The IFF Namespace of Spangraphs

The terminology t

- Need to consider bijections in the various colimit constructions.

Table 1: Terminology for the Namespace of Spangraphs

	Category	Functor	Natural Transformation	Adjunction
sgph	spangraph	vertex edge node name hypergraph	indication comediation projection bijection realization	adjunction- equivalence
sgph.fbr	spangraph	vertex edge node name hypergraph inclusion direct-image inverse-image	indication comediation projection bijection realization	adjunction- equivalence

	Class	Class Function	Other
sgph.obj	object	indication comediation projection vertex edge node name	
		hypergraph bijection realization	
sgph.mor	morphism isomorphism	source target indication comediation projection vertex edge node name	composable-opspan composable
		composition identity	
		hypergraph	

	Collection	Collection Function	Other
sgph.fbr.obj		object inclusion indication comediation projection vertex edge node name	
		hypergraph bijection realization inverse-image direct-image	
sgph.fbr.mor		morphism inclusion isomorphism	composable-opspan

		source target indication comediation projection vertex edge node name	composable
		composition identity	
		hypergraph inverse-image direct-image	

The IFF Lower Core Ontology (IFF-LCO)

For any category C in the IFF Lower Core (meta) Ontology (IFF-LCO), a module in the lower metalevel of the IFF, the assertion that "C *is a category*" cannot be made with the terminology of the IFF-LCO, since the appropriate categorical/functorial machinery is not present there. The IFF Category Theory Ontology (IFF-CAT) in the upper metalevel of the IFF provides that machinery. To make this assertion requires that we also describe or identify the components of a category. The expression is in terms of the object and morphism classes, the source and target functions, and the composition and identity functions. Proofs of some categorical properties, such as associativity of composition, will involve getting further into the details of the specific category C.

Likewise, for any functor $F : \mathsf{A} \rightarrow \mathsf{B}$ in the IFF-LCO, the assertion that "F *is a functor from* A *to* B" requires that we also describe or identify the components of a functor. These assertions cannot be made with the terminology of the IFF-LCO, since the appropriate categorical/functorial machinery is not present there. The IFF-CAT in the upper metalevel of the IFF provides that machinery. To make these assertions requires that we also describe or identify the components of functors. We use a description in terms of the object and morphism class functions. Proof of functoriality will involve getting further into the details of the specific functor F.

Also, for any natural transformation $\alpha : F \Rightarrow G : \mathsf{A} \rightarrow \mathsf{B}$ in the IFF-LCO, the assertion that "α *is a natural isomorphism*" cannot be made with the terminology of the IFF-LCO, since the appropriate categorical/functorial machinery is not present there. The IFF-CAT in the upper metalevel of the IFF provides that machinery. To make this assertion requires that we also describe or identify the components of a natural isomorphism. The expression is in terms of the source and target categories, the source and target functors, and the component function. Proofs of some natural transformation properties will involve getting further into the details of the specific natural transformation α; in particular, naturality, etc.

Finally, for any adjunction $\rho = \langle F, G, \eta, \varepsilon \rangle : \mathsf{A} \rightarrow \mathsf{B}$ in the IFF-LCO, the assertion that "ρ *is an adjunction*" cannot be made with the terminology of the IFF-LCO, since the appropriate categorical/functorial machinery is not present there. The IFF-CAT in the upper metalevel of the IFF provides that machinery. To make this assertion requires that we also describe or identify the components of an adjunction. The expression is in terms of the underlying and free categories, the left and right adjoint functors, and the unit and counit natural transformations. Proofs of some adjointness properties will involve getting further into the details of the specific adjunction ρ; in particular, universality, the triangular identities, etc.

Inner Core Categories

There is only one inner core category, Set the category of sets and functions. All other categories, functors and natural transformation of the IFF-LCO are based upon Set. Set is small complete and small cocomplete.

Middle Core Categories

Here is a list of categories (Figure 1) that are axiomatized in the inner and middle core of IFF Lower Core (meta) ontology (IFF-LCO).

○ Set : This is the most basic category whose objects are sets and morphisms are functions.

○ $\mathrm{Set}^{\cong}$: This is the target for the name functor. This I call the set *semi* category, the semi category of Set. It is the category whose objects are sets and morphisms are bijections. It is clearly a subcategory of Set. It is isomorphic to the category of inverse pairs, whose objects are sets and morphisms are inverse pairs of functions.

○ Set^{2} : This is called the set *pair category*, the pair category of Set. Currently it is not explicitly used. It is the category whose objects are pairs of sets and morphisms are pairs of functions compatible with source and target. It is the functor category Func[**2**, Set], where **2** is the category with two distinct objects and only identity arrows.

○ $\mathrm{Set}^{\perp} = \triangle\mathrm{Set}$: This is the target for the reference functor. I call this the set *semi-pair category*, the semi-pair category of Set. It is the category whose objects are pairs of sets and morphisms are bijection-function pairs that form a set semi-pair (not semi-quartet, as named in the old category theory for models document). It is a subcategory of Set^{2}.

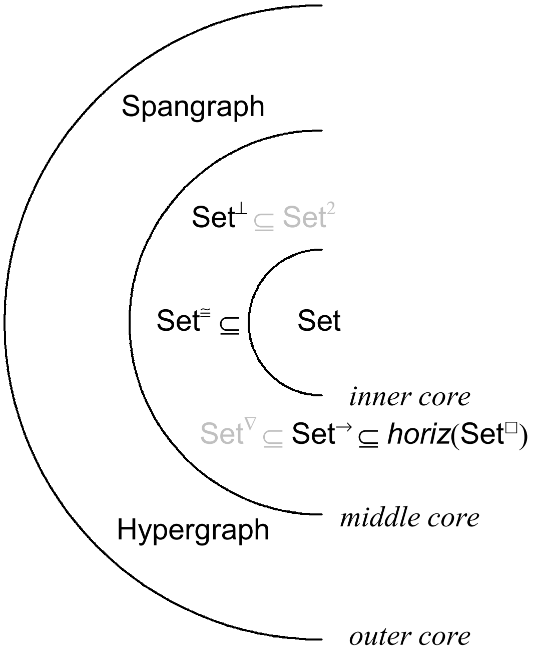


Figure 1: Shell Structure of the IFF-LCO

○ $\mathrm{Set}^{\rightarrow}$: This is the target for the signature functor. This is called the set *arrow category*, the arrow category of Set. It is the category whose objects are functions and morphisms are pairs of functions that form a set quartet with source and target. It is the functor category Func[**→**, Set], where **→** is the category with two distinct objects and one nonidentity arrow between them. It is also the horizontal category of the double category $\mathrm{Set}^{\square}$.

○ Set^{∇} : This is the target for the projection functors, which correspond to the projection natural transformations. As such, it is not strictly necessary, since we can get along with just describing and working with the natural transformations. I call this the set *hemi-arrow category*, the hemi-arrow category of Set. It is the category whose objects are functions and morphisms are function-bijection pairs that form a set hemiquartet with source and target. It is a subcategory of $\mathrm{Set}^{\rightarrow}$.

Here is the double category also axiomatized in IFF-LCO.

○ ($\mathrm{Set}^{\square} = \square\mathrm{Set}$: This is called the *quartet* double category, the double category of set squares. It is the category, whose objects are sets, squares are the commutative squares known as set quartets, and horizontal and vertical arrows are functions. The horizontal category is $\mathrm{Set}^{\rightarrow}$.

Middle Core Functors

Here are a list of the middle core functors (Figures 1a,b) that are axiomatized in IFF-LCO. The pair, semi-pair, arrow and hemi-arrow categories of Set have two projection functors π_1 and π_2 to the vertical source and target, respectively. There is also the important tuple functor on semi-pairs.

$$\begin{tikzcd}
 & \mathsf{Set}^{\cong} \arrow[r, phantom, "\subseteq"] & \mathsf{Set} \\
\mathsf{Set} & \mathsf{Set}^{\perp} \arrow[l, "tuple"'] \arrow[u, "\pi_1"] \arrow[d, "\pi_2"'] \arrow[r, phantom, "\subseteq"] & \mathsf{Set}^{2} \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"] \\
 & \mathsf{Set} \arrow[r, phantom, "="] & \mathsf{Set}
\end{tikzcd}$$

Figure 1a: Functors for the Pair and Semi-Pair Categories

$$\begin{tikzcd}
\mathsf{Set} \arrow[r, phantom, "="] & \mathsf{Set} \\
\mathsf{Set}^{\nabla} \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"'] \arrow[r, phantom, "\subseteq"] & \mathsf{Set}^{\rightarrow} = \mathit{horiz}(\mathsf{Set}^{\square}) \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"'] \\
\mathsf{Set}^{\cong} \arrow[r, phantom, "\subseteq"] & \mathsf{Set}
\end{tikzcd}$$

Figure 1b: Functors for the Arrow and Hemi-Arrow Categories

Here are the four Set pair projection functors.

- ○ $\pi_1 : \mathsf{Set}^2 \rightarrow \mathsf{Set}$: The first projection from the pair category Set^2 maps a set pair to the first set and maps a function pair to the first function.
- ○ $\pi_2 : \mathsf{Set}^2 \rightarrow \mathsf{Set}$: The second projection from the pair category Set^2 maps a set pair to the second set and maps a function pair to the second function.
- ○ $\pi_1 : \mathsf{Set}^{\perp} \rightarrow \mathsf{Set}^{\cong}$: The first projection from the semi-pair category $\mathsf{Set}^{\perp}$ maps a set pair to the first set and maps a bijection-function pair to the bijection. It is a restriction of the first pair projection.
- ○ $\pi_2 : \mathsf{Set}^{\perp} \rightarrow \mathsf{Set}$: The second projection from the semi-pair category $\mathsf{Set}^{\perp}$ maps a set pair to the second set and maps a bijection-function pair to the function. It is a restriction of the second pair projection.

Here are the four Set arrow projection functors.

- ○ $\pi_1 : \mathsf{Set}^{\rightarrow} \rightarrow \mathsf{Set}$: The first projection from the arrow category $\mathsf{Set}^{\rightarrow}$ maps a set function to the source set and maps a function pair to the first function. It is the projection to the vertical source.
- ○ $\pi_2 : \mathsf{Set}^{\rightarrow} \rightarrow \mathsf{Set}$: The second projection from the arrow category $\mathsf{Set}^{\rightarrow}$ maps a set function to the target set and maps a function pair to the second function. It is the projection to the vertical target.
- ○ $\pi_1 : \mathsf{Set}^{\nabla} \rightarrow \mathsf{Set}$: The first projection from the hemi-arrow category Set^{∇} maps a set function to the source set and maps a function-bijection pair to the function. It is a restriction of the first arrow projection.
- ○ $\pi_2 : \mathsf{Set}^{\nabla} \rightarrow \mathsf{Set}^{\cong}$: The second projection from the hemi-arrow category Set^{∇} maps a set function to the target set and maps a function-bijection pair to the bijection. It is a restriction of the second arrow projection.

Here is the important tuple functor.

- ○ $\mathit{tuple} : \mathsf{Set}^{\perp} \rightarrow \mathsf{Set}$: The *tuple* functor from the semi-pair category $\mathsf{Set}^{\perp}$ maps a set pair to the set of tuples whose arity is a subset of the first set of the pair and whose component values are elements of the second set of the pair. It maps a bijection-function pair to the function of tuples, which composes the tuple on the left with the inverse of the first function restricted to the arity and on the right with the second function. The requirement that the first function be a bijection comes from the need for this function of tuples to be well defined.

Outer Core Categories

There are just two outer core ontologies Spangraph and Hypergraph, and these are categorically equivalent via realization.

Outer Core Functors

$$\begin{tikzcd}
 & \mathrm{Set} & \\
\mathrm{Set} & \mathrm{Spangraph} \arrow[l, "\textit{edge}"'] \arrow[u, "\textit{vert}"'] \arrow[r, "\textit{node}"] \arrow[d, "\textit{name}"'] & \mathrm{Set} \\
 & \mathrm{Set}^{\cong} \subseteq \mathrm{Set} &
\end{tikzcd}$$

Figure 2a: Spangraph Category and Component Functors

$$\begin{tikzcd}
\mathrm{Set}^{\cong} & \mathrm{Set} & \mathrm{Set} \\
\mathrm{Set}^{\perp} \arrow[u, "\pi_1"] \arrow[d, "\pi_2"'] \arrow[drr, "\textit{tuple}"] & \mathrm{Hypergraph} \arrow[l, "* = \textit{refer}"'] \arrow[r, "\partial = \textit{sign}"] \arrow[ul, "\textit{name}"'] \arrow[u, "\textit{case}"'] \arrow[ur, "\textit{edge}"] \arrow[dl, "\textit{node}"] & \mathrm{Set}^{\rightarrow} \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"] \\
\mathrm{Set} & & \mathrm{Set}
\end{tikzcd}$$

Figure 2b: Hypergraph Category and Component Functors

Figures 2a,b illustrate the categories and component functors in the out core of the IFF-LCO. Figure 2b is a commutative diagram containing four equalities.

Here are the four Spangraph component functors. They have the same source category, and are subject to the bijection natural isomorphism ≅ : *vert* ≅ *hgph* ∘ *case*. See the functor identities in Table 1.

○ ***vert*** : Spangraph → Set : This is the vertex functor for the spangraph category Spangraph. It maps a spangraph to its vertex set and maps a spangraph morphism to its vertex function.

○ ***edge*** : Spangraph → Set : This is the edge functor for the spangraph category Spangraph. It maps a spangraph to its edge set and maps a spangraph morphism to its edge function.

○ ***node*** : Spangraph → Set : This is the node functor for the spangraph category Spangraph. It maps a spangraph to its node set and maps a spangraph morphism to its node function.

○ ***name*** : Spangraph → Set : This is the name functor for the spangraph category Spangraph. It maps a spangraph to its name set and maps a spangraph morphism to its name bijection.

Here are the two core Hypergraph component functors. They have the same source category Hypergraph, and are subject to the identity *sign* ∘ *proj2* = *refer* ∘ *tuple*., which is a pullback constraint for functors; that is, Hypergraph is the pullback of the opspan in the quasi-category CAT of categories and functors. See the functor identities in Table 1.

○ ∗ = *refer* : Hypergraph → Set$^{\perp}$: This is the reference functor for the hypergraph category Hypergraph. It maps a hypergraph to its reference pair of name and node sets and maps a hypergraph morphism to its reference pair of name bijection and node function.

○ ∂ = *sign* : Hypergraph → Set$^{\rightarrow}$: This is the signature functor for the hypergraph category Hypergraph. It maps a hypergraph to its signature function and maps a hypergraph morphism to its signature set quartet consisting of edge function as vertical source and tuple reference function as vertical target.

Here are the three Hypergraph component functors. They are defined in terms of the reference and signature functors. They are equivalent to the corresponding Spangraph component functors via the equivalence between the two categories. The three component functors have the same source category, and are subject to the bijection natural isomorphism ≅ : *vert* ≅ *hgph* ∘ *case*. See the functor identities in Table 1.

○ ***edge*** : Hypergraph → Set : This is the edge functor for the hypergraph category Hypergraph. It maps a hypergraph to its edge set and maps a hypergraph morphism to its edge function.

- ***node*** : Hypergraph → Set : This is the node functor for the hypergraph category Hypergraph. It maps a hypergraph to its node set and maps a hypergraph morphism to its node function.
- ***name*** : Hypergraph → Set : This is the name functor for the hypergraph category Hypergraph. It maps a hypergraph to its name set and maps a hypergraph morphism to its name bijection.

Here are the adjoint functors that are part of the categorical equivalence between Spangraph and Hypergraph. The hypergraph of the spangraph of a hypergraph is the original hypergraph, but the spangraph of the hypergraph of a spangraph is only isomorphic to the original spangraph through the realization spangraph isomorphism. See the functor identities in Table 1.

- ***hgph*** : Spangraph → Hypergraph : This is the hypergraph functor for the spangraph category. It maps a spangraph to its associated hypergraph, whose edge, node and name sets are the same, and whose reference and signature functions are defined in terms of the indication, comediation and projection functions of the spangraph.
- ***sgph*** : Hypergraph → Spangraph : This is the spangraph functor for the hypergraph category. It maps a hypergraph to its associated spangraph, whose edge, node and name sets are the same, whose vertex set is the case set of the hypergraph, whose indication, comediation and projection functions are the corresponding case functions of the hypergraph.

Table 1 lists the relevant functor identities for these functors.

Table 1: Functor Identities

$sgph \circ vert = case$	$hgph \circ edge = edge$ $sgph \circ edge = edge$	$hgph \circ node = node$ $sgph \circ node = node$	$hgph \circ name = name$ $sgph \circ name = name$
$refer \circ proj1 = name$ $refer \circ proj2 = node$	$sign \circ proj1 = edge$ $sign \circ proj2 = refer \circ tuple$		

Set Natural Transformation	Set Functor
$\langle : vert \Rightarrow edge$	$indic : \mathrm{Spangraph} \rightarrow \mathrm{Set}^{\rightarrow}$
$\rangle : vert \Rightarrow node$	$comed : \mathrm{Spangraph} \rightarrow \mathrm{Set}^{\rightarrow}$
$\underline{\downarrow} : vert \Rightarrow name\ .\ incl$	$proj : \mathrm{Spangraph} \rightarrow \mathrm{Set}^{\nabla}$
$\langle : case \Rightarrow edge$	$indic : \mathrm{Hypergraph} \rightarrow \mathrm{Set}^{\rightarrow}$
$\rangle : case \Rightarrow node$	$comed : \mathrm{Hypergraph} \rightarrow \mathrm{Set}^{\rightarrow}$
$\underline{\downarrow} : case \Rightarrow name\ .\ incl$	$proj : \mathrm{Hypergraph} \rightarrow \mathrm{Set}^{\nabla}$
$\cong : vert \cong hgph \circ case$	$bij : \mathrm{Spangraph} \rightarrow \mathrm{Set}^{\rightarrow}$

Table 2: Natural Transformation Identities

$\cong \bullet (1_{hgph} \circ \langle) = \langle$ $1_{sgph} \circ \langle = \langle$	$\cong \bullet (1_{hgph} \circ \rangle) = \rangle$ $1_{sgph} \circ \rangle = \rangle$	$\cong \bullet (1_{hgph} \circ \underline{\downarrow}) = \underline{\downarrow}$ $1_{sgph} \circ \underline{\downarrow} = \underline{\downarrow}$	$1_{sgph} \circ \cong = 1_{case}$ $1_{sgph} \circ \uparrow = 1_{sgph}$
$\uparrow \circ 1_{vert} = \cong$	$\uparrow \circ 1_{edge} = 1_{edge}$	$\uparrow \circ 1_{node} = 1_{node}$	$\uparrow \circ 1_{name} = 1_{name}$

Outer Core Natural Transformations

$$\begin{tikzcd}[column sep=large, row sep=huge]
& \text{Spangraph} \arrow[dl, bend right=60, "edge"'] \arrow[dl, "name"] \arrow[d, "vert"] \arrow[d, bend left=70, "node"] \arrow[d, phantom, shift right=10ex, "\overset{\langle}{\Leftarrow}"] \arrow[d, phantom, shift left=5ex, "\overset{\rangle}{\Rightarrow}"] \arrow[d, phantom, shift right=4ex, "\underline{\downarrow}\Downarrow"'] \\
\text{Set}^{\cong} \arrow[r, phantom, "\subseteq"] & \text{Set}
\end{tikzcd}$$

Figure 3a: Component Functors and Natural Transformations

$$\begin{tikzcd}[column sep=large, row sep=huge]
& \text{Hypergraph} \arrow[dl, bend right=60, "edge"'] \arrow[dl, "name"] \arrow[d, "case"] \arrow[d, bend left=70, "node"] \arrow[d, phantom, shift right=10ex, "\overset{\langle}{\Leftarrow}"] \arrow[d, phantom, shift left=5ex, "\overset{\rangle}{\Rightarrow}"] \arrow[d, phantom, shift right=4ex, "\underline{\downarrow}\Downarrow"'] \\
\text{Set}^{\cong} \arrow[r, phantom, "\subseteq"] & \text{Set}
\end{tikzcd}$$

Figure 3b: Component Functors and Natural Transformations

$$\begin{tikzcd}
\text{Spangraph} \arrow[dr, "hgph"] \arrow[dd, "vert"'] & \\
\overset{\cong}{\Rightarrow} & \text{Hypergraph} \arrow[dl, "case"] \\
\text{Set} &
\end{tikzcd}$$

Figure 4a: Bijection Natural Transformation

$$\begin{tikzcd}
\text{Spangraph} \arrow[dr, "hgph"] \arrow[dd, "id"'] & \\
\overset{\uparrow}{\Rightarrow} & \text{Hypergraph} \arrow[dl, "sgph"] \\
\text{Spangraph} &
\end{tikzcd}$$

Figure 4b: Realization Natural Transformation

The Context of Spangraphs

`sgph`

Categories

A useful category specified within the IFF Lower Core (meta) Ontology (IFF-LCO) is Spangraph, the category of spangraphs and their morphisms. This category is axiomatized in the spangraph namespace of the IFF-LCO.

The notion of a *spangraph* (labeled span) of sets is new. This is a specialization of the notion of a span, a basic notion in category theory. Spangraphs are used in the type language namespace, the model namespace and the namespace of classification spangraphs. The notion of a spangraph is closely related to the notion of a hypergraph. In fact, this document establishes the fact that the context of spangraphs is categorically equivalent to the context of hypergraphs. As a result, the terminology that we use for spangraphs is similar to the terminology that we use for hypergraphs. Historically, although the notion of bipartite graphs has been mentioned or used in various venues, the notion of spangraph is the more correct notion.

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

Functors

Here is a list of the outer shell functors of the IFF-LCO that are axiomatized in this namespace (Figure 1). The spangraph category has component vertex, edge and node functors to the category Set, and a component name functor to the bijection subcategory $\mathsf{Set}^{\cong}$. There is also a hypergraph functor linking the context of spangraphs with the categorically equivalent context of hypergraphs.

- ○ $\mathit{vert} : \mathsf{Spangraph} \rightarrow \mathsf{Set}$: The vertex functor maps a spangraph to its vertex set and maps a spangraph morphism to its vertex function.
- ○ $\mathit{edge} : \mathsf{Spangraph} \rightarrow \mathsf{Set}$: The edge functor maps a spangraph to its edge set and maps a spangraph morphism to its edge function.
- ○ $\mathit{node} : \mathsf{Spangraph} \rightarrow \mathsf{Set}$: The node functor maps a spangraph to its node set and maps a spangraph morphism to its node function.
- ○ $\mathit{name} : \mathsf{Spangraph} \rightarrow \mathsf{Set}^{\cong}$: The name functor maps a spangraph to its name set and maps a spangraph morphism to its name bijection.
- ○ $\mathit{hgph} : \mathsf{Spangraph} \rightarrow \mathsf{Hypergraph}$: The hypergraph functor connects the spangraph category to the hypergraph category. It maps a spangraph to its associated hypergraph. The reference pair is the name and node set pair. The arity is the composition of the indication fiber and the power of the projection. The signature is the restriction of the comediator to the arity. Any hypergraph has an associated spangraph, which is axiomatized in the hypergraph namespace. The hypergraph associated with the spangraph is the original hypergraph. Hence, the composition of the spangraph functor with the hypergraph functor is the identity functor on the context of hypergraphs. This property is one half of the categorical equivalence between spangraphs and hypergraphs.

$$\begin{tikzcd}
& \mathsf{Set} & \\
\mathsf{Set} & \mathsf{Spangraph} \arrow[l, "edge"'] \arrow[u, "vert"'] \arrow[r, "node"] \arrow[d, "name"'] & \mathsf{Set} \\
& \mathsf{Set}^{\cong} \subseteq \mathsf{Set} &
\end{tikzcd}$$

Figure 1: Spangraph Category and Component Functors

```
(2) (FUNC$functor vertex)
    (= (FUNC$source vertex) spangraph)
    (= (FUNC$target vertex) set$set)
    (= (FUNC$object vertex) sgph.obj$vertex)
    (= (FUNC$morphism vertex) sgph.mor$vertex)

(3) (FUNC$functor edge)
    (= (FUNC$source edge) spangraph)
    (= (FUNC$target edge) set$set)
    (= (FUNC$object edge) sgph.obj$edge)
    (= (FUNC$morphism edge) sgph.mor$edge)

(4) (FUNC$functor node)
    (= (FUNC$source node) spangraph)
    (= (FUNC$target node) set$set)
    (= (FUNC$object node) sgph.obj$node)
    (= (FUNC$morphism node) sgph.mor$node)

(5) (FUNC$functor name)
    (= (FUNC$source name) spangraph)
    (= (FUNC$target name) set$semi)
    (= (FUNC$object name) sgph.obj$name)
    (= (FUNC$morphism name) sgph.mor$name)

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

(7) (= (FUNC$composition [hgph$spangraph hypergraph])
       (FUNC$identity hgph$hypergraph]))
```

These functors satisfy several identities, which are used to partially define the hypergraph functor. These assert that the edge, node and name components are equivalently related through the hypergraph functor: $\mathit{hgph} \circ \mathit{edge} = \mathit{edge}$, $\mathit{hgph} \circ \mathit{node} = \mathit{node}$ and $\mathit{hgph} \circ \mathit{name} = \mathit{name}$.

```
(8) (= (FUNC$composition [hypergraph hgph$edge]) edge)
    (= (FUNC$composition [hypergraph hgph$node]) node)
    (= (FUNC$composition [hypergraph hgph$name]) name)
```

Natural Transformations

Here is a list of the outer shell natural transformations of the IFF-LCO that are axiomatized in this namespace. The indication, comediation and projection component functions of spangraphs define indication, comediation and projection component natural transformations (Figure 2a). The source functor for all three is the vertex 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 spangraph morphisms.

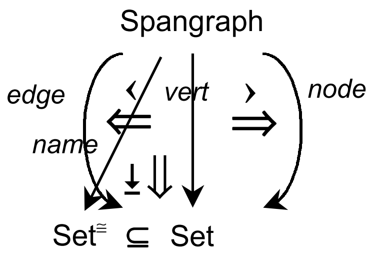


Figure 2a: Component Functors and Natural Transformations

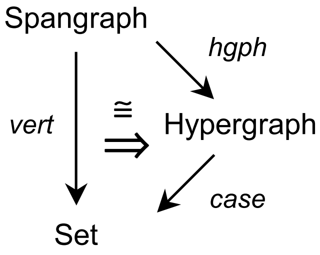


Figure 2b: Bijection Natural Transformation

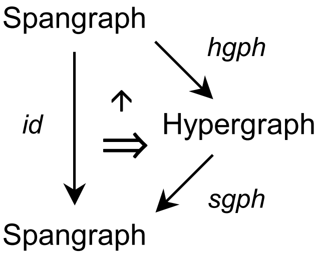


Figure 2c: Realization Natural Transformation

- ○ $\langle = \mathrm{indic} : \mathit{vert} \Rightarrow \mathit{edge} : \mathrm{Spangraph} \rightarrow \mathrm{Set}$: This is the *indication* natural transformation whose component at any spangraph is the indication function, which maps vertices to their (hyper) edges. It is equivalent to the hypergraph indication natural transformation.
- ○ $\rangle = \mathrm{comed} : \mathit{vert} \Rightarrow \mathit{node} : \mathrm{Spangraph} \rightarrow \mathrm{Set}$: This is the *comediation* natural transformation whose component at any spangraph is the comediation function, which maps vertices to their nodes. It is equivalent to the hypergraph comediation natural transformation.
- ○ $\underline{\downarrow} = \mathrm{proj} : \mathit{vert} \Rightarrow \mathit{name} \,.\, \mathit{incl} : \mathrm{Spangraph} \rightarrow \mathrm{Set}$: This is the *projection* natural transformation whose component at any spangraph is the projection function, which maps vertices to their names (labels). It is equivalent to the hypergraph projection natural transformation.

- $\cong$ = bij : $\mathit{vert} \Rightarrow \mathit{hgph} \circ \mathit{case}$: Spangraph → Set : This is the bijection *natural* isomorphism whose component at any spangraph $\boldsymbol{G}$ is the set bijection $\cong_{\boldsymbol{G}} : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{case}(\mathit{hgph}(\boldsymbol{G}))$, which maps vertices to their associated cases. This is the core of the equivalence between the categories Spangraph and Hypergraph.
- $\uparrow$ = real : $\mathit{id}_{\mathsf{Spangraph}} \Rightarrow \mathit{hgph} \circ \mathit{sgph}$: Spangraph → Spangraph : This is *realization* natural isomorphism whose component at any spangraph $\boldsymbol{G}$ is the realization spangraph isomorphism $\uparrow_{\boldsymbol{G}} = \mathit{real}(\boldsymbol{G}) : \boldsymbol{G} \rightarrow \mathit{sgph}(\mathit{hgph}(\boldsymbol{G}))$, which maps vertices to their names (labels). The exactness of the creation condition for spangraph morphisms means that the case function of the hypergraph morphism $\mathit{case}(\mathit{hgph}(\boldsymbol{f})) : \mathit{case}(\mathit{hgph}(\boldsymbol{G}) \rightarrow \mathit{case}(\mathit{hgph}(\boldsymbol{G}'))$ is "equivalent" to the vertex function of $\boldsymbol{f}$; that is, it commutes through the source/target realizations $\cong_{\boldsymbol{G}} : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{case}(\mathit{hgph}(\boldsymbol{G}))$ with the vertex function. This implies naturality for realization.

```
 (9) (NAT$natural-transformation indication)
     (= (NAT$source-category indication) spangraph)
     (= (NAT$target-category indication) set$set)
     (= (NAT$source-functor indication) vertex)
     (= (NAT$target-functor indication) edge)
     (= (NAT$component indication) sgph.obj$indication)

(10) (NAT$natural-transformation comediation)
     (= (NAT$source-category comediation) spangraph)
     (= (NAT$target-category comediation) set$set)
     (= (NAT$source-functor comediation) vertex)
     (= (NAT$target-functor comediation) node)
     (= (NAT$component comediation) sgph.obj$comediation)

(11) (NAT$natural-transformation projection)
     (= (NAT$source-category projection) spangraph)
     (= (NAT$target-category projection) set$set)
     (= (NAT$source-functor projection) vertex)
     (= (NAT$target-functor projection)
        (FUNC$composition [name (FUNC$inclusion [set$semi set$set])]))
     (= (NAT$component projection) sgph.obj$projection)

(12) (NAT$natural-isomorphism bijection)
     (= (NAT$source-category bijection) spangraph)
     (= (NAT$target-category bijection) set$set)
     (= (NAT$source-functor bijection) vertex)
     (= (NAT$target-functor bijection) (FUNC$composition [hypergraph hgph$case]))
     (= (NAT$component bijection) sgph.obj$bijection)

(13) (NAT$natural-isomorphism realization)
     (= (NAT$source-category realization) spangraph)
     (= (NAT$target-category realization) spangraph)
     (= (NAT$source-functor realization) (FUNC$identity spangraph))
     (= (NAT$target-functor realization) (FUNC$composition [hypergraph hgph$spangraph]))
     (= (NAT$component realization) sgph.obj$realization)
```

These natural transformations satisfy two kinds of identities. The first set of identities assert that the indication, comediation and projection components are equivalently related through the hypergraph functor and the bijection natural transformation: $\cong \bullet (1_{\mathit{hgph}} \circ \langle) = \langle$, $\cong \bullet (1_{\mathit{hgph}} \circ \rangle) = \rangle$, and $\cong \bullet (1_{\mathit{hgph}} \circ \underline{\downarrow}) = \underline{\downarrow}$.

```
(14) (= (NAT$vertical-composition
           [bijection
            (NAT$horizontal-composition
                [(NAT$vertical-identity hypergraph) hgph$indication])]) indication)

(15) (= (NAT$vertical-composition
           [bijection
            (NAT$horizontal-composition
                [(NAT$vertical-identity hypergraph) hgph$comediation])]) comediation)

(16) (= (NAT$vertical-composition
           [bijection
            (NAT$horizontal-composition
                [(NAT$vertical-identity hypergraph) hgph$projection])]) projection)
```

The second set of identities express the axiomatic definition of realization in terms of functors and natural transformations: the vertex of realization is bijection $\uparrow \circ \mathit{vert} = \cong$, whereas the edge, node and name of realization are identities: $\uparrow \circ \mathbf{1}_{edge} = \mathbf{1}_{edge}$, $\uparrow \circ \mathbf{1}_{node} = \mathbf{1}_{node}$ and $\uparrow \circ \mathbf{1}_{name} = \mathbf{1}_{name}$.

```
(17) (= (NAT$horizontal-composition [realization (NAT$vertical-identity edge)])
        (NAT$vertical-identity edge))

(18) (= (NAT$horizontal-composition [realization (NAT$vertical-identity node)])
        (NAT$vertical-identity node))

(19) (= (NAT$horizontal-composition [realization (NAT$vertical-identity name)])
        (NAT$vertical-identity name))
```

Adjunctions

We want to verify that the realization adjunction $\rho = \langle \mathit{hgph}, \mathit{sgph}, \uparrow, 1 \rangle : \mathsf{Spangraph} \rightarrow \mathsf{Hypergraph}$ forms an adjoint equivalence. This equivalence links the contexts of spangraphs and hypergraphs through realization – the category Hypergraph is categorically equivalent to the category Spangraph via the spangraph equivalence $\mathit{sgph} : \mathsf{Spangraph} \rightarrow \mathsf{Hypergraph}$.

We know that the spangraph functor $\mathit{sgph} : \mathsf{Hypergraph} \rightarrow \mathsf{Spangraph}$ is a left inverse of the hypergraph functor *hgph*; that is, $\mathit{hgph}(\mathit{sgph}(\boldsymbol{G})) = \boldsymbol{G}$ for each hypergraph $\boldsymbol{G}$, and $\mathit{hgph}(\mathit{sgph}(\boldsymbol{f})) = \boldsymbol{f}$ for each hypergraph morphism $\boldsymbol{f}$. In particular, *sgph* is faithful. We also know that any spangraph $\boldsymbol{G}$ is isomorphism to the spangraph $\mathit{sgph}(\mathit{hgph}(\boldsymbol{G}))$ via the realization spangraph isomorphism $\uparrow_{\boldsymbol{G}} : \boldsymbol{G} \cong \mathit{sgph}(\mathit{hgph}(\boldsymbol{G}))$. We need to show the following two conditions hold.

- ○ The spangraph functor $\mathit{sgph} : \mathsf{Hypergraph} \rightarrow \mathsf{Spangraph}$ is full and faithful.
- ○ Every spangraph $\boldsymbol{G}$ is isomorphic to $\mathit{sgph}(\boldsymbol{G}')$ for some hypergraph $\boldsymbol{G}'$.

The latter is immediately true, by setting $\boldsymbol{G}' = \mathit{hgph}(\boldsymbol{G})$. To show that *sgph* is a full functor, let $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ be two hypergraphs and let $\boldsymbol{f} = \langle \mathit{indic}(\boldsymbol{f}), \mathit{comed}(\boldsymbol{f}), \mathit{proj}(\boldsymbol{f}) \rangle : \mathit{sgph}(\boldsymbol{G}_1) \rightarrow \mathit{sgph}(\boldsymbol{G}_2)$ be a spangraph morphism. Consider the spangraph morphism $\mathit{sgph}(\mathit{hgph}(\boldsymbol{f})) : \mathit{sgph}(\boldsymbol{G}_1) \rightarrow \mathit{sgph}(\boldsymbol{G}_2)$. The edge, node and name functions are the same as the original spangraph morphism $\boldsymbol{f}$, since these do not change when we apply either *sgph* or *hgph*. Also, the indication, comediator and projection functions for source and target are the case functions. By the constraints on the quartets of $\boldsymbol{f}$, the vertex function is the case function and defined in terms of the edge and name functions. Hence, $\boldsymbol{f} = \mathit{sgph}(\mathit{hgph}(\boldsymbol{f}))$.

```
(20) (ADJ$adjunction adjoint-equivalence)
     (= (ADJ$underlying-category adjoint-equivalence) spangraph)
     (= (ADJ$free-category adjoint-equivalence) hgph$hypergraph)
     (= (ADJ$left-adjoint adjoint-equivalence) hypergraph)
     (= (ADJ$right-adjoint adjoint-equivalence) hgph$spangraph)
     (= (ADJ$unit adjoint-equivalence) realization)
     (= (ADJ$counit adjoint-equivalence)
        (NAT$vertical-identity (FUNC$identity hgph$hypergraph)))
```

The Contexts of Spangraph Fibers

sgph.fbr

Categories

For any set $\boldsymbol{X}$ representing a fixed set of names, there is a fiber *spangraph* subcategory

$$\mathsf{Spangraph}(\boldsymbol{X}) \subseteq \mathsf{Spangraph}$$

consisting of all spangraphs whose name set is $\boldsymbol{X}$ and all spangraph morphisms whose name bijection is $id_X : \boldsymbol{X} \rightarrow \boldsymbol{X}$. Since Spangraph is the origin, all the functors and natural transformations for the full category of spangraphs can be restricted to fibers.

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

$$\begin{tikzcd}
\mathrm{Y} \\
& \mathrm{X} \arrow[ul, "\mathrm{h}"']
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=large, row sep=large]
\boldsymbol{Spangraph}(\boldsymbol{Y}) \arrow[rr, shift left, "\mathbf{hgph}(\boldsymbol{Y})"] \arrow[dr, shift left, "\mathbf{sgph}(\boldsymbol{h}^{-1})"'] \arrow[ddr, "\mathbf{incl}(\boldsymbol{Y})"'] & & \boldsymbol{Hypergraph}(\boldsymbol{Y}) \arrow[ll, shift left, "\mathbf{sgph}(\boldsymbol{Y})"] \arrow[ll, phantom, "\equiv_{Y}"] \arrow[dr, shift left, "\mathbf{hgph}(\boldsymbol{h}^{-1})"'] \arrow[ddr, "\mathbf{incl}(\boldsymbol{Y})"] & \\
& \boldsymbol{Spangraph}(\boldsymbol{X}) \arrow[ul, shift left, "\mathbf{sgph}(\boldsymbol{h})"'] \arrow[ul, phantom, "\cong"] \arrow[rr, shift left, "\mathbf{hgph}(\boldsymbol{X})"] \arrow[d, "\mathbf{incl}(\boldsymbol{X})"'] & & \boldsymbol{Hypergraph}(\boldsymbol{X}) \arrow[ul, shift left, "\mathbf{hgph}(\boldsymbol{h})"'] \arrow[ul, phantom, "\cong"] \arrow[ll, shift left, "\mathbf{sgph}(\boldsymbol{X})"] \arrow[ll, phantom, "\equiv_{X}"] \arrow[d, "\mathbf{incl}(\boldsymbol{X})"] \\
& \boldsymbol{Spangraph} \arrow[rr, shift left, "\mathbf{hgph}"] & & \boldsymbol{Hypergraph} \arrow[ll, shift left, "\mathbf{sgph}"] \arrow[ll, phantom, "\equiv"]
\end{tikzcd}$$

Diagram 3: Fiber Categories, Functors and Adjunctions

Functors

For any set $\boldsymbol{X}$ representing a fixed set of names, there is an *inclusion* functor of the fiber into the full category of spangraphs.

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

For any set bijection $\boldsymbol{h} : \boldsymbol{X} \to \boldsymbol{Y}$ there is a *direct image* functor

$$\boldsymbol{h}^{+1} : \mathsf{Spangraph}(\boldsymbol{X}) \to \mathsf{Spangraph}(\boldsymbol{Y})$$

along $\boldsymbol{h}$ and an *inverse image* functor

$$\boldsymbol{h}^{-1} : \mathsf{Spangraph}(\boldsymbol{Y}) \to \mathsf{Spangraph}(\boldsymbol{X})$$

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

$$\mathsf{Spangraph}(\boldsymbol{X}) \cong \mathsf{Spangraph}(\boldsymbol{Y}).$$

```
(3) (KIF$function direct-image)
    (= (KIF$source direct-image) set.ftn$bijection)
    (= (KIF$target direct-image) FUNC$functor)
    (forall (?h (set.ftn$bijection ?h))
        (and (= (FUNC$source (direct-image ?h))   (spangraph (set.ftn$source ?h)))
             (= (FUNC$target (direct-image ?h))   (spangraph (set.ftn$target ?h)))
             (= (FUNC$object (direct-image ?h))   (sgph.fbr.obj$direct-image ?h))
             (= (FUNC$morphism (direct-image ?h)) (sgph.fbr.mor$direct-image ?h))))

(4) (KIF$function inverse-image)
    (= (KIF$source inverse-image) set.ftn$bijection)
    (= (KIF$target inverse-image) FUNC$functor)
    (forall (?h (set.ftn$bijection ?h))
        (and (= (FUNC$source (inverse-image ?h))   (spangraph (set.ftn$target ?h)))
             (= (FUNC$target (inverse-image ?h))   (spangraph (set.ftn$source ?h)))
             (= (FUNC$object (inverse-image ?h))   (sgph.fbr.obj$inverse-image ?h))
             (= (FUNC$morphism (inverse-image ?h)) (sgph.fbr.mor$inverse-image ?h))))

(5) (forall (?h (set.ftn$bijection ?h))
        (and (= (FUNC$composition [(direct-image ?h) (inverse-image ?h)])
                (FUNC$identity (spangraph (set.ftn$source ?h))))
             (= (FUNC$composition [(inverse-image ?h) (direct-image ?h)])
                (FUNC$identity (spangraph (set.ftn$target ?h))))
             (FUNC$isomorphic (spangraph (set.ftn$source ?h)) (spangraph (set.ftn$target ?h)))))
```

For any set $\boldsymbol{X}$ representing a fixed set of names, there are fiber vertex, edge, node, name and hypergraph functors, which are restrictions of the full vertex, edge, node, name and hypergraph functors. The first four component functors can be defined as the restriction of their un-fibered correspondents, or the composition with fiber inclusion functor. The fiber hypergraph functor commutes through inclusion with the full hypergraph functor – it is its restriction: $\mathit{hgph}(\boldsymbol{X}) \circ \mathit{incl}(\boldsymbol{X}) = \mathit{incl}(\boldsymbol{X}) \circ \mathit{hgph}$.

```
(6) (KIF$function vertex)
    (= (KIF$source vertex) set$set)
    (= (KIF$target vertex) FUNC$functor)
    (forall (?x (set$set ?x))
        (and (= (FUNC$source (vertex ?x)) (spangraph ?x))
             (= (FUNC$target (vertex ?x)) set$set)
             (= (FUNC$object (vertex ?x)) (sgph.fbr.obj$vertex ?x))
             (= (FUNC$morphism (vertex ?x)) (sgph.fbr.mor$vertex (set.ftn$identity ?x)))
             (= (vertex ?x) (FUNC$composition [(inclusion ?x) sgph$vertex]))))

(7) (KIF$function edge)
    (= (KIF$source edge) set$set)
    (= (KIF$target edge) FUNC$functor)
    (forall (?x (set$set ?x))
        (and (= (FUNC$source (edge ?x)) (spangraph ?x))
             (= (FUNC$target (edge ?x)) set$set)
             (= (FUNC$object (edge ?x)) (sgph.fbr.obj$edge ?x))
             (= (FUNC$morphism (edge ?x)) (sgph.fbr.mor$edge (set.ftn$identity ?x)))
             (= (edge ?x) (FUNC$composition [(inclusion ?x) sgph$edge]))))

(8) (KIF$function node)
    (= (KIF$source node) set$set)
    (= (KIF$target node) FUNC$functor)
    (forall (?x (set$set ?x))
        (and (= (FUNC$source (node ?x)) (spangraph ?x))
             (= (FUNC$target (node ?x)) set$set)
             (= (FUNC$object (node ?x)) (sgph.fbr.obj$node ?x))
             (= (FUNC$morphism (node ?x)) (sgph.fbr.mor$node (set.ftn$identity ?x)))
             (= (node ?x) (FUNC$composition [(inclusion ?x) sgph$node]))))

(9) (KIF$function name)
    (= (KIF$source name) set$set)
    (= (KIF$target name) FUNC$functor)
    (forall (?x (set$set ?x))
        (and (= (FUNC$source (name ?x)) (spangraph ?x))
             (= (FUNC$target (name ?x)) set$semi)
             (= (FUNC$object (name ?x)) (sgph.fbr.obj$name ?x))
             (= (FUNC$morphism (name ?x)) (sgph.fbr.mor$name (set.ftn$identity ?x)))
             (= (name ?x) (FUNC$composition [(inclusion ?x) sgph$name]))
             (= (name ?x) ((FUNC$diagonal [(spangraph ?x) set$semi]) ?x))))

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

(11) (forall (?x (set$set ?x))
         (= (FUNC$composition [(hypergraph ?x) (hgph.fbr$inclusion ?x)])
            (FUNC$composition [(inclusion ?x) sgph$hypergraph])))

(12) (forall (?x (set$set ?x))
         (= (FUNC$composition [(hgph.fib$spangraph ?x) (hypergraph ?x)])
            (FUNC$identity (hgph.fib$hypergrqph ?x))))
```

These fiber functors satisfy the same kind of identities as the full functors: $\mathit{hgph}(\boldsymbol{X}) \circ \mathit{edge}(\boldsymbol{X}) = \mathit{edge}(\boldsymbol{X})$, $\mathit{hgph}(\boldsymbol{X}) \circ \mathit{node}(\boldsymbol{X}) = \mathit{node}(\boldsymbol{X})$ and $\mathit{hgph}(\boldsymbol{X}) \circ \mathit{name}(\boldsymbol{X}) = \mathit{name}(\boldsymbol{X})$.

```
(13) (forall (?x (set$set ?x))
         (and (= (FUNC$composition [(hypergraph ?x) (hgph.fbr$edge ?x)]) (edge ?x))
              (= (FUNC$composition [(hypergraph ?x) (hgph.fbr$node ?x)]) (node ?x))
              (= (FUNC$composition [(hypergraph ?x) (hgph.fbr$name ?x)]) (name ?x))))
```

Natural Transformations

For any set ***X*** representing a fixed set of names, there are a fiber indication, comediation, projection, bijection and realization natural transformations, which are restrictions of the full indication, comediation, projection, bijection and realization natural transformations. The first four natural transformations can be defined as the restriction of their un-fibered correspondents, or the composition with fiber inclusion functor. The latter two are natural isomorphisms. These realization natural isomorphism satisfies the identity: $\mathit{real}(\boldsymbol{X}) \circ \mathit{incl}(\boldsymbol{X}) = \mathit{incl}(\boldsymbol{X}) \circ \mathit{real}$.

```
(14) (KIF$function indication)
     (= (KIF$source indication) set$set)
     (= (KIF$target indication) NAT$natural-transformation)
     (forall (?x (set$set ?x))
         (and (= (NAT$source-category (indication ?x)) (spangraph ?x))
              (= (NAT$target-category (indication ?x)) set$set)
              (= (NAT$source-functor (indication ?x)) (vertex ?x))
              (= (NAT$target-functor (indication ?x)) (edge ?x))
              (= (NAT$component (indication ?x)) (sgph.fbr.obj$indication ?x))
              (= (indication ?x)
                 (NAT$horizontal-composition
                     [(NAT$vertical-identity (inclusion ?x)) sgph$indication]))))

(15) (KIF$function comediation)
     (= (KIF$source comediation) set$set)
     (= (KIF$target comediation) NAT$natural-transformation)
     (forall (?x (set$set ?x))
         (and (= (NAT$source-category (comediation ?x)) (spangraph ?x))
              (= (NAT$target-category (comediation ?x)) set$set)
              (= (NAT$source-functor (comediation ?x)) (vertex ?x))
              (= (NAT$target-functor (comediation ?x)) (node ?x))
              (= (NAT$component (comediation ?x)) (sgph.fbr.obj$comediation ?x))
              (= (comediation ?x)
                 (NAT$horizontal-composition
                     [(NAT$vertical-identity (inclusion ?x)) sgph$comediation]))))

(16) (KIF$function projection)
     (= (KIF$source projection) set$set)
     (= (KIF$target projection) NAT$natural-transformation)
     (forall (?x (set$set ?x))
         (and (= (NAT$source-category (projection ?x)) (spangraph ?x))
              (= (NAT$target-category (projection ?x)) set$set)
              (= (NAT$source-functor (projection ?x)) (vertex ?x))
              (= (NAT$target-functor (projection ?x))
                 ((FUNC$diagonal [(spangraph ?x) set$set]) ?x))
              (= (NAT$component (projection ?x)) (sgph.fbr.obj$projection ?x))
              (= (projection ?x)
                 (NAT$horizontal-composition
                     [(NAT$vertical-identity (inclusion ?x)) sgph$projection]))))

(17) (KIF$function bijection)
     (= (KIF$source bijection) set$set)
     (= (KIF$target bijection) NAT$natural-isomorphism)
     (forall (?x (set$set ?x))
         (and (= (NAT$source-category (bijection ?x)) (spangraph ?x))
              (= (NAT$target-category (bijection ?x)) set$set)
              (= (NAT$source-functor (bijection ?x)) (vertex ?x))
              (= (NAT$target-functor (bijection ?x))
                 (FUNC$composition [(hypergraph ?x) (hgph.fbr$case ?x)]))
              (= (NAT$component (bijection ?x)) (sgph.fbr.obj$bijection ?x))
              (= (bijection ?x)
                 (NAT$horizontal-composition
                     [(NAT$vertical-identity (inclusion ?x)) sgph$bijection]))))

(18) (KIF$function realization)
     (= (KIF$source realization) set$set)
     (= (KIF$target realization) NAT$natural-isomorphism)
     (forall (?x (set$set ?x))
         (and (= (NAT$source-category (realization ?x)) (spangraph ?x))
              (= (NAT$target-category (realization ?x)) (spangraph ?x))
              (= (NAT$source-functor (realization ?x)) (FUNC$identity (spangraph ?x)))
              (= (NAT$target-functor (realization ?x))
                 (FUNC$composition [(hypergraph ?x) (hgph.fib$spangraph ?x)]))
              (= (NAT$component (realization ?x)) (sgph.fbr.obj$realization ?x))))

(19) (forall (?x (set$set ?x))
         (= (NAT$horizontal-composition
                [(NAT$vertical-identity (inclusion ?x)) sgph$realization])
            (NAT$horizontal-composition
                [(realization ?x) (NAT$vertical-identity (inclusion ?x))])))
```

Adjunctions

For any set ***X*** representing a fixed set of names, the realization adjunction

$$\rho(X) = \langle hgph(X), sgph(X), \eta(X), 1\rangle : \mathsf{Spangraph}(X) \rightarrow \mathsf{Hypergraph}(X),$$

forms an adjoint equivalence. This equivalence links the fiber contexts of spangraphs and hypergraphs through realization – the category Hypergraph(***X***) is categorically equivalent to the category Spangraph(***X***) with the fiber spangraph functor *sgph*(***X***) : Spangraph(***X***) → Hypergraph(***X***) as equivalence.

```
(20) (KIF$function adjoint-equivalence)
    (= (KIF$source adjoint-equivalence) set$set)
    (= (KIF$target adjoint-equivalence) ADJ$adjunction)
    (forall (?x (set$set ?x))
        (and (= (ADJ$underlying-category (adjoint-equivalence ?x)) (spangraph ?x))
             (= (ADJ$free-category (adjoint-equivalence ?x)) (hgph.fbr$hypergraph ?x))
             (= (ADJ$left-adjoint (adjoint-equivalence ?x)) (hypergraph ?x))
             (= (ADJ$right-adjoint (adjoint-equivalence ?x)) (hgph.fbr$spangraph ?x))
             (= (ADJ$unit (adjoint-equivalence ?x)) (realization ?x))
             (= (ADJ$counit (adjoint-equivalence ?x))
                (NAT$vertical-identity (FUNC$identity (hgph.fbr$hypergraph ?x))))))
```

Spangraphs

`sgph.obj`

$$\begin{tikzcd}
\text{nexus} & \text{prehensions} & \text{actualities} \\
\mathit{edge}(\boldsymbol{G}) & \mathit{vert}(\boldsymbol{G}) \arrow[l, "\langle_G"'] \arrow[r, "\rangle_G"] \arrow[d, "\underline{\downarrow}_G"'] & \mathit{node}(\boldsymbol{G}) \\
& \mathit{name}(\boldsymbol{G}) & \\
& \text{names} &
\end{tikzcd}$$

Figure 1a: Spangraph

$$\begin{tikzcd}
1 & 0 \arrow[l, "\underline{01}"'] \arrow[r, "\underline{02}"] \arrow[d, "\underline{03}"'] & 2 \\
& 3 &
\end{tikzcd}$$

Figure 1b: The Category T

This section discusses spangraphs. First, we give a concise mathematical definition, and then we discuss and formalize the various parts of this definition.

A *spangraph* $\boldsymbol{G} = \langle \mathit{indic}(\boldsymbol{G}), \mathit{comed}(\boldsymbol{G}), \mathit{proj}(\boldsymbol{G})\rangle$ (see Figure 1a) consists of

- an *indication* function $\langle_{\boldsymbol{G}} = \mathit{indic}(\boldsymbol{G})$,
- a *comediation* function $\rangle_{\boldsymbol{G}} = \mathit{comed}(\boldsymbol{G})$, and
- a *projection* function $\underline{\downarrow}_{\boldsymbol{G}} = \mathit{proj}(\boldsymbol{G})$,

that all share a common source set (and satisfy the multi-pullback constraints)

$\mathit{src}(\mathit{indic}(\boldsymbol{G})) = \mathit{src}(\mathit{comed}(\boldsymbol{G})) = \mathit{src}(\mathit{proj}(\boldsymbol{G}))$.

Hence, a spangraph is a T-shaped diagram of sets and functions. More abstractly, a spangraph $\boldsymbol{G}$ is a Set-valued functor $\boldsymbol{G} : \mathsf{T} \rightarrow \mathsf{Set}$ on the category T (Figure 1b), where $\mathit{indic}(\boldsymbol{G}) = \boldsymbol{G}(\underline{01})$, $\mathit{comed}(\boldsymbol{G}) = \boldsymbol{G}(\underline{02})$ and $\mathit{proj}(\boldsymbol{G}) = \boldsymbol{G}(\underline{03})$. A spangraph is determined by the function triple $\langle \mathit{indic}(\boldsymbol{G}), \mathit{comed}(\boldsymbol{G}), \mathit{proj}(\boldsymbol{G})\rangle$ of indication, comediation and projection.

```
(1) (SET$class object)

(2) (SET.FTN$function indication)
    (= (SET.FTN$source indication) object)
    (= (SET.FTN$target indication) set.ftn$function)

(3) (SET.FTN$function comediation)
    (= (SET.FTN$source comediation) object)
    (= (SET.FTN$target comediation) set.ftn$function)

(4) (SET.FTN$function projection)
    (= (SET.FTN$source projection) object)
    (= (SET.FTN$target projection) set.ftn$function)

(5) (= (SET.FTN$composition [indication set.ftn$source])
       (SET.FTN$composition [comediation set.ftn$source]))
    (= (SET.FTN$composition [comediation set.ftn$source])
       (SET.FTN$composition [projection set.ftn$source]))

(6) (forall (?g1 (object ?g1) ?g2 (object ?g2))
        (=> (and (= (indication ?g1) (indication ?g2))
                 (= (comediation ?g1) (comediation ?g2))
                 (= (projection ?g1) (projection ?g2)))
            (= ?g1 ?g2)))
```

For convenience of practical reference, we introduce additional spangraph terminology for the source and target components of these functions. The common source set is called the *vertex* set of $\boldsymbol{G}$ and denoted $\mathit{vert}(\boldsymbol{G})$. Other terminology might be vertex, prehension or arc set. The target of the indication function is called the *edge* set of $\boldsymbol{G}$ and denoted $\mathit{edge}(\boldsymbol{G})$. In terms of the equivalent hypergraph notion, these are hyperedges. Other terminology might be nexus set. The target set of the comediation function is called the *node* set of $\boldsymbol{G}$ and denoted $\mathit{node}(\boldsymbol{G})$. Other terminology might be set of actualities. The target set of the projection function is called the *name* set of $\boldsymbol{G}$ and denoted $\mathit{name}(\boldsymbol{G})$. In summary, we provide terminology for the following sets:

- the set of *vertices* $\mathit{vert}(\boldsymbol{G}) = \mathit{src}(\mathit{indic}(\boldsymbol{G})) = \mathit{src}(\mathit{comed}(\boldsymbol{G})) = \mathit{src}(\mathit{proj}(\boldsymbol{G}))$,
- the set of (hyper) *edges* $\mathit{edge}(\boldsymbol{G}) = \mathit{tgt}(\mathit{indic}(\boldsymbol{G}))$,
- the set of *nodes* $\mathit{node}(\boldsymbol{G}) = \mathit{tgt}(\mathit{comed}(\boldsymbol{G}))$, and
- the set of *names* $\mathit{name}(\boldsymbol{G}) = \mathit{tgt}(\mathit{proj}(\boldsymbol{G}))$.

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

- *indication* function $‹_{\boldsymbol{G}} = \mathit{indic}(\boldsymbol{G}) : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{edge}(\boldsymbol{G})$,
- *comediation* function $›_{\boldsymbol{G}} = \mathit{comed}(\boldsymbol{G}) : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{node}(\boldsymbol{G})$, and
- *projection* function $\pm_{\boldsymbol{G}} = \mathit{proj}(\boldsymbol{G}) : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{name}(\boldsymbol{G})$,

Hence, in long form a spangraph is a septuple $\boldsymbol{G} = \langle \mathit{vert}(\boldsymbol{G}), ‹_{\boldsymbol{G}}, \mathit{edge}(\boldsymbol{G}), ›_{\boldsymbol{G}}, \mathit{node}(\boldsymbol{G}), \pm_{\boldsymbol{G}}, \mathit{name}(\boldsymbol{G}) \rangle$. In the idea of a spangraph, the prehensile nature of vertices connects the nexus of (hyper)edges to participating nodes (actualities). Vertices could be called arcs, but that might possibly result in confusion with edges. The term "vertex" corresponds to the span nature of spangraphs. Names reference nodes within the nexus of a (hyper)edge. We assume that each spangraph has an adequate, possibly denumerable, set of names $\mathit{name}(\boldsymbol{G})$.

```
(7) (SET.FTN$function vertex)
    (= (SET.FTN$source vertex) object)
    (= (SET.FTN$target vertex) set$set)
    (= (SET.FTN$composition [indication set.ftn$source]) vertex)
    (= (SET.FTN$composition [target set.ftn$source]) vertex)
    (= (SET.FTN$composition [label set.ftn$source]) vertex)

(8) (SET.FTN$function edge)
    (= (SET.FTN$source edge) object)
    (= (SET.FTN$target edge) set$set)
    (= (SET.FTN$composition [indication set.ftn$target]) edge)

(9) (SET.FTN$function node)
    (= (SET.FTN$source node) object)
    (= (SET.FTN$target node) set$set)
    (= (SET.FTN$composition [comediation set.ftn$target]) node)

(10) (SET.FTN$function name)
     (= (SET.FTN$source name) object)
     (= (SET.FTN$target name) set$set)
     (= (SET.FTN$composition [projection set.ftn$target]) name)
```

Figure 2 illustrates a spangraph $\boldsymbol{eg} = \langle \mathit{vert}(\boldsymbol{eg}), ‹_{\boldsymbol{eg}}, \mathit{edge}(\boldsymbol{eg}), ›_{\boldsymbol{eg}}, \mathit{node}(\boldsymbol{eg}), \pm_{\boldsymbol{eg}}, \mathit{name}(\boldsymbol{eg}) \rangle$ with

- $\mathit{vert}(\boldsymbol{eg}) = \{(e_1, n_1), (e_1, n_2), (e_1, n_3), (e_2, n_2), (e_2, n_4), (e_3, n_3), (e_3, n_4)\}$,
- $\mathit{edge}(\boldsymbol{eg}) = \{e_1, e_2, e_3\}$,
- $\mathit{node}(\boldsymbol{eg}) = \{n_1, n_2, n_3, n_4\}$,
- $\mathit{name}(\boldsymbol{eg}) = \mathit{vert}(\boldsymbol{eg})$,
- $‹_{\boldsymbol{eg}} = \pi_{\mathit{edge}} : \mathit{vert}(\boldsymbol{eg}) \rightarrow \mathit{edge}(\boldsymbol{eg})$,
- $›_{\boldsymbol{eg}} = \pi_{\mathit{node}} : \mathit{vert}(\boldsymbol{eg}) \rightarrow \mathit{node}(\boldsymbol{eg})$, and
- $\pm_{\boldsymbol{eg}} = \mathit{id} : \mathit{vert}(\boldsymbol{eg}) \rightarrow \mathit{name}(\boldsymbol{eg})$,

where the vertices are edge-node pairs that are connected, any edge labels itself, the indication and comediation functions are the binary product projection functions, and the projection function is the identity. Here, (hyper) edges can be identified with subsets of nodes.

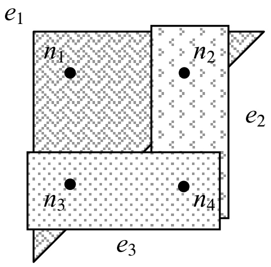


Figure 2: Example

```
(11) (object example)
     ((edge example) e1) ((edge example) e2) ((edge example) e3)
     ((node example) n1) ((node example) n2) ((node example) n3) ((node example) n3)
     (set.lim.prd2$pair example-pair)
     (= (set.lim.prd2$set1 example-pair) (edge example))
     (= (set.lim.prd2$set2 example-pair) (node example))
```

```
      (set$subset (vertex example) (set.lim.prd2$binary-product edge-node))
      ((vertex example) [e1 n1]) ((vertex example) [e1 n2]) ((vertex example) [e1 n3])
      ((vertex example) [e2 n2]) ((vertex example) [e2 n4]) ((vertex example) [e3 n3])
      ((vertex example) [e3 n4])
      (= (name example) (vertex example))
      (= (indication example)
         (set.ftn$composition
             [(set.ftn$inclusion [(vertex example) (set.lim.prd2$binary-product example-pair)])
              (set.lim.prd2$projection1 example-pair)]))
      (= (comediation example)
         (set.ftn$composition
             [(set.ftn$inclusion [(vertex example) (set.lim.prd2$binary-product example-pair)])
              (set.lim.prd2$projection2 example-pair)]))
      (= (projection example)
         (set.ftn$identity (vertex example)))
```

A spangraph must satisfy the *equivalence-identity condition*: if $\upsilon_1 \equiv_{\boldsymbol{G}} \upsilon_2$ then $\upsilon_1 = \upsilon_2$ for each pair of vertices $\upsilon_1, \upsilon_2 \in$ *vert*(***G***), where the equivalence relation $\upsilon_1 \equiv_{\boldsymbol{G}} \upsilon_2$ means that $‹_{\boldsymbol{G}}(\upsilon_1) = ‹_{\boldsymbol{G}}(\upsilon_2)$ and $⤓_{\boldsymbol{G}}(\upsilon_1) = ⤓_{\boldsymbol{G}}(\upsilon_2)$. To paraphrase this: an edge has only one vertex labeled with a particular name. This condition models the idea that a spangraph vertex is said to *belong* to its (hyper) edge, but to be (only) *attached* to its node. This means that the source-label product pairing function

$$(‹_{\boldsymbol{G}}, ⤓_{\boldsymbol{G}}) : \mathit{vert}(\boldsymbol{G}) \rightarrow \mathit{edge}(\boldsymbol{G})\times\mathit{name}(\boldsymbol{G})$$

is an injection. The equivalence-identity condition is equivalent to the condition that $⤓_{\boldsymbol{G}}$ is an injection on $‹_{\boldsymbol{G}}^{-1}(\varepsilon)$ for any edge $\varepsilon \in$ *edge*(***G***). The equivalence-identity condition allows a vertex $\upsilon \in$ *vert*(***G***) to be represented by its edge-name pair $(‹_{\boldsymbol{G}}, ⤓_{\boldsymbol{G}})(\upsilon) = (‹_{\boldsymbol{G}}(\upsilon), ⤓_{\boldsymbol{G}}(\upsilon)) = (\varepsilon, x)$; symbolize this as $\upsilon \cong (\varepsilon, x)$.

```
(12) (forall (?g (object ?g))
         (forall (?v1 ?v2 ((vertex ?g) ?v1) ((vertex ?g) ?v2))
             (=> (and (= ((indication ?g) ?v1) ((indication ?g) ?v2))
                      (= ((projection ?g) ?v1) ((projection ?g) ?v2)))
                 (= ?v1 ?v2))))
```

$$\begin{tikzcd}
\mathit{edge}(\boldsymbol{G}) \arrow[rr, "\#_{\mathit{hgph}(\boldsymbol{G})}"] \arrow[dr, "\partial_{\mathit{hgph}(\boldsymbol{G})}"'] & & \wp\,\mathit{name}(\boldsymbol{G}) & \mathit{name}(\boldsymbol{G}) \arrow[d, phantom, "\}\ *_{\mathit{hgph}(\boldsymbol{G})}"] \\
& \mathit{tuple}(*_{\mathit{hgph}(\boldsymbol{G})}) \arrow[ur, "\mathit{arity}(*_{\mathit{hgph}(\boldsymbol{G})})"] & & \mathit{node}(\boldsymbol{G})
\end{tikzcd}$$

Figure 3: Hypergraph of a Spangraph

Any spangraph ***G*** has an associated hypergraph *hgph*(***G***) (Figure 3.

```
(13) (SET.FTN$function hypergraph)
     (= (SET.FTN$source hypergraph) object)
     (= (SET.FTN$target hypergraph) hgph.obj$object)
```

The hypergraph and spangraph agree on three sets:

- *edge*(*hgph*(***G***)) = *edge*(***G***),
- *node*(*hgph*(***G***)) = *node*(***G***), and
- *name*(*hgph*(***G***)) = *name*(***G***).

The hypergraph edge set is the spangraph edge set; the hypergraph node set is the spangraph node set; and the hypergraph name set is the spangraph name set. The hypergraph ignores the spangraph vertex set.

```
(14) (= (SET.FTN$composition [hypergraph hgph.obj$edge]) edge)
     (= (SET.FTN$composition [hypergraph hgph.obj$node]) node)
     (= (SET.FTN$composition [hypergraph hgph.obj$name]) name)
```

The idea from the equivalence-identity condition of representing a vertex by its edge-name pair can be

$$\begin{tikzcd}
edge(\boldsymbol{G}) \arrow[dd, "id"'] & & vert(\boldsymbol{G}) \arrow[ll, "\text{‹}_{\boldsymbol{G}}"'] \arrow[rrr, "\text{›}_{\boldsymbol{G}}"] \arrow[dd, "\cong_{\boldsymbol{G}}"'] \arrow[dr, "\underline{\downarrow}_G"] & & & node(\boldsymbol{G}) \arrow[dd, "id"] \\
 & \text{‹}_{\uparrow} & & name(\boldsymbol{G}) \arrow[dd, "id", "\underline{\downarrow}_{\uparrow}"'] & \text{›}_{\uparrow} & \\
edge(\boldsymbol{G}) & & case(hgph(\boldsymbol{G})) \arrow[ll, "\text{‹}_{\boldsymbol{G}}"] \arrow[rrr, "\text{›}_{\boldsymbol{G}}"'] \arrow[dr, "\underline{\downarrow}_{\boldsymbol{G}}"'] & & & node(\boldsymbol{G}) \\
 & & & name(\boldsymbol{G}) & &
\end{tikzcd}$$

Figure ?: Realization Spangraph Morphism $\uparrow_G$

made more precise. Consider the range of the indication-projection product pairing function $(\text{‹}_{\boldsymbol{G}}, \underline{\downarrow}_{\boldsymbol{G}})$. This is precisely the set of cases of the hypergraph. The equivalence-identity condition can be expressed as a realization *bijection*

$\cong_{\boldsymbol{G}} : vert(\boldsymbol{G}) \rightarrow case(hgph(\boldsymbol{G}))$

from vertices to hypergraph cases. This is the vertex function of a *realization* spangraph isomorphism (Figure ?)

$\uparrow_{\boldsymbol{G}} : \boldsymbol{G} \rightarrow sgph(hgph(\boldsymbol{G}))$.

The edge, node and name functions for realization are the identities.

```
(15) (SET.FTN$function bijection)
     (= (SET.FTN$source bijection) object)
     (= (SET.FTN$target bijection) set.ftn$bijection)
     (= (SET.FTN$composition [bijection set.ftn$source]) vertex)
     (= (SET.FTN$composition [bijection set.ftn$target])
        (SET.FTN$composition [hypergraph hgph.obj$case]))

(16) (SET.FTN$function realization)
     (= (SET.FTN$source realization) object)
     (= (SET.FTN$target realization) sgph.mor$isomorphism)
     (= (SET.FTN$composition [realization vertex]) bijection)
     (= (SET.FTN$composition [realization sgph.mor$edge])
        (SET.FTN$composition [edge set.ftn$identity]))
     (= (SET.FTN$composition [realization sgph.mor$node])
        (SET.FTN$composition [node set.ftn$identity]))
     (= (SET.FTN$composition [realization sgph.mor$name])
        (SET.FTN$composition [name set.ftn$identity]))
```

We need to define the remaining components of the hypergraph of a spangraph

$\boldsymbol{G} = \langle node(\boldsymbol{G}), edge(\boldsymbol{G}), name(\boldsymbol{G}), *_{hgph(\boldsymbol{G})}, \#_{hgph(\boldsymbol{G})}, \partial_{hgph(\boldsymbol{G})} \rangle$:

- the *reference* set pair $*_{hgph(\boldsymbol{G})} = refer(hgph(\boldsymbol{G})) = \langle name(\boldsymbol{G}), node(\boldsymbol{G}) \rangle$,
- the *signature* function $\partial_{hgph(\boldsymbol{G})} = sign_{hgph(\boldsymbol{G})} : edge(\boldsymbol{G}) \rightarrow tuple(refer(hgph(\boldsymbol{G})))$, and
- the *edge-arity* function $\#_{hgph(\boldsymbol{G})} = arity_{hgph(\boldsymbol{G})} : edge(\boldsymbol{G}) \rightarrow \wp name(\boldsymbol{G})$.

The edge arity function $\#_{\boldsymbol{G}}$ and the signature function $\partial_{\boldsymbol{G}}$ satisfy the *arity condition* $\#_{\boldsymbol{G}}(\varepsilon) = arity(\partial_{\boldsymbol{G}}(\varepsilon))$ for all edges $\varepsilon \in edge(\boldsymbol{G})$. The reference set pair is already define by the components $name(\boldsymbol{G})$ and $node(\boldsymbol{G})$.

Arity: The equivalence-identity condition is equivalent to the condition that $\underline{\downarrow}_{\boldsymbol{G}}$ is an injection on $\text{‹}_{\boldsymbol{G}}^{-1}(\varepsilon)$ for any edge $\varepsilon \in edge(\boldsymbol{G})$. Hence, the edge arity function $\#_{hgph(\boldsymbol{G})} : edge(\boldsymbol{G}) \rightarrow \wp name(\boldsymbol{G})$ can be identified with the composition of the fiber of the source function and the power of the label function:

$\#_{hgph(\boldsymbol{G})}(\varepsilon) = \wp \underline{\downarrow}_{\boldsymbol{G}}(\text{‹}_{\boldsymbol{G}}^{-1}(\varepsilon))$ for any edge $\varepsilon \in edge(\boldsymbol{G})$.

Signature: For each edge $\varepsilon \in edge(G)$ the signature $\partial_{hgph(\boldsymbol{G})}(\varepsilon) : \#_{hgph(\boldsymbol{G})}(\varepsilon) \rightarrow node(\boldsymbol{G})$ can be defined as follows: $\partial_{hgph(\boldsymbol{G})}(\varepsilon)(x) = \text{›}_{\boldsymbol{G}}(\upsilon)$ for each name $x \in name(\boldsymbol{G})$ and vertex $\upsilon \in vert(\boldsymbol{G})$, where $\upsilon \cong (\varepsilon, x)$. By definition of $\#_{hgph(\boldsymbol{G})}(\varepsilon)$, there is such an vertex. By the equivalence-identity condition, there is only one such vertex. Hence, the hypergraph signature function $\partial_{hgph(\boldsymbol{G})} : edge(\boldsymbol{G}) \rightarrow tuple(refer(hgph(\boldsymbol{G})))$ is indirectly definable in terms of the target function.

```
(17) (forall (?g (object ?g))
         (and (= (hgph.obj$edge-arity (hypergraph ?g))
                 (set.ftn$composition
                     [(set.ftn$fiber (indication ?g))
                      (set.ftn$power (projection ?g))]))
              (forall (?v ((vertex ?g) ?v))
                  (= (((hgph.obj$signature (hypergraph ?g))
                         ((indication ?g) ?v))
                             ((projection ?g) ?v))
                     ((comediation ?g) ?v)))))
```

By virtue of the realization bijection between the set of vertices and hypergraph cases, various identifications can be made that further restrict realization.

Projection: The spangraph source function can be defined in terms of the hypergraph case indication, and the spangraph target function can be defined in terms of the hypergraph case comediation: for $\upsilon \cong (\varepsilon, x)$

$$indic_{hgph(\boldsymbol{G})}(\cong_{\boldsymbol{G}}(\upsilon)) = indic_{hgph(\boldsymbol{G})}(\varepsilon, x) = \upsilon = \langle_{\boldsymbol{G}}(\upsilon).$$

$$comed_{hgph(\boldsymbol{G})}(\cong_{\boldsymbol{G}}(\upsilon)) = comed_{hgph(\boldsymbol{G})}(\varepsilon, x) = \partial_{hgph(\boldsymbol{G})}(\varepsilon)(x) = \rangle_{\boldsymbol{G}}(\upsilon).$$

Labeling: The spangraph projection function can be defined in terms of the hypergraph case projection:

$$proj_{hgph(\boldsymbol{G})}(\cong_{\boldsymbol{G}}(\upsilon)) = proj_{hgph(\boldsymbol{G})}(\varepsilon, x) = x = \underline{\downarrow}_{\boldsymbol{G}}.$$

These properties express naturality for realization. They are half of the categorical equivalence between spangraphs and hypergraphs.

```
(18) (forall (?g (object ?g))
         (and (= (indication ?g)
                 (set.ftn$composition [(bijection ?g) (hgph.obj$indication (hypergraph ?g))]))
              (= (comediation ?g)
                 (set.ftn$composition [(bijection ?g) (hgph.obj$comediation (hypergraph ?g))]))
              (= (projection ?g)
                 (set.ftn$composition [(bijection ?g) (hgph.obj$projection (hypergraph ?g))]))))
```

Any hypergraph $\boldsymbol{G}$ has an associated spangraph $sgph(\boldsymbol{G})$, which is axiomatized in the hypergraph namespace. The hypergraph associated with the spangraph $sgph(\boldsymbol{G})$ is the original hypergraph:

$$hgph(sgph(\boldsymbol{G})) = \boldsymbol{G}.$$

This property is the other half of the categorical equivalence between spangraphs and hypergraphs.

```
(19) (= (SET.FTN$composition [hgph.obj$spangraph hypergraph])
        (SET.FTN$identity hgph.obj$object))
```

This identity and the naturality of realization imply the following fact: for any hypergraph $\boldsymbol{H}$, the realization of the spangraph $sgph(\boldsymbol{H})$ is the identity spangraph morphisms

$$\uparrow_{sgph(\boldsymbol{H})} : sgph(\boldsymbol{H}) \rightarrow sgph(hgph(sgph(\boldsymbol{H}))) = sgph(\boldsymbol{H}).$$

```
(20) (forall (?h (hgph.obj$object ?h))
         (= (realization (hgph.obj$spangraph ?h))
           (sgph.mor$identity (hgph.obj$spangraph ?h))))
```

Spangraph Fibers

`sgph.fbr.obj`

For any set $\boldsymbol{X}$ representing a fixed set of names, there is a *fiber* subclass of spangraphs $\mathsf{sgph}(\boldsymbol{X}) \subseteq \mathsf{sgph} = \mathit{obj}(\mathsf{Spangraph})$ whose name set is $\boldsymbol{X}$.

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

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

$$\mathit{incl}(\boldsymbol{X}) : \mathsf{sgph}(\boldsymbol{X}) \rightarrow \mathsf{sgph}$$

that injects the spangraph fiber class into the spangraph class.

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

For any set $\boldsymbol{X}$ representing a fixed set of names, there are indication, comediation and projection functions.

$$\mathit{indic}(\boldsymbol{X}) = \mathit{incl}(\boldsymbol{X}) \cdot \mathit{indic} : \mathsf{sgph}(\boldsymbol{X}) \rightarrow \mathsf{sgph} \rightarrow \mathsf{ftn}$$

$$\mathit{comed}(\boldsymbol{X}) = \mathit{incl}(\boldsymbol{X}) \cdot \mathit{comed} : \mathsf{sgph}(\boldsymbol{X}) \rightarrow \mathsf{sgph} \rightarrow \mathsf{ftn}$$

$$\mathit{proj}(\boldsymbol{X}) = \mathit{incl}(\boldsymbol{X}) \cdot \mathit{proj} : \mathsf{sgph}(\boldsymbol{X}) \rightarrow \mathsf{sgph} \rightarrow \mathsf{bftn}$$

```
(3) (KIF$function indication)
    (= (KIF$source indication) set$set)
    (= (KIF$target indication) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (indication ?x)) (object ?x))
             (= (SET.FTN$target (indication ?x)) set.ftn$function)
             (SET.FTN$restriction (indication ?x) sgph.obj$indication)
             (= (indication ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$indication]))))

(4) (KIF$function comediation)
    (= (KIF$source comediation) set$set)
    (= (KIF$target comediation) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (comediation ?x)) (object ?x))
             (= (SET.FTN$target (comediation ?x)) set.ftn$function)
             (SET.FTN$restriction (comediation ?x) sgph.obj$comediation)
             (= (comediation ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$comediation]))))

(5) (KIF$function projection)
    (= (KIF$source projection) set$set)
    (= (KIF$target projection) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (projection ?x)) (object ?x))
             (= (SET.FTN$target (projection ?x)) set.ftn$function)
             (SET.FTN$restriction (projection ?x) sgph.obj$projection)
             (= (projection ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$projection]))))

(6) (forall (?x (set$set ?x))
        (and (= (SET.FTN$composition [(indication ?x) set.ftn$source])
                (SET.FTN$composition [(comediation ?x) set.ftn$source]))
             (= (SET.FTN$composition [(comediation ?x) set.ftn$source])
                (SET.FTN$composition [(projection ?x) set.ftn$source]))))
```

For any set $\boldsymbol{X}$ representing a fixed set of names, we define fiber terminology for the vertex, edge, node and name sets.

$$\mathit{vert}(\mathbf{X}) = \mathit{incl}(\mathbf{X}) \cdot \mathit{vert} : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{set}$$

$$\mathit{edge}(\mathbf{X}) = \mathit{incl}(\mathbf{X}) \cdot \mathit{edge} : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{set}$$

$$\mathit{node}(\mathbf{X}) = \mathit{incl}(\mathbf{X}) \cdot \mathit{node} : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{set}$$

$$\mathit{name}(\mathbf{X}) = \mathit{incl}(\mathbf{X}) \cdot \mathit{name} : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{set}$$

```
(7) (KIF$function vertex)
    (= (KIF$source vertex) set$set)
    (= (KIF$target vertex) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (vertex ?x)) (object ?x))
             (= (SET.FTN$target (vertex ?x)) set$set)
             (SET.FTN$restriction (vertex ?x) sgph.obj$vertex)
             (= (vertex ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$vertex]))))

(8) (KIF$function edge)
    (= (KIF$source edge) set$set)
    (= (KIF$target edge) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (edge ?x)) (object ?x))
             (= (SET.FTN$target (edge ?x)) set$set)
             (SET.FTN$restriction (edge ?x) sgph.obj$edge)
             (= (edge ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$edge]))))

(9) (KIF$function node)
    (= (KIF$source node) set$set)
    (= (KIF$target node) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (node ?x)) (object ?x))
             (= (SET.FTN$target (node ?x)) set$set)
             (SET.FTN$restriction (node ?x) sgph.obj$node)
             (= (node ?x)
                (SET.FTN$composition [(inclusion ?x) sgph.obj$node]))))

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

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

$$\mathit{hgph}(\mathbf{X}) : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{hgph}(\mathbf{X})$$

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

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

We also define fiber terminology for realization bijection and realization

$$\cong(\mathbf{X}) = \cong_{\mathbf{X}} = \mathit{incl}(\mathbf{X}) \cdot \cong : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{ftn}$$

$$\mathit{real}(\mathbf{X}) = \mathit{incl}(\mathbf{X}) \cdot \mathit{real} : \mathrm{sgph}(\mathbf{X}) \rightarrow \mathrm{sgph} \rightarrow \mathrm{sgph.mor}.$$

The source and target of the realization bijection and realization itself can be expressed in terms of fiber functions: for any spangraph $\mathbf{G}$ in the fiber of $\mathbf{X}$,

$$\cong_{\mathbf{X}}(\mathbf{G}) : \mathit{vert}_{\mathbf{X}}(\mathbf{G}) \rightarrow \mathit{case}_{\mathbf{X}}(\mathit{hgph}_{\mathbf{X}}(\mathbf{G})).$$

$\uparrow_{X}(\boldsymbol{G}) : \boldsymbol{G} \rightarrow \mathit{sgph}_{\boldsymbol{X}}(\mathit{hgph}_{\boldsymbol{X}}(\boldsymbol{G}))$.

```
(12) (KIF$function bijection)
     (= (KIF$source bijection) set$set)
     (= (KIF$target bijection) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (bijection ?x)) (object ?x))
              (= (SET.FTN$target (bijection ?x)) set.ftn$bijection)
              (SET.FTN$restriction (bijection ?x) sgph.obj$bijection)
              (= (bijection ?x)
                 (SET.FTN$composition [(inclusion ?x) sgph.obj$bijection]))))

(13) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(bijection ?x) set.ftn$source]) (vertex ?x))
              (= (SET.FTN$composition [(bijection ?x) set.ftn$target])
                 (SET.FTN$composition [(hypergraph ?x) (hgph.fib.obj$case ?x)]))))

(14) (KIF$function realization)
     (= (KIF$source realization) set$set)
     (= (KIF$target realization) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (realization ?x)) (object ?x))
              (= (SET.FTN$target (realization ?x)) (sgph.fbr.mor$isomorphism ?x))
              (SET.FTN$restriction (realization ?x) sgph.obj$realization)))

(15) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$source ?x)])
                 (SET.FTN$identity (object ?x)))
              (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$target ?x)])
                 (SET.FTN$composition [(hypergraph ?x) (hgph.fib.obj$spangraph ?x)]))
              (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$vertex ?x)])
                 (bijection ?x))
              (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$edge ?x)])
                 (SET.FTN$composition [(edge ?x) set.ftn$identity]))
              (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$node ?x)])
                 (SET.FTN$composition [(node ?x) set.ftn$identity]))
              (= (SET.FTN$composition [(realization ?x) (sgph.fbr.mor$name ?x)])
                 (SET.FTN$composition [(name ?x) set.ftn$identity]))))
```

For any set function $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ the *inverse image* fiber operator (Diagram 4) $\boldsymbol{h}^{-1} : \mathsf{sgph}(\boldsymbol{Y}) \rightarrow \mathsf{sgph}(\boldsymbol{X})$ along $\boldsymbol{h}$ maps $\boldsymbol{Y}$-spangraphs to $\boldsymbol{X}$-spangraphs. The definitions of the components are as follows.

- *vertex* set $\mathit{vert}(\boldsymbol{h}^{-1}(\boldsymbol{G})) = \{(x, v) \mid x \in \boldsymbol{X}, v \in \mathit{vert}(\boldsymbol{G}), \boldsymbol{h}(x) = \pm_{G}(v)\} = \mathit{plbk}(\mathit{spn}(\boldsymbol{h}, \pm_{G}))$,
- *edge* set $\mathit{edge}(\boldsymbol{h}^{-1}(\boldsymbol{G})) = \mathit{edge}(\boldsymbol{G})$,
- *node* set $\mathit{node}(\boldsymbol{h}^{-1}(\boldsymbol{G})) = \mathit{node}(\boldsymbol{G})$,
- *name* set $\mathit{name}(\boldsymbol{h}^{-1}(\boldsymbol{G})) = \boldsymbol{X}$,
- *indication* function $‹_{\boldsymbol{h}-1(\boldsymbol{G})} = \mathit{vert}(\boldsymbol{h}) \cdot ‹_{G}$,
- *comediation* function $›_{\boldsymbol{h}-1(\boldsymbol{G})} = \mathit{vert}(\boldsymbol{h}) \cdot ›_{G}$, and
- *projection* function $\pm_{\boldsymbol{h}-1(\boldsymbol{G})} = \mathit{pr}_1(\mathit{spn}(\boldsymbol{h}, \pm_{G}))$,

where the vertex function is the second pullback projection $\mathit{vert}(\boldsymbol{h}) = \mathit{pr}_2(\mathit{spn}(\boldsymbol{h}, \pm_{G}))$.

$$\begin{tikzcd}
 & & \mathit{node}(\boldsymbol{G}) \\
\mathit{vert}(\boldsymbol{h}^{-1}(\boldsymbol{G})) \arrow[urr, "›_{\boldsymbol{h}-1(\boldsymbol{G})}"] \arrow[rr, "\mathit{vert}(\boldsymbol{h})"] \arrow[dr, "‹_{\boldsymbol{h}-1(\boldsymbol{G})}"] \arrow[dd, "\pm_{\boldsymbol{h}-1(\boldsymbol{G})}"'] & & \mathit{vert}(\boldsymbol{G}) \arrow[u, "›_{G}"'] \arrow[dl, "‹_{G}"'] \arrow[dd, "\pm_{G}"] \\
 & \mathit{edge}(\boldsymbol{G}) & \\
\mathit{name}(\boldsymbol{h}^{-1}(\boldsymbol{G})) = \boldsymbol{X} \arrow[rr, "\boldsymbol{h}"'] & & \boldsymbol{Y} = \mathit{name}(\boldsymbol{G})
\end{tikzcd}$$

Diagram 4: Inverse image

Check the equivalence-identity condition:

if $(x_1, v_1) \equiv_{\boldsymbol{h}-1(\boldsymbol{G})} (x_2, v_2)$ then $(x_1, v_1) = (x_2, v_2)$
for each pair of vertices $(x_1, v_1), (x_2, v_2) \in \mathit{vert}(\boldsymbol{h}^{-1}(\boldsymbol{G}))$.

This equivalence means that $‹_{\boldsymbol{h}-1(\boldsymbol{G})}(x_1, v_1) = ‹_{\boldsymbol{h}-1(\boldsymbol{G})}(x_2, v_2)$ and $\pm_{\boldsymbol{h}-1(\boldsymbol{G})}(x_1, v_1) = \pm_{\boldsymbol{h}-1(\boldsymbol{G})}(x_2, v_2)$. By definition of the indication and projection functions for the inverse image, the equivalence means $‹_{\boldsymbol{G}}(v_1) = ‹_{\boldsymbol{G}}(v_2)$ and $x_1 = x_2$. The definition of the vertex set for the inverse image (pullback condition) implies $\pm_{G}(v_1) = \boldsymbol{h}(x_1) = \boldsymbol{h}(x_2) = \pm_{G}(v_2)$. Finally, by the equivalence-identity condition for $\boldsymbol{G}$, we have $v_1 = v_2$.

Although this gives a well-defined inverse image, we do not axiomatize this for several reasons. It has no generalized inverse. There is no connecting link between the original spangraph and the inverse image. It is non-intuitive in concrete cases. The first two reasons occur based on our need for bijective name functions. Instead, we restrict the inverse image operator to bijections. Then all three objections disappear. Of course, this case is just a renaming, and the components of the inverse image reduce almost to a triviality.

For any set bijection $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ the *direct image* fiber operator $\boldsymbol{h}^{+1} : \mathsf{sgph}(\boldsymbol{X}) \rightarrow \mathsf{sgph}(\boldsymbol{Y})$ along $\boldsymbol{h}$ maps $\boldsymbol{X}$-spangraphs to $\boldsymbol{Y}$-spangraphs by composition with projection. For any spangraph $\boldsymbol{G_X}$ with name set $\boldsymbol{X}$, the direct image $\boldsymbol{h}^{+1}(\boldsymbol{G_X})$ is identical to $\boldsymbol{G_X}$, except that the name set is $\boldsymbol{Y}$ and the projection function is $\underline{\downarrow}_{\boldsymbol{h}-1(\boldsymbol{G})} = \underline{\downarrow}_{G} \cdot \boldsymbol{h}$.

There is a *direct connection* hypergraph morphism $\blacktriangleright_{\boldsymbol{h}}(\boldsymbol{G_X}) : \boldsymbol{G_X} \rightarrow \boldsymbol{h}^{+1}(\boldsymbol{G_X})$, whose vertex, edge and node functions are the identity functions and whose name function is $\boldsymbol{h}$.

```
(17) (KIF$function direct-image)
     (= (KIF$source direct-image) set.ftn$function)
     (= (KIF$target direct-image) SET.FTN$function)
     (forall (?h (set.ftn$function ?h))
         (and (= (SET.FTN$source (direct-image ?x)) (object (set.ftn$source ?h)))
              (= (SET.FTN$target (direct-image ?x)) (object (set.ftn$target ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (vertex (set.ftn$target ?h))])
                 (vertex (set.ftn$source ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (edge (set.ftn$target ?h))])
                 (edge (set.ftn$source ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (node (set.ftn$target ?h))])
                 (node (set.ftn$source ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (name (set.ftn$target ?h))])
                 ((set.ftn$constant [(object (set.ftn$source ?h)) set$set]) (set.ftn$target ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (indication (set.ftn$target ?h))])
                 (indication (set.ftn$source ?h)))
              (= (SET.FTN$composition [(direct-image ?h) (comediation (set.ftn$target ?h))])
                 (comediation (set.ftn$source ?h)))
              (forall (?g ((object (set.ftn$source ?h)) ?g)
                  (= (projection ((direct-image ?x) ?g))
                     (set.ftn$composition [(projection ?g) ?h]))))))

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

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

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

The direct image of an identity is the identity operator:

$(\mathit{id}_{\boldsymbol{X}})^{+1} = \mathit{id}$ for any set $\boldsymbol{X}$.

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

(20) (forall (?x (set$set ?x))
         (= (direct-image (set.ftn$identity ?x))
            (SET.FTN$identity (object ?x))))
```

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

$\blacktriangleright_{(\boldsymbol{f} \cdot \boldsymbol{g})}(\boldsymbol{G_X}) : \boldsymbol{G_X} \rightarrow (\boldsymbol{f} \cdot \boldsymbol{g})^{+1}(\boldsymbol{G_X}) = (\boldsymbol{f}^{+1} \cdot \boldsymbol{g}^{+1})(\boldsymbol{G_X}) = \boldsymbol{g}^{+1}(\boldsymbol{f}^{+1}(\boldsymbol{G_X}))$

$= \blacktriangleright_{\boldsymbol{f}}(\boldsymbol{G_X}) \cdot_{[\boldsymbol{f},\, \boldsymbol{g}]} \blacktriangleright_{\boldsymbol{g}}(\boldsymbol{f}^{+1}(\boldsymbol{G_X})) : \boldsymbol{G_X} \rightarrow \boldsymbol{f}^{+1}(\boldsymbol{G_X}) \rightarrow \boldsymbol{g}^{+1}(\boldsymbol{f}^{+1}(\boldsymbol{G_X}))$

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

The direct connection of an identity is the identity operator:

$\blacktriangleright_{id\boldsymbol{X}}(\boldsymbol{G_X}) = id(\boldsymbol{G_X})$ for any set $\boldsymbol{X}$.

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

$\blacktriangleright_{\boldsymbol{h}}(\boldsymbol{G_X}) \cdot_{[\boldsymbol{h},\, \boldsymbol{h}-1]} \blacktriangleright_{\boldsymbol{h}-1}(\boldsymbol{h}^{+1}(\boldsymbol{G_X})) = \blacktriangleright_{(\boldsymbol{h} \cdot \boldsymbol{h}-1)}(\boldsymbol{G_X}) = \blacktriangleright_{id\boldsymbol{X}}(\boldsymbol{G_X}) = id(\boldsymbol{G_X})$

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

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

(22) (forall (?x (set$set ?x)
              ?G ((object ?x) ?G))
         (= ((direct-connection (set.ftn$identity ?x)) ?G)
            ((sgph.fbr.mor$identity ?x) ?G)))
```

For any set bijection $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ the *inverse image* $\boldsymbol{h}^{-1}$: sgph($\boldsymbol{Y}$) → sgph($\boldsymbol{X}$) along $\boldsymbol{h}$ is a fiber operator that maps $\boldsymbol{Y}$-spangraphs to $\boldsymbol{X}$-spangraphs. For any spangraph $\boldsymbol{G_X}$ with name set $\boldsymbol{Y}$, the inverse image $\boldsymbol{h}^{-1}(\boldsymbol{G_X})$ is identical to $\boldsymbol{G_X}$, except that the name set is $\boldsymbol{X}$ and the projection function is $\underline{\downarrow}_{\boldsymbol{h}-1(\boldsymbol{G}X)} = \underline{\downarrow}_{\boldsymbol{G}X} \cdot \boldsymbol{h}^{-1}$. Clearly, the inverse image operator can be defined as the direct image along the inverse $\boldsymbol{h}^{-1} : \boldsymbol{Y} \rightarrow \boldsymbol{X}$.

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

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

(24) (KIF$function inverse-connection)
     (= (KIF$source inverse-connection) set.ftn$bijection)
     (= (KIF$target inverse-connection) SET.FTN$function)
     (forall (?h (set.ftn$bijection ?h))
         (and (= (SET.FTN$source (inverse-connection ?h))
                 (object (set.ftn$target ?h)))
              (= (SET.FTN$target (inverse-connection ?h))
                 (sgph.fib.mor$morphism (set.ftn$inverse ?h)))
              (= (SET.FTN$composition [(inverse-connection ?h) (sgph.fib.mor$target ?h)])
                 (SET.FTN$identity (object (set.ftn$target ?h))))
              (= (SET.FTN$composition [(inverse-connection ?h) (sgph.fib.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: $id_{sgph(\boldsymbol{X})} = \boldsymbol{h}^{+1} \cdot \boldsymbol{h}^{-1}$ and $id_{sgph(\boldsymbol{Y})} = \boldsymbol{h}^{-1} \cdot \boldsymbol{h}^{+1}$ for any set function $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$.

The direct and inverse connection operators are inverse to each other.

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

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

Spangraph Morphisms

`sgph.mor`

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

$$\begin{tikzcd}
\mathit{edge}(\boldsymbol{G}) \arrow[d, "\boldsymbol{edge}(\boldsymbol{f})"'] \arrow[dr, phantom, "‹_{\boldsymbol{f}}"] & \mathit{vert}(\boldsymbol{G}) \arrow[l, "‹_{\boldsymbol{G}}"'] \arrow[rr, "›_{\boldsymbol{G}}"] \arrow[dr, "\underline{\downarrow}_{G}"] \arrow[d, "\boldsymbol{vert}(\boldsymbol{f})"'] \arrow[drr, phantom, "›_{\boldsymbol{f}}"] & & \mathit{node}(\boldsymbol{G}) \arrow[d, "\boldsymbol{node}(\boldsymbol{f})"] \\
 & & \mathit{name}(\boldsymbol{G}) \arrow[d, "l"', "\cong"] \arrow[dl, phantom, "\underline{\downarrow}_{\boldsymbol{f}}"] & \\
\mathit{edge}(\boldsymbol{G'}) & \mathit{vert}(\boldsymbol{G'}) \arrow[l, "‹_{\boldsymbol{G'}}"] \arrow[rr, "›_{\boldsymbol{G'}}"'] \arrow[r, "\underline{\downarrow}_{\boldsymbol{G'}}"'] & \mathit{name}(\boldsymbol{G'}) & \mathit{node}(\boldsymbol{G'})
\end{tikzcd}$$

Figure 4: Spangraph Morphism

A *spangraph morphism* $\boldsymbol{f} = \langle \mathit{indic}(\boldsymbol{f}), \mathit{comed}(\boldsymbol{f}), \mathit{proj}(\boldsymbol{f}) \rangle : \boldsymbol{G}_1 \rightarrow \boldsymbol{G}_2$ from hypergraph $\boldsymbol{G}_1$ to hypergraph $\boldsymbol{G}_2$ (Figure 4) is a three dimensional construction consisting of

- an *indication* quartet $‹_{\boldsymbol{f}} = \mathit{indic}(\boldsymbol{f})$,
- a *comediation* quartet $›_{\boldsymbol{f}} = \mathit{comed}(\boldsymbol{f})$ and
- a *projection* hemiquartet $\underline{\downarrow}_{\boldsymbol{f}} = \mathit{proj}(\boldsymbol{f})$,

which share a common vertical source function

$\mathit{vert\text{-}src}(\mathit{indic}(\boldsymbol{f})) = \mathit{vert\text{-}src}(\mathit{comed}(\boldsymbol{f})) = \mathit{vert\text{-}src}(\mathit{proj}(\boldsymbol{f}))$.

Hence, a spangraph morphism is a morphism of T-shaped diagrams of sets and functions. More abstractly, a spangraph morphism ***f*** is either of two equivalent notions:

- a functor $\boldsymbol{f} : \mathsf{T} \rightarrow \mathsf{Set}^{\rightarrow}$ satisfying $\boldsymbol{f} \circ \boldsymbol{\pi}_1 = \boldsymbol{G}_1$ and $\boldsymbol{f} \circ \boldsymbol{\pi}_2 = \boldsymbol{G}_2$, where $\mathit{indic}(\boldsymbol{f}) = \boldsymbol{f}(\underline{01})$, $\mathit{comed}(\boldsymbol{f}) = \boldsymbol{f}(\underline{02})$ and $\mathit{proj}(\boldsymbol{f}) = \boldsymbol{f}(\underline{03})$ is a hemiquartet;
- a natural transformation $\boldsymbol{f} : \boldsymbol{G}_1 \Rightarrow \boldsymbol{G}_2 : \mathsf{T} \rightarrow \mathsf{Set}$, where $\mathit{vert}(\boldsymbol{f}) = \boldsymbol{f}(0)$, $\mathit{edge}(\boldsymbol{f}) = \boldsymbol{f}(1)$, $\mathit{node}(\boldsymbol{f}) = \boldsymbol{f}(2)$ and $\mathit{name}(\boldsymbol{f}) = \boldsymbol{f}(3)$ is a bijection.

A spangraph is determined by the function triple $\langle \mathit{indic}(\boldsymbol{G}), \mathit{comed}(\boldsymbol{G}), \mathit{proj}(\boldsymbol{G}) \rangle$ of indication, comediation and projection.

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

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

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

(4) (SET.FTN$function indication)
    (= (SET.FTN$source indication) morphism)
    (= (SET.FTN$target indication) set.qtt$quartet)

(5) (SET.FTN$function comediation)
    (= (SET.FTN$source comediation) morphism)
    (= (SET.FTN$target comediation) set.qtt$quartet)

(6) (SET.FTN$function projection)
    (= (SET.FTN$source projection) morphism)
    (= (SET.FTN$target projection) set.hqtt$quartet)
```

```
(7) (= (SET.FTN$composition [indication set.qtt$vertical-source])
       (SET.FTN$composition [comediation set.qtt$vertical-source]))
    (= (SET.FTN$composition [comediation set.qtt$vertical-source])
       (SET.FTN$composition [projection set.qtt$vertical-source]))

(8) (forall (?f1 (morphism ?f1) ?f2 (morphism ?f2))
        (=> (and (= (indication ?f1) (indication ?f2))
                 (= (comediation ?f1) (comediation ?f2))
                 (= (projection ?f1) (projection ?f2)))
            (= ?f1 ?f2)))
```

For convenience of reference, we introduce additional spangraph terminology for the vertical source and vertical target functions of these quartets. The common vertical source function is called the *vertex* function of ***f*** and denoted *vert*(***f***). The vertical target function of the indication quartet is called the *edge* function of ***f*** and denoted *edge*(***f***). The vertical target function of the comediation quartet is called the *node* function of ***f*** and denoted *node*(***f***). The vertical target bijection of the projection hemiquartet is called the *name* function of ***f*** and denoted *name*(***f***). In summary, a spangraph morphism has the following component functions:

- a *vertex* function *v* = *vert*(***f***) : *vert*(***G***) → *vert*(***G′***),
- an *edge* function *e* = *edge*(***f***) : *edge*(***G***) → *edge*(***G′***),
- a *node* function *n* = *node*(***f***) : *node*(***G***) → *node*(***G′***), and
- a *name* bijection *l* = *name*(***f***) : *name*(***G***) → *name*(***G′***).

This results in the following presentations of the indication, comediation and projection quartets:

- *indication* quartet $‹_{f}$ = *indic*(***f***) = ⟨*vert*(***f***), *edge*(***f***)⟩ : *indic*(*src*(***f***)) → *indic*(*tgt*(***f***)),
- *comediation* quartet $›_{f}$ = *comed*(***f***) = ⟨*vert*(***f***), *node*(***f***)⟩ : *comed*(*src*(***f***)) → *comed*(*tgt*(***f***)), and
- *projection* hemiquartet $⤓_{f}$ = *proj*(***f***) = ⟨*vert*(***f***), *name*(***f***)⟩ : *proj*(*src*(***f***)) → *proj*(*tgt*(***f***)).

Therefore, a spangraph morphism can be displayed as in Figure 2.

The indication quartet asserts the preservation of indication (edges): $v \cdot ‹_{G2} = ‹_{G1} \cdot e$. This means that the ***G***$_2$-indication (edge) of the image of any vertex υ ∈ *vert*(***G***$_1$) is the image of the ***G***$_1$-indication (edge) of the vertex: $‹_{G2}$(*vert*(*f*)(υ)) = *edge*(*f*)($‹_{G1}$(υ)).

The projection quartet asserts the preservation of projection (names): $v \cdot ⤓_{G2} = ⤓_{G1} \cdot l$. This means that the ***G***$_2$-projection (name) of the image of any vertex υ ∈ *vert*(***G***$_1$) is the image of the ***G***$_1$-projection (name) of the vertex: $⤓_{G2}$(*vert*(*f*)(υ)) = *name*(*f*)($⤓_{G1}$(υ)).

Symbolically these two preservation equations mean that,

if υ ≅ (ε, *x*) then *vert*(*f*)(υ) ≅ (*edge*(*f*)(ε), *name*(*f*)(*x*)).

The comediation quartet asserts the preservation of comediation (nodes): $v \cdot ›_{G2} = ›_{G1} \cdot n$. This means that the ***G***$_2$-comediation (node) of the image of any vertex υ ∈ *vert*(***G***$_1$) is the image of the ***G***$_1$-comediation (node) of the vertex: $›_{G2}$(*vert*(*f*)(υ)) = *node*(*f*)($›_{G1}$(υ)); or,

if $›_{G1}$(υ) = ν then $›_{G2}$(*vert*(*f*)(υ)) = *node*(*f*)(ν).

```
 (9) (SET.FTN$function vertex)
     (= (SET.FTN$source vertex) morphism)
     (= (SET.FTN$target vertex) set.ftn$function)
     (= vertex (SET.FTN$composition [indication set.qtt$vertical-source]))
     (= vertex (SET.FTN$composition [comediation set.qtt$vertical-source]))
     (= vertex (SET.FTN$composition [projection set.hqtt$vertical-source]))

(10) (SET.FTN$function edge)
     (= (SET.FTN$source edge) morphism)
     (= (SET.FTN$target edge) set.ftn$function)
     (= edge (SET.FTN$composition [indication set.qtt$vertical-target]))

(11) (SET.FTN$function node)
     (= (SET.FTN$source node) morphism)
     (= (SET.FTN$target node) set.ftn$function)
     (= node (SET.FTN$composition [comediation set.qtt$vertical-target]))

(12) (SET.FTN$function name)
```

```
      (= (SET.FTN$source name) morphism)
      (= (SET.FTN$target name) set.ftn$bijection)
      (= name (SET.FTN$composition [projection set.hqtt$vertical-target]))
```

A spangraph morphism $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{G}'$ must satisfy the *creation condition*: for each target vertex $\upsilon' \in \mathsf{vert}(\boldsymbol{G}')$, source edge $\varepsilon \in \mathsf{edge}(\boldsymbol{G})$ and source name $x \in \mathsf{name}(\boldsymbol{G})$, if $e(\varepsilon) = ‹_{\boldsymbol{G}'}(\upsilon')$ and $⤓_{\boldsymbol{G}'}(\upsilon') = l(x)$ then there exists a unique source vertex $\upsilon \in \mathsf{vert}(\boldsymbol{G})$ satisfying $\upsilon' = v(\upsilon)$, $\varepsilon = ‹_{\boldsymbol{G}}(\upsilon)$ and $⤓_{\boldsymbol{G}}(\upsilon) = x$. The creation condition means that the set of vertices $\mathsf{vert}(\boldsymbol{G})$ is the abstract limit (multi-pullback) $\mathsf{vert}(\boldsymbol{G}) = \mathit{limit}(D)$ of the W-shaped diagram $D = (e, \mathsf{edge}(\boldsymbol{G}'), ‹_{\boldsymbol{G}'}, \mathsf{vert}(\boldsymbol{G}'), ⤓_{\boldsymbol{G}'}, \mathsf{name}(\boldsymbol{G}'), l)$ of sets and functions, and hence unique up to isomorphism (bijection).

```
(13) (forall (?f (morphism ?f))
         (forall (?v2 ((sgph.obj$vertex (target ?f)) ?v2)
                  ?e1 ((sgph.obj$edge (source ?f)) ?e1)
                  ?x1 ((sgph.obj$name (source ?f)) ?x1))
             (=> (and (= ((edge ?f) ?e1) ((sgph.obj$indication (target ?f)) ?v2))
                      (= ((sgph.obj$projection (target ?f)) ?v2) ((name ?f) ?x1)))
                 (exists-unique (?v1 ((sgph.obj$vertex (source ?f)) ?v1))
                     (and (= ?v2 ((name ?f) ?v1))
                          (= ?e1 ((sgph.obj$indication (source ?f)) ?v1))
                          (= ?x1 ((sgph.obj$projection (source ?f)) ?v1)))))))
```

$$\begin{tikzcd}
\wp\mathsf{name}(\boldsymbol{G}) \arrow[d, "\wp l"', "\cong"] & \mathsf{edge}(\boldsymbol{G}) \arrow[l, "\#_{\mathit{hgph}(\boldsymbol{G})}"'] \arrow[r, "\partial_{\mathit{hgph}(\boldsymbol{G})}"] \arrow[d, "e"] & \mathsf{tuple}(\mathsf{refer}(\mathsf{hgph}(\boldsymbol{G}))) \arrow[d, "\mathsf{tuple}(\mathsf{refer}(\boldsymbol{f}))"] & \mathsf{node}(\boldsymbol{G}) \arrow[d, "n"'] & \mathsf{name}(\boldsymbol{G}) \arrow[d, "l"', "\cong"] \\
\wp\mathsf{name}(\boldsymbol{G}') & \mathsf{edge}(\boldsymbol{G}') \arrow[l, "\#_{\mathit{hgph}(\boldsymbol{G}')}"] \arrow[r, "\partial_{\mathit{hgph}(\boldsymbol{G}')}"'] & \mathsf{tuple}(\mathsf{refer}(\mathsf{hgph}(\boldsymbol{G}'))) & \mathsf{node}(\boldsymbol{G}') & \mathsf{name}(\boldsymbol{G}')
\end{tikzcd}$$

Figure 5: Hypergraph Morphism

Any spangraph morphism $\boldsymbol{f} = \langle v, e, n, l\rangle : \boldsymbol{G} \rightarrow \boldsymbol{G}'$ has an associated hypergraph morphism (Figure 5) $\mathsf{hgph}(\boldsymbol{f}) = \langle e, n, l\rangle = \langle \mathsf{edge}(\boldsymbol{f}), \mathsf{node}(\boldsymbol{f}), \mathsf{name}(\boldsymbol{f})\rangle : \mathsf{hgph}(\boldsymbol{G}) \rightarrow \mathsf{hgph}(\boldsymbol{G}')$.

```
(14) (SET.FTN$function hypergraph)
     (= (SET.FTN$source hypergraph) morphism)
     (= (SET.FTN$target hypergraph) hgph.mor$morphism)
     (= (SET.FTN$composition [hypergraph hgph.mor$source])
        (SET.FTN$composition [source sgph.obj$hypergraph]))
     (= (SET.FTN$composition [hypergraph hgph.mor$target])
        (SET.FTN$composition [target sgph.obj$hypergraph]))
```

The hypergraph morphism and spangraph morphism agree on three functions:

– $\mathsf{edge}(\mathsf{hgph}(\boldsymbol{f})) = \mathsf{edge}(\boldsymbol{f})$,

– $\mathsf{node}(\mathsf{hgph}(\boldsymbol{f})) = \mathsf{node}(\boldsymbol{f})$, and

– $\mathsf{name}(\mathsf{hgph}(\boldsymbol{f})) = \mathsf{name}(\boldsymbol{f})$.

The hypergraph morphism edge function is the spangraph morphism edge function; the hypergraph morphism node function is the spangraph morphism node function; and the hypergraph morphism name bijection is the spangraph morphism name bijection. The hypergraph morphism ignores the spangraph morphism vertex function.

```
(15) (= (SET.FTN$composition [hypergraph hgph.mor$edge]) edge)
     (= (SET.FTN$composition [hypergraph hgph.mor$node]) node)
     (= (SET.FTN$composition [hypergraph hgph.mor$name]) name)
```

The exactness of the creation condition can be made more precise and abstract (Diagram 5). The case function of the hypergraph morphism $\mathsf{case}(\mathsf{hgph}(\boldsymbol{f}))$ is "equivalent" to the vertex function of $\boldsymbol{f}$; that is, it commutes through the source/target realizations with the vertex function.

$$\begin{tikzcd}
\mathsf{vert}(\boldsymbol{G}) \arrow[r, "\cong_{\boldsymbol{G}}"] \arrow[d, "v"'] & \mathsf{case}(\mathsf{hgph}(\boldsymbol{G})) \arrow[d, "\mathsf{case}(\mathsf{hgph}(\boldsymbol{f}))"] & \boldsymbol{G} \arrow[d, "\boldsymbol{f}"'] \\
\mathsf{vert}(\boldsymbol{G}') \arrow[r, "\cong_{\boldsymbol{G}'}"'] & \mathsf{case}(\mathsf{hgph}(\boldsymbol{G}')) & \boldsymbol{G}'
\end{tikzcd}$$

Diagram 5: Abstract Creation Condition

This implies the naturality condition for realization, the unit of the categorical equivalence between the spangraph and hypergraph contexts.

```
(16) (forall (?f (morphism ?f))
         (= (set.ftn$composition
                [(sgph.obj$bijection (source ?f)) (hgph.mor$case (hypergraph ?f))])
            (set.ftn$composition
                [(vertex ?f) (sgph.obj$bijection (target ?f))])))
```

We need to define the remaining components of the hypergraph morphism of a spangraph morphism $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{G'}$:

- the *reference* semiquartet

  $*_{hgph(\boldsymbol{f})} = refer(hgph(\boldsymbol{f})) = \langle l, n \rangle = \langle name(\boldsymbol{f}), node(\boldsymbol{f}) \rangle : *_{hgph(\boldsymbol{G})} \rightarrow *_{hgph(\boldsymbol{G'})}$,

- the *signature* quartet

  $\partial_{hgph(\boldsymbol{f})} = sign(hgph(\boldsymbol{f})) = \langle e, tuple(*_{hgph(\boldsymbol{f})}) \rangle$

  $= \langle edge(\boldsymbol{f}), tuple(refer(hgph(\boldsymbol{f}))) \rangle : \partial_{hgph(\boldsymbol{G})} \rightarrow \partial_{hgph(\boldsymbol{G'})}$, and

- the *edge-arity* quartet

  $\#_{hgph(\boldsymbol{f})} = edge\text{-}arity(\boldsymbol{f}) = \langle e, \wp l \rangle = \langle edge(\boldsymbol{f}), \wp name(\boldsymbol{f}) \rangle : \#_{hgph(\boldsymbol{G})} \rightarrow \#_{hgph(\boldsymbol{G'})}$.

```
(17) (forall (?f (morphism ?f))
         (and (= (set.sqtt$source (hgph.mor$reference (hypergraph ?f)))
                 (hgph.obj$reference (sgph.obj$hypergraph (source ?f))))
              (= (set.sqtt$target (hgph.mor$reference (hypergraph ?f)))
                 (hgph.obj$reference (sgph.obj$hypergraph (target ?f))))
              (= (set.sqtt$function1 (hgph.mor$reference (hypergraph ?f))) (name ?f))
              (= (set.sqtt$function2 (hgph.mor$reference (hypergraph ?f))) (node ?f))
              (= (set.qtt$horizontal-source (hgph.mor$signature (hypergraph ?f)))
                 (hgph.obj$signature (sgph.obj$hypergraph (source ?f))))
              (= (set.qtt$horizontal-target (hgph.mor$signature (hypergraph ?f)))
                 (hgph.obj$signature (sgph.obj$hypergraph (target ?f))))
              (= (set.qtt$vertical-source (hgph.mor$signature (hypergraph ?f))) (edge ?f))
              (= (set.qtt$vertical-target (hgph.mor$signature (hypergraph ?f)))
                 (set.sqtt$tuple (hgph.mor$reference (hypergraph ?f))))
              (= (set.qtt$horizontal-source (hgph.mor$edge-arity (hypergraph ?f)))
                 (hgph.obj$edge-arity (sgph.obj$hypergraph (source ?f))))
              (= (set.qtt$horizontal-target (hgph.mor$edge-arity (hypergraph ?f)))
                 (hgph.obj$edge-arity (sgph.obj$hypergraph (target ?f))))
              (= (set.qtt$vertical-source (hgph.mor$edge-arity (hypergraph ?f))) (edge ?f))
              (= (set.qtt$vertical-target (hgph.mor$edge-arity (hypergraph ?f)))
                 (set.ftn$power (name ?f)))))
```

The reference semiquartet and the arity and signature quartets are already define by the components $name(\boldsymbol{f})$, $node(\boldsymbol{f})$, $edge(\boldsymbol{f})$, $tuple(refer(\boldsymbol{f}))$ and $\wp name(\boldsymbol{f})$. The following preservation conditions must also be satisfied (needed by the arity and signature quartets):

$\#_{hgph(\boldsymbol{G})} \cdot \wp l = e \cdot \#_{hgph(\boldsymbol{G'})}$ (arity preservation)

$\partial_{hgph(\boldsymbol{G})} \cdot tuple(refer(hgph(\boldsymbol{f}))) = e \cdot \partial_{hgph(\boldsymbol{G'})}$ (signature preservation).

Arity preservation: Arity preservation means $\#_{hgph(\boldsymbol{G})}(\varepsilon) \cong \#_{hgph(\boldsymbol{G'})}(e(\varepsilon))$ that for any edge $\varepsilon \in edge(\boldsymbol{G})$ by means of the bijection l; in words, "the arity of ε is in bijective correspondence with the arity of the edge image $e(\varepsilon)$".

Signature preservation: Signature preservation means that $tuple(*_{\boldsymbol{f}})(\partial_{hgph(\boldsymbol{G})}(\varepsilon)) = \partial_{hgph(\boldsymbol{G'})}(e(\varepsilon))$ for any edge $\varepsilon \in edge(\boldsymbol{G})$, and hence that $e(\partial_{hgph(\boldsymbol{G})}(\varepsilon)(x)) = \partial_{hgph(\boldsymbol{G'})}(e(\varepsilon))(x)$ for every $x \in arity_{hgph(\boldsymbol{G})}(\varepsilon)$; in words, "the node image of the x^{th} coordinate of the source signature $\partial_{hgph(\boldsymbol{G})}(\varepsilon)$ is equal to the x^{th} coordinate of the target signature of the edge image $e(\varepsilon)$".

```
(18) (forall (?f (morphism ?f))
         (and (= (set.ftn$composition
                     [(hgph.obj$edge-arity (sgph.obj$hypergraph (source ?f)))
                      (set.ftn$power (name ?f))])
                 (set.ftn$composition
                     [(edge ?f)
                      (hgph.obj$edge-arity (sgph.obj$hypergraph (target ?f)))]))
              (= (set.ftn$composition
                     [(hgph.obj$signature (sgph.obj$hypergraph (source ?f)))
                      (set.sqtt$tuple (hgph.mor$reference (hypergraph ?f)))])
                 (set.ftn$composition
                     [(edge ?f)
                      (hgph.obj$signature (sgph.obj$hypergraph (target ?f)))])))))
```

Any hypergraph morphism $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{G}'$ has an associated spangraph $sgph(\boldsymbol{f}) : sgph(\boldsymbol{G}) \rightarrow sgph(\boldsymbol{G}')$, which is axiomatized in the hypergraph namespace. The hypergraph associated with the spangraph $sgph(\boldsymbol{f})$ is the original hypergraph morphism:

– $hgph(sgph(\boldsymbol{f})) = \boldsymbol{f}$.

This property is one half of the categorical equivalence between spangraphs and hypergraphs.

```
(19) (= (SET.FTN$composition [hgph.mor$spangraph hypergraph])
        (SET.FTN$identity hgph.mor$morphism))
```

Two spangraph morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable spangraph morphisms $\boldsymbol{f}_1 : \boldsymbol{G} \rightarrow \boldsymbol{G}'$ and $\boldsymbol{f}_2 : \boldsymbol{G}' \rightarrow \boldsymbol{G}''$ is defined in terms of the composition of their indication, comediation and projection component quartets.

```
(20) (SET.GIM.PBK$opspan composable-opspan)
     (= (SET.GIM.PBK$class1 composable-opspan) morphism)
     (= (SET.GIM.PBK$class2 composable-opspan) morphism)
     (= (SET.GIM.PBK$opvertex composable-opspan) sgph.obj$object)
     (= (SET.GIM.PBK$first composable-opspan) target)
     (= (SET.GIM.PBK$second composable-opspan) source)

(21) (REG$edge composable)
     (= (REG$class1 composable) morphism)
     (= (REG$class2 composable) morphism)
     (= (REG$extent composable) (SET.GIM.PBK$pullback composable-opspan))

(22) (SET.FTN$function composition)
     (= (SET.FTN$source composition) (SET.GIM.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))
              (= (indication (composition [?f1 ?f2]))
                 (set.qtt$composition [(indication ?f1) (indication ?f2)]))
              (= (comediation (composition [?f1 ?f2]))
                 (set.qtt$composition [(comediation ?f1) (comediation ?f2)]))
              (= (projection (composition [?f1 ?f2]))
                 (set.hqtt$composition [(projection ?f1) (projection ?f2)]))))
```

Composition satisfies the usual *associative law*.

```
(23) (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 spangraph $\boldsymbol{G}$, there is an *identity* spangraph morphism.

```
(24) (SET.FTN$function identity)
     (= (SET.FTN$source identity) sgph.obj$object)
     (= (SET.FTN$target identity) morphism)
     (forall (?g (sgph.obj$object ?g))
         (and (= (source (identity ?g)) ?g)
              (= (target (identity ?g)) ?g)
              (= (indication (identity ?g))
                 (set.qtt$identity (sgph.obj$indication ?g)))
              (= (comediation (identity ?g))
                 (set.qtt$identity (sgph.obj$comediation ?g)))
              (= (projection (identity ?g))
                 (set.hqtt$identity (sgph.obj$projection ?g)))))
```

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

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

A spangraph morphism $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{G'}$ is an *isomorphism* when all component functions are bijections.

```
(26) (SET$class isomorphism)
     (SET$subclass isomorphism morphism)
     (forall (?f (morphism ?f))
         (<=> (isomorphism ?f)
              (and (set.ftn$bijection (vertex ?f))
                   (set.ftn$bijection (edge ?f))
                   (set.ftn$bijection (node ?f)))))
```

Spangraph Morphism Fibers

`sgph.fbr.mor`

For any set bijection $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ representing a fixed function of names, we define the fiber for the name class function, so that sgphmor($\boldsymbol{h}$) = *name*$^{-1}$($\boldsymbol{h}$) ⊆ sgphmor = *mor*(Spangraph) is the class of spangraph morphisms whose name function is $\boldsymbol{h}$. 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 spangraph morphisms is used in the name fiber namespace.

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

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

incl($\boldsymbol{h}$) : sgphmor($\boldsymbol{h}$) → sgphmor

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

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

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

src($\boldsymbol{h}$) : sgphmor($\boldsymbol{h}$) → sgph(*src*($\boldsymbol{h}$))

tgt($\boldsymbol{h}$) : sgphmor($\boldsymbol{h}$) → sgph(*tgt*($\boldsymbol{h}$)),

and indication, comediation and projection functions

indic($\boldsymbol{h}$) = *incl*($\boldsymbol{h}$) · *indic* : sgphmor($\boldsymbol{h}$) → sgphmor → qtt

comed($\boldsymbol{h}$) = *incl*($\boldsymbol{h}$) · *comed* : sgphmor($\boldsymbol{h}$) → sgphmor → qtt

proj($\boldsymbol{h}$) = *incl*($\boldsymbol{h}$) · *proj* : sgphmor($\boldsymbol{h}$) → sgphmor → bqtt.

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

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

(5) (KIF$function indication)
    (= (KIF$source indication) set.ftn$bijection)
    (= (KIF$target indication) SET.FTN$function)
    (forall (?h (set.ftn$bijection ?h))
        (and (= (SET.FTN$source (indication ?h)) (morphism ?h))
             (= (SET.FTN$target (indication ?h)) set.qtt$quartet)
             (= (SET.FTN$composition [(indication ?h) set.sqtt$source])
                (SET.FTN$composition [(source ?h) (sgph.fbr.obj$indication (set.ftn$source ?h))]))
             (= (SET.FTN$composition [(indication ?h) set.sqtt$target])
                (SET.FTN$composition [(target ?h) (sgph.fbr.obj$indication (set.ftn$target ?h))]))
             (SET.FTN$restriction (indication ?h) sgph.mor$indication)
             (= (indication ?h)
                (SET.FTN$composition [(inclusion ?h) sgph.mor$indication]))))
```

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

(7) (KIF$function projection)
    (= (KIF$source projection) set.ftn$bijection)
    (= (KIF$target projection) SET.FTN$function)
    (forall (?h (set.ftn$bijection ?h))
        (and (= (SET.FTN$source (projection ?h)) (morphism ?h))
             (= (SET.FTN$target (projection ?h)) set.hqtt$hemiquartet)
             (= (SET.FTN$composition [(projection ?h) set.hqtt$horizontal-source])
                (SET.FTN$composition [(source ?h) (sgph.fbr.obj$projection (set.ftn$source ?h))]))
             (= (SET.FTN$composition [(projection ?h) set.hqtt$horizontal-target])
                (SET.FTN$composition [(target ?h) (sgph.fbr.obj$projection (set.ftn$target ?h))])))))
             (SET.FTN$restriction (projection ?h) sgph.mor$projection)
             (= (projection ?h)
                (SET.FTN$composition [(inclusion ?h) sgph.mor$projection]))))

(8) (forall (?h (set.ftn$bijection ?h))
        (and (= (SET.FTN$composition [(indication ?h) set.qtt$vertical-source])
                (SET.FTN$composition [(comediation ?h) set.qtt$vertical-source]))
             (= (SET.FTN$composition [(comediation ?h) set.qtt$vertical-source])
                (SET.FTN$composition [(projection ?h) set.qtt$vertical-source]))))
```

We define terminology for the vertex, edge, node and name functions for hypergraph morphisms in the fiber of $\boldsymbol{h}$:

$$\mathit{vert}(\boldsymbol{h}) = \mathit{incl}(\boldsymbol{h}) \cdot \mathit{vert} : \mathrm{sgphmor}(\boldsymbol{h}) \rightarrow \mathrm{sgphmor} \rightarrow \mathrm{ftn}$$

$$\mathit{edge}(\boldsymbol{h}) = \mathit{incl}(\boldsymbol{h}) \cdot \mathit{edge} : \mathrm{sgphmor}(\boldsymbol{h}) \rightarrow \mathrm{sgphmor} \rightarrow \mathrm{ftn}$$

$$\mathit{node}(\boldsymbol{h}) = \mathit{incl}(\boldsymbol{h}) \cdot \mathit{node} : \mathrm{sgphmor}(\boldsymbol{h}) \rightarrow \mathrm{sgphmor} \rightarrow \mathrm{ftn}$$

$$\mathit{name}(\boldsymbol{h}) = \mathit{incl}(\boldsymbol{h}) \cdot \mathit{name} : \mathrm{sgphmor}(\boldsymbol{h}) \rightarrow \mathrm{sgphmor} \rightarrow \mathrm{ftn}$$

```
 (9) (KIF$function vertex)
     (= (KIF$source vertex) set.ftn$bijection)
     (= (KIF$target vertex) SET.FTN$function)
     (forall (?h (set.ftn$bijection ?h))
         (and (= (SET.FTN$source (vertex ?h)) (morphism ?h))
              (= (SET.FTN$target (vertex ?h)) set.ftn$function)
              (SET.FTN$restriction (vertex ?h) sgph.mor$vertex)
              (= (vertex ?h)
                 (SET.FTN$composition [(inclusion ?h) sgph.mor$vertex]))))

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

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

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

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

$$\mathit{hgph}(\boldsymbol{h}) : \mathrm{sgphmor}(\boldsymbol{h}) \rightarrow \mathrm{hgphmor}(\boldsymbol{h})$$

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

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

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

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

$$\begin{tikzcd}
\boldsymbol{G}_1 \arrow[r, "\boldsymbol{m}"] \arrow[d, "\blacktriangleright_{\boldsymbol{h}}(\boldsymbol{G}_1)"'] & \boldsymbol{G}_2 \arrow[d, "\blacktriangleright_{\boldsymbol{h}}(\boldsymbol{G}_2)"] \\
\boldsymbol{h}^{+1}(\boldsymbol{G}_1) \arrow[r, "\boldsymbol{h}^{+1}(\boldsymbol{m})"'] & \boldsymbol{h}^{+1}(\boldsymbol{G}_2)
\end{tikzcd}$$

Diagram 6a: Direct Image - abstract

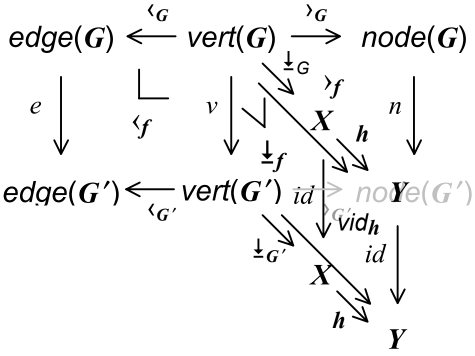


Diagram 6b: Direct Image - details

For any set bijection $\boldsymbol{h} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ representing a fixed name function, the *direct image* fiber operator (Diagram 6) $\boldsymbol{h}^{+1} : \mathrm{sgphmor}(\mathit{id}_{\boldsymbol{X}}) \rightarrow \mathrm{sgphmor}(\mathit{id}_{\boldsymbol{Y}})$ along $\boldsymbol{h}$ maps $\mathit{id}_{\boldsymbol{X}}$-spangraph morphisms to $\mathit{id}_{\boldsymbol{Y}}$-spangraph morphisms. For any spangraph morphism $\boldsymbol{m} : \boldsymbol{G}_1 \rightarrow \boldsymbol{G}_2$ with name function $id_{\boldsymbol{X}}$, the direct image $\boldsymbol{h}^{+1}(\boldsymbol{m})$ has the same edge and node function as $\boldsymbol{m}$, but has the name function $id_{\boldsymbol{Y}}$. We use the direct connection operator to define this via the abstract commuting diagram above. This is well-defined, since the direct connection is an isomorphism.

```
(15) (KIF$function direct-image)
     (= (KIF$source direct-image) set.ftn$function)
     (= (KIF$target direct-image) SET.FTN$function)
     (forall (?h (set.ftn$bijection ?h))
         (and (= (SET.FTN$source (direct-image ?h))
                 (morphism (set.ftn$identity (set.ftn$source ?h))))
              (= (SET.FTN$target (direct-image ?h))
                 (morphism (set.ftn$identity (set.ftn$target ?h))))
              (forall ?m ((morphism (set.ftn$identity (set.ftn$source ?h))) ?m))
                  (= (sgph.mor$composition
                         [((direct-connection ?h)
                              ((source (set.ftn$identity (set.ftn$source ?h))) ?m))
                          ((direct-image ?h) ?m)])
                     (sgph.mor$composition
                         [?m
                          ((direct-connection ?h)
                              ((target (set.ftn$identity (set.ftn$source ?h))) ?m))])))))
```

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

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

The direct image of an identity is the identity operator:

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

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

(17) (forall (?x (set$set ?x))
         (= (direct-image (set.ftn$identity ?x))
            (SET.FTN$identity (morphism (set.ftn$identity ?x)))))
```

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

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

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

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

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

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

$\mathit{cmpbl}([\boldsymbol{f}, \boldsymbol{g}]) \subseteq \mathsf{sgphmor}(\boldsymbol{f}) \times \mathsf{sgphmor}(\boldsymbol{g})$

$\mathit{comp}([\boldsymbol{f}, \boldsymbol{g}]) : \mathsf{plbk}([\boldsymbol{f}, \boldsymbol{g}]) = \mathsf{sgphmor}(\boldsymbol{f}) \times_{\mathsf{sgph}(\boldsymbol{Y})} \mathsf{sgphmor}(\boldsymbol{g}) \rightarrow \mathsf{sgphmor}(\boldsymbol{f} \cdot \boldsymbol{g})$.

```
(20) (KIF$function composable-opspan)
     (= (KIF$source composable-opspan) (REG$extent set.ftn$composable))
     (= (KIF$target composable-opspan) SET.GIM.PBK$opspan)
     (forall (?f (set.ftn$function ?f) ?g (set.ftn$function ?g) (set.ftn$composable ?f ?g))
         (and (= (SET.GIM.PBK$class1 (composable-opspan [?f ?g])) (morphism ?f))
              (= (SET.GIM.PBK$class2 (composable-opspan [?f ?g])) (morphism ?g))
              (= (SET.GIM.PBK$opvertex (composable-opspan [?f ?g]))
                 (sgph.fbr.obj$object (set.ftn$target ?f)))
              (= (SET.GIM.PBK$first (composable-opspan [?f ?g])) (target ?f))
              (= (SET.GIM.PBK$second (composable-opspan [?f ?g])) (source ?g))))
```

```
(21) (KIF$function composable)
     (= (KIF$source composable) (REG$extent set.ftn$composable))
     (= (KIF$target composable) REG$edge)
     (forall (?f (set.ftn$function ?f) ?g (set.ftn$function ?g) (set.ftn$composable ?f ?g))
         (and (= (REG$class1 (composable [?f ?g])) (morphism ?f))
              (= (REG$class2 (composable [?f ?g])) (morphism ?g))
              (= (REG$extent (composable [?f ?g]))
                 (SET.GIM.PBK$pullback (composable-opspan [?f ?g])))
              (REG$abridgement (composable [?f ?g]) sgph.mor$composable)))

(22) (KIF$function composition)
     (= (KIF$source composition) (REG$extent set.ftn$composable))
     (= (KIF$target composition) SET.FTN$function)
     (forall (?f (set.ftn$function ?f) ?g (set.ftn$function ?g) (set.ftn$composable ?f ?g))
         (and (= (SET.FTN$source (composition [?f ?g])) (REG$extent (composable [?f ?g])))
              (= (SET.FTN$target (composition [?f ?g]))
                 (morphism (set.ftn$composition [?f ?g])))
              (SET.FTN$restriction (composition [?f ?g]) sgph.mor$composition)))
```

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

$ident(\boldsymbol{X}) : \mathrm{sgph}(\boldsymbol{X}) \rightarrow \mathrm{sgphmor}(id_{\boldsymbol{X}})$

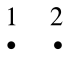
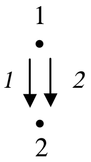
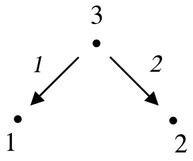
```
(23) (KIF$function identity)
     (= (KIF$source identity) set$set)
     (= (KIF$target identity) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (identity ?x)) (sgph.fbr.obj$object ?x))
              (= (SET.FTN$target (identity ?x)) (morphism (set.ftn$identity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

Colimits for Spangraphs

`sgph.col`

Colimits of spangraphs 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 Spangraph(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 Spangraph(X) and Spangraph(Y) are isomorphic, and hence have isomorphic colimits. For any set X representing a fixed set of names, the category Hypergraph(X) is categorically equivalent to the category Spangraph(X), and hence is also cocomplete.

Table 2: Shapes for Colimit Diagrams

			
empty (intitial)	*two* (binary coproduct)	*parallel pair* (coequalizer)	*span* (pushout)

We axiomatize colimits for finite diagrams *D* 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 spangraph, (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 Spangraph(X) can be composed with the vertex, edge and node component functors, and also with the indication, comediation and projection component natural transformations. These compositions result in corresponding vertex, edge and node component diagrams in Set, and indication, comediation and projection component diagram morphisms. Composition with the name functor results in the constant functor $\Delta(X) : path(G) \rightarrow \text{Set}$. 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 spangraphs are concrete, and defined in terms of colimits for their component diagrams and diagram morphisms.

$$\begin{tikzcd}
path(\mathrm{G}) \arrow[d, "D^{\#}"'] \\
\mathrm{Spangraph}(X) \arrow[d, bend right=80, "edge(X)"', ""{name=E}] \arrow[d, bend right=35, "\Delta(X)"{description}, ""{name=D}] \arrow[d, "vert(X)"{description}, ""{name=V}] \arrow[d, bend left=70, "node(X)", ""{name=N}] \arrow[from=V, to=E, Rightarrow, "‹_X"] \arrow[from=V, to=D, Rightarrow, "\underline{\downarrow}_X"'] \arrow[from=V, to=N, Rightarrow, "›_X"] \\
\mathrm{Set}
\end{tikzcd}$$

Figure 3a: Component Diagrams and Diagram Morphisms

Initial Spangraph

For any set X representing a fixed set of names, there is a special spangraph 0_X (see Figure 6, where arrows denote functions) called the *initial spangraph*, which has the required set of names X, but no indices, no (hyper) edges and no nodes. That is, the spangraph 0_X consists of:

- the empty set of *vertices* $vert(0_X) = \varnothing$;
- the empty set of *edges* $edge(0_X) = \varnothing$;
- the empty set of *nodes* $node(0_X) = \varnothing$; and
- the given set of *names* is $name(0_X) = X$.

It has the following empty component functions:

- the counique *indication* function ‹ $_{0X} = !_{\varnothing} : \varnothing \rightarrow \varnothing$;

$$\begin{tikzcd}
\varnothing \arrow[d, "!_{edge(G)}"'] & \varnothing \arrow[l, "\varnothing = !_{\varnothing}"'] \arrow[rr, "\varnothing = !_{\varnothing}"] \arrow[d, "!_{ind(G)}"'] \arrow[dr, "!_X"] & & \varnothing \arrow[d, "!_{node(G)}"] \\
& & X \arrow[dd, "id"'] & \\
edge(G) & vert(G) \arrow[l, "‹_{X,G}"] \arrow[rr, "›_{X,G}"'] \arrow[dr, "\underline{\downarrow}_{X,G}"'] & & node(G) \\
& & X &
\end{tikzcd}$$

Figure 6: Initial Type Spangraph and Counique Type Spangraph Morphism

- the counique *comediation* function $›_{\mathbf{0}X} = !_{\varnothing} : \varnothing \rightarrow \varnothing$; and
- the counique *projection* function $⭳_{\mathbf{0}X} = !_X : \varnothing \rightarrow X$.

The initial spangraph 0_X in the fibered category Spangraph(X) is the “first” spangraph in the following sense: for any spangraph G in Spangraph(X), there is a (*co*) *unique* spangraph morphism $!_{X,G} : 0_X \rightarrow G$ in Spangraph(X), (see Figure 6, where arrows denote functions). This is the unique spangraph morphism from 0_X to G with identity name function on X. The vertex, edge and node functions of this spangraph morphism are the counique functions (the empty functions) from the initial set $\varnothing$ to the respective set *vert*(G), *edge(G)* or *node(G)*:

- the counique (empty) *vertex* function $vert(!_{X,G}) = !_{ind(G)} : \varnothing \rightarrow vert(G)$;
- the counique (empty) *edge* function $edge(!_{X,G}) = !_{edge(G)} : \varnothing \rightarrow edge(G)$; and
- the counique (empty) *node* function $node(!_{X,G}) = !_{node(G)} : \varnothing \rightarrow node(G)$.

```
(1) (SET.FTN$function initial)
    (= (SET.FTN$source initial) set$set)
    (= (SET.FTN$target initial) sgph.obj$object)
    (forall (?x (set$set ?x))
        (and ((sgph.fbr.obj$object ?x) (initial ?x))
             (= (sgph.obj$vertex (initial ?x)) set.col$initial)
             (= (sgph.obj$edge (initial ?x)) set.col$initial)
             (= (sgph.obj$node (initial ?x)) set.col$initial)
             (= (sgph.obj$name (initial ?x)) ?x)
             (= (sgph.obj$indication (initial ?x)) (set.col$counique set.col$initial))
             (= (sgph.obj$comediation (initial ?x)) (set.col$counique set.col$initial))
             (= (sgph.obj$projection (initial ?x)) (set.col$counique ?x))
             ((sgph.fbr.obj$object ?x) (initial ?x))))

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

Binary Coproducts

`sgph.col.coprod2`

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

$$\begin{tikzcd}
\boldsymbol{G}_1 \arrow[r, "\iota_1"] \arrow[dr, "f_1"'] & \boldsymbol{G}_1+\boldsymbol{G}_2 \arrow[d, "{[f_1,f_2]}"'] & \boldsymbol{G}_2 \arrow[l, "\iota_2"'] \arrow[dl, "f_2"] \\
& \boldsymbol{G} &
\end{tikzcd}$$

Figure 7a: Binary Coproduct Spangraph, Injections, Cocone and Comediator – abstract version

$$\begin{tikzcd}
\mathit{edge}(\boldsymbol{G}_1)+\mathit{edge}(\boldsymbol{G}_2) & \mathit{vert}(\boldsymbol{G}_1)+\mathit{vert}(\boldsymbol{G}_2) \arrow[l, "\langle_{X,G1+G2}"'] \arrow[r, "\rangle_{X,G1+G2}"] \arrow[d, "\underline{\downarrow}_{X,G1+G2}"'] & \mathit{node}(\boldsymbol{G}_1)+\mathit{node}(\boldsymbol{G}_2) \\
& \boldsymbol{X} &
\end{tikzcd}$$

Figure 7b: Binary Coproduct Spangraph – concrete version

Let $\boldsymbol{X}$ be any set representing a fixed set of names. Let $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ be two spangraphs with common name set $\boldsymbol{X}$. The *binary coproduct (sum)* spangraph $\boldsymbol{G}_1+_{\boldsymbol{X}}\boldsymbol{G}_2$ is defined as follows (Figure 7b).

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

$\mathit{vert}_X(\boldsymbol{G}_1+_X\boldsymbol{G}_2) = \mathit{vert}_X(\boldsymbol{G}_1)+\mathit{vert}_X(\boldsymbol{G}_2)$,

$\mathit{edge}_X(\boldsymbol{G}_1+_X\boldsymbol{G}_2) = \mathit{edge}_X(\boldsymbol{G}_1)+\mathit{edge}_X(\boldsymbol{G}_2)$, and

$\mathit{node}_X(\boldsymbol{G}_1+_X\boldsymbol{G}_2) = \mathit{node}_X(\boldsymbol{G}_1)+\mathit{node}_X(\boldsymbol{G}_2)$.

The name set is is the fixed set $\boldsymbol{X}$,

$\mathit{name}_X(\boldsymbol{G}_1+_X\boldsymbol{G}_2) = \mathit{name}_X(\boldsymbol{G}_1) = \mathit{name}_X(\boldsymbol{G}_2) = \boldsymbol{X}$.

The indication and comediation functions are sums of their components

$\langle_{X,\, G1+G2} = \langle_{X,\, G1} + \langle_{X,\, G2} : \mathit{vert}_X(\boldsymbol{G}_1)+\mathit{vert}_X(\boldsymbol{G}_2) \rightarrow \mathit{edge}_X(\boldsymbol{G}_1)+\mathit{edge}_X(\boldsymbol{G}_2)$ and

$\rangle_{X,\, G1+G2} = \rangle_{X,\, G1} + \rangle_{X,\, G2} : \mathit{vert}_X(\boldsymbol{G}_1)+\mathit{vert}_X(\boldsymbol{G}_2) \rightarrow \mathit{node}_X(\boldsymbol{G}_1)+\mathit{node}_X(\boldsymbol{G}_2)$.

The projection function is the coproduct copairing of the component projection functions

$\underline{\downarrow}_{X,\, G1+G2} = [\underline{\downarrow}_{X,\, G1}, \underline{\downarrow}_{X,\, G2}] : \mathit{vert}_X(\boldsymbol{G}_1)+\mathit{vert}_X(\boldsymbol{G}_2) \rightarrow \boldsymbol{X}$.

The bijection is the composition of the bijection sum and a bijective auxiliary function

$\cong_{X,\, (G1+G2)} = (\cong_{X,G1} +_X \cong_{X,G2}) \cdot a$

$: \mathit{vert}_X(\boldsymbol{G}_1)+\mathit{vert}_X(\boldsymbol{G}_2) \rightarrow \mathit{case}_X(\mathit{hgph}_X(\boldsymbol{G}_1))+\mathit{case}_X(\mathit{hgph}_X(\boldsymbol{G}_2)) \cong \mathit{case}_X(\mathit{hgph}_X(\boldsymbol{G}_1+_X\boldsymbol{G}_2))$

is clearly a bijection.

The auxiliary function (a bijection) is

$$a = [\mathit{case}_{X}(\mathit{hgph}_{X}(\mathit{in}_{X,\,G1})), \mathit{case}_{X}(\mathit{hgph}_{X}(\mathit{in}_{X,\,G2}))]$$
$$: \mathit{case}_{X}(\mathit{hgph}_{X}(\boldsymbol{G}_1))+\mathit{case}_{X}(\mathit{hgph}_{X}(\boldsymbol{G}_2)) \cong \mathit{case}_{X}(\mathit{hgph}_{X}(\boldsymbol{G}_1+_{X}\boldsymbol{G}_2))$$

where $a((1, (\varepsilon_1, x))) = ((1, \varepsilon_1), x)$ for any edge $\varepsilon_1 \in \mathit{edge}_{X}(\boldsymbol{G}_1)$ and name $x \in \boldsymbol{X}$ where $(\varepsilon_1, x) \in \mathit{case}_{X}(\boldsymbol{G}_1)$ and $a((2, (\varepsilon_2, x))) = ((2, \varepsilon_2), x)$ for any edge $\varepsilon_2 \in \mathit{rel}_{X}(\boldsymbol{G}_2)$ and name $x \in \boldsymbol{X}$ where $(\varepsilon_2, x) \in \mathit{case}_{X}(\boldsymbol{G}_2)$.

The *sum injection* spangraph morphism $\mathit{in}_{X}(\boldsymbol{G}_1) = \langle \mathit{in}_{\mathit{vert}(\boldsymbol{G}1)}, \mathit{in}_{\mathit{edge}(\boldsymbol{G}1)}, \mathit{in}_{\mathit{node}(\boldsymbol{G}1)}, \mathit{in}_{\mathit{name}(\boldsymbol{G}1)}\rangle : \boldsymbol{G}_1 \rightarrow \boldsymbol{G}_1+_{X}\boldsymbol{G}_2$ from the first component spangraph $\boldsymbol{G}_1$ to the sum spangraph $\boldsymbol{G}_1+_{X}\boldsymbol{G}_2$ is described as follows.

- The vertex function is the sum injection
  $\mathit{vert}_{X}(\mathit{in}_{X}(\boldsymbol{G}_1)) = \mathit{in}_{\mathit{vert}(\boldsymbol{G}1)} : \mathit{vert}_{X}(\boldsymbol{G}_1) \rightarrow \mathit{vert}_{X}(\boldsymbol{G}_1+_{X}\boldsymbol{G}_2) = \mathit{vert}_{X}(\boldsymbol{G}_1)+\mathit{vert}_{X}(\boldsymbol{G}_2)$;
- the edge function is the sum injection
  $\mathit{edge}_{X}(\mathit{in}_{X}(\boldsymbol{G}_1)) = \mathit{in}_{\mathit{edge}(\boldsymbol{G}1)} : \mathit{edge}_{X}(\boldsymbol{G}_1) \rightarrow \mathit{edge}_{X}(\boldsymbol{G}_1+_{X}\boldsymbol{G}_2) = \mathit{edge}_{X}(\boldsymbol{G}_1)+\mathit{edge}_{X}(\boldsymbol{G}_2)$;
- the node function is the sum injection
  $\mathit{node}(\mathit{in}_{X}(\boldsymbol{G}_1)) = \mathit{in}_{\mathit{node}(\boldsymbol{G}1)} : \mathit{node}_{X}(\boldsymbol{G}_1) \rightarrow \mathit{node}_{X}(\boldsymbol{G}_1+_{X}\boldsymbol{G}_2) = \mathit{node}_{X}(\boldsymbol{G}_1)+\mathit{node}_{X}(\boldsymbol{G}_2)$; and
- the name function is the identity
  $\mathit{name}(\mathit{in}_{X}(\boldsymbol{G}_1)) = \mathit{id}_{X} : \boldsymbol{X} \rightarrow \mathit{name}_{X}(\boldsymbol{G}_1+_{X}\boldsymbol{G}_2) = \boldsymbol{X}$.

The sum $\boldsymbol{G}_1+_{X}\boldsymbol{G}_2$ is a coproduct in the category $\mathsf{Spangraph}(\boldsymbol{X})$. In summary, all sums exist in any fiber $\mathit{name}^{-1}(\boldsymbol{X}) = \mathsf{Spangraph}(\boldsymbol{X})$ of the name functor $\mathit{name} : \mathsf{Spangraph} \rightarrow \mathsf{Set}$. The spangraph functor

$$\mathit{sgph}(\boldsymbol{X}) : \mathsf{Hypergraph}(\boldsymbol{X}) \rightarrow \mathsf{Spangraph}(\boldsymbol{X})$$

and the hypergraph functor

$$\mathit{hgph}(\boldsymbol{X}) : \mathsf{Spangraph}(\boldsymbol{X}) \rightarrow \mathsf{Hypergraph}(\boldsymbol{X})$$

preserve sums.

A *binary coproduct (sum)* is a finite colimit in the category $\mathsf{Spangraph}(\boldsymbol{X})$ for a diagram of shape *two* = • •. Such a diagram (of spangraphs and spangraph morphisms) is called a *pair* of spangraphs. For any set $\boldsymbol{X}$ representing a fixed set of names, a *pair* of spangraphs consists of spangraphs *spangraph1* and *spangraph2*. We use either the generic term '`diagram`' or the specific term '`pair`' to denote the pair class. Pairs are determined by their two component spangraphs.

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

(2) (KIF$function spangraph1)
    (= (KIF$source spangraph1) set$set)
    (= (KIF$target spangraph1) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (spangraph1 ?x)) (diagram ?x))
             (= (SET.FTN$target (spangraph1 ?x)) (sgph.fbr.obj$object ?x))))

(3) (KIF$function spangraph2)
    (= (KIF$source spangraph2) set$set)
    (= (KIF$target spangraph2) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (spangraph2 ?x)) (diagram ?x))
             (= (SET.FTN$target (spangraph2 ?x)) (sgph.fbr.obj$object ?x))))

(4) (forall (?x (set$set ?x)
             ?p1 ((diagram ?x) ?p1) ?p2 ((diagram ?x) ?p2))
        (=> (and (= ((spangraph1 ?x) ?p1) ((spangraph1 ?x) ?p2))
                 (= ((spangraph2 ?x) ?p1) ((spangraph2 ?x) ?p2)))
            (= ?p1 ?p2)))
```

For any set $\boldsymbol{X}$ representing a fixed set of names, there is a *vertex pair* or *vertex diagram* function, which maps a pair of spangraphs to the underlying pair of vertex sets. Similarly, there is an *edge pair* or *edge diagram* function, which maps a pair of spangraphs to the underlying pair of edge sets; and there is a *node pair* or *node diagram* function, which maps a pair of spangraphs to the underlying pair of node sets.

```
(5) (KIF$function vertex-diagram)
    (= (KIF$source vertex-diagram) set$set)
    (= (KIF$target vertex-diagram) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (vertex-diagram ?x)) (diagram ?x))
             (= (SET.FTN$target (vertex-diagram ?x)) set.col.coprd2$diagram)
             (= (SET.FTN$composition [(vertex-diagram ?x) set.col.coprd2$set1])
                (SET.FTN$composition [(spangraph1 ?x) (sgph.fbr.obj$vertex ?x)]))
             (= (SET.FTN$composition [(vertex-diagram ?x) set.col.coprd2$set2])
                (SET.FTN$composition [(spangraph2 ?x) (sgph.fbr.obj$vertex ?x)]))))

(6) (KIF$function edge-diagram)
    (= (KIF$source edge-diagram) set$set)
    (= (KIF$target edge-diagram) SET.FTN$function)
    (forall (?x (set$set ?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 [(spangraph1 ?x) (sgph.fbr.obj$edge ?x)]))
             (= (SET.FTN$composition [(edge-diagram ?x) set.col.coprd2$set2])
                (SET.FTN$composition [(spangraph2 ?x) (sgph.fbr.obj$edge ?x)]))))

(7) (KIF$function node-diagram)
    (= (KIF$source node-diagram) set$set)
    (= (KIF$target node-diagram) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (node-diagram ?x)) (diagram ?x))
             (= (SET.FTN$target (node-diagram ?x)) set.col.coprd2$diagram)
             (= (SET.FTN$composition [(node-diagram ?x) set.col.coprd2$set1])
                (SET.FTN$composition [(spangraph1 ?x) (sgph.fbr.obj$node ?x)]))
             (= (SET.FTN$composition [(node-diagram ?x) set.col.coprd2$set2])
                (SET.FTN$composition [(spangraph2 ?x) (sgph.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 spangraph morphisms) consists of a pair of spangraph morphisms called *opfirst* and *opsecond*. These are required to have a common target spangraph called the *opvertex* of the cocone. Each binary coproduct cocone is under a pair of spangraphs.

```
 (8) (KIF$function cocone)
     (= (KIF$source cocone) set$set)
     (= (KIF$target cocone) SET$class)

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

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

(11) (KIF$function opfirst)
     (= (KIF$source opfirst) set$set)
     (= (KIF$target opfirst) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (opfirst ?x)) (cocone ?x))
              (= (SET.FTN$target (opfirst ?x))
                 (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (SET.FTN$composition [(cocone-diagram ?x) (spangraph1 ?x)]))
              (= (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (opvertex ?x))))

(12) (KIF$function opsecond)
```

```
     (= (KIF$source opsecond) set$set)
     (= (KIF$target opsecond) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (opsecond ?x)) (cocone ?x))
              (= (SET.FTN$target (opsecond ?x))
                 (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (SET.FTN$composition [(cocone-diagram ?x) (spangraph2 ?x)]))
              (= (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (opvertex ?x))))
```

For any set ***X*** representing a fixed set of names, there is a *vertex cocone* function, which maps a binary coproduct cocone of spangraphs and spangraph morphisms to the underlying binary coproduct cocone of vertex sets and vertex functions. There are also *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.

```
(13) (KIF$function vertex-cocone)
     (= (KIF$source vertex-cocone) set$set)
     (= (KIF$target vertex-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (vertex-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (vertex-cocone ?x)) set.col.coprd2$cocone)
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coprd2$cocone-diagram])
                 (SET.FTN$composition [(cocone-diagram ?x) (vertex-diagram ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coprd2$opvertex])
                 (SET.FTN$composition [(opvertex ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coprd2$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coprd2$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))))

(14) (KIF$function edge-cocone)
     (= (KIF$source edge-cocone) set$set)
     (= (KIF$target edge-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (edge-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (edge-cocone ?x)) set.col.coprd2$cocone)
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.coprd2$cocone-diagram])
                 (SET.FTN$composition [(cocone-diagram ?x) (edge-diagram ?x)]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.coprd2$opvertex])
                 (SET.FTN$composition [(opvertex ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.coprd2$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.coprd2$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))))

(15) (KIF$function node-cocone)
     (= (KIF$source node-cocone) set$set)
     (= (KIF$target node-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (node-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (node-cocone ?x)) set.col.coprd2$cocone)
              (= (SET.FTN$composition [(node-cocone ?x) set.col.coprd2$cocone-diagram])
                 (SET.FTN$composition [(cocone-diagram ?x) (node-diagram ?x)]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.coprd2$opvertex])
                 (SET.FTN$composition [(opvertex ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.coprd2$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.coprd2$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))))
```

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

```
(16) (KIF$function colimiting-cocone)
     (= (KIF$source colimiting-cocone) set$set)
     (= (KIF$target colimiting-cocone) SET.FTN$function)
     (forall (?x (set$set ?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)))))

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

(18) (KIF$function injection1)
     (= (KIF$source injection1) set$set)
     (= (KIF$target injection1) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (injection1 ?x)) (diagram ?x))
              (= (SET.FTN$target (injection1 ?x))
                 (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (spangraph1 ?x))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (colimit ?x))
              (= (injection1 ?x)
                 (SET.FTN$composition [(colimiting-cocone ?x) (opfirst ?x)]))))

(19) (KIF$function injection2)
     (= (KIF$source injection2) set$set)
     (= (KIF$target injection2) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (injection2 ?x)) (diagram ?x))
              (= (SET.FTN$target (injection2 ?x))
                 (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (spangraph2 ?x))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$target (set.ftn$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 three axioms are the necessary conditions that the vertex, edge and node functors preserve concrete colimits. These explicitly ensure that a binary coproduct of spangraphs is specific – that its vertex, edge and node sets are exactly the disjoint unions of the corresponding sets of the pair of spangraphs. In addition, these explicitly ensure that the two coproduct injection spangraph morphisms are specific – that their vertex, edge and node functions are exactly the coproduct injections of the vertex, edge and node set pairs of the pair of spangraphs.

```
(20) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(colimiting-cocone ?x) (vertex-cocone ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coprod2$colimiting-cone]))
              (= (SET.FTN$composition [(colimit ?x) (sgph.fbr.obj$vertex ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coprod2$colimit]))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coprod2$injection1]))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coprod2$projection2]))))

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

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

For any set $\boldsymbol{X}$ representing a fixed set of names and for any binary coproduct cocone, there is a *comediator* spangraph morphism $[\boldsymbol{f}_1, \boldsymbol{f}_2] : \boldsymbol{G}_1+\boldsymbol{G}_2 \rightarrow \boldsymbol{G}$ (see Figure 7a, where arrows denote spangraph morphisms) from the binary coproduct of the underlying diagram (pair of spangraphs) to the opvertex of the cocone. This is the unique spangraph morphism, which commutes with the opfirst $\boldsymbol{f}_1$ and the opsecond $\boldsymbol{f}_2$. We define this by using the comediators of the underlying vertex, edge and node cocones. Existence and uniqueness represents the universality of the binary coproduct operator.

```
(23) (KIF$function comediator)
     (= (KIF$source comediator) set$set)
     (= (KIF$target comediator) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (comediator ?x)) (cocone ?x))
              (= (SET.FTN$target (comediator ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(comediator ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (SET.FTN$composition [(cocone-diagram ?x) (colimit ?x)]))
              (= (SET.FTN$composition [(comediator ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (opvertex ?x))
              (= (SET.FTN$composition [(comediator ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-cocone ?x) set.col.coprd2$comediator]))
              (= (SET.FTN$composition [(comediator ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))])
                 (SET.FTN$composition [(edge-cocone ?x) set.col.coprd2$comediator]))
              (= (SET.FTN$composition [(comediator ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))])
                 (SET.FTN$composition [(node-cocone ?x) set.col.coprd2$comediator]))))
```

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

```
(24) (forall (?x (set$set ?x)
              ?s ((cocone ?x) ?s))
         (= ((comediator ?x) ?s)
            (the (?f ((sgph.fbr.mor$morphism (set.ftn$identity ?x)) ?f))
                (and (= ((sgph.fbr.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                            [((injection1 ?x) ((cocone-diagram ?x) ?s)) ?f])
                        ((opfirst ?x) ?s))
                     (= ((sgph.fbr.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                            [((injection2 ?x) ((cocone-diagram ?x) ?s)) ?f])
                        ((opsecond ?x) ?s))))))
```

Endorelations and Quotients

sgph.col.endo

$$\begin{tikzcd}
\mathit{edge}(\boldsymbol{G}) \arrow[dd, "\boldsymbol{edge}(f)"'] \arrow[dr, phantom, "\langle_f"] & \mathit{vert}(\boldsymbol{G}) \arrow[l, "\langle_{\boldsymbol{G}}"'] \arrow[rr, "\rangle_{\boldsymbol{G}}"] \arrow[dr, "\underline{\downarrow}_{G}"'] \arrow[dd, "\boldsymbol{vert}(f)"'] \arrow[drr, phantom, "\rangle_f"] \arrow[ddr, phantom, "\underline{\downarrow}_f"] & & \mathit{node}(\boldsymbol{G}) \arrow[dd, "\boldsymbol{node}(f)"] \\
& & \mathit{name}(\boldsymbol{G}) \arrow[dd, "l"', "\cong"] & \\
\mathit{edge}(\boldsymbol{G'}) & \mathit{vert}(\boldsymbol{G'}) \arrow[l, "\langle_{\boldsymbol{G'}}"] \arrow[rr, gray, "\rangle_{\boldsymbol{G'}}"'] \arrow[dr, "\underline{\downarrow}_{\boldsymbol{G'}}"'] & & \mathit{node}(\boldsymbol{G'}) \\
& & \mathit{name}(\boldsymbol{G'}) &
\end{tikzcd}$$

Figure 4: Spangraph Morphism

Endorelations are definable on fibers of the category Spangraph. Let $\boldsymbol{X}$ be any set representing a fixed set of names. A spangraph *endorelation* is a quadruple $\boldsymbol{J} = \langle \mathit{sgph}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}) \rangle$ consisting of an underlying spangraph $\mathit{sgph}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{G}$ with name set $\boldsymbol{X}$, a binary endorelation $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{V} \subseteq \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G}) \times \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})$ on the vertices of $\boldsymbol{G}$, a binary endorelation $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{E} \subseteq \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G}) \times \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})$ on the edges of $\boldsymbol{G}$, and a binary endorelation $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{N} \subseteq \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G}) \times \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G})$ on the nodes of $\boldsymbol{G}$, that satisfy the following constraints. Let $\equiv_{\boldsymbol{V}}$, $\equiv_{\boldsymbol{E}}$ and $\equiv_{\boldsymbol{N}}$ denote the equivalence relations generated by the $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J})$, $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J})$ and $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{J})$ relations, respectively.

– **[vertex constraints]** for any pair of vertices $\upsilon_1, \upsilon_2 \in \mathit{vert}(\boldsymbol{G})$,
  if $(\upsilon_1, \upsilon_2) \in \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J})$, then
  * $\langle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_1) \equiv_E \langle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_2)$,
  * $\rangle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_1) \equiv_N \rangle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_2)$ and
  * $\underline{\downarrow}_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_1) = \underline{\downarrow}_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_2)$;

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

– [**equivalence-identity constraints**] for any pair of vertices $\upsilon_1, \upsilon_2 \in \mathit{vert}(\boldsymbol{G})$,
  if $\langle_{\boldsymbol{G}}(\upsilon_1) \equiv_R \langle_{\boldsymbol{G}}(\upsilon_2)$ and $\underline{\downarrow}_{\boldsymbol{G}}(\upsilon_1) = \underline{\downarrow}_{\boldsymbol{G}}(\upsilon_2)$ then
  * $\upsilon_1 \equiv_V \upsilon_2$.

The equivalence-identity constraint implies that the "bijection" map is a well-define bijection. The equivalence-identity constraint is equivalent to the following alternate equivalence-identity condition.

– [**alternate equivalence-identity condition**] for any pair of edges $\varepsilon_1, \varepsilon_2 \in \mathit{edge}(\boldsymbol{G})$,
  if $\varepsilon_1 \equiv_R \varepsilon_2$, $\upsilon_1 \cong (\varepsilon_1, x)$ and $\upsilon_2 \cong (\varepsilon_2, x)$ then
  * $\upsilon_1 \equiv_V \upsilon_2$.

These following two alternate arity/signature conditions are equivalent and we identify them. The vertex constraint and alternate arity/signature conditions 1&2 imply the arity/signature and equivalence-identity constraints. The arity/signature and equivalence-identity constraints imply the alternate arity/signature-1&2 conditions.

– **[alternate arity/signature condition 1]** for any vertex $\upsilon_1 \in \mathit{vert}(\boldsymbol{G})$ and any edge $\varepsilon_2 \in \mathit{edge}(\boldsymbol{G})$,
  if $\langle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_1) \equiv_E \varepsilon_2$ then there is another vertex $\upsilon_2 \in \mathit{vert}(\boldsymbol{G})$ such that
  * $\upsilon_1 \equiv_V \upsilon_2$ and
  * $\langle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon_2) = \varepsilon_2$.

– **[alternate arity/signature condition 2]** for any pair of edges $\varepsilon_1, \varepsilon_2 \in \mathit{edge}(\boldsymbol{G})$,
  if $\varepsilon_1 \equiv_R \varepsilon_2$ and $\upsilon_1 \cong (\varepsilon_1, x)$ then there is another vertex $\upsilon_2 \in \mathit{vert}(\boldsymbol{G})$ such that
  * $\upsilon_1 \equiv_V \upsilon_2$ and
  * $\upsilon_2 \cong (\varepsilon_2, x)$.

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

```
(1) (KIF$function endorelation)
    (= (KIF$source endorelation) set$set)
    (= (KIF$target endorelation) SET$class)

(2) (KIF$function spangraph)
    (KIF$function base)
    (= base spangraph)
    (= (KIF$source spangraph) set$set)
    (= (KIF$target spangraph) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (spangraph ?x)) (endorelation ?x))
             (= (SET.FTN$target (spangraph ?x)) (sgph.fbr.obj$object ?x))))

(3) (KIF$function vertex)
    (= (KIF$source vertex) set$set)
    (= (KIF$target vertex) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (vertex ?x)) (endorelation ?x))
             (= (SET.FTN$target (vertex ?x)) rel.endo$endorelation)
             (= (SET.FTN$composition [(vertex ?x) rel.endo$set])
                (SET.FTN$composition [(spangraph ?x) (sgph.fbr.obj$vertex ?x)]))))

(4) (KIF$function edge)
    (= (KIF$source edge) set$set)
    (= (KIF$target edge) SET.FTN$function)
    (forall (?x (set$set ?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 [(spangraph ?x) (sgph.fbr.obj$edge ?x)]))))

(5) (KIF$function node)
    (= (KIF$source node) set$set)
    (= (KIF$target node) SET.FTN$function)
    (forall (?x (set$set ?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 [(spangraph ?x) (sgph.fbr.obj$node ?x)]))))

(6) (forall (?x (set$set ?x)
           ?j ((endorelation ?x) ?j))
        (forall (?v1 (((sgph.fbr.obj$vertex ?x) ((spangraph ?x) ?j)) ?v1)
                 ?v2 (((sgph.fbr.obj$vertex ?x) ((spangraph ?x) ?j)) ?v2))
            (=> (((vertex ?x) ?j) ?i1 ?i2)
                (and ((rel.endo$equivalence-closure ((edge ?x) ?j))
                         (((sgph.fbr.obj$indication ?x) ((spangraph ?x) ?j)) ?v1)
                         (((sgph.fbr.obj$indication ?x) ((spangraph ?x) ?j)) ?v2))
                     ((rel.endo$equivalence-closure ((node ?x) ?j))
                         (((sgph.fbr.obj$comediation ?x) ((spangraph ?x) ?j)) ?v1)
                         (((sgph.fbr.obj$comediation ?x) ((spangraph ?x) ?j)) ?v2))
                     (=  (((sgph.fbr.obj$projection ?x) ((spangraph ?x) ?j)) ?v1)
                         (((sgph.fbr.obj$projection ?x) ((spangraph ?x) ?j)) ?v2))))))

(7) (forall (?x (set$set ?x)
           ?j ((endorelation ?x) ?j))
        (forall (?v1 (((sgph.fbr.obj$vertex ?x) ((spangraph ?x) ?j)) ?v1)
                 ?e2 (((sgph.fbr.obj$edge ?x) ((spangraph ?x) ?j)) ?e2))
            (=> ((rel.endo$equivalence-closure ((edge ?x) ?j))
                    (((sgph.fbr.obj$indication ?x) ((spangraph ?x) ?j)) ?v1)
                    ?e2)
                (exists (?v2 (((sgph.fbr.obj$vertex ?x) ((spangraph ?x) ?j)) ?v2))
                    (and ((rel.endo$equivalence-closure ((vertex ?x) ?j)) ?v1 ?v2)
                         (= (((sgph.fbr.obj$indication ?x) ((spangraph ?x) ?j)) ?v2) ?e2))))))
```

By the equivalence-identity condition for $\boldsymbol{G}$, any vertex $\upsilon \in \mathsf{vert}_{\boldsymbol{X}}(\boldsymbol{G})$ is in bijective correspondence with a case pair $(\varepsilon, x) \in \mathsf{case}_{\boldsymbol{X}}(\boldsymbol{G})$, where $(‹_{\boldsymbol{G}}, \underline{\downarrow}_{\boldsymbol{G}})(\upsilon) = (‹_{\boldsymbol{G}}(\upsilon), \underline{\downarrow}_{\boldsymbol{G}}(\upsilon)) = (\varepsilon, x)$; that is $\upsilon \cong (\varepsilon, x)$.

Let $\boldsymbol{X}$ be any set representing a fixed set of names. The class $\mathsf{endo}(\boldsymbol{X})$ of endorelations at $\boldsymbol{X}$ is ordered. For any two endorelations $\boldsymbol{J}_1, \boldsymbol{J}_2 \in \mathsf{endo}(\boldsymbol{X})$, $\boldsymbol{J}_1$ is a subrelation of $\boldsymbol{J}_2$, $\boldsymbol{J}_1 \leq \boldsymbol{J}_2$, when

- $\mathsf{sgph}_{\boldsymbol{X}}(\boldsymbol{J}_1) = \mathsf{sgph}_{\boldsymbol{X}}(\boldsymbol{J}_2)$;

- $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}_1) \subseteq \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}_2)$;
- $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}_1) \subseteq \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}_2)$; and
- $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}_1) \subseteq \mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}_2)$.

```
(8) (KIF$function subrelation)
     (= (KIF$source subrelation) set$set)
     (= (KIF$target subrelation) ORD$partial-order)
     (forall (?x (set$set ?x))
         (and (= (ORD$class (subrelation ?x)) (endorelation ?x))
              (forall (?j1 ((endorelation ?x) ?j1)
                       ?j2 ((endorelation ?x) ?j2))
                  (<=> ((subrelation ?x) ?j1 ?j2)
                       (and (= ((spangraph ?x) ?j1) ((hypergraph ?x) ?j2))
                            (ord$suborder ((vertex ?x) ?j1) ((vertex ?x) ?j2))
                            (ord$suborder ((edge ?x) ?j1) ((edge ?x) ?j2))
                            (ord$suborder ((node ?x) ?j1) ((node ?x) ?j2)))))))
```

$$\begin{tikzcd}
\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G}) \arrow[d, "{\mathit{edge}(\tau_{X,J}) = \mathit{canon}(\boldsymbol{E})}"'] & \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G}) \arrow[l, "{\langle_{X,G}}"'] \arrow[rr, "{\rangle_{X,G}}"] \arrow[dr, "{\underline{\downarrow}_{X,G}}"] \arrow[dd, "{\mathit{vert}(\tau_{X,J}) = \mathit{canon}(\boldsymbol{V})}"'] & & \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G}) \arrow[dd, "{\mathit{node}(\tau_{X,J}) = \mathit{canon}(\boldsymbol{N})}"] \\
 & & \boldsymbol{X} \arrow[dd, "id"] & \\
\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{E}} & \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{V}} \arrow[l, "{\langle_{X,G/J}}"] \arrow[rr, "{\rangle_{X,G/J}}"'] \arrow[dr, "{\underline{\downarrow}_{X,G/J}}"'] & & \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{N}} \\
 & & \boldsymbol{X} &
\end{tikzcd}$$

Figure 8: Type Spangraph Quotient and Canonical Morphism – details

$$\begin{tikzcd}
\boldsymbol{G} \arrow[r, "\tau_{X,J}"] \arrow[dr, "\boldsymbol{f}"'] & \boldsymbol{G}/\boldsymbol{J} \arrow[d, "\tilde{\boldsymbol{f}}^{X}"] \\
 & \boldsymbol{K}
\end{tikzcd}$$

Diagram 7: Canonical Morphism and Universality of the Quotient – abstract

Often, the endorelations $\boldsymbol{V}$, $\boldsymbol{E}$ and $\boldsymbol{N}$ are equivalence relations on the vertices, 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 spangraph morphisms (coequalizer diagrams) do not have component equivalence relations. For any edge $\varepsilon \in \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})$, write $[\varepsilon]_{\boldsymbol{E}}$ for the $\boldsymbol{E}$-equivalence class of ε. Same for vertices and nodes. Given any fixed set of names $\boldsymbol{X}$, a simple inductive proof shows that the quadruple $\hat{\boldsymbol{J}} = \langle \boldsymbol{G}, \equiv_{\boldsymbol{V}}, \equiv_{\boldsymbol{E}}, \equiv_{\boldsymbol{R}} \rangle$ is also an endorelation.

Quotient

Given any fixed set of names $\boldsymbol{X}$, the *quotient* $\boldsymbol{G}/\boldsymbol{J}$ of an endorelation $\boldsymbol{J} = \langle \boldsymbol{G}, \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{N} \rangle = \langle \mathit{sgph}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}) \rangle$ on a name set $\boldsymbol{X}$ (see Figure 8, where arrows denote set functions; or Diagram 7, where arrows denote spangraph morphisms) is the spangraph defined as follows:

- The set of *vertices* is $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G}/\boldsymbol{J}) = \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{V}}$.
- The set of *edges* is $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G}/\boldsymbol{J}) = \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{E}}$.
- The set of *nodes* is $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{G}/\boldsymbol{J}) = \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{N}}$.
- The set of *names* is $\mathit{name}_{\boldsymbol{X}}(\boldsymbol{G}/\boldsymbol{J}) = \mathit{name}_{\boldsymbol{X}}(\boldsymbol{G}) = \boldsymbol{X}$.

The following function definitions are well defined by the endorelation constraints.

- The *indication* function

  $\langle_{\boldsymbol{X},\boldsymbol{G}/\boldsymbol{J}} : \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{V}} \rightarrow \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{E}}$

  is defined pointwise as follows: $\langle_{\boldsymbol{X},\boldsymbol{G}/\boldsymbol{J}}([\upsilon]_{\boldsymbol{V}}) = [\langle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon)]_{\boldsymbol{E}}$ for all vertices $\upsilon \in \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})$.
- The *comediation* function

  $\rangle_{\boldsymbol{X},\boldsymbol{G}/\boldsymbol{J}} : \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{V}} \rightarrow \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{N}}$

  is defined pointwise as follows: $\rangle_{\boldsymbol{X},\boldsymbol{G}/\boldsymbol{J}}([\upsilon]_{\boldsymbol{V}}) = [\rangle_{\boldsymbol{X},\boldsymbol{G}}(\upsilon)]_{\boldsymbol{N}}$ for all vertices $\upsilon \in \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})$.
- The *projection* function

  $\underline{\downarrow}_{\boldsymbol{X},\boldsymbol{G}/\boldsymbol{J}} : \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})/\equiv_{\boldsymbol{V}} \rightarrow \mathit{var}_{\boldsymbol{X}}(\boldsymbol{G}) = \boldsymbol{X}$

is defined pointwise as follows: $⤓_{\mathbf{X},\mathbf{G}/\mathbf{J}}([\upsilon]_{V}) = ⤓_{\mathbf{X},\mathbf{G}}(\upsilon)$ for all vertices $\upsilon \in \mathit{vert}_{\mathbf{X}}(\mathbf{G})$.

- The *bijection*

  $\cong_{\mathbf{G}} : \mathit{vert}_{\mathbf{X}}(\mathbf{G})/\equiv_{V} \rightarrow \mathit{case}_{\mathbf{X}}(\mathit{hgph}_{\mathbf{X}}(\mathbf{G}/\mathbf{J}))$

  is defined pointwise as follows: $\cong_{\mathbf{G}}([\upsilon]_{V}) = ([\varepsilon]_{\mathbf{E}}, x)$, where $\upsilon \cong (\varepsilon, x)$.

Canon

There is a *canon*ical quotient spangraph morphism

$\boldsymbol{\tau}_{X,J} = \langle \mathit{vert}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}), \mathit{edge}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}), \mathit{node}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}), \mathit{id}(\mathbf{X})\rangle : \mathbf{G} \rightarrow \mathbf{G}/\mathbf{J}$

whose vertex function is the canonical surjection

$\mathit{vert}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}) = \mathit{canon}(\mathbf{V}) = [\text{-}]_{V} : \mathit{vert}_{\mathbf{X}}(\mathbf{G}) \rightarrow \mathit{vert}_{\mathbf{X}}(\mathbf{G})/\equiv_{\mathbf{V}}$,

whose edge function is the canonical surjection

$\mathit{edge}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}) = \mathit{canon}(\mathbf{E}) = [\text{-}]_{\mathbf{E}} : \mathit{edge}_{\mathbf{X}}(\mathbf{G}) \rightarrow \mathit{edge}_{\mathbf{X}}(\mathbf{G})/\equiv_{\mathbf{E}}$, and

whose node function is the canonical surjection

$\mathit{node}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J}) = \mathit{canon}(\mathbf{N}) = [\text{-}]_{N} : \mathit{node}_{\mathbf{X}}(\mathbf{G}) \rightarrow \mathit{node}_{\mathbf{X}}(\mathbf{G})/\equiv_{\mathbf{N}}$.

The defining properties of the components of this spangraph morphism are trivial, by the definition of the quotient spangraph.

We show that the canonical quotient spangraph morphism $\boldsymbol{\tau}_{X,J} : \mathbf{G} \rightarrow \mathbf{G}/\mathbf{J}$ satisfies the *creation condition*: for each target vertex ($\mathbf{V}$ equivalence class) $\upsilon' \in \mathit{vert}(\mathbf{G}/\mathbf{J})$, source edge $\varepsilon \in \mathit{edge}(\mathbf{G})$ and name $x \in \mathbf{X}$, if $[\varepsilon]_{\mathbf{E}} = \mathit{edge}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J})(\varepsilon) = ‹_{\mathbf{X},\mathbf{G}/\mathbf{J}}(\upsilon')$ and $⤓_{\mathbf{X},\mathbf{G}/\mathbf{J}}(\upsilon') = x$ then there exists a unique source vertex $\underline{\upsilon} \in \mathit{vert}(\mathbf{G})$ satisfying $\upsilon' = \mathit{vert}_{\mathbf{X}}(\boldsymbol{\tau}_{X,J})(\underline{\upsilon}) = [\underline{\upsilon}]_{V}$, $\varepsilon = ‹_{\mathbf{G}}(\underline{\upsilon})$ and $⤓_{\mathbf{G}}(\underline{\upsilon}) = x$. We express this using equivalence: for each vertex $\upsilon \in \mathit{vert}(\mathbf{G})$ and any edge $\varepsilon \in \mathit{edge}(\mathbf{G})$ and name $x \in \mathbf{X}$, if $‹_{\mathbf{X},\mathbf{G}}(\upsilon) \equiv_{E} \varepsilon$ and $⤓_{\mathbf{X},\mathbf{G}}(\upsilon) = x$ then there exists a unique source vertex $\underline{\upsilon} \in \mathit{vert}(\mathbf{G})$ satisfying $\underline{\upsilon} \equiv_{V} \upsilon$, $\varepsilon = ‹_{\mathbf{G}}(\underline{\upsilon})$ and $⤓_{\mathbf{G}}(\underline{\upsilon}) = x$. But this follows using the arity/signature condition 1 and the vertex conditions.

```
(9) (KIF$function quotient)
    (= (KIF$source quotient) set$set)
    (= (KIF$target quotient) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (quotient ?x)) (endorelation ?x))
             (= (SET.FTN$target (quotient ?x)) (sgph.fbr.obj$object ?x))
             (forall (?j ((endorelation ?x) ?j))
                 (and (= ((sgph.fbr.obj$vertex ?x) ((quotient ?x) ?j))
                         (rel.endo$quotient (rel.endo$equivalence-closure ((vertex ?x) ?j))))
                      (= ((sgph.fbr.obj$edge ?x) ((quotient ?x) ?j))
                         (rel.endo$quotient (rel.endo$equivalence-closure ((edge ?x) ?j))))
                      (= ((sgph.fbr.obj$node ?x) ((quotient ?x) ?j))
                         (rel.endo$quotient (rel.endo$equivalence-closure ((node ?x) ?j))))
                      (= (set.ftn$composition
                             [(rel.endo$canon ((vertex ?x) ?j))
                              ((sgph.fbr.obj$indication ?x) ((quotient ?x) ?j))])
                         (set.ftn$composition
                             [((sgph.fbr.obj$indication ?x) ((spangraph ?x) ?j))
                              (rel.endo$canon ((edge ?x) ?j))]))
                      (= (set.ftn$composition
                             [(rel.endo$canon ((vertex ?x) ?j))
                              ((sgph.fbr.obj$comediation ?x) ((quotient ?x) ?j))])
                         (set.ftn$composition
                             [((sgph.fbr.obj$comediation ?x) ((spangraph ?x) ?j))
                              (rel.endo$canon ((node ?x) ?j))]))
                      (= (set.ftn$composition
                             [(rel.endo$canon ((vertex ?x) ?j))
                              ((sgph.fbr.obj$projection ?x) ((quotient ?x) ?j))])
                         ((sgph.fbr.obj$projection ?x) ((spangraph ?x) ?j)))))))

(10) (KIF$function canon)
     (= (KIF$source canon) set$set)
     (= (KIF$target canon) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (canon ?x)) (endorelation ?x))
              (= (SET.FTN$target (canon ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (forall (?j ((endorelation ?x) ?j))
                  (and (= (SET.FTN$composition
                              [(canon ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                          (spangraph ?x))
                       (= (SET.FTN$composition
                              [(canon ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                          (quotient ?x))
                       (= (SET.FTN$composition
                              [(canon ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                          (SET.FTN$composition [(vertex ?x) rel.endo$canon]))
                       (= (SET.FTN$composition
                              [(canon ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))])
                          (SET.FTN$composition [(edge ?x) rel.endo$canon]))
                       (= (SET.FTN$composition
                              [(canon ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))])
                          (SET.FTN$composition [(node ?x) rel.endo$canon]))))))
```

Any spangraph morphism $\boldsymbol{f} = \langle v, e, n, id\rangle = \langle \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{node}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{id}_{\boldsymbol{X}}\rangle : \boldsymbol{G} \rightarrow \boldsymbol{G}'$, whose name function is the identity on $\boldsymbol{X}$, defines a spangraph endorelation $\mathit{ker}(\boldsymbol{f}) = \boldsymbol{J}_{\boldsymbol{f}} = \langle \boldsymbol{G}, \boldsymbol{V}_{\boldsymbol{f}}, \boldsymbol{E}_{\boldsymbol{f}}, \boldsymbol{N}_{\boldsymbol{f}}\rangle$ over $\boldsymbol{X}$ called the *kernel* of $\boldsymbol{f}$, where the base spangraph is the source spangraph, the vertex endorelation is the kernel of the vertex function, the edge endorelation is the kernel of the edge function, and the node endorelation is the kernel of the node function:

- $\mathit{hgph}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{G}$;
- $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{V}_{\boldsymbol{f}} = \{(\upsilon_1, \upsilon_2) \mid \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f})(\upsilon_1) = \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f})(\upsilon_2)\}$;
- $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{E}_{\boldsymbol{f}} = \{(\varepsilon_1, \varepsilon_2) \mid \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_1) = \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_2)\}$; and
- $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{J}) = \boldsymbol{N}_{\boldsymbol{f}} = \{(\varepsilon_1, \varepsilon_2) \mid \mathit{node}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_1) = \mathit{node}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_2)\}$.

The vertex, arity/signature and equivalence-identity constraints can (and should) all be checked. These are implied by the spangraph morphism preservation and creation conditions.

$$\begin{tikzcd}
\boldsymbol{G} \arrow[r, "\tau_J"] \arrow[d, "\varepsilon_f"'] \arrow[dr, "\boldsymbol{f}"] & \boldsymbol{G}/\boldsymbol{J} \arrow[d, "\tilde{\boldsymbol{f}}^X"] \arrow[dl, "\boldsymbol{f}_J"'] \\
\boldsymbol{G}/\boldsymbol{J}_f \arrow[r, "\mu_f"'] & \boldsymbol{G}'
\end{tikzcd}$$

Diagram 8: Factorization

The notion of a kernel gives a canonical approach (Diagram 8) for an epi-mono factorization system for spangraph morphisms. By definition, any spangraph morphism respects its kernel. Let ε_f denote the canonical morphism of the kernel of $\boldsymbol{f}$, and let μ_f denote the unique mediating morphism such that $\varepsilon_f \circ \mu_f = \boldsymbol{f}$. This is an "image factorization" of $\boldsymbol{f}$. The kernel of any spangraph morphism $\boldsymbol{f}$ is the "largest" spangraph endorelation that $\boldsymbol{f}$ respects. If $\boldsymbol{f}$ respects a spangraph endorelation $\boldsymbol{J}$, then there is a unique spangraph morphism $\boldsymbol{f}_{\boldsymbol{J}} : \boldsymbol{G}/\boldsymbol{J} \rightarrow \boldsymbol{G}/\boldsymbol{J}_{\boldsymbol{f}}$ such that $\boldsymbol{\tau}_{X,J} \circ \boldsymbol{f}_{\boldsymbol{J}} = \varepsilon_f$ and $\boldsymbol{f}_{\boldsymbol{J}} \circ \mu_f = \tilde{\boldsymbol{f}}^X$.

```
(11) (KIF$function kernel)
     (= (KIF$source kernel) set$set)
     (= (KIF$target kernel) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (kernel ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$target (kernel ?x)) (endorelation ?x))
              (forall (?f ((sgph.fbr.mor$morphism (set.ftn$identity ?x)) ?f))
                  (and (= ((spangraph ?x) ((kernel ?x) ?f))
                          ((sgph.fbr.mor$source (set.ftn$identity ?x)) ?f))
                       (= ((vertex ?x) ((kernel ?x) ?f))
                          (set.ftn$kernel ((sgph.fbr.mor$vertex (set.ftn$identity ?x)) ?f))
                       (= ((edge ?x) ((kernel ?x) ?f))
                          (set.ftn$kernel ((sgph.fbr.mor$edge (set.ftn$identity ?x)) ?f))
                       (= ((node ?x) ((kernel ?x) ?f))
                          (set.ftn$kernel ((sgph.fbr.mor$node (set.ftn$identity ?x)) ?f))))
```

Let $\boldsymbol{X}$ be any fixed set of names, and let $\boldsymbol{J} = \langle \boldsymbol{G}, \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{N}\rangle = \langle \mathit{sgph}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J}), \mathit{node}_{\boldsymbol{X}}(\boldsymbol{J})\rangle$ be a spangraph endorelation on $\boldsymbol{X}$. A spangraph morphism $\boldsymbol{f} = \langle \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{node}_{\boldsymbol{X}}(\boldsymbol{f}), \mathit{id}_{\boldsymbol{X}}\rangle : \boldsymbol{G} \rightarrow \boldsymbol{K}$ in the fibered category $\mathsf{Spangraph}(\boldsymbol{X})$ *respects* $\boldsymbol{J}$ when $\boldsymbol{J} \leq \mathit{ker}(\boldsymbol{f})$, $\boldsymbol{J}$ is a subrelation of the kernel of $\boldsymbol{f}$. In more detail,

- if $(\upsilon_0, \upsilon_1) \in \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{J})$ then $\mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f})(\upsilon_0) = \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{f})(\upsilon_1)$, for any two vertices $\upsilon_0, \upsilon_1 \in \mathit{vert}_{\boldsymbol{X}}(\boldsymbol{G})$;
- if $(\varepsilon_0, \varepsilon_1) \in \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{J})$ then $\mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_0) = \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{f})(\varepsilon_1)$, for any two edges $\varepsilon_0, \varepsilon_1 \in \mathit{edge}_{\boldsymbol{X}}(\boldsymbol{G})$; and
- if $(\alpha_0, \alpha_1) \in \mathit{node}_{\boldsymbol{X}}(\boldsymbol{J})$ then $\mathit{node}_{\boldsymbol{X}}(\boldsymbol{f})(\alpha_0) = \mathit{node}_{\boldsymbol{X}}(\boldsymbol{f})(\alpha_1)$, for any two nodes $\alpha_0, \alpha_1 \in \mathit{node}_{\boldsymbol{X}}(\boldsymbol{G})$.

```
(12) (KIF$function matches-opspan)
```

```
     (= (KIF$source matches-opspan) set$set)
     (= (KIF$target matches-opspan) SET.LIM.PBK$opspan)
     (forall (?x (set$set ?x))
         (and (= (SET.LIM.PBK$class1 (matches-opspan ?x))
                 (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.LIM.PBK$class2 (matches-opspan ?x)) (endorelation ?x))
              (= (SET.LIM.PBK$opvertex (matches-opspan ?x)) (sgph.fbr.obj$object ?x))
              (= (SET.LIM.PBK$opfirst (matches-opspan ?x))
                 (sgph.fbr.mor$source (set.ftn$identity ?x)))
              (= (SET.LIM.PBK$opsecond (matches-opspan ?x)) (spangraph ?x))))

(13) (KIF$function matches)
     (= (KIF$source matches) set$set)
     (= (KIF$target matches) REL$relation)
     (forall (?x (set$set ?x))
         (and (= (REL$class1 (matches ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (REL$class2 (matches ?x)) (endorelation ?x))
              (= (REL$extent (matches ?x)) (SET.LIM.PBK$pullback (matches-opspan ?x)))))

(14) (KIF$function respects)
     (= (KIF$source respects) set$set)
     (= (KIF$target respects) REL$relation)
     (forall (?x (set$set ?x))
         (and (= (REL$class1 (respects ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (REL$class2 (respects ?x)) (endorelation ?x))
              (REL$subrelation (respects ?x) (matches ?x))
              (forall (?f ((sgph.fbr.mor$morphism ?x) ?f)
                       ?j ((endorelation ?x) ?j) ((matches ?x) ?f ?j))
                  (<=> ((respects ?x) ?f ?j)
                       ((subrelation ?x) ?j ((kernel ?x) ?f))))))
```

Proposition. *For every endorelation* $\boldsymbol{J}$ *on name set* $\boldsymbol{X}$ *and every* $\boldsymbol{X}$*-fibered spangraph morphism* $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{K}$ *that respects* $\boldsymbol{J}$, *there is a unique comediating* $\boldsymbol{X}$*-fibered spangraph morphism* $\tilde{\boldsymbol{f}}^{X} : \boldsymbol{G}/\boldsymbol{J} \rightarrow \boldsymbol{K}$ *such that* $\boldsymbol{\tau}_{X,J} \circ \tilde{\boldsymbol{f}}^{X} = \boldsymbol{f}$.

If $\boldsymbol{f} : \boldsymbol{G} \rightarrow \boldsymbol{K}$ respects $\boldsymbol{J}$, then it factors uniquely through the quotient: there is a unique spangraph morphism $\tilde{\boldsymbol{f}}^{X} : \boldsymbol{G}/\boldsymbol{J} \rightarrow \boldsymbol{K}$ in the fibered category Spangraph($\boldsymbol{X}$) such that $\boldsymbol{\tau}_{X,J} \circ \tilde{\boldsymbol{f}}^{X} = \boldsymbol{f}$. This means that:

$[]_{V} \cdot \mathit{vert}_{X}(\tilde{\boldsymbol{f}}^{X}) = \mathit{vert}_{X}(\boldsymbol{f})$, or $\mathit{vert}_{X}(\tilde{\boldsymbol{f}}^{X})([\upsilon]_{V}) = \mathit{vert}_{X}(\boldsymbol{f})(\upsilon)$ for all vertices $\upsilon \in \mathit{vert}_{X}(\boldsymbol{G})$;

$[]_{\boldsymbol{E}} \cdot \mathit{edge}_{X}(\tilde{\boldsymbol{f}}^{X}) = \mathit{edge}_{X}(\boldsymbol{f})$, or $\mathit{edge}_{X}(\tilde{\boldsymbol{f}}^{X})([\varepsilon]_{\boldsymbol{E}}) = \mathit{edge}_{X}(\boldsymbol{f})(\varepsilon)$ for all edges $\varepsilon \in \mathit{edge}_{X}(\boldsymbol{G})$; and

$[]_{\boldsymbol{R}} \cdot \mathit{node}_{X}(\tilde{\boldsymbol{f}}^{X}) = \mathit{node}_{X}(\boldsymbol{f})$, or $\mathit{node}_{X}(\tilde{\boldsymbol{f}}^{X})([\alpha]_{\boldsymbol{E}}) = \mathit{node}_{X}(\boldsymbol{f})(\alpha)$ for all nodes $\alpha \in \mathit{node}_{X}(\boldsymbol{G}_{X})$.

The respectful constraints on $\boldsymbol{f}$ imply that the components of $\tilde{\boldsymbol{f}}^{X}$ are a well defined.

We next show that $\tilde{\boldsymbol{f}}^{X}$ satisfies the creation condition. Let $\upsilon' \in \mathit{vert}(\boldsymbol{K})$ be a target vertex, let $[\varepsilon]_{\boldsymbol{E}} \in \mathit{edge}(\boldsymbol{G}/\boldsymbol{E})$ for some edge $\varepsilon \in \mathit{edge}(\boldsymbol{G})$ and let $x \in \boldsymbol{X}$ be a name, where $\mathit{edge}_{X}(\tilde{\boldsymbol{f}}^{X})([\varepsilon]_{\boldsymbol{E}}) = \mathit{edge}_{X}(\boldsymbol{f})(\varepsilon) = \langle_{\boldsymbol{K}}(\upsilon')$ and $\pm_{\boldsymbol{K}}(\upsilon') = x$. Since, $\boldsymbol{f}$ satisfies the creation condition, there exists a unique source vertex $\upsilon \in \mathit{vert}(\boldsymbol{G})$ satisfying $\upsilon' = \mathit{vert}_{X}(\boldsymbol{f})(\upsilon)$, $\varepsilon = \langle_{\boldsymbol{G}}(\upsilon)$ and $\pm_{\boldsymbol{G}}(\upsilon) = x$. Then, $[\upsilon]_{V} \in \mathit{vert}_{X}(\boldsymbol{G}/\boldsymbol{E})$ satisfies $\upsilon' = \mathit{vert}_{X}(\tilde{\boldsymbol{f}}^{X})([\upsilon]_{V}) = \mathit{vert}_{X}(\boldsymbol{f})(\upsilon)$, $[\varepsilon]_{\boldsymbol{E}} = \langle_{X,G/J}([\upsilon]_{V})$ and $\pm_{X,G/J}([\upsilon]_{V}) = x$. Is it unique? If not, there is a vertex $\underline{\upsilon} \in \mathit{vert}_{X}(\boldsymbol{G})$ satisfying, $\upsilon' = \mathit{vert}_{X}(\boldsymbol{f})(\underline{\upsilon})$, $\varepsilon \equiv_{\boldsymbol{E}} \langle_{X,G}(\underline{\upsilon})$ and $\pm_{X,G}(\underline{\upsilon}) = x$, but not $\upsilon \equiv_{V} \underline{\upsilon}$. By the arity/signature constraint 1, there is an there is another vertex $\upsilon_2 \in \mathit{vert}(\boldsymbol{G})$ such that $\underline{\upsilon} \equiv_{V} \upsilon_2$ and $\langle_{X,G}(\upsilon_2) = \varepsilon$. Since $\boldsymbol{f}$ respects $\boldsymbol{J}$, we have $\upsilon' = \mathit{vert}_{X}(\boldsymbol{f})(\underline{\upsilon}) = \mathit{vert}_{X}(\boldsymbol{f})(\upsilon_2)$, $\varepsilon = \langle_{\boldsymbol{G}}(\upsilon_2)$ and $\pm_{\boldsymbol{G}}(\upsilon_2) = x$. But this violates the uniqueness of $\upsilon \in \mathit{vert}(\boldsymbol{G})$, since $\underline{\upsilon} \equiv_{V} \upsilon_2$ implies $\upsilon_2 \neq \upsilon$.

The canonical quotient spangraph morphism $\boldsymbol{\tau}_{X,J} : \boldsymbol{G} \rightarrow \boldsymbol{G}/\boldsymbol{J}$ respects $\boldsymbol{J}$, and its unique morphism is the identity. Based on this proposition, a definite description is used to define a *comediator* function (Diagram 7) that maps a pair ($\boldsymbol{f}$, $\boldsymbol{J}$) consisting of an endorelation and a respectful spangraph morphism to their comediator $\tilde{\boldsymbol{f}}^{X}$.

```
(15) (KIF$function comediator)
     (= (KIF$source comediator) set$set)
     (= (KIF$target comediator) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (comediator ?x)) (REL$extent (respects ?x)))
              (= (SET.FTN$target (comediator ?x)) (sgph.fbr.mor$morphism ?x))
              (forall (?f ((sgph.fbr.mor$morphism (set.ftn$identity ?x)) ?f)
                       ?j ((endorelation ?x) ?j)
                       ((respects ?x) ?f ?j))
                  (= ((comediator ?x) [?f ?j])
                     (the (?ft ((sgph.fbr.mor$morphism (set.ftn$identity ?x)) ?ft))
                          (and (= ((sgph.fbr.mor$source (set.ftn$identity ?x)) ?ft)
                                  ((quotient ?x) ?j))
                               (= ((sgph.fbr.mor$target (set.ftn$identity ?x)) ?ft)
                                  ((sgph.fbr.mor$target (set.ftn$identity ?x)) ?f))
                               (= ((sgph.fbr.mor$composition
                                      [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                                          [((canon ?x) ?j) ?ft])
                                   ?f)))))))
```

Coequalizers

`sgph.col.coeq`

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

Let $\boldsymbol{X}$ be any fixed set of names, a *parallel pair* $\boldsymbol{f} = \langle \boldsymbol{f}_1, \boldsymbol{f}_2 \rangle : \boldsymbol{O} \rightarrow \boldsymbol{D}$ of spangraph morphisms is a pair of spangraph morphisms sharing a common origin spangraph $\boldsymbol{O}$ and a common destination spangraph $\boldsymbol{D}$. See Diagram 10, where arrows denote spangraph morphisms.

$$\begin{tikzcd}
\boldsymbol{O} \arrow[r, shift left, "\boldsymbol{f}_1"] \arrow[r, shift right, "\boldsymbol{f}_2"'] & \boldsymbol{D} \arrow[r, "\tau_{X,f}"] \arrow[dr, "\boldsymbol{g}"'] & \boldsymbol{T}/\boldsymbol{J}_{X,f} \arrow[d, "\hat{\boldsymbol{g}}"] \\
& & \boldsymbol{G}
\end{tikzcd}$$

Diagram 10: Coequalizer of a Parallel Pair

Let $\boldsymbol{X}$ be any fixed set of names, a *parallel pair* $\boldsymbol{f} = \langle \boldsymbol{f}_1, \boldsymbol{f}_2 \rangle : \boldsymbol{O} \rightarrow \boldsymbol{D}$ of spangraph morphisms is the appropriate base diagram for a coequalizer. Each parallel pair consists of a pair of spangraph morphisms called *morphism1* and *morphism2* with name function id_X that share the same *origin* and *destination* spangraphs. 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 spangraph morphisms.

```
(1) (KIF$function diagram)
    (KIF$function parallel-pair)
    (= parallel-pair diagram)
    (= (KIF$source diagram) set$set)
    (= (KIF$target diagram) SET$class)

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

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

(4) (KIF$function morphism1)
    (= (KIF$source morphism1) set$set)
    (= (KIF$target morphism1) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (morphism1 ?x)) (diagram ?x))
             (= (SET.FTN$target (morphism1 ?x))
                (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
             (= (SET.FTN$composition
                    [(morphism1 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                (origin ?x))
             (= (SET.FTN$composition
                    [(morphism1 ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                (destination ?x))))

(5) (KIF$function morphism2)
    (= (KIF$source morphism2) set$set)
    (= (KIF$target morphism2) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (morphism2 ?x)) (diagram ?x))
             (= (SET.FTN$target (morphism2 ?x))
                (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
             (= (SET.FTN$composition
                    [(morphism2 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                (origin ?x))
             (= (SET.FTN$composition
                    [(morphism2 ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                (destination ?x))))
```

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

A set coequalizer diagram, a parallel pair of set functions, consists of two sets called origin and destination and two functions whose common source is the origin set and whose common target is the destination set. A morphism of parallel pairs of set functions from a source diagram to a target diagram consists of two functions called origin and destination, where the source of the origin is the origin of the source and the source of the destination is the destination of the source. These functions must commute with the component functions of source and target diagrams. These are both axiomatized in the set coequalizer namespace of the IFF Lower Core Ontology.

$$\begin{tikzcd}
\boldsymbol{O} \arrow[r, shift left, "\boldsymbol{f}_1"] \arrow[r, shift right, "\boldsymbol{f}_2"'] & \boldsymbol{D}
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
 & & \mathit{vert}(\boldsymbol{O}) \arrow[r, shift left, "\mathit{vert}(\boldsymbol{f}_1)"] \arrow[r, shift right, "\mathit{vert}(\boldsymbol{f}_2)"'] \arrow[dll, "\langle_S"'] \arrow[drr, "\rangle_S"] \arrow[dd, "\underline{\downarrow}_S"] & \mathit{vert}(\boldsymbol{D}) \arrow[dl, "\langle_T"'] \arrow[drr, "\rangle_T"] \arrow[dd, "\underline{\downarrow}_T"'] & & \\
\mathit{edge}(\boldsymbol{O}) \arrow[r, shift left, "\mathit{edge}(\boldsymbol{f}_1)"] \arrow[r, shift right, "\mathit{edge}(\boldsymbol{f}_2)"'] & \mathit{edge}(\boldsymbol{D}) & & & \mathit{node}(\boldsymbol{O}) \arrow[r, shift left, "\mathit{node}(\boldsymbol{f}_1)"] \arrow[r, shift right, "\mathit{node}(\boldsymbol{f}_2)"'] & \mathit{node}(\boldsymbol{D}) \\
 & & \boldsymbol{X} \arrow[r, shift left, "\mathit{id}(\boldsymbol{X})"] \arrow[r, shift right, "\mathit{id}(\boldsymbol{X})"'] & \boldsymbol{X} & &
\end{tikzcd}$$

Figure 10: Component Diagrams and Diagram Morphisms – details

There are *vertex*, *edge* and *node* diagram functions, which map parallel pairs of spangraph morphisms to the underlying vertex, 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 vertex, edge and node set functors. In addition, there are *indication*, *comediation* and *projection* diagram morphism functions, which map parallel pairs of spangraph morphisms to the underlying indication, comediation and projection morphism of parallel pairs, respectively. The source of indication is the vertex diagram and the target of indication is the edge diagram. The source of comediation is the vertex diagram and the target of comediation is the node diagram. The source of projection is the vertex 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 component part of the compositions of the diagram with the indication, comediation and projection set natural transformations.

$$\begin{tikzcd}
 & \mathit{path}(\boldsymbol{PP}) \arrow[d, "D^{\#}"'] \\
 & \mathrm{Spangraph}(\boldsymbol{X}) \arrow[dl, bend right=50, "\mathit{edge}(\boldsymbol{X})"'] \arrow[dl, "\mathit{name}(\boldsymbol{X})"] \arrow[d, "\mathit{vert}(\boldsymbol{X})"] \arrow[d, bend left=80, "\mathit{node}(\boldsymbol{X})"] \arrow[d, phantom, shift right=8, "\overset{\langle_X}{\Leftarrow}\;\underline{\downarrow}_X\Downarrow"] \arrow[d, phantom, shift left=8, "\overset{\rangle_X}{\Rightarrow}"] \\
\mathrm{Set}^{\cong} \arrow[r, phantom, "\subseteq"] & \mathrm{Set}
\end{tikzcd}$$

Figure 3a: Component Diagrams and Diagram Morphisms

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

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

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

(??) (KIF$function indication)
     (= (KIF$source indication) set$set)
     (= (KIF$target indication) SET.FTN$function)
     (forall (?x (set$set ?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$source])
                 (vertex-diagram ?x))
              (= (SET.FTN$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) (sgph.fbr.obj$indication ?x)]))
              (= (SET.FTN$composition [(indication ?x) set.col.coeq$destination])
                 (SET.FTN$composition [(destination ?x) (sgph.fbr.obj$indication ?x)]))))

(??) (KIF$function comediation)
     (= (KIF$source comediation) set$set)
     (= (KIF$target comediation) SET.FTN$function)
     (forall (?x (set$set ?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$source])
                 (vertex-diagram ?x))
              (= (SET.FTN$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) (sgph.fbr.obj$comediation ?x)]))
              (= (SET.FTN$composition [(comediation ?x) set.col.coeq$destination])
                 (SET.FTN$composition [(destination ?x) (sgph.fbr.obj$comediation ?x)]))))

(??) (KIF$function projection)
     (= (KIF$source projection) set$set)
     (= (KIF$target projection) SET.FTN$function)
     (forall (?x (set$set ?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$source])
                 (vertex-diagram ?x))
              (= (SET.FTN$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) (sgph.fbr.obj$projection ?x)]))
              (= (SET.FTN$composition [(projection ?x) set.col.coeq$destination])
                 (SET.FTN$composition [(destination ?x) (sgph.fbr.obj$projection ?x)]))))
```

$$\begin{tikzcd}
\boldsymbol{O} \arrow[r, shift left, "\boldsymbol{f}_1"] \arrow[r, shift right, "\boldsymbol{f}_2"'] & \boldsymbol{D}
\end{tikzcd}$$

Figure 11a: Parallel Pair – abstract

$$\begin{tikzcd}
edge(\boldsymbol{O}) \arrow[d, shift right, "e_1"'] \arrow[d, shift left, "e_2"] & vert(\boldsymbol{O}) \arrow[l, "‹_{\boldsymbol{O}}"'] \arrow[rr, "›_{\boldsymbol{O}}"] \arrow[dr, "\underline{\downarrow}_{O}"] \arrow[dd, shift right, "v_1"'] \arrow[dd, shift left, "v_2"] & & node(\boldsymbol{O}) \arrow[dd, shift right, "n_1"'] \arrow[dd, shift left, "n_2"] \\
& & \boldsymbol{X} \arrow[dd, "\cong"] & \\
edge(\boldsymbol{D}) & vert(\boldsymbol{D}) \arrow[l, "‹_{\boldsymbol{D}}"] \arrow[rr, "id"] \arrow[dr, "\underline{\downarrow}_{\boldsymbol{D}}"'] & & node(\boldsymbol{D}) \\
& & \boldsymbol{X} &
\end{tikzcd}$$

Figure 11b: Parallel Pair – details

Let $\boldsymbol{X}$ be any fixed set of names, and let $\boldsymbol{f} = \langle \boldsymbol{f}_1, \boldsymbol{f}_2 \rangle) : \boldsymbol{O} \rightarrow \boldsymbol{D}$ (Figure 11) be a coequalizer diagram or parallel pair of spangraph morphisms. The information in $\boldsymbol{f}$ is equivalently expressed as an *endorelation* $\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}} = \langle \boldsymbol{D}, \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{N} \rangle = \langle \boldsymbol{D}, vert_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}), edge_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}), node_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}) \rangle$ based at the destination spangraph of the parallel pair. The vertex endorelation is the coequalizer endorelation of the vertex diagram

$vert_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}) = \{(vert(\boldsymbol{f}_1)(\upsilon), vert(\boldsymbol{f}_2)(\upsilon)) \mid \upsilon \in vert(\boldsymbol{O})\}$,

the edge endorelation is the coequalizer endorelation of the edge diagram

$edge_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}) = \{(edge(\boldsymbol{f}_1)(\varepsilon), edge(\boldsymbol{f}_2)(\varepsilon)) \mid \varepsilon \in edge(\boldsymbol{O})\}$, and

the node endorelation is the coequalizer endorelation of the node diagram

$node_{\boldsymbol{X}}(\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}) = \{(node(\boldsymbol{f}_1)(\nu), node(\boldsymbol{f}_2)(\nu)) \mid \nu \in node(\boldsymbol{O})\}$.

The commuting diagrams for the indication, comediation and projection quartets verify the vertex constraints for the endorelation $\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}$: $‹_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_1)(\upsilon)) \equiv_E ‹_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_2)(\upsilon))$, $›_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_1)(\upsilon)) \equiv_N ›_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_2)(\upsilon))$ and $\underline{\downarrow}_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_1)(\upsilon)) = \underline{\downarrow}_{\boldsymbol{X},\boldsymbol{G}}(vert(\boldsymbol{f}_2)(\upsilon))$.

The commuting diagrams for the edge-arity and signature quartets of the hypergraph morphisms of $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ verify the arity/signature constraints for the endorelation $\boldsymbol{J}_{\boldsymbol{X},\boldsymbol{f}}$: $arity_{\boldsymbol{X}}(\boldsymbol{G})(edge(\boldsymbol{f}_1)(\varepsilon)) = arity_{\boldsymbol{X}}(\boldsymbol{G})(edge(\boldsymbol{f}_2)(\varepsilon))$ and $sign_{\boldsymbol{X}}(\boldsymbol{G})(edge(\boldsymbol{f}_1)(\varepsilon)) \equiv_\partial sign_{\boldsymbol{X}}(\boldsymbol{G})(edge(\boldsymbol{f}_2)(\varepsilon))$.

The bijection between vertices and cases of the hypergraph of O and D verifies the equivalence-identity constraints: for any pair of vertices $\upsilon_1, \upsilon_2 \in vert(\boldsymbol{D})$, if $‹_{\boldsymbol{G}}(\upsilon_1) = edge(\boldsymbol{f}_1)(\varepsilon)$, $‹_{\boldsymbol{G}}(\upsilon_2) = edge(\boldsymbol{f}_2)(\varepsilon)$ and $\underline{\downarrow}_{\boldsymbol{G}}(\upsilon_1) = \underline{\downarrow}_{\boldsymbol{G}}(\upsilon_2) = x$ then $\upsilon_1 = vert(\boldsymbol{f}_1)(\upsilon)$ and $\upsilon_2 = vert(\boldsymbol{f}_2)(\upsilon)$ for $\upsilon \cong (\varepsilon, x)$.

```
(10) (KIF$function endorelation)
     (= (KIF$source endorelation) set$set)
     (= (KIF$target endorelation) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (endorelation ?x)) (diagram ?x))
              (= (SET.FTN$target (endorelation ?x)) (sgph.col.endo$endorelation ?x))
              (= (SET.FTN$composition [(endorelation ?x) (sgph.col.endo$spangraph ?x)])
                 (destination ?x))
              (= (SET.FTN$composition [(endorelation ?x) (sgph.col.endo$vertex ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coeq$endorelation]))
              (= (SET.FTN$composition [(endorelation ?x) (sgph.col.endo$edge ?x)])
                 (SET.FTN$composition [(edge-diagram ?x) set.col.coeq$endorelation]))
              (= (SET.FTN$composition [(endorelation ?x) (sgph.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 11, where arrows denote spangraph morphisms) is situated under a coequalizer *diagram* or parallel pair of spangraph morphisms. Each coequalizer cocone has an *opvertex* spangraph $\boldsymbol{C}$, and a spangraph *morphism* $\boldsymbol{f} : \boldsymbol{A} \rightarrow \boldsymbol{C}$, whose source spangraph is the target spangraph of the spangraph morphisms in the underlying cocone diagram (parallel-pair) and whose target spangraph is the opvertex. Since Figure 11 is a commutative diagram, the composite spangraph morphism is not needed.

$$\begin{tikzcd}
\boldsymbol{B} \arrow[r, shift left, "\boldsymbol{f}_1"] \arrow[r, shift right, "\boldsymbol{f}_2"'] \arrow[dr, "\boldsymbol{f}_1 \cdot \boldsymbol{f} = \boldsymbol{f}_2 \cdot \boldsymbol{f}"'] & \boldsymbol{A} \arrow[d, "\boldsymbol{f}"] \\
& \boldsymbol{C}
\end{tikzcd}$$

Figure 11: Coequalizer Cocone

```
(11) (KIF$function cocone)
     (= (KIF$source cocone) set$set)
     (= (KIF$target cocone) SET$class)

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

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

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

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

The *vertex* coequalizer cocone function maps a coequalizer cocone of spangraphs and spangraph morphisms to the underlying coequalizer cocone of vertex sets and vertex set functions. The *edge* coequalizer cocone function maps a coequalizer cocone of spangraphs and spangraph morphisms to the underlying coequalizer cocone of edge sets and edge set functions. The *node* coequalizer cocone function maps a coequalizer cocone of spangraphs and spangraph morphisms to the underlying coequalizer cocone of node sets and node set functions.

```
(16) (KIF$function vertex-cocone)
     (= (KIF$source vertex-cocone) set$set)
     (= (KIF$target vertex-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (vertex-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (vertex-cocone ?x)) set.col.coeq$cocone)
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coeq$cocone-diagram])
                 (SET.FTN$composition [(cocone-diagram ?x) (vertex-diagram ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coeq$opvertex])
                 (SET.FTN$composition [(opvertex ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.coeq$function])
                 (SET.FTN$composition
                     [(morphism ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))))
```

```
(17) (KIF$function edge-cocone)
     (= (KIF$source edge-cocone) set$set)
     (= (KIF$target edge-cocone) SET.FTN$function)
     (forall (?x (set$set ?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$composition [(opvertex ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.coeq$function])
                 (SET.FTN$composition
                     [(morphism ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))))

(18) (KIF$function node-cocone)
     (= (KIF$source node-cocone) set$set)
     (= (KIF$target node-cocone) SET.FTN$function)
     (forall (?x (set$set ?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$composition [(opvertex ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.coeq$function])
                 (SET.FTN$composition
                     [(morphism ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))))
```

It is important to observe that the spangraph 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.

```
(19) (forall (?x (set$set ?x)
              ?s ((cocone ?x) ?s))
         ((sgph.col.endo$respects ?x)
             ((morphism ?x) ?s)
             (endorelation ((cocone-diagram ?x) ?s))))
```

There is a class function '`colimiting-cone`' that maps a parallel pair of spangraph morphisms to its coequalizer (colimiting coequalizer cocone) (see Figure 34, where arrows denote spangraph morphisms). The totality of this function, along with the universality of the comediator spangraph morphism, implies that a coequalizer exists for any parallel pair of spangraph morphisms. The opvertex of the colimiting coequalizer cocone is a specific *coequalizer* spangraph. It comes equipped with a *canon* spangraph 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.

$$\begin{tikzcd}
\boldsymbol{B} \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] \arrow[dr] & \boldsymbol{A} \arrow[d, "\boldsymbol{\gamma}"] \\
 & \bar{\boldsymbol{A}}
\end{tikzcd}$$

Figure 34: Colimiting Cocone

```
(20) (KIF$function colimiting-cocone)
     (= (KIF$source colimiting-cocone) set$set)
     (= (KIF$target colimiting-cocone) SET.FTN$function)
     (forall (?x (set$set ?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)))))

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

(22) (KIF$function canon)
```

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

(23) (forall (?x (set$set ?x))
         (and (= (colimit ?x)
                 (SET.FTN$composition [(endorelation ?x) (sgph.col.endo$quotient ?x)]))
              (= (canon ?x)
                 (SET.FTN$composition [(endorelation ?x) (sgph.col.endo$canon ?x)]))))
```

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

```
(24) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(colimiting-cocone ?x) (vertex-cocone x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coeq$colimiting-cocone]))
              (= (SET.FTN$composition [(colimit ?x) (sgph.fbr.obj$vertex ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coeq$colimit]))
              (= (SET.FTN$composition [(canon ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coeq$canon]))))

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

(26) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(colimiting-cocone ?x) (node-cocone x)])
                 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$colimiting-cocone]))
              (= (SET.FTN$composition [(colimit ?x) (sgph.fbr.obj$node ?x)])
                 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$colimit]))
              (= (SET.FTN$composition [(canon ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))])
                 (SET.FTN$composition [(node-diagram ?x) set.col.coeq$canon]))))
```

The *comediator* spangraph morphism, from the coequalizer of a parallel pair of spangraph morphisms to the opvertex of a cocone under the parallel pair (see Figure 35, where arrows denote spangraph morphisms), is the unique spangraph morphism that commutes with cocone spangraph morphisms. This is defined abstractly by using a definite description, and is defined concretely as the comediator of the associated (morphism, endorelation) pair.

Figure 35: Comediator

$$\begin{tikzcd}
B \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] & A \arrow[r, "\gamma"] \arrow[dr, "g"'] & \bar{A} \arrow[d, "\bar{a}"] \\
 & & C
\end{tikzcd}$$

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

(28) (forall (?x (set$set ?x)
              ?s ((cocone ?x) ?s))
         (= ((comediator ?x) ?s)
            ((sgph.col.endo$comediator ?x)
                [((morphism ?x) ?s) ((endorelation ?x) ((cocone-diagram ?x) ?s))])))
```

Pushouts

sgph.col.psh

A *pushout* is a finite colimit in the category Spangraph for a diagram of shape *span* = • ← • → •. Such a diagram of spangraphs and spangraph morphisms is called a *span* (see Figure 36, where arrows denote spangraph morphisms)

$$\begin{tikzcd} & B \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\ A_1 & & A_2 \end{tikzcd}$$

Figure 36: Pushout Diagram = Span

A *span* of spangraphs and spangraph morphisms is the appropriate base diagram for a pushout. Each span consists of a pair of spangraph morphisms called *first* and *second*. These are required to have a common source spangraph called the *vertex*. (Note that this vertex is different from the vertex used in the spangraph object namespace, which was a vertex set. The vertex here is a spangraph, which itself has a vertex set. The namespace prefixes can be used to disambiguate.) The target spangraphs for first and second are given the names *spangraph*$_1$ and *spangraph*$_2$. 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 graph that is also named span. Spans are determined by their pair of component spangraph morphisms.

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

(2) (KIF$function spangraph1)
    (= (KIF$source spangraph1) set$set)
    (= (KIF$target spangraph1) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (spangraph1 ?x)) (diagram ?x))
             (= (SET.FTN$target (spangraph1 ?x)) (sgph.fbr.obj$object ?x))))

(3) (KIF$function spangraph2)
    (= (KIF$source spangraph2) set$set)
    (= (KIF$target spangraph2) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (spangraph2 ?x)) (diagram ?x))
             (= (SET.FTN$target (spangraph2 ?x)) (sgph.fbr.obj$object ?x))))

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

(5) (KIF$function first)
    (= (KIF$source first) set$set)
    (= (KIF$target first) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (first ?x)) (diagram ?x))
             (= (SET.FTN$target (first ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
             (= (SET.FTN$composition [(first ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                (vertex ?x))
             (= (SET.FTN$composition [(first ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                (spangraph1 ?x))))

(6) (KIF$function second)
    (= (KIF$source second) set$set)
    (= (KIF$target second) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (second ?x)) (diagram ?x))
             (= (SET.FTN$target (second ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
             (= (SET.FTN$composition [(second ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                (vertex ?x))
             (= (SET.FTN$composition [(second ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                (spangraph2 ?x))))
```

```
(7) (forall (?x (set$set ?x)
             ?d1 ((diagram ?x) ?d1) ?d2 ((diagram ?x) ?d2))
        (=> (and (= ((first ?x) ?d1) ((first ?x) ?d2))
                 (= ((second ?x) ?d1) ((second ?x) ?d2)))
            (= ?d1 ?d2)))
```

The *pair* of target spangraphs (suffixing discrete diagram) underlying any span of spangraphs 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$set)
    (= (KIF$target pair) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (pair ?x)) (diagram ?x))
             (= (SET.FTN$target (pair ?x)) (sgph.col.coprd2$diagram ?x))
             (= (SET.FTN$composition [(pair ?x) (sgph.col.coprd2$spangraph1 ?x)])
                (spangraph1 ?x))
             (= (SET.FTN$composition [(pair ?x) (sgph.col.coprd2$spangraph2 ?x)])
                (spangraph2 ?x))))
```

Every span has an *opposite*.

```
(9) (KIF$function opposite)
    (= (KIF$source opposite) set$set)
    (= (KIF$target opposite) SET.FTN$function)
    (forall (?x (set$set ?x))
        (and (= (SET.FTN$source (opposite ?x)) (diagram ?x))
             (= (SET.FTN$target (opposite ?x)) (diagram ?x))
             (= (SET.FTN$composition [(opposite ?x) (spangraph1 ?x)]) (spangraph2 ?x))
             (= (SET.FTN$composition [(opposite ?x) (spangraph2 ?x)]) (spangraph1 ?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$set ?x))
         (= (SET.FTN$composition [(opposite ?x) (opposite ?x)])
            (SET.FTN$identity (diagram ?x))))
```

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

```
(11) (KIF$function vertex-diagram)
     (= (KIF$source vertex-diagram) set$set)
     (= (KIF$target vertex-diagram) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (vertex-diagram ?x)) (diagram ?x))
              (= (SET.FTN$target (vertex-diagram ?x)) set.col.psh$diagram)
              (= (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$set1])
                 (SET.FTN$composition [(spangraph1 ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$set2])
                 (SET.FTN$composition [(spangraph2 ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$vertex])
                 (SET.FTN$composition [(vertex ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$first])
                 (SET.FTN$composition [(first ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$second])
                 (SET.FTN$composition [(second ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))))

(12) (KIF$function edge-diagram)
     (= (KIF$source edge-diagram) set$set)
     (= (KIF$target edge-diagram) SET.FTN$function)
     (forall (?x (set$set ?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 [(spangraph1 ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$set2])
                 (SET.FTN$composition [(spangraph2 ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$vertex])
                 (SET.FTN$composition [(vertex ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$first])
                 (SET.FTN$composition [(first ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(edge-diagram ?x) set.col.psh$second])
                 (SET.FTN$composition [(second ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))))

(13) (KIF$function node-diagram)
     (= (KIF$source node-diagram) set$set)
     (= (KIF$target node-diagram) SET.FTN$function)
     (forall (?x (set$set ?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 [(spangraph1 ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$set2])
                 (SET.FTN$composition [(spangraph2 ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$vertex])
                 (SET.FTN$composition [(vertex ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$first])
                 (SET.FTN$composition [(first ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(node-diagram ?x) set.col.psh$second])
                 (SET.FTN$composition [(second ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))))
```

The *parallel pair* or *coequalizer diagram* function maps a spangraph pushout diagram to the associated spangraph coequalizer diagram. See Figure 37, where arrows denote spangraph morphisms. The parallel pair of spangraph morphisms in this coequalizer diagram is the composite of the first and second spangraph morphisms of the pushout diagram with the coproduct injection spangraph morphisms for the binary coproduct of the pair of component spangraphs in the pushout diagram. The coequalizer and canon of the coequalizer diagram will be used to give a specific definition for the pushout.

$$\begin{tikzcd} & B \arrow[dl, "f_1"'] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr, "f_2"] & \\ A_1 \arrow[r, "\iota_1"'] & A_1+A_2 & A_2 \arrow[l, "\iota_2"] \end{tikzcd}$$

Figure 37: Coequalizer Diagram

```
(14) (KIF$function coequalizer-diagram)
     (KIF$function parallel-pair)
     (= parallel-pair coequalizer-diagram)
     (= (KIF$source coequalizer-diagram) set$set)
     (= (KIF$target coequalizer-diagram) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (coequalizer-diagram ?x)) (diagram ?x))
              (= (SET.FTN$target (coequalizer-diagram ?x)) (sgph.col.coeq$diagram ?x))
              (= (SET.FTN$composition [(coequalizer-diagram ?x) (sgph.col.coeq$origin ?x)])
                 (vertex ?x))
              (= (SET.FTN$composition [(coequalizer-diagram ?x) (sgph.col.coeq$destination ?x)])
                 (SET.FTN$composition [(pair ?x) (sgph.col.coprd2$binary-coproduct ?x)]))
              (forall (?d ((diagram ?x) ?d))
                  (and (= ((sgph.col.coeq$morphism1 ?x) ((coequalizer-diagram ?x) ?d))
                          ((sgph.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                              [((first ?x) ?d) ((sgph.col.coprd2$injection1 ?x) ((pair ?x) ?d))]))
                       (= ((sgph.col.coeq$morphism2 ?x) ((coequalizer-diagram ?x) ?d))
                          ((sgph.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                              [((second ?x) ?d) ((sgph.col.coprd2$injection2 ?x) ((pair ?x) ?d))]))))))
```

Here we assert three important categorical identities (not just isomorphisms). For any pushout diagram (in Spangraph), these relate the vertex, edge or node diagrams (in Set) of the coequalizer diagram (in Spangraph) 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$set ?x))
         (and (= (SET.FTN$composition [(coequalizer-diagram ?x) (sgph.col.coeq$vertex-diagram ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.coeq$coequalizer-diagram]))
              (= (SET.FTN$composition [(coequalizer-diagram ?x) (sgph.col.coeq$edge-diagram ?x)])
                 (SET.FTN$composition [(edge-diagram  ?x) set.col.coeq$coequalizer-diagram]))
              (= (SET.FTN$composition [(coequalizer-diagram ?x) (sgph.col.coeq$node-diagram])
                 (SET.FTN$composition [(node-diagram  ?x) set.col.coeq$coequalizer-diagram]))))

(16) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(SET.FTN$composition
                     [(coequalizer-diagram ?x)
```

```
                      (sgph.col.coeq$colimiting-cocone ?x)])
                      (sgph.col.coeq$vertex-diagram ?x)])
                 (SET.FTN$composition [(SET.FTN$composition
                     [(vertex-diagram ?x)
                       set.col.psh$coequalizer-diagram])
                       set.col.coeq$colimiting-cocone]))
              (= (SET.FTN$composition [(SET.FTN$composition
                     [(coequalizer-diagram ?x)
                      (sgph.col.coeq$colimiting-cocone ?x)])
                      (sgph.col.coeq$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)
                      (sgph.col.coeq$colimiting-cocone ?x)])
                      (sgph.col.coeq$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 38, where arrows denote spangraph morphisms) has the following constituents:

- an overlying *diagram* (the shaded part of Figure 38),
- an *opvertex* spangraph ***A***, and
- a pair of spangraph morphisms called *opfirst* and *opsecond*.

The common target spangraph of opfirst and opsecond is the opvertex. The source spangraphs of opfirst and opsecond are the target spangraphs of the spangraph morphisms in the pushout diagram. The opfirst and opsecond spangraph 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.

$$\begin{tikzcd}
 & \boldsymbol{B} \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\
\boldsymbol{A}_1 \arrow[dr, "op1st"'] & & \boldsymbol{A}_2 \arrow[dl, "op2nd"] \\
 & \boldsymbol{A} &
\end{tikzcd}$$

Figure 38: Pushout Cocone

```
(17) (KIF$function cocone)
     (= (KIF$source cocone) set$set)
     (= (KIF$target cocone) SET$class)

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

(20) (KIF$function opfirst)
     (= (KIF$source opfirst) set$set)
     (= (KIF$target opfirst) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (opfirst ?x)) (cocone ?x))
              (= (SET.FTN$target (opfirst ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (SET.FTN$composition [(cocone-diagram ?x) (spangraph1 ?x)]))
              (= (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (opvertex ?x))))

(21) (KIF$function opsecond)
```

```
     (= (KIF$source opsecond) set$set)
     (= (KIF$target opsecond) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (opsecond ?x)) (cocone ?x))
              (= (SET.FTN$target (opsecond ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (SET.FTN$composition [(cocone-diagram ?x) (spangraph2 ?x)]))
              (= (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (opvertex ?x))))

(22) (forall (?x (set$set ?x)
              ?s ((cocone ?x) ?s))
         (= ((sgph.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                [((first ?x) ((cocone-diagram ?x) ?s)) ((opfirst ?x) ?s)])
            ((sgph.mor$composition [(set.ftn$identity ?x) (set.ftn$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$set)
     (= (KIF$target binary-coproduct-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (binary-coproduct-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (binary-coproduct-cocone ?x)) (sgph.col.coprd2$cocone ?x))
              (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (sgph.col.coprd2$cocone-diagram ?x)])
                 (SET.FTN$composition [(cocone-diagram ?x) (pair ?x)]))
              (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (sgph.col.coprd2$opvertex ?x)])
                 (opvertex ?x))
              (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (sgph.col.coprd2$opfirst ?x)])
                 (opfirst ?x))
              (= (SET.FTN$composition [(binary-coproduct-cocone ?x) (sgph.col.coprd2$opsecond ?x)])
                 (opsecond ?x))))
```

The *coequalizer cocone* function maps a spangraph pushout cocone to the associated spangraph coequalizer cocone. See Figure 39, where arrows denote spangraph 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 spangraph morphism of the coequalizer cocone is the binary coproduct comediator of the opfirst and opsecond spangraph morphisms with respect to the binary coproduct pair diagram of the cocone.

$$\begin{tikzcd}
 & \boldsymbol{B} \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\
\boldsymbol{A}_1 \arrow[r, "\iota_1"'] \arrow[dr, "op1^{st}"'] & \boldsymbol{A}_1+\boldsymbol{A}_2 \arrow[d, "\gamma"] & \boldsymbol{A}_2 \arrow[l, "\iota_2"] \arrow[dl, "op2^{nd}"] \\
 & \boldsymbol{A} &
\end{tikzcd}$$

Figure 39: 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 39 demonstrates that this cocone is well defined.

$$\boldsymbol{f}_1 \cdot \boldsymbol{\iota}_1 \cdot \boldsymbol{\gamma} = \boldsymbol{f}_1 \cdot op1^{st} = \boldsymbol{f}_2 \cdot op2^{nd} = \boldsymbol{f}_2 \cdot \boldsymbol{\iota}_2 \cdot \boldsymbol{\gamma}$$

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

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

```
(24) (KIF$function vertex-cocone)
     (= (KIF$source vertex-cocone) set$set)
     (= (KIF$target vertex-cocone) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (vertex-cocone ?x)) (cocone ?x))
              (= (SET.FTN$target (vertex-cocone ?x)) set.col.psh$cocone)
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.psh$cocone-diagram])
                 (SET.FTN$composition [(cocone-diagram ?x) (vertex-diagram ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.psh$opvertex])
                 (SET.FTN$composition [(opvertex ?x) (sgph.fbr.obj$vertex ?x)]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.psh$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(vertex-cocone ?x) set.col.psh$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))]))))
```

```
(25) (KIF$function edge-cocone)
     (= (KIF$source edge-cocone) set$set)
     (= (KIF$target edge-cocone) SET.FTN$function)
     (forall (?x (set$set ?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$composition [(opvertex ?x) (sgph.fbr.obj$edge ?x)]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(edge-cocone ?x) set.col.psh$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$edge (set.ftn$identity ?x))]))))
```

```
(26) (KIF$function node-cocone)
     (= (KIF$source node-cocone) set$set)
     (= (KIF$target node-cocone) SET.FTN$function)
     (forall (?x (set$set ?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$composition [(opvertex ?x) (sgph.fbr.obj$node ?x)]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$opfirst])
                 (SET.FTN$composition [(opfirst ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))
              (= (SET.FTN$composition [(node-cocone ?x) set.col.psh$opsecond])
                 (SET.FTN$composition [(opsecond ?x) (sgph.fbr.mor$node (set.ftn$identity ?x))]))))
```

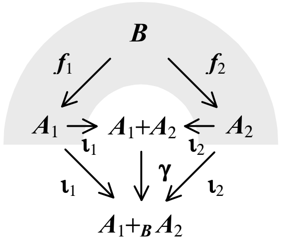


Figure 40: Colimiting Cocone

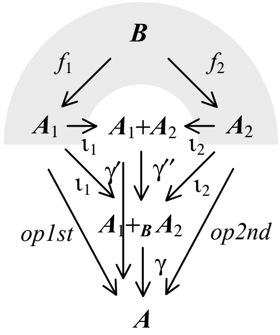


Figure 41: Comediator

The *colimiting cocone* function maps a pushout diagram to its pushout (colimiting pushout cocone). See Figure 40, where arrows denote spangraph 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* spangraph, which comes equipped with two *injection* spangraph morphisms. The last axiom expresses concreteness of the colimit – it expresses pushouts in terms of coproducts and coequalizers (Figure 41): 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.

```
(27) (KIF$function colimiting-cocone)
     (= (KIF$source colimiting-cocone) set$set)
     (= (KIF$target colimiting-cocone) SET.FTN$function)
     (forall (?x (set$set ?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)))))

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

(29) (KIF$function injection1)
     (= (KIF$source injection1) set$set)
     (= (KIF$target injection1) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (injection1 ?x)) (diagram ?x))
              (= (SET.FTN$target (injection1 ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (spangraph1 ?x))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (colimit ?x))
              (= (injection1 ?x) (SET.FTN$composition [(colimiting-cocone ?x) (opfirst ?x)]))))

(30) (KIF$function injection2)
     (= (KIF$source injection2) set$set)
     (= (KIF$target injection2) SET.FTN$function)
     (forall (?x (set$set ?x))
         (and (= (SET.FTN$source (injection2 ?x)) (diagram ?x))
              (= (SET.FTN$target (injection2 ?x)) (sgph.fbr.mor$morphism (set.ftn$identity ?x)))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$source (set.ftn$identity ?x))])
                 (spangraph2 ?x))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$target (set.ftn$identity ?x))])
                 (colimit ?x))
              (= (injection2 ?x) (SET.FTN$composition [(colimiting-cocone ?x) (opsecond ?x)]))))

(31) (forall (?x (set$set ?x)
              ?d ((diagram ?x) ?d))
         (and (= (colimit ?d)
                 ((sgph.col.coeq$coequalizer ?x) ((coequalizer-diagram ?x) ?d)))
              (= ((injection1 ?x) ?d)
                 ((sgph.fbr.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                     [((sgph.col.coprd2$injection1 ?x) ((pair ?x) ?d))
                      ((sgph.col.coeq$canon ?x) ((coequalizer-diagram ?x) ?d))]))
              (= ((injection2 ?x) ?d)
                 ((sgph.fbr.mor$composition [(set.ftn$identity ?x) (set.ftn$identity ?x)])
                     [((sgph.col.coprd2$injection2 ?x) ((pair ?x) ?d))
                      ((sgph.col.coeq$canon ?x) ((coequalizer-diagram ?x) ?d))]))))
```

The following three axioms are the necessary conditions that the vertex, 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 spangraph morphisms are specific.

```
(32) (forall (?x (set$set ?x))
         (and (= (SET.FTN$composition [(colimiting-cocone ?x) (vertex-cocone ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$colimiting-cocone]))
              (= (SET.FTN$composition [(colimit ?x) (sgph.fbr.obj$vertex ?x)])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$colimit]))
              (= (SET.FTN$composition [(injection1 ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$injection1]))
              (= (SET.FTN$composition [(injection2 ?x) (sgph.fbr.mor$vertex (set.ftn$identity ?x))])
                 (SET.FTN$composition [(vertex-diagram ?x) set.col.psh$injection2]))))

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

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

The *comediator* spangraph morphism, from the pushout of a pushout diagram to the opvertex of a pushout cocone under the diagram (see Figure 41, where arrows denote spangraph morphisms), is the unique spangraph 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.

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

(36) (forall (?x (set$set ?x))
         (= (comediator ?x)
            (SET.FTN$composition [(coequalizer-cocone ?x) (sgph.col.coeq$comediator ?x)])))
```