
(2) (KIF$function counique)
 (= (KIF$source counique) set.obj$object)
 (= (KIF$target counique) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (counique ?x)) (hgph.fbr.obj$object ?x))
 (= (SET.FTN$target (counique ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(counique ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 ((SET.FTN$constant [(hgph.fbr.obj$object ?x) (hgph.fbr.obj$object ?x)]) (initial ?x)))
 (= (SET.FTN$composition [(counique ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (SET.FTN$identity (hgph.fbr.obj$object ?x)))
 (= (SET.FTN$composition [(counique ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(hgph.fbr.obj$edge ?x) set.col$counique]))
 (= (SET.FTN$composition [(counique ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(hgph.fbr.obj$node ?x) set.col$counique]))
 (forall (?g ((hgph.fbr.obj$object ?x) ?g))
 (= ((counique ?x) ?g)
 (the (?m ((hgph.fbr.mor$morphism (set.mor$identity ?x)) ?m))
 (and (= ((hgph.fbr.mor$source (set.mor$identity ?x)) ?m) (initial ?x))
 (= ((hgph.fbr.mor$target (set.mor$identity ?x)) ?m) ?g))))))
```



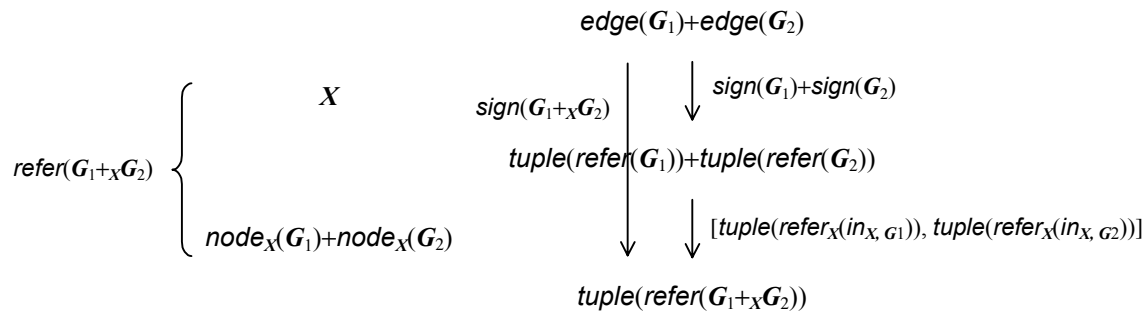
## Binary Coproducts

### hgph.col.coprod2

Sums (binary coproducts) do not exist for an arbitrary pair of hypergraphs; instead, hypergraphs need to be restricted to a subcategory of hypergraphs with name sets that are in bijection. For example, any two hypergraphs in the subcategory with countable name sets can be summed. More precisely, the subcategory  $\text{Hypergraph}(\alpha)$  of all hypergraphs whose name set has cardinality  $\alpha$  is cocomplete. We simplify the discussion of sums by assuming all hypergraphs under discussion have the same name set  $X$ ; that is, instead of considering  $\text{Hypergraph}(\alpha)$ , we only consider the fiber category  $\text{Hypergraph}(X)$  for some set of names  $X$ . This is a subcategory  $\text{Hypergraph}(X) \subseteq \text{Hypergraph}(\alpha)$  for  $\|X\| = \alpha$ . For the countable case  $\|X\| = \aleph$ , the set of natural numbers  $X = \text{natno}$  would work. We then argue that all  $\text{Hypergraph}(\alpha)$  are partitioned into isomorphic fiber categories.



**Figure 12a: Binary Coproduct Hypergraph, Injections, Cocone and Comediator – abstract**



**Figure 12b: Binary Coproduct Hypergraph – concrete version**

Let  $X$  be any set representing a fixed set of names. Let  $G_1$  and  $G_2$  be two hypergraphs with common name set  $X$ . The *binary coproduct (sum)* hypergraph  $G_1+XG_2$  is defined as follows (Figure 12b).

The edge and node sets are the sums or binary coproducts (disjoint unions) of their components

$$\text{edge}_X(G_1+XG_2) = \text{edge}_X(G_1) + \text{edge}_X(G_2), \text{ and}$$

$$\text{node}_X(G_1+XG_2) = \text{node}_X(G_1) + \text{node}_X(G_2).$$

The name set is the fixed set  $X$ ,

$$\text{name}_X(G_1+XG_2) = \text{name}_X(G_1) = \text{name}_X(G_2) = X.$$

The reference set pair is

$$*_{X, G_1+XG_2} = \text{refer}_X(G_1+XG_2) = \langle X, \text{node}_X(G_1) + \text{node}_X(G_2) \rangle.$$

The signature function

$$\begin{aligned} \partial_{X, G_1+XG_2} &= [\partial_{X, G_1} \cdot \text{tuple}(\text{refer}_X(\text{in}_{X, G_1})), \partial_{X, G_2} \cdot \text{tuple}(\text{refer}_X(\text{in}_{X, G_2}))] \\ &= (\partial_{X, G_1} + \partial_{X, G_2}) \cdot [\text{tuple}(\text{refer}_X(\text{in}_{X, G_1})), \text{tuple}(\text{refer}_X(\text{in}_{X, G_2}))] \\ &: \text{edge}_X(G_1) + \text{edge}_X(G_2) \rightarrow \text{tuple}(\text{refer}_X(G_1+XG_2)), \end{aligned}$$

The edge-arity function

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 42

May 8, 2003

$$\#_{X, G_1+XG_2} = [\#_{X, G_1}, \#_{X, G_2}] : \text{edge}_X(G_1) + \text{edge}_X(G_2) \rightarrow \wp X.$$

The *sum injection* hypergraph morphism  $\text{in}_X(G_1) = \langle \text{in}_{\text{edge}(G_1)}, \text{in}_{\text{node}(G_1)}, \text{id}_X \rangle : G_1 \rightarrow G_1+XG_2$  from the first component hypergraph  $G_1$  to the sum hypergraph  $G_1+XG_2$  is described as follows.

- The edge function is the sum injection

$$\text{in}_{\text{edge}(G_1)} : \text{edge}_X(G_1) \rightarrow \text{edge}_X(G_1+XG_2) = \text{edge}_X(G_1) + \text{edge}_X(G_2);$$

- the node function is the sum injection

$$\text{in}_{\text{node}(G_1)} : \text{node}_X(G_1) \rightarrow \text{node}_X(G_1+XG_2) = \text{node}_X(G_1) + \text{node}_X(G_2); \text{ and}$$

- the name function is the identity

$$\text{id}_X : X \rightarrow \text{name}_X(G_1+XG_2) = X.$$

The sum  $G_1+XG_2$  is a coproduct in the category  $\text{Hypergraph}(X)$ . In summary, all sums exist in any fiber  $\text{name}^{-1}(X) = \text{Hypergraph}(X)$  of the name functor  $\text{name} : \text{Hypergraph} \rightarrow \text{Set}$ . The hypergraph functor

$$\text{hgph}(X) : \text{Spangraph}(X) \rightarrow \text{Hypergraph}(X)$$

and the spangraph functor

$$\text{sgph}(X) : \text{Hypergraph}(X) \rightarrow \text{Spangraph}(X)$$

preserve sums.

A *binary coproduct (sum)* is a finite colimit in the category  $\text{Hypergraph}(X)$  for a diagram of shape  $\text{two} = \bullet \bullet$ . Such a diagram (of hypergraphs and hypergraph morphisms) is called a *pair* of hypergraphs. For any set  $X$  representing a fixed set of names, a *pair* of hypergraphs consists of hypergraphs *hypergraph1* and *hypergraph2*. We use either the generic term ‘diagram’ or the specific term ‘pair’ to denote the pair class. Pairs are determined by their two component hypergraphs.

```
(1) (KIF$function diagram)
 (KIF$function pair)
 (= pair diagram)
 (= (KIF$source diagram) set.obj$object)
 (= (KIF$target diagram) SET$class)

(2) (KIF$function hypergraph1)
 (= (KIF$source hypergraph1) set.obj$object)
 (= (KIF$target hypergraph1) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph1 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph1 ?x)) (hgph.fbr.obj$object ?x))))

(3) (KIF$function hypergraph2)
 (= (KIF$source hypergraph2) set.obj$object)
 (= (KIF$target hypergraph2) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph2 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph2 ?x)) (hgph.fbr.obj$object ?x))))

(4) (forall (?x (set.obj$object ?x))
 ?d1 ((diagram ?x) ?d1) ?d2 ((diagram ?x) ?d2))
 (=> (and (= ((hypergraph1 ?x) ?d1) ((hypergraph1 ?x) ?d2))
 (= ((hypergraph2 ?x) ?d1) ((hypergraph2 ?x) ?d2)))
 (= ?d1 ?d2)))
```

For any set  $X$  representing a fixed set of names, there is an *edge pair* or *edge diagram* function, which maps a pair of hypergraphs to the underlying pair of edge sets. Similarly, there is a *node pair* or *node diagram* function, which maps a pair of hypergraphs to the underlying pair of node sets.

```
(5) (KIF$function edge-diagram)
 (= (KIF$source edge-diagram) set.obj$object)
 (= (KIF$target edge-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (edge-diagram ?x)) set.col.coprd2$diagram)
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coprd2$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coprd2$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$edge ?x)]))))
```

```
(SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$edge ?x)]))
```

```
(6) (KIF$function node-diagram)
(= (KIF$source node-diagram) set.obj$object)
(= (KIF$target node-diagram) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (node-diagram ?x)) set.col.copr2$diagram)
 (= (SET.FTN$composition [(node-diagram ?x) set.col.copr2$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.copr2$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$node ?x)]))))
```

For any set  $X$  representing a fixed set of names, a *binary coproduct cocone* is the appropriate cocone for a binary coproduct over  $X$ . A coproduct cocone (see Figure 7a, where arrows denote hypergraph morphisms) consists of a pair of hypergraph morphisms called *opfirst* and *opsecond*. These are required to have a common target hypergraph called the *opvertex* of the cocone. Each binary coproduct cocone is under a pair of hypergraphs.

```
(7) (KIF$function cocone)
(= (KIF$source cocone) set.obj$object)
(= (KIF$target cocone) SET$class)

(8) (KIF$function cocone-diagram)
(= (KIF$source cocone-diagram) set.obj$object)
(= (KIF$target cocone-diagram) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (cocone-diagram ?x)) (cocone ?x))
 (= (SET.FTN$target (cocone-diagram ?x)) (diagram ?x))))

(9) (KIF$function opvertex)
(= (KIF$source opvertex) set.obj$object)
(= (KIF$target opvertex) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opvertex ?x)) (cocone ?x))
 (= (SET.FTN$target (opvertex ?x)) (hgph.fbr.obj$object ?x))))

(10) (KIF$function opfirst)
(= (KIF$source opfirst) set.obj$object)
(= (KIF$target opfirst) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opfirst ?x)) (cocone ?x))
 (= (SET.FTN$target (opfirst ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (hypergraph1 ?x)]))
 (= (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))))

(11) (KIF$function opsecond)
(= (KIF$source opsecond) set.obj$object)
(= (KIF$target opsecond) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opsecond ?x)) (cocone ?x))
 (= (SET.FTN$target (opsecond ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (hypergraph2 ?x)]))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))))
```

For any set  $X$  representing a fixed set of names, there are *edge cocone* and *node cocone* functions, which map a binary coproduct cocone to the underlying binary coproduct cocone of edge sets and edge functions and node sets and node functions, respectively.

```
(12) (KIF$function edge-cocone)
(= (KIF$source edge-cocone) set.obj$object)
(= (KIF$target edge-cocone) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-cocone ?x)) (cocone ?x))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 44

May 8, 2003

```
(= (SET.FTN$target (edge-cocone ?x)) set.col.copr2$cocone)
(= (SET.FTN$composition [(edge-cocone ?x) set.col.copr2$cocone-diagram])
 (SET.FTN$target [(cocone-diagram ?x) (edge-diagram ?x)]))
(= (SET.FTN$composition [(edge-cocone ?x) set.col.copr2$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$edge ?x)]))
(= (SET.FTN$composition [(edge-cocone ?x) set.col.copr2$opfirst])
 (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))]))
(= (SET.FTN$composition [(edge-cocone ?x) set.col.copr2$opsecond])
 (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))]))))

(13) (KIF$function node-cocone)
 (= (KIF$source node-cocone) set.obj$object)
 (= (KIF$target node-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (node-cocone ?x)) set.col.copr2$cocone)
 (= (SET.FTN$composition [(node-cocone ?x) set.col.copr2$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (node-diagram ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.copr2$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.copr2$opfirst])
 (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$node (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.copr2$opsecond])
 (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])))))
```

For any set  $X$  representing a fixed set of names, the *colimiting cocone* function that maps a pair of hypergraphs to its binary coproduct colimiting cocone. The totality of this function, along with the universality of the comediator hypergraph morphism, implies that a binary coproduct exists for any pair of hypergraphs. The opvertex of the binary coproduct colimiting cocone is a specific *binary coproduct* hypergraph. It comes equipped with two *injection* hypergraph morphisms.

```
(14) (KIF$function colimiting-cocone)
 (= (KIF$source colimiting-cocone) set.obj$object)
 (= (KIF$target colimiting-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimiting-cocone ?x)) (diagram ?x))
 (= (SET.FTN$target (colimiting-cocone ?x)) (cocone ?x))
 (= (SET.FTN$composition [(colimiting-cocone ?x) (cocone-diagram ?x)])
 (SET.FTN$identity (diagram ?x)))))

(15) (KIF$function colimit)
 (KIF$function binary-coproduct)
 (= binary-coproduct colimit)
 (= (KIF$source colimit) set.obj$object)
 (= (KIF$target colimit) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimit ?x)) (diagram ?x))
 (= (SET.FTN$target (colimit ?x)) (hgph.fbr.obj$object ?x))
 (= (colimit ?x)
 (SET.FTN$composition [(colimiting-cocone ?x) (opvertex ?x)])))

(16) (KIF$function injection1)
 (= (KIF$source injection1) set.obj$object)
 (= (KIF$target injection1) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (injection1 ?x)) (diagram ?x))
 (= (SET.FTN$target (injection1 ?x))
 (sgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (hypergraph1 ?x))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (colimit ?x))
 (= (injection1 ?x)
 (SET.FTN$composition [(colimiting-cocone ?x) (opfirst ?x)])))

(17) (KIF$function injection2)
 (= (KIF$source injection2) set.obj$object)
 (= (KIF$target injection2) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (injection2 ?x)) (diagram ?x))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 45

May 8, 2003

```
(= (SET.FTN$target (injection2 ?x))
 (sgph.fbr.mor$morphism (set.mor$identity ?x)))
(= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (hypergraph2 ?x))
(= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (colimit ?x))
(= (injection2 ?x)
 (SET.FTN$composition [(colimiting-cocone ?x) (opsecond ?x)])))
```

For any set  $X$  representing a fixed set of names, the binary coproduct and injections are expressed both abstractly by their defining axioms and concretely by the following axioms. These axioms ensure that the binary coproduct is specific. The following two axioms are the necessary conditions that the edge and node functors preserve concrete colimits. These explicitly ensure that a binary coproduct of hypergraphs is specific – that its edge and node sets are exactly the disjoint unions of the corresponding sets of the pair of hypergraphs. In addition, these explicitly ensure that the two coproduct injection hypergraph morphisms are specific – that their edge and node functions are exactly the coproduct injections of the edge and node set pairs of the pair of hypergraphs.

```
(18) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (edge-cocone ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coprod2$colimiting-cone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$edge ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coprod2$colimit]))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coprod2$injection1]))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coprod2$projection2]))))

(19) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (node-cocone ?x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.coprod2$colimiting-cone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$node ?x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.coprod2$colimit]))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-diagram ?x) set.col.coprod2$injection1]))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-diagram ?x) set.col.coprod2$projection2]))))
```

For any set  $X$  representing a fixed set of names and for any binary coproduct cocone, there is a *comediator* hypergraph morphism  $[f_1, f_2] : G_1 + G_2 \rightarrow G$  (see Figure 7a, where arrows denote hypergraph morphisms) from the binary coproduct of the underlying diagram (pair of hypergraphs) to the opvertex of the cocone. This is the unique hypergraph morphism, which commutes with the opfirst  $f_1$  and the opsecond  $f_2$ . We define this by using the comediators of the underlying edge and node cocones. Existence and uniqueness represents the universality of the binary coproduct operator.

```
(20) (KIF$function comediator)
 (= (KIF$source comediator) set.obj$object)
 (= (KIF$target comediator) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (comediator ?x)) (cocone ?x))
 (= (SET.FTN$target (comediator ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (colimit ?x)]))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-cocone ?x) set.col.coprod2$comediator]))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-cocone ?x) set.col.coprod2$comediator]))))
```

It can be verified that the comediation is the unique hypergraph morphism that makes the diagram in Figure 7a commutative. We use a definite description to express this fact.

```
(21) (forall (?x (set.obj$object ?x)
 ?s ((cocone ?x) ?s))
 (= ((comediator ?x) ?s)
 (the (?f ((hgph.fbr.mor$morphism (set.mor$identity ?x)) ?f))
 (and (= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(injection1 ?x) ((cocone-diagram ?x) ?s)) ?f])
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 46

May 8, 2003

```
((opfirst ?x) ?s))
(= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [((injection2 ?x) ((cocone-diagram ?x) ?s)) ?f])
 ((opsecond ?x) ?s))))
```

## Endorelations and Quotients

hgph.col.endo

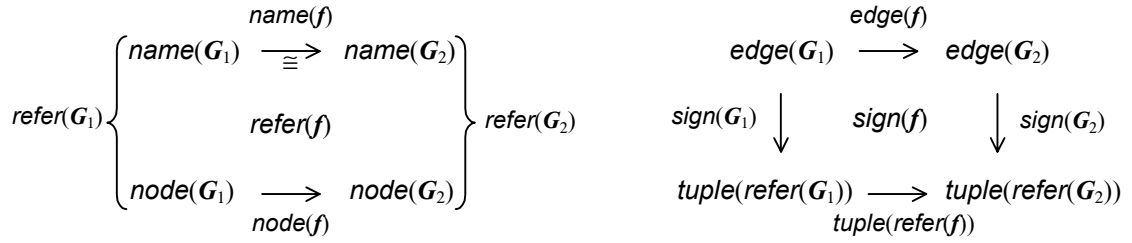


Figure 13: Hypergraph Morphism

Endorelations are definable on fibers of the category **Hypergraph**. Let  $X$  be any set representing a fixed set of names. Let  $\text{endo}(X)$  denote the class of all hypergraph endorelations at  $X$ . A hypergraph *endorelation* is a triple  $J = \langle \text{hgph}_X(J), \text{edge}_X(J), \text{node}_X(J) \rangle$  consisting of a hypergraph  $\text{hgph}_X(J) = G$  with name set  $X$ , a binary endorelation  $\text{edge}_X(J) = E_X \subseteq \text{edge}_X(G) \times \text{edge}_X(G)$  on edges of  $G$ , and a binary endorelation  $\text{node}_X(J) = N_X \subseteq \text{node}_X(G) \times \text{node}_X(G)$  on nodes of  $G$ , that satisfy the following constraints on arity and signature functions of  $G$ . Here  $\equiv_E$  is the equivalence relation generated by  $\text{edge}_X(J)$  and  $\equiv_N$  is the equivalence relation generated by  $\text{node}_X(J)$ .

- **[arity/signature constraints]** for any pair of edges  $\varepsilon_1, \varepsilon_2 \in \text{edge}(G)$ ,  
if  $(\varepsilon_1, \varepsilon_2) \in \text{edge}_X(J)$ , then
  - \*  $\text{arity}_X(G)(\varepsilon_1) = \text{arity}_X(G)(\varepsilon_2)$  and
  - \*  $\text{sign}_X(G)(\varepsilon_1) \equiv_0 \text{sign}_X(G)(\varepsilon_2)$ , where  $\equiv_0 \subseteq \text{sign}(\text{refer}(G)) \times \text{sign}(\text{refer}(G))$  is the equivalence relation on tuple defined as follows:  $(\tau_1, \tau_2) \in \equiv_0$  when  
 $\text{arity}(\text{refer}(G))(\tau_1) = \text{arity}(\text{refer}(G))(\tau_2)$ , and  
 $\tau_1(x_1) \equiv_N \tau_2(x_2)$  for all  $x \in \text{arity}(\text{refer}(G))(\tau_1) = \text{arity}(\text{refer}(G))(\tau_2)$ .

The hypergraph  $G$  is called the *base hypergraph* of  $J$  – the hypergraph on which  $J$  is based. A hypergraph endorelation is determined by the triple consisting of its base hypergraph and two endorelations. A hypergraph endorelation on  $G$  determines a spangraph endorelation on its associated spangraph  $\text{hgph}(G)$ .

- (1) (KIF\$function endorelation)  
 (= (KIF\$source endorelation) set.obj\$object)  
 (= (KIF\$target endorelation) SET\$class)
- (2) (KIF\$function hypergraph)  
 (KIF\$function base)  
 (= base hypergraph)  
 (= (KIF\$source hypergraph) set.obj\$object)  
 (= (KIF\$target hypergraph) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
   (and (= (SET.FTN\$source (hypergraph ?x)) (endorelation ?x))  
 (= (SET.FTN\$target (hypergraph ?x)) (sgph.fbr.obj\$object ?x))))
- (3) (KIF\$function edge)  
 (= (KIF\$source edge) set.obj\$object)  
 (= (KIF\$target edge) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
   (and (= (SET.FTN\$source (edge ?x)) (endorelation ?x))  
 (= (SET.FTN\$target (edge ?x)) rel.endo\$endorelation)  
 (= (SET.FTN\$composition [(edge ?x) rel.endo\$set])  
 (SET.FTN\$composition [(hypergraph ?x) (hgph.fbr.obj\$edge ?x)]))))
- (4) (KIF\$function node)  
 (= (KIF\$source node) set.obj\$object)  
 (= (KIF\$target node) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
   (and (= (SET.FTN\$source (node ?x)) (endorelation ?x))

```

(= (SET.FTN$target (node ?x)) rel.endo$endorelation)
(= (SET.FTN$composition [(node ?x) rel.endo$set])
 (SET.FTN$composition [(hypergraph ?x) (hgph.fbr.obj$node ?x)])))

(5) (forall (?x (set.obj$object ?x)
 ?j ((endorelation ?x) ?j))
 (forall (?e1 ((hgph.fbr.obj$edge ?x) ((hypergraph ?x) ?j)) ?e1)
 ?e2 ((hgph.fbr.obj$edge ?x) ((hypergraph ?x) ?j)) ?e2))
 (=> ((edge ?x) ?j) ?e1 ?e2)
 (and (= ((hgph.fbr.obj$edge-arity ?x) ((hypergraph ?x) ?j)) ?e1)
 ((hgph.fbr.obj$edge-arity ?x) ((hypergraph ?x) ?j)) ?e2))
 (forall (?x (((hgph.fbr.obj$edge-arity ?x) ((hypergraph ?x) ?j)) ?e1) ?x)
 ((rel.endo$equivalence-closure ((node ?x) ?j))
 (((hgph.fbr.obj$signature ?x) ((hypergraph ?x) ?j)) ?e1) ?x)
 (((hgph.fbr.obj$signature ?x) ((hypergraph ?x) ?j)) ?e2) ?x))))))

```

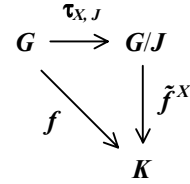
Let  $X$  be any set representing a fixed set of names. The class  $\mathbf{endo}(X)$  of endorelations at  $X$  is ordered. For any two endorelations  $J_1, J_2 \in \mathbf{endo}(X)$ ,  $J_1$  is a subrelation of  $J_2$ ,  $J_1 \leq J_2$ , when

- $hgph_X(J_1) = hgph_X(J_2)$ ,
- $edge_X(J_1) \subseteq edge_X(J_2)$ , and
- $node_X(J_1) \subseteq node_X(J_2)$ .

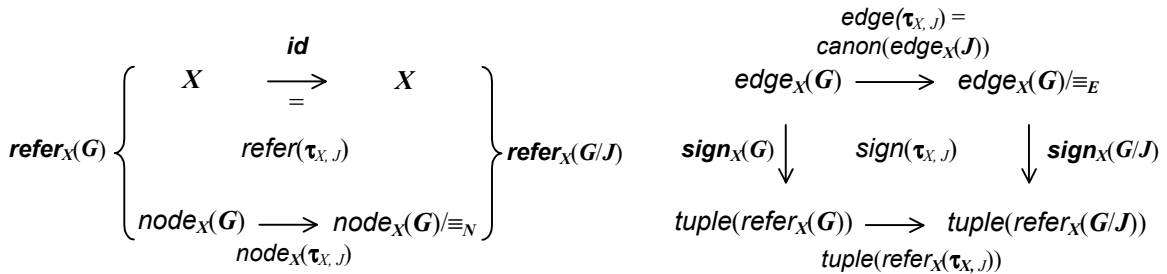
```

(6) (KIF$function subrelation)
 (= (KIF$source subrelation) set.obj$object)
 (= (KIF$target subrelation) ORD$partial-order)
 (forall (?x (set.obj$object ?x))
 (and (= (ORD$class (subrelation ?x)) (endorelation ?x))
 (forall (?j1 ((endorelation ?x) ?j1)
 ?j2 ((endorelation ?x) ?j2))
 (<=> ((subrelation ?x) ?j1 ?j2)
 (and (= ((hypergraph ?x) ?j1) ((hypergraph ?x) ?j2))
 (ord$suborder ((edge ?x) ?j1) ((edge ?x) ?j2))
 (ord$suborder ((node ?x) ?j1) ((node ?x) ?j2)))))))

```



**Diagram 15: Canonical Morphism and Universality of the Quotient**



**Figure 14: Hypergraph Quotient and Canonical Morphism - details**

Often, the endorelations  $E_X$  and  $N_X$  are equivalence relations on the edges and nodes, respectively. However, it is not only convenient but also very important not to require this. In particular, the endorelations defined by parallel pairs of hypergraph morphisms (coequalizer diagrams) do not have component equivalence relations. For any edge  $\varepsilon \in \mathbf{edge}_X(G)$ , write  $[\varepsilon]_E$  for the  $E$ -equivalence class of  $\varepsilon$ . Same for nodes.



# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 49

May 8, 2003

Given any fixed set of names  $X$ , a simple inductive proof shows that the triple  $\hat{J} = \langle G, \equiv_E, \equiv_N \rangle$  is also an endorelation.

## Quotient

Given any fixed set of variables  $X$ , the *quotient*  $G/J$  of an endorelation  $J = \langle G, E_X, N_X \rangle = \langle hgph_X(J), edge_X(J), node_X(J) \rangle$  on a variable set  $X$  (see Figure 14, where arrows denote set functions; or Diagram 1, where arrows denote hypergraph morphisms) is the hypergraph defined as follows:

- The set of *edges* is  $edge_X(G/J) = edge_X(G)/\equiv_E$ .
- The set of *nodes* is  $node_X(G/J) = node_X(G)/\equiv_N$ .
- The set of *names* is  $name_X(G/J) = name_X(G) = X$ .

The following definitions follow the diagrams above.

- The *reference* set pair

$$*_{X,G/J} = refer_X(G/J) = \langle X, node_X(G)/\equiv_N \rangle.$$

- The *edge arity* function

$$\#_{X,G/J} = edge-arity_X(G/J) : edge_X(G)/\equiv_E \rightarrow \wp X$$

is defined pointwise as follows

$$\#_{X,G/J}[\varepsilon]_E = \#_{X,G}(\varepsilon) \text{ for all edges } \varepsilon \in edge_X(G).$$

- The *signature* function

$$\partial_{X,G/J} = sign_X(G/J) : edge_X(G)/\equiv_E \rightarrow tuple(refer_X(G/J))$$

is defined pointwise as follows

$$\partial_{G/J}[\varepsilon]_E(x) = [\partial_G(\varepsilon)(x)]_N \text{ for all edges } \varepsilon \in edge_X(G) \text{ and names } x \in \#_G(\varepsilon).$$

## Canon

There is a *canonical* quotient hypergraph morphism

$$\tau_{X,J} = \langle refer_X(\tau_{X,J}), edge-arity_X(\tau_{X,J}), sign_X(\tau_{X,J}) \rangle : G \rightarrow L/J$$

whose edge function is the canonical surjection

$$edge_X(\tau_{X,J}) = canon(edge_X(J)) = [-]_E : edge_X(G) \rightarrow edge_X(G)/\equiv_E, \text{ and}$$

whose node function is the canonical surjection

$$node_X(\tau_{X,J}) = canon(node_X(J)) = [-]_N : node_X(G) \rightarrow node_X(G)/\equiv_N.$$

The fundamental property for this hypergraph morphism is trivial, given the definition of the quotient hypergraph above.

```
(7) (KIF$function quotient)
 (= (KIF$source quotient) set.obj$object)
 (= (KIF$target quotient) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (quotient ?x)) (endorelation ?x))
 (= (SET.FTN$target (quotient ?x)) (hgph.fbr.obj$object ?x))
 (forall (?j ((endorelation ?x) ?j))
 (and (= ((hgph.fbr.obj$node ?x) ((quotient ?x) ?j))
 (rel.endo$quotient (rel.endo$equivalence-closure ((node ?x) ?j))))
 (= ((hgph.fbr.obj$edge ?x) ((quotient ?x) ?j))
 (rel.endo$quotient (rel.endo$equivalence-closure ((relation ?x) ?j))))
 (= (set.pr$set1 ((hgph.fbr.obj$reference ?x) ((quotient ?x) ?j))) ?x)
 (= (set.pr$set2 ((hgph.fbr.obj$reference ?x) ((quotient ?x) ?j)))
 ((hgph.fbr.obj$node ?x) ((quotient ?x) ?j)))
 (= (set.ftn$composition
 [(rel.endo$canon ((edge ?x) ?j))
 ((hgph.fbr.obj$edge-arity ?x) ((quotient ?x) ?j))])
 ((hgph.fbr.obj$edge-arity ?x) ((hypergraph ?x) ?j)))
 (= (set.ftn$composition
 [(rel.endo$canon ((edge ?x) ?j))
 ((hgph.fbr.obj$signature ?x) ((quotient ?x) ?j))])
 (set.ftn$composition
 [(hgph.fbr.obj$signature ?x) ((hypergraph ?x) ?j)]))
```

```

(set.sqtt$tuple
 ((hgph.fbr.mor$reference (set.mor$identity ?x))
 ((canon ?x) ?j)))))))))

(8) (KIF$function canon)
 (= (KIF$source canon) set.obj$object)
 (= (KIF$target canon) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (canon ?x)) (endorelation ?x))
 (= (SET.FTN$target (canon ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (hypergraph ?x))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (quotient ?x))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition (SET.FTN$composition
 [(edge ?x) rel.endo$equivalence-closure] rel.endo$canon)))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(SET.FTN$composition
 [(node ?x) rel.endo$equivalence-closure] rel.endo$canon)])))

```

Any hypergraph morphism  $f = \langle e, n, id \rangle = \langle edge_X(f), node_X(f), id_X \rangle : G \rightarrow G'$ , whose name function is the identity on  $X$ , defines a hypergraph endorelation  $J_f = \langle G, E_f, N_f \rangle$  over  $X$  called the *kernel* of  $f$ , where the base hypergraph is the source hypergraph, the edge endorelation is the kernel of the edge function, and the node endorelation is the kernel of the node function:

- $hgph_X(J) = G$ ;
- $edge_X(J) = E_f = \{(\varepsilon_1, \varepsilon_2) \mid edge_X(f)(\varepsilon_1) = edge_X(f)(\varepsilon_2)\}$ ; and
- $node_X(J) = N_f = \{(\varepsilon_1, \varepsilon_2) \mid node_X(f)(\varepsilon_1) = node_X(f)(\varepsilon_2)\}$ .

Let us verify the arity/signature constraints. Let  $\varepsilon_1, \varepsilon_2 \in edge(D)$  be any pair of edges in the destination hypergraph, and assume that  $(\varepsilon_1, \varepsilon_2) \in edge_X(J_{X,f})$ ; that is, assume that  $edge(f)(\varepsilon_1) = edge(f)(\varepsilon_2)$ . The commuting diagrams for the edge-arity and signature quartets of  $f$  imply the required constraints:  $arity_X(G)(\varepsilon_1) = arity_X(G)(\varepsilon_2)$  and  $sign_X(G)(\varepsilon_1) \equiv_{\partial} sign_X(G)(\varepsilon_2)$ .

This notion of a hypergraph morphism kernel gives a canonical approach (Figure 15) for an epi-mono factorization system for hypergraph morphisms. By definition, any language morphism respects its kernel. Let  $\varepsilon_f$  denote the canonical morphism of the kernel of  $f$ , and let  $\mu_f$  denote the unique mediating morphism such that  $\varepsilon_f \circ \mu_f = f$ . This is an “image factorization” of  $f$ . The kernel of any language morphism  $f$  is the “largest” hypergraph endorelation that  $f$  respects. If  $f$  respects a hypergraph endorelation  $J$ , then there is a unique language morphism  $f_J : L_X/J \rightarrow L_X/J_f$  such that  $\tau_{X,J} \circ f_J = \varepsilon_f$  and  $f_J \circ \mu_f = \tilde{f}^X$ .

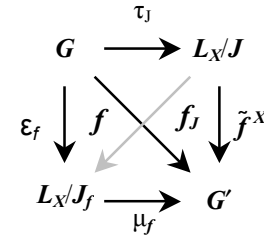


Figure 15: Factorization

```

(9) (KIF$function kernel)
 (= (KIF$source kernel) set.obj$object)
 (= (KIF$target kernel) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (kernel ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$target (kernel ?x)) (endorelation ?x))
 (forall (?f ((hgph.fbr.mor$morphism (set.mor$identity ?x)) ?f))
 (and (= ((hypergraph ?x) ((kernel ?x) ?f))
 ((hgph.fbr.mor$source (set.mor$identity ?x)) ?f))
 (= ((edge ?x) ((kernel ?x) ?f))
 (set.ftn$kernel ((hgph.fbr.mor$edge (set.mor$identity ?x)) ?f))
 (= ((node ?x) ((kernel ?x) ?f))
 (set.ftn$kernel ((hgph.fbr.mor$node (set.mor$identity ?x)) ?f))))

```

Let  $X$  be any fixed set of names, and let  $J = \langle G, E, N \rangle = \langle hgph_X(J), edge_X(J), node_X(J) \rangle$  be a hypergraph endorelation on  $X$ . A hypergraph morphism  $f = \langle edge_X(f), node_X(f), id_X \rangle : G \rightarrow K$  in the fibered category  $Hypergraph(X)$  respects  $J$  when:

- if  $(\varepsilon_0, \varepsilon_1) \in edge_X(J)$  then  $edge_X(f)(\varepsilon_0) = edge_X(f)(\varepsilon_1)$ , for any two edges  $\varepsilon_0, \varepsilon_1 \in edge_X(G)$ ; and
- if  $(\alpha_0, \alpha_1) \in node_X(J)$  then  $node_X(f)(\alpha_0) = node_X(f)(\alpha_1)$ , for any two nodes  $\alpha_0, \alpha_1 \in node_X(G)$ .

```

(10) (KIF$function matches-opspan)
 (= (KIF$source matches-opspan) set.obj$object)
 (= (KIF$target matches-opspan) SET.LIM.PBK$opspan)

```

## Robert E. Kent

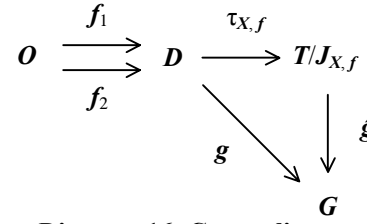
[illegible]

## Coequalizers

### hgph.col.coeq

A *coequalizer* is a finite colimit in the category **Hypergraph** for a diagram of shape *parallel-pair* =  $\bullet \rightrightarrows \bullet$ . Such a diagram is called a *parallel pair* of hypergraph morphisms.

Let  $X$  be any fixed set of names, a *parallel pair*  $f = \langle f_1, f_2 \rangle : O \rightarrow D$  of hypergraph morphisms is a pair of hypergraph morphisms sharing a common origin hypergraph  $O$  and a common destination hypergraph  $D$ . See Diagram 16, where arrows denote hypergraph morphisms.



**Diagram 16: Coequalizer of a Parallel Pair**

Let  $X$  be any fixed set of names, a *parallel pair*  $f = \langle f_1, f_2 \rangle : O \rightarrow D$  of hypergraph morphisms is the appropriate base diagram for a coequalizer. Each parallel pair consists of a pair of hypergraph morphisms called *morphism1* and *morphism2* with the identity name function  $X$  that share the same *origin* and *destination* hypergraphs. We use either the generic term ‘diagram’ or the specific term ‘parallel-pair’ to denote the parallel pair class. Parallel pairs are determined by their two component hypergraph morphisms.

- (1) (KIF\$function diagram)  
 (= (KIF\$function parallel-pair)  
 (= parallel-pair diagram)  
 (= (KIF\$source diagram) set.obj\$object)  
 (= (KIF\$target diagram) SET\$class))
- (2) (KIF\$function origin)  
 (= (KIF\$source origin) set.obj\$object)  
 (= (KIF\$target origin) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
 (and (= (SET.FTN\$source (origin ?x)) (diagram ?x))  
 (= (SET.FTN\$target (origin ?x)) (hgph.fbr.obj\$object ?x))))
- (3) (KIF\$function destination)  
 (= (KIF\$source destination) set.obj\$object)  
 (= (KIF\$target destination) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
 (and (= (SET.FTN\$source (destination ?x)) (diagram ?x))  
 (= (SET.FTN\$target (destination ?x)) (hgph.fbr.obj\$object ?x))))
- (4) (KIF\$function morphism1)  
 (= (KIF\$source morphism1) set.obj\$object)  
 (= (KIF\$target morphism1) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
 (and (= (SET.FTN\$source (morphism1 ?x)) (diagram ?x))  
 (= (SET.FTN\$target (morphism1 ?x))  
 (hgph.fbr.mor\$morphism (set.mor\$identity ?x)))  
 (= (SET.FTN\$composition  
 [(morphism1 ?x) (hgph.fbr.mor\$source (set.mor\$identity ?x))]  
 (origin ?x))  
 (= (SET.FTN\$composition  
 [(morphism1 ?x) (hgph.fbr.mor\$target (set.mor\$identity ?x))]  
 (destination ?x))))
- (5) (KIF\$function morphism2)  
 (= (KIF\$source morphism2) set.obj\$object)  
 (= (KIF\$target morphism2) SET.FTN\$function)  
 (forall (?x (set.obj\$object ?x))  
 (and (= (SET.FTN\$source (morphism2 ?x)) (diagram ?x))  
 (= (SET.FTN\$target (morphism2 ?x))  
 (hgph.fbr.mor\$morphism (set.mor\$identity ?x)))  
 (= (SET.FTN\$composition  
 [(morphism2 ?x) (hgph.fbr.mor\$source (set.mor\$identity ?x))]  
 (origin ?x))  
 (= (SET.FTN\$composition  
 [(morphism2 ?x) (hgph.fbr.mor\$target (set.mor\$identity ?x))]  
 (destination ?x))))

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 53

May 8, 2003

```
(6) (forall (?x (set.obj$object ?x)
 ?p ((diagram ?x) ?p) ?q ((diagram ?x) ?q))
 (= (and (= ((morphism1 ?x) ?p) ((morphism1 ?x) ?q))
 (= ((morphism2 ?x) ?p) ((morphism2 ?x) ?q)))
 (= ?p ?q)))
```

There are *case*, *edge* and *node* diagram functions, which map parallel pairs of hypergraph morphisms to the underlying case, edge and node parallel pairs of set functions, respectively. When the diagram is regarded as a graph morphism or functor, these are the object part of the compositions of the diagram with the case, edge and node set functors (Figure 10). In addition, there are *indication*, *comediation* and *projection* diagram morphism functions, which map parallel pairs of hypergraph morphisms to the underlying indication, comediation and projection morphism of parallel pairs, respectively. The source of indication is the case diagram and the target of indication is the edge diagram. The source of comediation is the case diagram and the target of comediation is the node diagram. The source of projection is the case diagram and the target of projection is the constant diagram for  $X$ . When the diagram is regarded as a graph morphism or functor, these are the morphism part of the compositions of the diagram with the indication, comediation and projection set natural transformations.

```
(7) (KIF$function case-diagram)
 (= (KIF$source case-diagram) set.obj$object)
 (= (KIF$target case-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (case-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (case-diagram ?x)) set.col.coeq$diagram)
 (= (SET.FTN$composition [(case-diagram ?x) set.col.coeq$origin]
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.coeq$destination]
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.coeq$function1]
 (SET.FTN$composition
 [(morphism1 ?x) (hgph.fbr.mor$case (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.coeq$function2]
 (SET.FTN$composition
 [(morphism2 ?x) (hgph.fbr.mor$case (set.mor$identity ?x))])))))

(8) (KIF$function edge-diagram)
 (= (KIF$source edge-diagram) set.obj$object)
 (= (KIF$target edge-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (edge-diagram ?x)) set.col.coeq$diagram)
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$origin]
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$destination]
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$function1]
 (SET.FTN$composition
 [(morphism1 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$function2]
 (SET.FTN$composition
 [(morphism2 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])))))

(9) (KIF$function node-diagram)
 (= (KIF$source node-diagram) set.obj$object)
 (= (KIF$target node-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (node-diagram ?x)) set.col.coeq$diagram)
 (= (SET.FTN$composition [(node-diagram ?x) set.col.coeq$origin]
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.coeq$destination]
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.coeq$function1]
 (SET.FTN$composition
 [(morphism1 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.coeq$function2]
 (SET.FTN$composition
 [(morphism2 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])))))
```

```

(10) (KIF$function indication)
 (= (KIF$source indication) set.obj$object)
 (= (KIF$target indication) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (indication ?x)) (diagram ?x))
 (= (SET.FTN$target (indication ?x)) set.col.coeq.mor$diagram-morphism)
 (= (SET.FTN$composition [(indication ?x) set.col.coeq.mor$source])
 (case-diagram ?x))
 (= (SET.FTN$composition [(indication ?x) set.col.coeq.mor$target])
 (edge-diagram ?x))
 (= (SET.FTN$composition [(indication ?x) set.col.coeq.mor$origin])
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$indication ?x)]))
 (= (SET.FTN$composition [(indication ?x) set.col.coeq$destination])
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$indication ?x)]))))))

(11) (KIF$function comediation)
 (= (KIF$source comediation) set.obj$object)
 (= (KIF$target comediation) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (comediation ?x)) (diagram ?x))
 (= (SET.FTN$target (comediation ?x)) set.col.coeq.mor$diagram-morphism)
 (= (SET.FTN$composition [(comediation ?x) set.col.coeq.mor$source])
 (case-diagram ?x))
 (= (SET.FTN$composition [(comediation ?x) set.col.coeq.mor$target])
 (node-diagram ?x))
 (= (SET.FTN$composition [(comediation ?x) set.col.coeq.mor$origin])
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$comediation ?x)]))
 (= (SET.FTN$composition [(comediation ?x) set.col.coeq$destination])
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$comediation ?x)]))))))

(12) (KIF$function projection)
 (= (KIF$source projection) set.obj$object)
 (= (KIF$target projection) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (projection ?x)) (diagram ?x))
 (= (SET.FTN$target (projection ?x)) set.col.coeq.mor$diagram-morphism)
 (= (SET.FTN$composition [(projection ?x) set.col.coeq.mor$source])
 (case-diagram ?x))
 (= (SET.FTN$composition [(projection ?x) set.col.coeq.mor$target])
 (edge-diagram ?x))
 (= (SET.FTN$composition [(projection ?x) set.col.coeq.mor$origin])
 (SET.FTN$composition [(origin ?x) (hgph.fbr.obj$projection ?x)]))
 (= (SET.FTN$composition [(projection ?x) set.col.coeq$destination])
 (SET.FTN$composition [(destination ?x) (hgph.fbr.obj$projection ?x)]))))))

```

$$\begin{array}{ccc}
 & f_1 & \\
 O & \rightrightarrows & D \\
 & f_2 & 
 \end{array}$$

**Figure 16a: Parallel Pair**  
– abstract

$$\begin{array}{ccc}
 \begin{array}{c} \text{refer}(O) \left\{ \begin{array}{ccc} X & \xrightarrow[id]{=} & X \\ & \searrow & \nearrow \\ & \text{node}(f_1) & \\ \text{node}(O) & \xrightarrow[\text{node}(f_2)]{} & \text{node}(D) \end{array} \right. & & \text{refer}(D) \end{array} & \begin{array}{ccc} \text{edge}(O) & \xrightarrow[\text{edge}(f_2)]{\text{edge}(f_1)} & \text{edge}(D) \\ \text{sign}(O) \downarrow & & \downarrow \text{sign}(D) \\ \text{tuple}(\text{refer}(O)) & \xrightarrow{\text{tuple}(\text{refer}(f_1))} & \text{tuple}(\text{refer}(D)) \\ & \xrightarrow{\text{tuple}(\text{refer}(f_2))} & \end{array}
 \end{array}$$

**Figure 16b: Parallel Pair**  
– details

Let  $X$  be any fixed set of names, and let  $f = \langle f_1, f_2 \rangle : O \rightarrow D$  (Figure 16) be a coequalizer diagram or parallel pair of hypergraph morphisms. The information in  $f$  is equivalently expressed as an *endorelation*  $J_{X,f}$  =

$\langle D, E, N \rangle = \langle D, \text{edge}_X(J_{X,f}), \text{node}_X(J_{X,f}) \rangle$  based at the destination hypergraph of the parallel pair. The edge endorelation is the coequalizer endorelation of the edge diagram

$$\text{edge}_X(J_{X,f}) = \{(\text{edge}(f_1)(\varepsilon), \text{edge}(f_2)(\varepsilon)) \mid \varepsilon \in \text{edge}(O)\}, \text{ and}$$

the node endorelation is the coequalizer endorelation of the node diagram

$$\text{node}_X(J_{X,f}) = \{(\text{node}(f_1)(v), \text{node}(f_2)(v)) \mid v \in \text{node}(O)\}.$$

The commuting diagrams for the edge-arity and signature quartets of  $f_1$  and  $f_2$  imply the arity/signature constraints for the endorelation  $J_{X,f}$ :  $\text{arity}_X(G)(\text{edge}(f_1)(\varepsilon)) = \text{arity}_X(G)(\text{edge}(f_2)(\varepsilon))$  and  $\text{sign}_X(G)(\text{edge}(f_1)(\varepsilon)) \equiv \text{sign}_X(G)(\text{edge}(f_2)(\varepsilon))$ .

```
(13) (KIF$function endorelation)
 (= (KIF$source endorelation) set.obj$object)
 (= (KIF$target endorelation) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (endorelation ?x)) (diagram ?x))
 (= (SET.FTN$target (endorelation ?x)) (hgph.col.endo$endorelation ?x))
 (= (SET.FTN$composition [(endorelation ?x) (hgph.col.endo$hypergraph ?x)])
 (destination ?x))
 (= (SET.FTN$composition [(endorelation ?x) (hgph.col.endo$edge ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$endorelation]))
 (= (SET.FTN$composition [(endorelation ?x) (hgph.col.endo$node ?x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$endorelation])))))
```

The notion of a coequalizer *cocone* is used to specify and axiomatize coequalizers. Each coequalizer cocone (see Figure 17, where arrows denote hypergraph morphisms) is situated under a coequalizer *diagram* or parallel pair of hypergraph morphisms. Each coequalizer cocone has an *opvertex* hypergraph  $C$ , and a hypergraph *morphism*  $f: A \rightarrow C$ , whose source hypergraph is the target hypergraph of the hypergraph morphisms in the underlying cocone diagram (parallel-pair) and whose target hypergraph is the opvertex. Since Figure 17 is a commutative diagram, the composite hypergraph morphism is not needed.

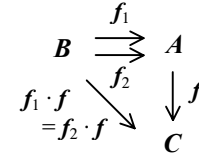


Figure 17: Coequalizer Cocone

```
(14) (KIF$function cocone)
 (= (KIF$source cocone) set.obj$object)
 (= (KIF$target cocone) SET$class)

(15) (KIF$function cocone-diagram)
 (= (KIF$source cocone-diagram) set.obj$object)
 (= (KIF$target cocone-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (cocone-diagram ?x)) (cocone ?x))
 (= (SET.FTN$target (cocone-diagram ?x)) (diagram ?x))))

(16) (KIF$function opvertex)
 (= (KIF$source opvertex) set.obj$object)
 (= (KIF$target opvertex) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opvertex ?x)) (cocone ?x))
 (= (SET.FTN$target (opvertex ?x)) (hgph.fbr.obj$object ?x))))

(17) (KIF$function morphism)
 (= (KIF$source morphism) set.obj$object)
 (= (KIF$target morphism) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (morphism ?x)) (cocone ?x))
 (= (SET.FTN$target (morphism ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition
 [(morphism ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (destination ?x)]))
 (= (SET.FTN$composition
 [(morphism ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x)))))

(18) (forall (?x (set.obj$object ?x)
 ?s ((cocone ?x) ?s)))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 56

May 8, 2003

```
(= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [(morphism1 ?x) ((cocone-diagram ?x) ?s)) ((morphism ?x) ?s)])
 ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [(morphism2 ?x) ((cocone-diagram ?x) ?s)) ((morphism ?x) ?s)]))
```

The *case* coequalizer cocone function maps a coequalizer cocone of hypergraphs and hypergraph morphisms to the underlying coequalizer cocone of case sets and case set functions. The *edge* coequalizer cocone function maps a coequalizer cocone of hypergraphs and hypergraph morphisms to the underlying coequalizer cocone of edge sets and edge set functions. The *node* coequalizer cocone function maps a coequalizer cocone of hypergraphs and hypergraph morphisms to the underlying coequalizer cocone of node sets and node set functions.

```
(19) (KIF$function case-cocone)
 (= (KIF$source case-cocone) set.obj$object)
 (= (KIF$target case-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (case-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (case-cocone ?x)) set.col.coeq$cocone)
 (= (SET.FTN$composition [(case-cocone ?x) set.col.coeq$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (case-diagram ?x)]))
 (= (SET.FTN$composition [(case-cocone ?x) set.col.coeq$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-cocone ?x) set.col.coeq$function])
 (SET.FTN$composition
 [(morphism ?x) (hgph.fbr.mor$case (set.mor$identity ?x))])))))

(20) (KIF$function edge-cocone)
 (= (KIF$source edge-cocone) set.obj$object)
 (= (KIF$target edge-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (edge-cocone ?x)) set.col.coeq$cocone)
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.coeq$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (edge-diagram ?x)]))
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.coeq$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.coeq$function])
 (SET.FTN$composition
 [(morphism ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])))))

(21) (KIF$function node-cocone)
 (= (KIF$source node-cocone) set.obj$object)
 (= (KIF$target node-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (node-cocone ?x)) set.col.coeq$cocone)
 (= (SET.FTN$composition [(node-cocone ?x) set.col.coeq$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (node-diagram ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.coeq$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.coeq$function])
 (SET.FTN$composition
 [(morphism ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])))))
```

It is important to observe that the hypergraph morphism in a cocone respects the endorelation of the underlying diagram of the cocone. This fact is needed to help define the comediator of a cocone.

```
(22) (forall (?x (set.obj$object ?x)
 ?s ((cocone ?x) ?s))
 ((hgph.col.endo$respects ?x)
 ((morphism ?x) ?s)
 (endorelation ((cocone-diagram ?x) ?s))))
```

There is a class function ‘colimiting-cone’ that maps a parallel pair of hypergraph morphisms to its coequalizer (colimiting coequalizer cocone) (see Figure 18, where arrows denote hypergraph morphisms). The totality of this function, along with the universality of the comediator hypergraph morphism, implies that a coequalizer exists for any parallel pair of hypergraph morphisms. The opvertex of the colimiting coequalizer co-

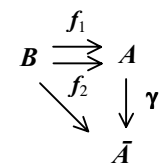


Figure 18: Colimiting Cocone



# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 57

May 8, 2003

cone is a specific *coequalizer* hypergraph. It comes equipped with a *canon* hypergraph morphism. The coequalizer and canon are expressed both abstractly, and, in the last axiom, concretely as the quotient and canon of the endorelation of the diagram.

```
(23) (KIF$function colimiting-cocone)
 (= (KIF$source colimiting-cocone) set.obj$object)
 (= (KIF$target colimiting-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimiting-cocone ?x)) (diagram ?x))
 (= (SET.FTN$target (colimiting-cocone ?x)) (cocone ?x))
 (= (SET.FTN$composition [(colimiting-cocone ?x) (cocone-diagram ?x)])
 (SET.FTN$identity (diagram ?x)))))

(24) (KIF$function colimit)
 (KIF$function coequalizer)
 (= coequalizer colimit)
 (= (KIF$source colimit) set.obj$object)
 (= (KIF$target colimit) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimit ?x)) (diagram ?x))
 (= (SET.FTN$target (colimit ?x)) (hgph.fbr.obj$object ?x))
 (= (colimit ?x)
 (SET.FTN$composition [(colimiting-cocone ?x) (opvertex ?x)])))

(25) (KIF$function canon)
 (= (KIF$source canon) set.obj$object)
 (= (KIF$target canon) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (canon ?x)) (diagram ?x))
 (= (SET.FTN$target (canon ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition
 [(canon ?x) (hgph.fbr.mor$source (set.mor$identity ?x))]
 (destination ?x))
 (= (SET.FTN$composition
 [(canon ?x) (hgph.fbr.mor$target (set.mor$identity ?x))]
 (colimit ?x))
 (= (canon ?x)
 (SET.FTN$composition [(colimiting-cocone ?x) (morphism ?x)])))

(26) (forall (?x (set.obj$object ?x))
 (and (= (colimit ?x)
 (SET.FTN$composition [(endorelation ?x) (hgph.col.endo$quotient ?x)]))
 (= (canon ?x)
 (SET.FTN$composition [(endorelation ?x) (hgph.col.endo$canon ?x)])))
```

The following three axioms are the necessary conditions that the case, edge and node functors preserve concrete colimits. These, in addition to the endorelation axiom above, ensure that both this coequalizer hypergraph and its canon hypergraph morphism are specific.

```
(27) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (case-cocone x)])
 (SET.FTN$composition [(case-diagram ?x) set.col.coeq$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$case ?x)])
 (SET.FTN$composition [(case-diagram ?x) set.col.coeq$colimit]))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$case (set.mor$identity ?x))])
 (SET.FTN$composition [(case-diagram ?x) set.col.coeq$canon]))))

(28) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (edge-cocone x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$edge ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$colimit]))
 (= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$canon]))))

(29) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (node-cocone x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$node ?x)]))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

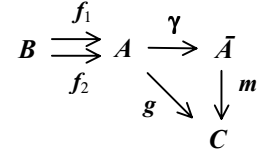
Page 58

May 8, 2003

```
(SET.FTN$composition [(node-diagram ?x) set.col.coeq$colimit]))
(= (SET.FTN$composition [(canon ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$canon]))))
```

The *comediator* hypergraph morphism, from the coequalizer of a parallel pair of hypergraph morphisms to the opvertex of a cocone under the parallel pair (see Figure 19, where arrows denote hypergraph morphisms), is the unique hypergraph morphism that commutes with cocone hypergraph morphisms. This is defined abstractly by using a definite description, and is defined concretely as the comediator of the associated (morphism, endorelation) pair.

**Figure 19: Comediator**



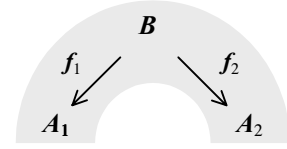
```
(30) (KIF$function comediator)
 (= (KIF$source comediator) set.obj$object)
 (= (KIF$target comediator) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (comediator ?x)) (cocone ?x))
 (= (SET.FTN$target (comediator ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (colimit ?x)]))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))
 (forall (?s ((cocone ?x) ?s))
 (= ((comediator ?x) ?s)
 (the (?m ((hgph.fbr.mor$morphism (set.mor$identity ?x)) ?m))
 (= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [((canon ?x) ((cocone-diagram ?x) ?s)) ?m])
 ((morphism ?x) ?s)))))))

(31) (forall (?x (set.obj$object ?x)
 ?s ((cocone ?x) ?s))
 (= ((comediator ?x) ?s)
 ((hgph.col.endo$comediator ?x)
 [((morphism ?x) ?s) ((endorelation ?x) ((cocone-diagram ?x) ?s))])))
```

## Pushouts

### hgph.col.psh

A *pushout* is a finite colimit in the category **Hypergraph** for a diagram of shape  $\text{span} = \bullet \leftarrow \bullet \rightarrow \bullet$ . Such a diagram of hypergraphs and hypergraph morphisms is called a *span* (see Figure 20, where arrows denote hypergraph morphisms)



**Figure 20: Pushout Diagram**  
= Span

A *span* of hypergraphs and hypergraph morphisms is the appropriate base diagram for a pushout. Each span consists of a pair of hypergraph morphisms called *first* and *second*. These are required to have a common source hypergraph called the *vertex*. The target hypergraphs for first and second are given the names *hypergraph1* and *hypergraph2*. We use either the generic term ‘*diagram*’ or the specific term ‘*span*’ to denote the span class. A span is the special case of a general diagram whose shape is the span graph. Spans are determined by their pair of component hypergraph morphisms.

```
(1) (KIF$function diagram)
 (= (KIF$function span)
 (= span diagram)
 (= (KIF$source diagram) set.obj$object)
 (= (KIF$target diagram) SET$class))

(2) (KIF$function hypergraph1)
 (= (KIF$source hypergraph1) set.obj$object)
 (= (KIF$target hypergraph1) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph1 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph1 ?x)) (hgph.fbr.obj$object ?x))))

(3) (KIF$function hypergraph2)
 (= (KIF$source hypergraph2) set.obj$object)
 (= (KIF$target hypergraph2) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph2 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph2 ?x)) (hgph.fbr.obj$object ?x))))

(4) (KIF$function vertex)
 (= (KIF$source vertex) set.obj$object)
 (= (KIF$target vertex) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (vertex ?x)) (diagram ?x))
 (= (SET.FTN$target (vertex ?x)) (hgph.fbr.obj$object ?x))))

(5) (KIF$function first)
 (= (KIF$source first) set.obj$object)
 (= (KIF$target first) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (first ?x)) (diagram ?x))
 (= (SET.FTN$target (first ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(first ?x) (hgph.fbr.mor$source (set.mor$identity ?x))]
 (vertex ?x))
 (= (SET.FTN$composition [(first ?x) (hgph.fbr.mor$target (set.mor$identity ?x))]
 (hypergraph1 ?x)))))

(6) (KIF$function second)
 (= (KIF$source second) set.obj$object)
 (= (KIF$target second) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (second ?x)) (diagram ?x))
 (= (SET.FTN$target (second ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(second ?x) (hgph.fbr.mor$source (set.mor$identity ?x))]
 (vertex ?x))
 (= (SET.FTN$composition [(second ?x) (hgph.fbr.mor$target (set.mor$identity ?x))]
 (hypergraph2 ?x)))))

(7) (forall (?x (set.obj$object ?x))
 ?d1 ((diagram ?x) ?d1) ?d2 ((diagram ?x) ?d2))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 60

May 8, 2003

```
(=> (and (= ((first ?x) ?d1) ((first ?x) ?d2))
 (= ((second ?x) ?d1) ((second ?x) ?d2)))
 (= ?d1 ?d2)))
```

The *pair* of target hypergraphs (sufficing discrete diagram) underlying any span of hypergraphs is named. This construction is derived from the fact that the pair shape is a subshape of the span shape.

```
(8) (KIF$function pair)
 (= (KIF$source pair) set.obj$object)
 (= (KIF$target pair) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (pair ?x)) (diagram ?x))
 (= (SET.FTN$target (pair ?x)) (hgph.col.copr2$diagram ?x))
 (= (SET.FTN$composition [(pair ?x) (hgph.col.copr2$hypergraph1 ?x)])
 (hypergraph1 ?x))
 (= (SET.FTN$composition [(pair ?x) (hgph.col.copr2$hypergraph2 ?x)])
 (hypergraph2 ?x))))
```

Every span has an *opposite*.

```
(9) (KIF$function opposite)
 (= (KIF$source opposite) set.obj$object)
 (= (KIF$target opposite) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opposite ?x)) (diagram ?x))
 (= (SET.FTN$target (opposite ?x)) (diagram ?x))
 (= (SET.FTN$composition [(opposite ?x) (hypergraph1 ?x)]) (hypergraph2 ?x))
 (= (SET.FTN$composition [(opposite ?x) (hypergraph2 ?x)]) (hypergraph1 ?x))
 (= (SET.FTN$composition [(opposite ?x) (vertex ?x)]) (vertex ?x))
 (= (SET.FTN$composition [(opposite ?x) (first ?x)]) (second ?x))
 (= (SET.FTN$composition [(opposite ?x) (second ?x)]) (first ?x))))
```

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

```
(10) (forall (?x (set.obj$object ?x))
 (= (SET.FTN$composition [(opposite ?x) (opposite ?x)])
 (SET.FTN$identity (diagram ?x))))
```

The *case span* or *case diagram* function maps a diagram of hypergraphs and hypergraph morphisms to the underlying set colimit pushout diagram of case sets and set functions. The *edge span* or *edge diagram* function and the *node span* or *node diagram* function are similar.

```
(11) (KIF$function case-diagram)
 (= (KIF$source case-diagram) set.obj$object)
 (= (KIF$target case-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (case-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (case-diagram ?x)) set.col.psh$diagram)
 (= (SET.FTN$composition [(case-diagram ?x) set.col.psh$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.psh$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.psh$vertex])
 (SET.FTN$composition [(vertex ?x) (hgph.fbr.obj$case ?x)]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.psh$first])
 (SET.FTN$composition [(first ?x) (hgph.fbr.mor$case (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(case-diagram ?x) set.col.psh$second])
 (SET.FTN$composition [(second ?x) (hgph.fbr.mor$case (set.mor$identity ?x))]))))

(12) (KIF$function edge-diagram)
 (= (KIF$source edge-diagram) set.obj$object)
 (= (KIF$target edge-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (edge-diagram ?x)) set.col.psh$diagram)
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$vertex])
 (SET.FTN$composition [(vertex ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$first])
 (SET.FTN$composition [(edge-diagram ?x) set.col.psh$first]))))
```

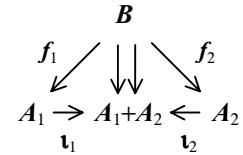
```

 (SET.FTN$composition [(first ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$second])
 (SET.FTN$composition [(second ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])))

(13) (KIF$function node-diagram)
 (= (KIF$source node-diagram) set.obj$object)
 (= (KIF$target node-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (node-diagram ?x)) set.col.psh$diagram)
 (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$vertex])
 (SET.FTN$composition [(vertex ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$first])
 (SET.FTN$composition [(first ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$second])
 (SET.FTN$composition [(second ?x) (hgph.fbr.mor$node (set.mor$identity ?x))]))))

```

The *parallel pair* or *coequalizer diagram* function maps a hypergraph pushout diagram to the associated hypergraph coequalizer diagram. See Figure 21, where arrows denote hypergraph morphisms. The parallel pair of hypergraph morphisms in this coequalizer diagram is the composite of the first and second hypergraph morphisms of the pushout diagram with the coproduct injection hypergraph morphisms for the binary coproduct of the pair of component hypergraphs in the pushout diagram. The coequalizer and canon of the coequalizer diagram will be used to give a specific definition for the pushout.



**Figure 21: Coequalizer Diagram**

```

(14) (KIF$function coequalizer-diagram)
 (KIF$function parallel-pair)
 (= parallel-pair coequalizer-diagram)
 (= (KIF$source coequalizer-diagram) set.obj$object)
 (= (KIF$target coequalizer-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (coequalizer-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (coequalizer-diagram ?x)) (hgph.col.coeq$diagram ?x))
 (= (SET.FTN$composition [(coequalizer-diagram ?x) (hgph.col.coeq$origin ?x)]
 (vertex ?x))
 (= (SET.FTN$composition [(coequalizer-diagram ?x) (hgph.col.coeq$destination ?x)]
 (SET.FTN$composition [(pair ?x) (hgph.col.copr2$binary-coproduct ?x)]))
 (forall (?d ((diagram ?x) ?d))
 (and (= ((hgph.col.coeq$morphism1 ?x) ((coequalizer-diagram ?x) ?d))
 ((hgph.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [((first ?x) ?d) ((hgph.col.copr2$injection1 ?x) ((pair ?x) ?d)]))
 (= ((hgph.col.coeq$morphism2 ?x) ((coequalizer-diagram ?x) ?d))
 ((hgph.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [((second ?x) ?d) ((hgph.col.copr2$injection2 ?x) ((pair ?x) ?d)])))))

```

Here we assert three important categorical identities (not just isomorphisms). For any pushout diagram (in Hypergraph), these relate the case, edge or node diagrams (in Set) of the coequalizer diagram (in Hypergraph) to the coequalizer diagram (in Set) of the vertex, edge or node diagrams of the pushout diagram, respectively. These identities are assumed in the definition of the colimiting cocone below.

```

(15) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(coequalizer-diagram ?x) (hgph.col.coeq$case-diagram ?x)]
 (SET.FTN$composition [(case-diagram ?x) set.col.coeq$coequalizer-diagram]))
 (= (SET.FTN$composition [(coequalizer-diagram ?x) (hgph.col.coeq$edge-diagram ?x)]
 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$coequalizer-diagram]))
 (= (SET.FTN$composition [(coequalizer-diagram ?x) (hgph.col.coeq$node-diagram)]
 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$coequalizer-diagram]))))

(16) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(SET.FTN$composition
 [(coequalizer-diagram ?x)
 (hgph.col.coeq$colimiting-cocone ?x)]
 (hgph.col.coeq$case-diagram ?x)]
 (SET.FTN$composition [(SET.FTN$composition

```

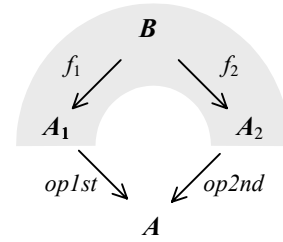
```

((case-diagram ?x)
 set.col.psh$coequalizer-diagram))
 set.col.coeq$colimiting-cocone]))
(= (SET.FTN$composition [(SET.FTN$composition
 [(coequalizer-diagram ?x)
 (hgph.col.coeq$colimiting-cocone ?x)])
 (hgph.col.coeq$edge-diagram ?x)])
 (SET.FTN$composition [(SET.FTN$composition
 [(edge-diagram ?x)
 set.col.psh$coequalizer-diagram))
 set.col.coeq$colimiting-cocone]))
(= (SET.FTN$composition [(SET.FTN$composition
 [(coequalizer-diagram ?x)
 (hgph.col.coeq$colimiting-cocone ?x)])
 (hgph.col.coeq$node-diagram ?x)])
 (SET.FTN$composition [(SET.FTN$composition
 [(node-diagram ?x)
 set.col.psh$coequalizer-diagram))
 set.col.coeq$colimiting-cocone]]))

```

*Pushout cocones* are used to specify and axiomatize pushouts. Each pushout cocone (see Figure 22, where arrows denote hypergraph morphisms) has the following constituents:

- an overlying *diagram* (the shaded part of Figure 22),
- an *opvertex* hypergraph  $A$ , and
- a pair of hypergraph morphisms called *opfirst* and *opsecond*.



**Figure 22: Pushout Cocone**

The common target hypergraph of *opfirst* and *opsecond* is the *opvertex*. The source hypergraphs of *opfirst* and *opsecond* are the target hypergraphs of the hypergraph morphisms in the pushout diagram. The *opfirst* and *opsecond* hypergraph morphisms form a commutative diagram with the overlying pushout diagram. A pushout cocone is the very special case of a colimiting cocone under a pushout diagram. The term ‘cocone’ denotes the pushout cocone class. The term ‘cocone-diagram’ represents the underlying pushout diagram.

```

(17) (KIF$function cocone)
 (= (KIF$source cocone) set.obj$object)
 (= (KIF$target cocone) SET$class)

(18) (KIF$function cocone-diagram)
 (= (KIF$source cocone-diagram) set.obj$object)
 (= (KIF$target cocone-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (cocone-diagram ?x)) (cocone ?x))
 (= (SET.FTN$target (cocone-diagram ?x)) (diagram ?x))))

(19) (KIF$function opvertex)
 (= (KIF$source opvertex) set.obj$object)
 (= (KIF$target opvertex) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opvertex ?x)) (cocone ?x))
 (= (SET.FTN$target (opvertex ?x)) (hgph.fbr.obj$object ?x))))

(20) (KIF$function opfirst)
 (= (KIF$source opfirst) set.obj$object)
 (= (KIF$target opfirst) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opfirst ?x)) (cocone ?x))
 (= (SET.FTN$target (opfirst ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (hypergraph1 ?x)]))
 (= (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x)))))

(21) (KIF$function opsecond)
 (= (KIF$source opsecond) set.obj$object)
 (= (KIF$target opsecond) SET.FTN$function)
 (forall (?x (set.obj$object ?x))

```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 63

May 8, 2003

```
(and (= (SET.FTN$source (opsecond ?x)) (cocone ?x))
 (= (SET.FTN$target (opsecond ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (hypergraph2 ?x)]))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))))

(22) (forall (?x (set.obj$object ?x)
 ?s ((cocone ?x) ?s))
 (= ((hgph.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [(first ?x) ((cocone-diagram ?x) ?s)]) ((opfirst ?x) ?s))
 ((hgph.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(second ?x) ((cocone-diagram ?x) ?s)]) ((opsecond ?x) ?s)))))
```

The *binary-coproduct cocone* underlying any cocone (pushout diagram) is named.

```
(23) (KIF$function binary-coproduct-cocone)
 (= (KIF$source binary-coproduct-cocone) set.obj$object)
 (= (KIF$target binary-coproduct-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (binary-coproduct-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (binary-coproduct-cocone ?x)) (hgph.col.coprd2$cocone ?x))
 (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (hgph.col.coprd2$cocone-diagram ?x)])
 (SET.FTN$composition [(cocone-diagram ?x) (pair ?x)]))
 (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (hgph.col.coprd2$opvertex ?x)])
 (opvertex ?x))
 (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (hgph.col.coprd2$opfirst ?x)])
 (opfirst ?x))
 (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (hgph.col.coprd2$opsecond ?x)])
 (opsecond ?x)))))
```

The *coequalizer cocone* function maps a hypergraph pushout cocone to the associated hypergraph coequalizer cocone. See Figure 23, where arrows denote hypergraph morphisms. The diagram overlying the coequalizer cocone is the coequalizer diagram associated with the overlying pushout diagram. The opvertex of the coequalizer cocone is the opvertex. The hypergraph morphism of the coequalizer cocone is the binary coproduct comediator of the opfirst and opsecond hypergraph morphisms with respect to the binary coproduct pair diagram of the cocone.

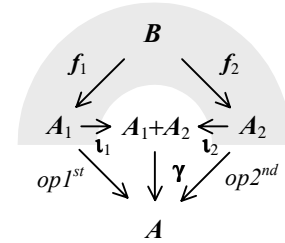


Figure 23: Coequalizer Cocone

This is the first step in the definition of the pushout of a cocone. The following string of equalities using the notation in Figure 23 demonstrates that this cocone is well defined.

$$f_1 \cdot u_1 \cdot \gamma = f_1 \cdot op1^{st} = f_2 \cdot op2^{nd} = f_2 \cdot u_2 \cdot \gamma$$

```
(24) (KIF$function coequalizer-cocone)
 (= (KIF$source coequalizer-cocone) set.obj$object)
 (= (KIF$target coequalizer-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (coequalizer-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (coequalizer-cocone ?x)) (hgph.col.coeq$cocone ?x))
 (= (SET.FTN$composition [(coequalizer-cocone ?x) (hgph.col.coeq$cocone-diagram ?x)])
 (SET.FTN$composition [(cocone-diagram ?x) (coequalizer-diagram ?x)]))
 (= (SET.FTN$composition [(coequalizer-cocone ?x) (hgph.col.coeq$opvertex ?x)])
 (opvertex ?x))
 (= (SET.FTN$composition [(coequalizer-cocone ?x) (hgph.col.coeq$morphism ?x)])
 (SET.FTN$composition [(binary-coproduct-cocone ?x) (hgph.col.coprd2$comediator ?x)])))))
```

The *case cocone* function maps a hypergraph pushout cocone to the underlying set pushout cocone. The *edge cocone* function and the *node cocone* function are defined similarly.

```
(25) (KIF$function case-cocone)
 (= (KIF$source case-cocone) set.obj$object)
 (= (KIF$target case-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (case-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (case-cocone ?x)) set.col.psh$cocone)
 (= (SET.FTN$composition [(case-cocone ?x) set.col.psh$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (case-diagram ?x)]))
```

```

(= (SET.FTN$composition [(case-cocone ?x) set.col.psh$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$case ?x)]))
(= (SET.FTN$composition [(case-cocone ?x) set.col.psh$opfirst])
 (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$case (set.mor$identity ?x))]))
(= (SET.FTN$composition [(case-cocone ?x) set.col.psh$opsecond])
 (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$case (set.mor$identity ?x))]))))

(26) (KIF$function edge-cocone)
 (= (KIF$source edge-cocone) set.obj$object)
 (= (KIF$target edge-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (edge-cocone ?x)) set.col.psh$cocone)
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (edge-diagram ?x)]))
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$opfirst])
 (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$opsecond])
 (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])))))

(27) (KIF$function node-cocone)
 (= (KIF$source node-cocone) set.obj$object)
 (= (KIF$target node-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-cocone ?x)) (cocone ?x))
 (= (SET.FTN$target (node-cocone ?x)) set.col.psh$cocone)
 (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$cocone-diagram])
 (SET.FTN$composition [(cocone-diagram ?x) (node-diagram ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$opvertex])
 (SET.FTN$composition [(opvertex ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$opfirst])
 (SET.FTN$composition [(opfirst ?x) (hgph.fbr.mor$node (set.mor$identity ?x))]))
 (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$opsecond])
 (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])))))

```

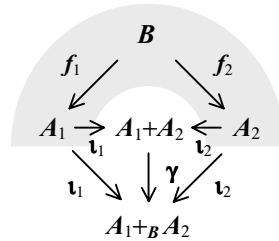


Figure 24: Colimiting Cocone

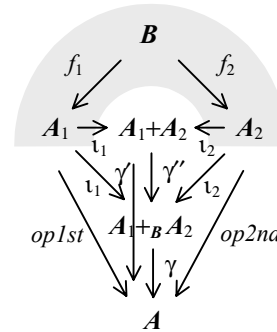


Figure 25: Comediator

The *colimiting cocone* function maps a pushout diagram to its pushout (colimiting pushout cocone). See Figure 24, where arrows denote hypergraph morphisms. For convenience of reference, we define three terms that represent the components of this colimiting pushout cocone. The *opvertex* of the pushout cocone is a specific *pushout* hypergraph, which comes equipped with two *injection* hypergraph morphisms. The last axiom expresses concreteness of the colimit – it expresses pushouts in terms of coproducts and coequalizers (Figure 25): the colimit of the coequalizer diagram is (not just isomorphic but) equal to the pushout; likewise, the compositions of the binary coproduct injections of the binary coproduct pair diagram with the canon of the coequalizer diagram are equal to the pushout injections.

```

(28) (KIF$function colimiting-cocone)
 (= (KIF$source colimiting-cocone) set.obj$object)
 (= (KIF$target colimiting-cocone) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimiting-cocone ?x)) (diagram ?x))

```



# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 65

May 8, 2003

```
(= (SET.FTN$target (colimiting-cocone ?x)) (cocone ?x))
(= (SET.FTN$composition [(colimiting-cocone ?x) (cocone-diagram ?x)])
 (SET.FTN$identity (diagram ?x))))

(29) (KIF$function colimit)
 (KIF$function pushout)
 (= pushout colimit)
 (= (KIF$source colimit) set.obj$object)
 (= (KIF$target colimit) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (colimit ?x)) (diagram ?x))
 (= (SET.FTN$target (colimit ?x)) (hgph.fbr.obj$object ?x))
 (= (colimit ?x) (SET.FTN$composition [(colimiting-cocone ?x) (opvertex ?x)])))

(30) (KIF$function injection1)
 (= (KIF$source injection1) set.obj$object)
 (= (KIF$target injection1) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (injection1 ?x)) (diagram ?x))
 (= (SET.FTN$target (injection1 ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (hypergraph1 ?x))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (colimit ?x))
 (= (injection1 ?x) (SET.FTN$composition [(colimiting-cocone ?x) (opfirst ?x)]))))

(31) (KIF$function injection2)
 (= (KIF$source injection2) set.obj$object)
 (= (KIF$target injection2) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (injection2 ?x)) (diagram ?x))
 (= (SET.FTN$target (injection2 ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (hypergraph2 ?x))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (colimit ?x))
 (= (injection2 ?x) (SET.FTN$composition [(colimiting-cocone ?x) (opsecond ?x)]))))

(32) (forall (?x (set.obj$object ?x)
 ?d ((diagram ?x) ?d))
 (and (= (colimit ?d)
 ((hgph.col.coeq$coequalizer ?x) ((coequalizer-diagram ?x) ?d)))
 (= ((injection1 ?x) ?d)
 ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(hgph.col.copr2$injection1 ?x) ((pair ?x) ?d)]
 ((hgph.col.coeq$canon ?x) ((coequalizer-diagram ?x) ?d)])))
 (= ((injection2 ?x) ?d)
 ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(hgph.col.copr2$injection2 ?x) ((pair ?x) ?d)]
 ((hgph.col.coeq$canon ?x) ((coequalizer-diagram ?x) ?d)]))))

The following three axioms are the necessary conditions that the case, edge and node functors preserve
concrete colimits. These, in addition to the coequalizer axiom above, ensure that both this pushout and its
two pushout injection hypergraph morphisms are specific.
```

```
(33) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (case-cocone ?x)])
 (SET.FTN$composition [(case-diagram ?x) set.col.psh$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$case ?x)])
 (SET.FTN$composition [(case-diagram ?x) set.col.psh$colimit]))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$case (set.mor$identity ?x))])
 (SET.FTN$composition [(case-diagram ?x) set.col.psh$injection1]))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$case (set.mor$identity ?x))])
 (SET.FTN$composition [(case-diagram ?x) set.col.psh$injection2]))))

(34) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (edge-cocone ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.psh$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$edge ?x)])
 (SET.FTN$composition [(edge-diagram ?x) set.col.psh$colimit]))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-diagram ?x) set.col.psh$edge]))))
```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 66

May 8, 2003

```
(SET.FTN$composition [(edge-diagram ?x) set.col.psh$injection1]))
(= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))])
 (SET.FTN$composition [(edge-diagram ?x) set.col.psh$injection2]))))

(35) (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$composition [(colimiting-cocone ?x) (node-cocone ?x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.psh$colimiting-cocone]))
 (= (SET.FTN$composition [(colimit ?x) (hgph.fbr.obj$node ?x)])
 (SET.FTN$composition [(node-diagram ?x) set.col.psh$colimit]))
 (= (SET.FTN$composition [(injection1 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-diagram ?x) set.col.psh$injection1]))
 (= (SET.FTN$composition [(injection2 ?x) (hgph.fbr.mor$node (set.mor$identity ?x))])
 (SET.FTN$composition [(node-diagram ?x) set.col.psh$injection2]))))
```

The *comediator* hypergraph morphism, from the colimit of the overlying diagram of a pushout cocone to the opvertex of the cocone (see Figure 25, where arrows denote hypergraph morphisms), is the unique hypergraph morphism that commutes with opfirst and opsecond. This is defined abstractly by using a definite description, and is defined concretely as the comediator of coequalizer cocone.

```
(36) (KIF$function comediator)
 (= (KIF$source comediator) set.obj$object)
 (= (KIF$target comediator) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (comediator ?x)) (cocone ?x))
 (= (SET.FTN$target (comediator ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (colimit ?x)]))
 (= (SET.FTN$composition [(comediator ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x)))
 (forall (?s ((cocone ?x) ?s))
 (= ((comediator ?x) ?s)
 (the (?m ((hgph.fbr.mor$morphism (set.mor$identity ?x)) ?m))
 (and (= ((hgph.fbr.mor$composition
 [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(injection1 ?x) ((cocone-diagram ?x) ?s)) ?m])
 ((opfirst ?x) ?s))
 (= ((hgph.fbr.mor$composition
 [(set.mor$identity ?x) (set.mor$identity ?x)]
 [(injection2 ?x) ((cocone-diagram ?x) ?s)) ?m])
 ((opsecond ?x) ?s))))))))))


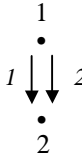
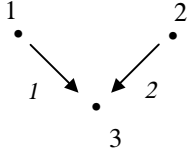
(37) (forall (?x (set.obj$object ?x))
 (= (comediator ?x)
 (SET.FTN$composition [(coequalizer-cocone ?x) (hgph.col.coeq$comediator ?x)])))
```

## Limits for Hypergraphs

`hgph.lim`

Limits of hypergraphs are basic. They underlie the colimit construction for models and logics. Thus, they are used for constructing the *instance pole* of object-level ontologies. As demonstrated below, since  $\text{Hypergraph}(X)$  has all products and equalizers, it has all limits – it is complete. For any set bijection  $h : X \rightarrow Y$  representing a fixed function of names, the categories  $\text{Hypergraph}(X)$  and  $\text{Hypergraph}(Y)$  are isomorphic, and hence have isomorphic limits. For any set  $X$  representing a fixed set of names, the category  $\text{Spangraph}(X)$  is categorically equivalent to the category  $\text{Hypergraph}(X)$ , and hence is also complete.

**Table 3: Shapes for Diagrams**

|                            |                                                                                   |                                                                                   |                                                                                    |
|----------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
|                            |  |  |  |
| <i>empty</i><br>(terminal) | <i>two</i><br>(binary product)                                                    | <i>parallel pair</i><br>(equalizer)                                               | <i>opspan</i><br>(pullback)                                                        |

We axiomatize limits for finite diagrams  $D : G \rightarrow |\text{Hypergraph}(X)|$  of the following diagram shape graphs  $G$  (Table 3): (i) *empty*, (ii) *two*, (iii) *parallel-pair* and (iv) *opspan*. These limits are called (i) the final (terminal) hypergraph, (ii) a binary product, (iii) an equalizer, and (iv) a pullback.

The extension functor of a diagram  $D$  of a particular shape  $G$  in the fiber category  $\text{Hypergraph}(X)$  can be composed with the case, edge and node component functors, and also with the indication, comediation and projection component natural transformations. These compositions result in corresponding case, edge and node component diagrams in  $\text{Set}$ , and indication, comediation and projection component diagram morphisms. Composition with the name functor results in the constant functor  $\Delta(X) : \text{path}(G) \rightarrow \text{Set}$ . These components will be axiomatized in the namespaces for each shape. The limits of various shapes are axiomatized both abstractly and concretely. The concrete axiomatization is this component axiomatization. That is, limits for hypergraphs are concrete, and defined in terms of limits for their component diagrams and diagram morphisms.

## Final (Terminal) Hypergraph

$$\begin{array}{ccccc}
 \wp X & \xleftarrow{\#_{X,G}} & \text{edge}_X(G) & \xrightarrow{\partial_{X,G}} & \text{tuple}(\text{refer}_X(G)) & \text{node}_X(G) & X \\
 \text{id} \downarrow & & \downarrow \#_{X,G} & & \downarrow \text{tuple}(\text{refer}(!_X G)) & \downarrow !_X \text{node}(G) & \downarrow \text{id} \\
 \wp X & \xleftarrow{\text{id}} & \wp X & \xrightarrow{\text{tuple}(!_X)} & \text{tuple}(\text{refer}_X(1_X)) \cong \wp X & 1 & X
 \end{array}$$

**Figure 26: Final (Terminal) Hypergraph and Unique Hypergraph Morphism**

For any set  $X$  representing a fixed set of variables, there is a special hypergraph  $1_X$  (see Figure 26, where arrows denote functions) called the *terminal (final) hypergraph*, which has the required set of variables  $X$ , the power set  $\wp X$  as edges and one node:

- the set of *edges*  $\text{edge}(1_X) = \wp X$ ;
- the set of *nodes*  $\text{node}(1_X) = 1$ ; and
- the set of *names* is  $\text{name}(1_X) = X$ .

It has the following reference set pair, and signature

- the *reference* set pair  $*_{1X} = \langle X, 1 \rangle$ ;
- the *signature* function (bijection)  $\partial_{1X} : \wp X \rightarrow \wp X$  where  $\text{tuple}(!_X)(Y)(y) = !_Y(y)$  for all  $Y \in \wp X$ ;
- the *edge-arity* function  $\#_{1X} = \text{id}_{\wp X} : \wp X \rightarrow \wp X$ .

In the set pair namespace:

- Add the two bijections, *arity* and *signature*, which establish the isomorphism  $\text{tuple}(\text{refer}(1_X)) \cong \wp X$ .
- One of these functions  $\partial_{1X} : \wp X \rightarrow \wp X$  serves as signature function for the terminal hypergraph.

The terminal hypergraph  $1_X$  in the fibered category  $\text{Hypergraph}(X)$  is the “last” hypergraph in the following sense: for any hypergraph  $G$  in  $\text{Hypergraph}(X)$  there is a *unique* hypergraph morphism  $!_{X,G} : G \rightarrow 1_X$  in  $\text{Hypergraph}(X)$ , (see Figure 1, where arrows denote functions) the unique hypergraph morphism from  $G$  to  $1_X$  with identity name function on  $X$ . The edge function of this hypergraph morphism is the edge arity function of the hypergraph  $G$ . The node function of this hypergraph morphism is the unique function (the constant function) from the node set  $\text{node}(G)$  to the terminal set 1.

- The unique (empty) *edge* function is  $\text{edge}(!_X G) = \#_{X,G} : \text{edge}(G) \rightarrow \wp X$ ; and
- The unique (empty) *node* function is  $\text{node}(!_X G) = !_X \text{node}(G) : \text{node}(G) \rightarrow 1$ .

The function  $\text{tuple}(\text{refer}(!_X G)) : \text{tuple}(\text{refer}(G)) \rightarrow \text{tuple}(\text{refer}(1_X)) \cong \wp X$  maps any tuple to the tuple with the same arity, which is constant at all components. This function is isomorphic to the tuple-arity function  $\text{tuple}(\text{refer}(G)) \rightarrow \wp X$ , which maps a tuple to its arity; in fact, it is the composition of the tuple-arity function with the signature bijection  $\partial_{1X}$ . The reference and signature quartets for the counique hypergraph morphism are trivial.

```

(1) (SET.FTN$function terminal)
 (SET.FTN$function final)
 (= final terminal)
 (= (SET.FTN$source terminal) set.obj$object)
 (= (SET.FTN$target terminal) hgph.obj$object)
 (forall (?x (set.obj$object ?x))
 (and ((hgph.fbr.obj$object ?x) (terminal ?x))
 (= (hgph.obj$edge (terminal ?x)) (set$power ?x))
 (= (hgph.obj$node (terminal ?x)) set.lim$terminal)
 (= (hgph.obj$name (terminal ?x)) ?x)
 (= (set.pr$set1 (hgph.obj$reference (terminal ?x))) ?x)
 (= (set.pr$set2 (hgph.obj$reference (terminal ?x))) set.lim$terminal)
 (= (hgph.obj$signature (terminal ?x)) (set.pr$signature ?x))
 (= (hgph.obj$edge-arity (terminal ?x))
 (set.mor$identity (set$power ?x))))))

(2) (KIF$function unique)
 (= (KIF$source unique) set.obj$object)
 (= (KIF$target unique) SET.FTN$function)

```

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 69

May 8, 2003

```
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (unique ?x)) (hgph.fbr.obj$object ?x))
 (= (SET.FTN$target (unique ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(unique ?x) (hgph.fbr.mor$source (set.mor$identity ?x))]
 (SET.FTN$identity (hgph.fbr.obj$object ?x)))
 (= (SET.FTN$composition [(unique ?x) (hgph.fbr.mor$target (set.mor$identity ?x))]
 (terminal ?x))
 (= (SET.FTN$composition [(unique ?x) (hgph.fbr.mor$edge (set.mor$identity ?x))]
 (hgph.fbr.obj$edge-arity ?x))
 (= (SET.FTN$composition [(unique ?x) (hgph.fbr.mor$node (set.mor$identity ?x))]
 (SET.FTN$composition [(hgph.fbr.obj$node ?x) set.col$counique]])))))
```

## Binary Products

### hgph.lim.prd2

Binary products do not exist for an arbitrary pair of hypergraphs; instead, hypergraphs need to be restricted to a subcategory of hypergraphs with name sets that are in bijection. For example, any two hypergraphs in the subcategory with countable name sets can be producted. More precisely, the subcategory **Hypergraph**( $\alpha$ ) of all hypergraphs whose name set has cardinality  $\alpha$  is complete. We simplify the discussion of products by assuming all hypergraphs under discussion have the same name set  $X$ ; that is, instead of considering **Hypergraph**( $\alpha$ ), we only consider the fiber category **Hypergraph**( $X$ ) for some set of names  $X$ . This is a subcategory **Hypergraph**( $X$ )  $\subseteq$  **Hypergraph**( $\alpha$ ) for  $\|X\| = \alpha$ . For the countable case  $\|X\| = \aleph$ , the set of natural numbers  $X = \text{natno}$  would work. We then argue that all **Hypergraph**( $\alpha$ ) are partition into isomorphic fiber categories.

The following goes just after the limiting cone section, in order to aid in the concrete specification of the limit.

There is a *pairing* hypergraph morphism

$$pr_{G_1, G_2} : \text{tuple}(G_1) \times_X \text{tuple}(G_2) \rightarrow \text{tuple}(G_1 \times_X G_2)$$

from the product of the tuples of two hypergraphs to the tuple of the product of the two hypergraphs.

There is an *unpairing*

$$unpr_{G_1, G_2} : \text{tuple}(G_1 \times_X G_2) \rightarrow \text{tuple}(G_1) \times_X \text{tuple}(G_2)$$

hypergraph morphism from the tuple of the product of the two hypergraphs to the product of the tuples of two hypergraphs.

Pairing is inverse to unpairing (Figure 27).

```
(--) (KIF$function pairing)
 (= (KIF$source pairing) set.obj$object)
 (= (KIF$target pairing) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (pairing ?x)) (diagram ?x))
 (= (SET.FTN$target (pairing ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(pairing ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(tuple-diagram ?x) (binary-product ?x)]))
 (= (SET.FTN$composition [(pairing ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (SET.FTN$composition [(binary-product ?x) (hgph.fbr.obj$tuple ?x)]))
 (= (SET.FTN$composition [(pairing ?x) (hgph.fbr.mor$reference (set.mor$identity ?x))])
 (SET.FTN$composition [(SET.FTN$composition [
 (reference-diagram ?x)
 (set.sqtt.lim.prd2$binary-product ?x)]
 (set.sqtt.fbr.mor$identity ?x)])))

 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (unpairing ?x)) (diagram ?x))
 (= (SET.FTN$target (unpairing ?x)) (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(pairing ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(binary-product ?x) (hgph.fbr.obj$tuple ?x)]))
 (= (SET.FTN$composition [(pairing ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (SET.FTN$composition [(tuple-diagram ?x) (binary-product ?x)])))

 (forall (?x (set.obj$object ?x))
 ?p ((diagram ?x) ?p))
 (and (= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
 [(pairing ?x) ?p] ((unpairing ?x) ?p))
 ((hgph.fbr.mor$identity ?x)
 ((binary-product ?x) ((tuple-diagram ?x) ?p))))
 (= ((hgph.fbr.mor$composition [(set.mor$identity ?x) (set.mor$identity ?x)])
```

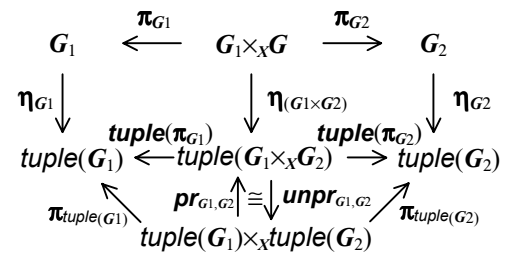
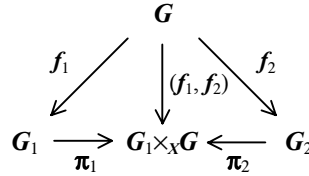


Figure 27: Binary Product & Tuple

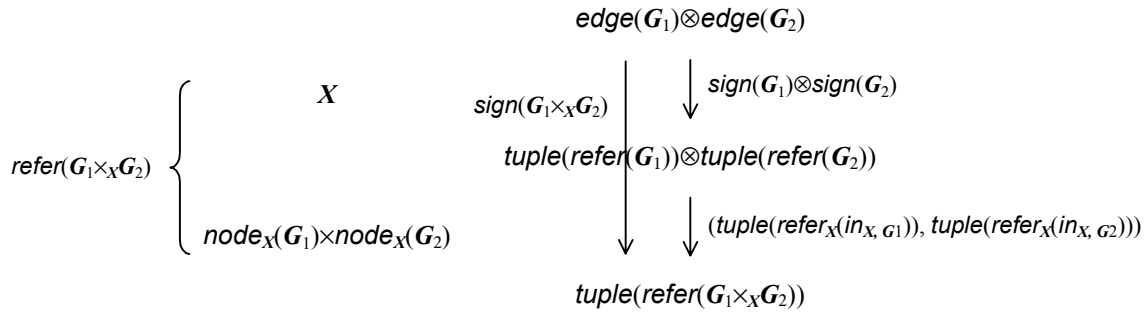
```

[((unpairing ?x) ?p) ((pairing ?x) ?p)]
((hgph.fbr.mor$identity ?x)
 ((hgph.fbr.obj$tuple ?x) ((binary-product ?x) ?p)))

```



**Figure 28a: Binary Product Hypergraph, Projections, Cone and Mediator – abstract version**



**Figure 28b: Binary Product Hypergraph – concrete version**

Let  $X$  be any set representing a fixed set of names. Let  $G_1$  and  $G_2$  be two hypergraphs with common name set  $X$ . The *binary product* hypergraph  $G_1 \times_X G_2$  is defined as follows (Figure 7b).

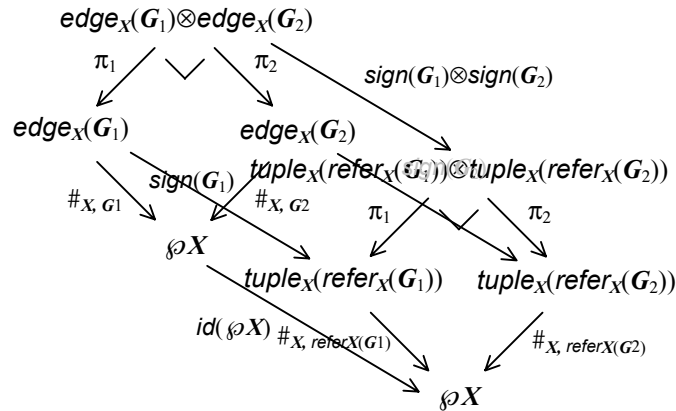
The node set is the binary Cartesian product of its components

$$\text{node}_X(G_1 \times_X G_2) = \text{node}_X(G_1) \times \text{node}_X(G_2).$$

The edge set is the pullback of the component edge arity functions

$$\begin{aligned} \text{edge}_X(G_1 \times_X G_2) \\ = \text{edge}_X(G_1) \otimes \text{edge}_X(G_2) = \{(\varepsilon_1, \varepsilon_2) \mid \varepsilon_1 \in \text{edge}_X(G_1), \varepsilon_2 \in \text{edge}_X(G_2), \text{arity}_X(\varepsilon_1) = \text{arity}_X(\varepsilon_2)\}, \end{aligned}$$

which is the subset of those pairs of edges that have the same arity. Denote the inclusion function into the Cartesian product by



**Figure 28c: Signature as Pullback Mediator**

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 72

May 8, 2003

$$inc : edge(A) \otimes edge(A') \rightarrow edge(A) \times edge(A').$$

The name set is the fixed set  $X$ ,

$$name_X(G_1 \times_X G_2) = name_X(G_1) = name_X(G_2) = X.$$

The reference set pair is

$$*_{X, (G_1 \times_X G_2)} = refer_X(G_1 \times_X G_2) = \langle X, node_X(G_1) \times node_X(G_2) \rangle.$$

The signature function

$$\partial_{X, (G_1 \times_X G_2)} = sign_X(G_1 \times_X G_2) : edge(A) \otimes edge(A') \rightarrow tuple(refer_X(G_1 \times_X G_2)),$$

is defined componentwise by  $\partial_{X, (G_1 \times_X G_2)}(\varepsilon_1, \varepsilon_2)_x = (\partial_{X, G_1}(\varepsilon_1)_x, \partial_{X, G_2}(\varepsilon_2)_x)$  for any variable  $x \in arity_X(G_1)(\varepsilon_1) = arity_X(G_2)(\varepsilon_2)$ .

We will define this pointwise. However, it may be helpful to define a notion of binary product in the fiber category of set semi-pairs based at first set  $X$ . Then define an analogous pullback for tuple arity.

The edge-arity function

$$\#_{X, (G_1 \times_X G_2)} = \pi_1 \cdot \#_{X, G_1} = edge\_arity_X(G_1 \times_X G_2) : edge_X(G_1) \otimes edge_X(G_2) \rightarrow \wp X.$$

maps a pullback pair to their common arity.

The *product projection* hypergraph morphism  $pr_X(G_1) = \langle \pi_1, \pi_1, id_X \rangle : G_1 \times_X G_2 \rightarrow G_1$  from the product hypergraph  $G_1 \times_X G_2$  to the first component hypergraph  $G_1$  is described as follows.

- The edge function is the edge pullback projection

$$\pi_1 = pr_{edge(G_1)} : edge_X(G_1 \times_X G_2) = edge_X(G_1) \otimes edge_X(G_2) \rightarrow edge_X(G_1);$$

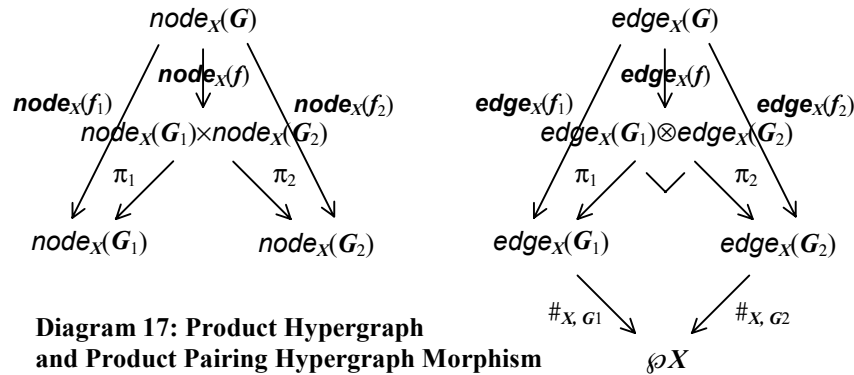
- the node function is the node product projection

$$\pi_1 = pr_{node(G_1)} : node_X(G_1) \times node_X(G_2) = node_X(G_1 \times_X G_2) \rightarrow node_X(G_1); \text{ and}$$

- the name function is the identity

$$id_X : X \rightarrow name_X(G_1 \times_X G_2) = X.$$

The hypergraph morphism condition holds, since  $\partial_{G_1}(pr_{G_1}(inc(\varepsilon_1, \varepsilon_2)))_x = \partial_{G_1}(\varepsilon_1)_x = tuple(pr_{G_1})(\partial_{G_1}(\varepsilon_1, \varepsilon_2))_x$  for any variable  $x \in arity_X(\varepsilon_1)$ .



```
(??) (KIF$function hypergraph-pair)
(= (KIF$source hypergraph-pair) set.obj$object)
(= (KIF$target hypergraph-pair) SET.LIM.PRD2$diagram)
(forall (?x (set.obj$object ?x))
 (and (= (SET.LIM.PRD2$class1 (hypergraph-pair ?x)) (hgph.fbr.obj$object ?x))
 (= (SET.LIM.PRD2$class2 (hypergraph-pair ?x)) (hgph.fbr.obj$object ?x))))

(SET.LIM.PRD2$binary-product (hypergraph-pair ?x))
(SET.LIM.PRD2$projection1 (hypergraph-pair ?x))
(SET.LIM.PRD2$projection2 (hypergraph-pair ?x))

(??) (KIF$function arity-opspan)
(= (KIF$source arity-opspan) set.obj$object)
(= (KIF$target arity-opspan) SET.FTN$function)
(forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (arity-opspan ?x))
```



# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 73

May 8, 2003

```
(SET.LIM.PR2$binary-product (hypergraph-pair ?x))
(= (SET.FTN$target (arity-opspan ?x) set.lim.pbk$diagram)
 (SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$set1])
 (SET.FTN$composition
 [(SET.LIM.PR2$projection1 (hypergraph-pair ?x) (hgph.fbr.obj$edge ?x))])
 (SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$set2])
 (SET.FTN$composition
 [(SET.LIM.PR2$projection2 (hypergraph-pair ?x) (hgph.fbr.obj$edge ?x))])
 (SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$opvertex])
 ((SET.FTN$constant
 [(SET.LIM.PR2$binary-product (hypergraph-pair ?x) set.obj$object]) (set$power
?x))
 (SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$opfirst])
 (SET.FTN$composition
 [(SET.LIM.PR2$projection1 (hypergraph-pair ?x) (hgph.fbr.obj$edge-arity ?x))])
 (SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$opsecond])
 (SET.FTN$composition
 [(SET.LIM.PR2$projection2 (hypergraph-pair ?x) (hgph.fbr.obj$edge-arity ?x))]))
(SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$pullback])
(SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$projection1])
(SET.FTN$composition [(arity-opspan ?x) set.lim.pbk$projection2])

(= (set.lim.pbk$opfirst (arity-opspan ?x)
 ((hgph.fbr.obj$edge-arity ?x) ?g1))
 (set.lim.pbk$opsecond (arity-opspan ?x)
 ((hgph.fbr.obj$edge-arity ?x) ?g2))

(set.lim.pbk$pullback ((arity-opspan ?x) [?g1 ?g2]))
(set.lim.pbk$projection1 ((arity-opspan ?x) [?g1 ?g2]))
(set.lim.pbk$projection2 ((arity-opspan ?x) [?g1 ?g2]))
```

Let  $f_1 = \langle \text{refer}_X(f_1), \text{sign}_X(f_1) \rangle : G \rightarrow G_1$  and  $f_2 = \langle \text{refer}_X(f_2), \text{sign}_X(f_2) \rangle : G \rightarrow G_2$  be any two hypergraph morphisms with a common source. Define the *mediator* (product pairing) hypergraph morphism

$$f = [f_1, f_2] = \langle \text{refer}_X(f), \text{sign}_X(f) \rangle : G \rightarrow G_1 \times_X G_2$$

as follows:

$\text{node}_X(f) = (\text{node}_X(f_1), \text{node}_X(f_2))$  is the node Cartesian product pairing; and

$\text{edge}_X(f) = (\text{edge}_X(f_1), \text{edge}_X(f_2))$  is the edge pullback pairing.

The commuting diagram,  $\text{edge}_X(f) \cdot \partial_{X, (G_1 \times_X G_2)} = \partial_{X, G} \cdot \text{tuple}(\text{refer}_X(f))$ , in the signature quartet of  $f$  holds since the definition of the product signature map implies

$$\begin{aligned} & \partial_{X, (G_1 \times_X G_2)}(\text{edge}_X(f)(\varepsilon))_x \\ &= (\partial_{X, G_1}(\text{edge}_X(f_1)(\varepsilon))_x, \partial_{X, G_2}(\text{edge}_X(f_2)(\varepsilon))_x) \\ &= (\text{node}_X(f_1)(\partial_{X, G}(\varepsilon))_x, \text{node}_X(f_2)(\partial_{X, G}(\varepsilon))_x) \\ &= \text{tuple}(\text{refer}_X(f))(\partial_{X, G}(\varepsilon))_x \end{aligned}$$

for any name  $x \in \text{arity}_X(G)(\varepsilon)$ . This is the unique mediating hypergraph morphism satisfying  $f \circ \text{pr}_{X, G_1} = f_1$  and  $f \circ \text{pr}_{X, G_2} = f_2$ .

The product  $G_1 \times_X G_2$  is a product in the category  $\text{Hypergraph}(X)$ . In summary, all products exist in any fiber  $\text{name}^{-1}(X) = \text{Hypergraph}(X)$  of the name functor  $\text{name} : \text{Hypergraph} \rightarrow \text{Set}$ . The hypergraph functor

$$\text{hgph}(X) : \text{Spangraph}(X) \rightarrow \text{Hypergraph}(X)$$

and the spangraph functor

$$\text{sgph}(X) : \text{Hypergraph}(X) \rightarrow \text{Spangraph}(X)$$

preserve products.

A *binary product* is a finite limit in the category  $\text{Hypergraph}(X)$  for a diagram of shape  $\text{two} = \bullet \bullet$ . Such a diagram (of hypergraphs and hypergraph morphisms) is called a *pair* of hypergraphs. For any set  $X$  repre-

## The IFF Namespace of Hypergraphs

Robert E. Kent

Page 74

May 8, 2003

senting a fixed set of names, a *pair* of hypergraphs consists of hypergraphs *hypergraph1* and *hypergraph2*. We use either the generic term ‘diagram’ or the specific term ‘pair’ to denote the pair class. Pairs are determined by their two component hypergraphs.

```
(1) (KIF$function diagram)
 (KIF$function pair)
 (= pair diagram)
 (= (KIF$source diagram) set.obj$object)
 (= (KIF$target diagram) SET$class)

(2) (KIF$function hypergraph1)
 (= (KIF$source hypergraph1) set.obj$object)
 (= (KIF$target hypergraph1) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph1 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph1 ?x)) (hgph.fbr.obj$object ?x))))

(3) (KIF$function hypergraph2)
 (= (KIF$source hypergraph2) set.obj$object)
 (= (KIF$target hypergraph2) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (hypergraph2 ?x)) (diagram ?x))
 (= (SET.FTN$target (hypergraph2 ?x)) (hgph.fbr.obj$object ?x))))

(4) (forall (?x (set.obj$object ?x)
 ?d1 ((diagram ?x) ?d1) ?d2 ((diagram ?x) ?d2))
 (= (> (and (= ((hypergraph1 ?x) ?d1) ((hypergraph1 ?x) ?d2))
 (= ((hypergraph2 ?x) ?d1) ((hypergraph2 ?x) ?d2)))
 (= ?d1 ?d2))))
```

For any set *X* representing a fixed set of names, there is an *edge pair* or *edge diagram* function, which maps a pair of hypergraphs to the underlying pair of edge sets. Similarly, there is a *node pair* or *node diagram* function, which maps a pair of hypergraphs to the underlying pair of node sets.

```
(5) (KIF$function edge-diagram)
 (= (KIF$source edge-diagram) set.obj$object)
 (= (KIF$target edge-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (edge-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (edge-diagram ?x)) set.col.copr2$diagram)
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.copr2$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$edge ?x)]))
 (= (SET.FTN$composition [(edge-diagram ?x) set.col.copr2$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$edge ?x)]))))

(6) (KIF$function node-diagram)
 (= (KIF$source node-diagram) set.obj$object)
 (= (KIF$target node-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (node-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (node-diagram ?x)) set.col.copr2$diagram)
 (= (SET.FTN$composition [(node-diagram ?x) set.col.copr2$set1])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$node ?x)]))
 (= (SET.FTN$composition [(node-diagram ?x) set.col.copr2$set2])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$node ?x)]))))
```

There is a *reference diagram* function, which maps a diagram to the binary product diagram of reference set pairs.

```
(--) (KIF$function reference-diagram)
 (= (KIF$source reference-diagram) set.obj$object)
 (= (KIF$target reference-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (reference-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (reference-diagram ?x)) (set.pr.lim.prd2$diagram ?x))
 (= (SET.FTN$composition [(reference-diagram ?x) (pair1 ?x)])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$reference ?x)]))
 (= (SET.FTN$composition [(reference-diagram ?x) (pair2 ?x)])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$reference ?x)]))))
```

There is a *tuple diagram* function, which applies the tuple operator to each component of a hypergraph pair.

# The IFF Namespace of Hypergraphs

Robert E. Kent

Page 75

May 8, 2003

```
(--) (KIF$function tuple-diagram)
 (= (KIF$source tuple-diagram) set.obj$object)
 (= (KIF$target tuple-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (tuple-diagram ?x)) (diagram ?x))
 (= (SET.FTN$target (tuple-diagram ?x)) (diagram ?x))
 (= (SET.FTN$composition [(tuple-diagram ?x) (hypergraph1 ?x)])
 (SET.FTN$composition [(hypergraph1 ?x) (hgph.fbr.obj$tuple ?x)]))
 (= (SET.FTN$composition [(tuple-diagram ?x) (hypergraph2 ?x)])
 (SET.FTN$composition [(hypergraph2 ?x) (hgph.fbr.obj$tuple ?x)]))))
```

For any set  $X$  representing a fixed set of names, a *binary product cone* is the appropriate cone for a binary product over  $X$ . A product cone (see Figure 7a, where arrows denote hypergraph morphisms) consists of a pair of hypergraph morphisms called *first* and *second*. These are required to have a common source hypergraph called the *vertex* of the cone. Each binary product cone is over a pair of hypergraphs.

```
(7) (KIF$function cone)
 (= (KIF$source cone) set.obj$object)
 (= (KIF$target cone) SET$class)

(8) (KIF$function cone-diagram)
 (= (KIF$source cone-diagram) set.obj$object)
 (= (KIF$target cone-diagram) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (cone-diagram ?x)) (cone ?x))
 (= (SET.FTN$target (cone-diagram ?x)) (diagram ?x))))

(9) (KIF$function vertex)
 (= (KIF$source vertex) set.obj$object)
 (= (KIF$target vertex) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (vertex ?x)) (cone ?x))
 (= (SET.FTN$target (vertex ?x)) (hgph.fbr.obj$object ?x))))

(10) (KIF$function first)
 (= (KIF$source first) set.obj$object)
 (= (KIF$target first) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (first ?x)) (cocone ?x))
 (= (SET.FTN$target (first ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(first ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (vertex ?x))
 (= (SET.FTN$composition [(first ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (SET.FTN$composition [(cone-diagram ?x) (hypergraph1 ?x)]))))

(11) (KIF$function second)
 (= (KIF$source second) set.obj$object)
 (= (KIF$target second) SET.FTN$function)
 (forall (?x (set.obj$object ?x))
 (and (= (SET.FTN$source (opsecond ?x)) (cocone ?x))
 (= (SET.FTN$target (opsecond ?x))
 (hgph.fbr.mor$morphism (set.mor$identity ?x)))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$source (set.mor$identity ?x))])
 (SET.FTN$composition [(cocone-diagram ?x) (hypergraph2 ?x)]))
 (= (SET.FTN$composition [(opsecond ?x) (hgph.fbr.mor$target (set.mor$identity ?x))])
 (opvertex ?x))))
```