The IFF Lower Core (meta) Ontology

A Rough Cut!

This meta-ontology has been released in order to show the full architecture of the SUO IFF. A more polished complete ontology will be released later.

Things to do

- Define diagram morphisms and discrete diagram morphisms.
- Define (discrete) diagram-morphism conversion operator in the function namespace.
- Define $\sum \boldsymbol{H} : \sum \boldsymbol{F}_1 \to \sum \boldsymbol{F}_2$ in the coproduct namespace, where $\boldsymbol{H} : \sum \boldsymbol{F}_1 \to \sum \boldsymbol{F}_2$ is a morphism of discrete diagrams.

Table 1 lists the terminology in the IFF Lower Core Ontology. The blue area colors the terminology for the function double category: sets are 0-cells, functions are both horizontal and vertical arrows (1-cells), and quartets are squares (2-cells). Make this a true double category:

* Identify the 'function' namespace with the 'set function' namespace.

* Change 'function-morphism' to 'square'.

Table 1: Lower Metalevel Core Ontology

	Class	Function	Other
set	tuple	arity type type-fiber classification vertical-identity	
set .ftn	function	vertical-identity domain codomain arrow fiber power singleton designation vertical-composition signature arity assign substitution source target	vertically-composable-opspan vertically-composable
set .qtt	function-morphism fibration	source target domain codomain opspan cone fiber composition identity vertical-composition vertical-identity signature assign	composable-opspan composable vertically-composable-opspan vertically-composable
rel			

ord			
gph			

The Namespace of Sets

Sets

set

In order to define ontological and information flow languages we assume an "adequate" (possibly denumerable) collection of variables. Variables are used for indexing and naming the (functional) participants of relations. They correspond to the lines of identity and lambda variables of conceptual graphs.

- For any set A the collection of all function $\mathsf{F}A$ whose source and target sets are in the power set $\wp A$, is a set-theoretically small collection; that is, it is a set. We call this the *morphic power* of A. We give the special names *domain* and *codomain* for the source and target of such functions.

```
(1) (SET.FTN$function morphic-power)
    (= (SET.FTN$source morphic-power) set)
    (= (SET.FTN$target morphic-power) set)
    (forall (?a (set ?a))
        (and (set$subset (morphic-power ?a) set.ftn$function)
             (forall (?f (set.ftn$function ?f))
                 (<=> ((morphic-power ?a) ?h)
                      (and (set$subset (set.ftn$source ?f) (power ?a))
                           (set$subset (set.ftn$target ?f) (power ?a))
                           (= ((domain ?a) ?f) (set.ftn$source ?f))
                           (= ((codomain ?a) ?f) (set.ftn$target ?f)))))))

(2) (SET.FTN$function domain)
    (= (SET.FTN$source domain) set)
    (= (SET.FTN$target domain) set.ftn$function)
    (= (SET.FTN$composition [domain set.ftn$source]) morphic-power)
    (= (SET.FTN$composition [domain set.ftn$target]) power)

(3) (SET.FTN$function codomain)
    (= (SET.FTN$source codomain) set)
    (= (SET.FTN$target codomain) set.ftn$function)
    (= (SET.FTN$composition [codomain set.ftn$source]) morphic-power)
    (= (SET.FTN$composition [codomain set.ftn$target]) power)
```

- The set pairc function maps a set A to its diagonal set pair $\mathit{pr}(A) = \langle A, A \rangle$.

```
(4) (SET.FTN$function pair)
    (= (SET.FTN$source pair) set)
    (= (SET.FTN$target pair) set.pr$pair)
    (= (SET.FTN$composition [pair set2])
       (SET.FTN$identity set))
    (= (SET.FTN$composition [pair set1])
       (SET.FTN$identity set))
```

- The classification function maps a set A to its associated identity classification $\mathit{cls}(A) = \langle A, A, \models_A \rangle$. The set pair underlying this classification is the diagonal set pair.

```
(5) (SET.FTN$function classification)
    (= (SET.FTN$source classification) set)
    (= (SET.FTN$target classification) cls$classification)
    (= (SET.FTN$composition [classification instance])
       (SET.FTN$identity set))
    (= (SET.FTN$composition [classification type])
       (SET.FTN$identity set))
    (= classification rel$identity)

   (= (SET.FTN$composition [classification cls$pair]) pair)
```

Colimits

Coproducts

`set.col.coprd`

A *coproduct* (Diagram 1) is a colimit for a discrete diagram (diagram of discrete shape). Discrete diagrams are in bijective correspondence with tuple sets (Figure 1).

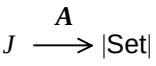


Figure 1: Tuple of Sets

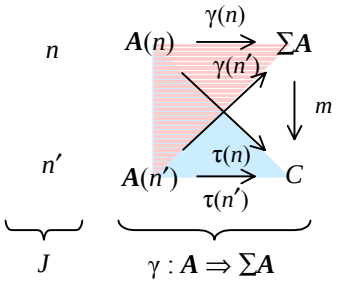


Diagram 1: Coproduct

- A *tuple of sets* $\boldsymbol{A} = \{\boldsymbol{A}(n) \mid n \in J\}$ (Figure 1) is a function to |Set| whose source is a particular *index*ing set J. So any tuple of sets is a class function. Thus, all the tuple-sets form a subcollection of the collection of class functions. Set-theoretically, this is a generic (larger-than-large) collection. Compare this to the class of tuples, which is just a large collection (class).
- A *tuple of sets* $\boldsymbol{A} = \langle \mathit{index}(\boldsymbol{A}), \mathit{ftn}(\boldsymbol{A}) \rangle$ consists of an *index* (source) set $J = \mathit{index}(\boldsymbol{A})$, and is a *function* $\boldsymbol{A} : J \to |\mathsf{Set}|$; in other notation, $\boldsymbol{A} = \{\boldsymbol{A}(n) \mid n \in J\}$. Since each tuple set is a class function, the collection of all the tuple sets forms a collection.

```
(1) (KIF$collection tuple-set)
    (KIF$subcollection tuple-set SET.FTN$function)

(2) (KIF$function index)
    (= (KIF$source index) tuple-set)
    (= (KIF$target index) set$set)
    (= index SET.FTN$source)
```

- For any tuple of sets $\boldsymbol{A} : J \to |\mathsf{Set}|$ there is a discrete *diagram* whose shape is the discrete graph of the arity class J, whose object function (set) is A and whose morphism function (function) is empty.

```
(3) (KIF$function diagram)
    (= (KIF$source diagram) tuple-set)
    (= (KIF$target diagram) dgm$diagram)
    (forall (?a (tuple-set ?a))
        (and (= (dgm$shape (diagram ?a)) (gph$graph (index ?a)))
             (= (dgm$set (diagram ?a)) ?a)))
```

- The diagram function is injective – the tuple set of the diagram of a tuple set is the original.

```
(forall (?a (tuple-set ?a))
    (= (dgm$tuple-set (diagram ?a)) ?a))
```

- A *cocone* (Diagram 1) consists of a tuple-set $\boldsymbol{A}$, an *opvertex* C, and a collection τ of *component* functions indexed by the nodes in the index set of the tuple-set (= diagram). The cocone is situated under the tuple-set.

```
(4) (KIF$class cocone)

(5) (KIF$function cocone-tuple-set)
    (= (KIF$source cocone-tuple-set) cocone)
    (= (KIF$target cocone-tuple-set) tuple-set)

(6) (KIF$function opvertex)
    (= (KIF$source opvertex) cocone)
    (= (KIF$target opvertex) set$set)

(7) (KIF$function component)
    (= (KIF$source component) cocone)
    (= (KIF$target component) SET.FTN$function)
    (forall (?s (cocone ?s))
        (and (= (SET.FTN$source (component ?s)) (index (cocone-tuple-set ?s)))
             (= (SET.FTN$target (component ?s)) set.ftn$function)
             (forall (?n ((index (cocone-tuple-set ?s)) ?n))
                 (and (= (set.ftn$source ((component ?s) ?n))
```

```
      ((cocone-tuple-set ?s) ?n))
   (= (set.ftn$target ((component ?s) ?n))
      (opvertex ?s))))))
```

- The KIF function '`colimiting-cocone`' maps a tuple set (discrete diagram) $\boldsymbol{A}$ to its coproduct (colimiting coproduct cocone) (Diagram 1)

  $$\begin{aligned} \mathit{coprd}(\boldsymbol{A}) &= \textstyle\sum_{n \in \mathit{index}(\boldsymbol{A})} \boldsymbol{A}(n) \\ &= \{(n, x) \mid n \in \mathit{index}(\boldsymbol{A}), x \in \boldsymbol{A}(n)\}. \end{aligned}$$

  A colimiting coproduct cocone is the special case of a general colimiting cocone over a coproduct diagram.

  For any tuple set $\boldsymbol{A}$ and any index $n \in \mathit{index}(\boldsymbol{A})$ there is an *injection* function

  $$\mathit{inj}(\boldsymbol{A})(n) : \boldsymbol{A}(n) \to \mathit{coprd}(\boldsymbol{A})$$

  defined by $\mathit{inj}(\boldsymbol{A})(n)(x) = (n, x)$ for all indices $n \in \mathit{index}(\boldsymbol{A})$ and all elements $x \in \boldsymbol{A}(n)$. Obviously, the injections are injective.

```
(8) (KIF$function colimiting-cocone)
    (= (KIF$source colimiting-cocone) tuple-set)
    (= (KIF$target colimiting-cocone) cocone)
    (forall (?a (tuple-set ?a))
        (= ?a (cocone-tuple-set (colimiting-cocone ?a))))

(9) (KIF$function colimit)
    (KIF$function coproduct)
    (= coproduct colimit)
    (= (KIF$source colimit) tuple-set)
    (= (KIF$target colimit) set$set)
    (forall (?a (tuple-set ?a))
        (= (colimit ?a) (opvertex (colimiting-cocone ?a))))

(10) (KIF$function injection)
     (= (KIF$source injection) tuple-set)
     (= (KIF$target injection) SET.FTN$function)
     (forall (?a (tuple-set ?a)
         (= (injection ?a) (component (colimiting-cocone ?a))))
```

- There is a *comediator* function (Diagram 1) to the opvertex of a coproduct cocone over a tuple set from the coproduct of the tuple set. This is the unique function that commutes with the component functions of the cocone. We have also introduced a "convenience term" cotupling. With an tuple set parameter, this maps a tuple of set functions, which form a coproduct cocone with the tuple set, to their comediator (or *cotupling*) function.

```
(11) (KIF$function comediator)
     (= (KIF$source comediator) cocone)
     (= (KIF$target comediator) set.ftn$function)
     (forall (?s (cocone ?s))
         (and (= (set.ftn$source (comediator ?s))
                 (colimit (cocone-tuple-set ?s)))
              (= (set.ftn$target (comediator ?s)) (opvertex ?s))))

(12) (KIF$function cotupling-cocone)
     (KIF$source cotupling-cocone) tuple-set)
     (KIF$target cotupling-cocone) SET.FTN$partial-function)
     (forall (?a (tuple-set ?a))
         (and (= (SET.FTN$source (cotupling-cocone ?a))
                 (SET.FTN$power [(index ?a) set.ftn$function]))
              (= (SET.FTN$target (cotupling-cocone ?a)) cocone)
              (forall (?f ((SET.FTN$power [(index ?a) set.ftn$function]) ?f))
                  (<=> ((SET.FTN$domain (cotupling-cocone ?a)) ?f)
                       (and (forall (?n ((index ?a) ?n))
                                (= (set.ftn$source (?f ?n)) (?a ?n)))
                            (exists (?c (set$set ?c))
                                (forall (?n ((index ?a) ?n))
                                    (= (set.ftn$target (?f ?n)) ?c)))))))))

(13) (KIF$function cotupling)
```

```
(= (KIF$source cotupling) tuple-set)
(= (KIF$target cotupling) SET.FTN$partial-function)
(forall (?a (tuple-set ?a))
    (and (= (SET.FTN$source (cotupling ?a))
            (SET.FTN$power [(index ?a) set.ftn$function]))
         (= (SET.FTN$target (cotupling ?a)) set.ftn$function)
         (= (SET.FTN$domain (cotupling ?a))
            (SET.FTN$domain (cotupling-cocone ?a)))
         (forall ?f ((SET.FTN$domain (cotupling ?a)) ?f))
             (= ((cotupling ?a) ?f)
                (comediator ((cotupling-cocone ?a) ?f))))))
```

- There is an *indication* function (special for coproducts) (Diagram 2)

  $indic(\boldsymbol{A}) : coprd(\boldsymbol{A}) \to index(\boldsymbol{A})$

  from the coproduct to the tuple set index, which is defined by $indic(\boldsymbol{A})(n, x) = n$.

  The indication function is the cotupling of the collection of constant index functions

  $\{\Delta_{\boldsymbol{A}}(n) : \boldsymbol{A}(n) \to index(\boldsymbol{A}) \mid n \in index(\boldsymbol{A})\}$.

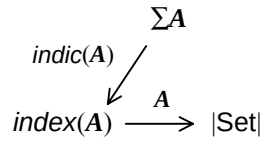


Diagram 2: Colimit and Indication Function of a Tuple Set

```
(14) (KIF$function constant-index)
     (= (KIF$source constant-index) tuple-set)
     (= (KIF$target constant-index) SET.FTN$function)
     (forall (?a (tuple-set ?a))
         (and (= (SET.FTN$source (constant-index ?a)) (index ?a))
              (= (SET.FTN$target (constant-index ?a)) set.ftn$function)
              (forall (?n ((index ?a) ?n))
                  (and (= (set.ftn$source ((constant-index ?a) ?n)) (?a ?n))
                       (= (set.ftn$target ((constant-index ?a) ?n)) (index ?a))
                       (= ((constant-index ?a) ?n)
                          ((set.ftn$constant [(?a ?n) (index ?a)]) ?n))))))

(15) (SET.FTN$function indication)
     (= (SET.FTN$source indication) tuple-set)
     (= (SET.FTN$target indication) set.ftn$function)
     (forall (?a (tuple-set ?a)
         (and (= (set.ftn$source (indication ?a)) (coproduct ?a))
              (= (set.ftn$target (indication ?a)) (index ?a))
              (= (indication ?a) ((cotupling ?a) (constant-index ?a)))))
```

Coproduct Morphisms

`set.col.coprd.mor`

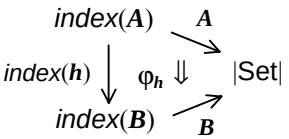
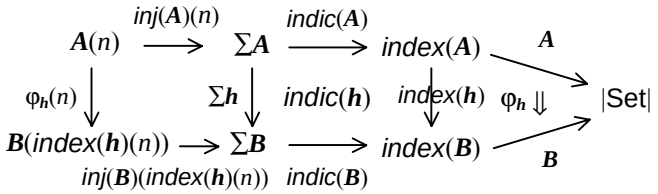


Diagram 3: Tuple Set Morphism

- A *morphism* of tuple sets $\boldsymbol{h} = \langle \mathit{index}(\boldsymbol{h}), \mathit{tfm}(\boldsymbol{h}) \rangle : \boldsymbol{A} \to \boldsymbol{B}$ consists of
  - an index function $\mathit{index}(\boldsymbol{h}) : \mathit{index}(\boldsymbol{A}) \to \mathit{index}(\boldsymbol{B})$ and
  - a collection of transform functions
    $\mathit{tfm}(\boldsymbol{h}) = \varphi(\boldsymbol{h}) = \{\varphi_{\boldsymbol{h}}(n) : \boldsymbol{A}(n) \to \boldsymbol{B}(\mathit{index}(\boldsymbol{h})(n)) \mid n \in J\}$
    indexed by $\mathit{index}(\boldsymbol{A})$, whose source sets are given by $\boldsymbol{A}$ and whose target sets are given by $\mathit{index}(\boldsymbol{h}) \cdot \boldsymbol{B}$.

```
(1) (KIF$collection tuple-set-morphism)

(2) (KIF$function source)
    (= (KIF$source source) tuple-set-morphism)
    (= (KIF$target source) set.col.coprd$tuple-set)

(3) (KIF$function target)
    (= (KIF$source target) tuple-set-morphism)
    (= (KIF$target target) set.col.coprd$tuple-set)

(4) (KIF$function index)
    (= (KIF$source index) tuple-set-morphism)
    (= (KIF$target index) set.ftn$function)
    (forall (?h (tuple-set-morphism ?h))
        (and (= (set.ftn$source (index ?h))
                (set.col.coprd$index (source ?h)))
             (= (set.ftn$target (index ?h))
                (set.col.coprd$index (target ?h)))))

(5) (KIF$function transform)
    (= (KIF$source transform) tuple-set-morphism)
    (= (KIF$target transform) SET.FTN$function)
    (forall (?h (tuple-set-morphism ?h))
        (and (= (SET.FTN$source (transform ?h)) (index (source ?h)))
             (= (SET.FTN$target (transform ?h)) set.ftn$function)
             (= (SET.FTN$composition [(transform ?h) set.ftn$source]) (source ?h))
             (= (SET.FTN$composition [(transform ?h) set.ftn$target])
                (SET.FTN$composition [(index ?h) (target ?h)])))))
```



- Any morphism of tuple sets $\boldsymbol{h} = \langle \mathit{index}(\boldsymbol{h}), \mathit{tfm}(\boldsymbol{h}) \rangle : \boldsymbol{A} \to \boldsymbol{B}$ defines a *coproduct* function

  $\mathit{coprd}(\boldsymbol{h}) = \sum \boldsymbol{h} : \mathit{coprd}(\boldsymbol{A}) = \sum \boldsymbol{A} \to \sum \boldsymbol{B} = \mathit{coprd}(\boldsymbol{B})$.

  The pointwise definition is:

  $\mathit{coprd}(\boldsymbol{h})((n, x)) = (\mathit{index}(\boldsymbol{h})(n), \mathit{tfm}(\boldsymbol{h})(n)(x)))$

  for any index $n \in \mathit{arity}(\boldsymbol{A})$ and element $x \in \boldsymbol{A}(n)$. This is well defined by definition of the transform $\varphi_{\boldsymbol{h}}$. The coproduct function is the cotupling of the injection of transforms:

  $\{\varphi_{\boldsymbol{h}}(n) \cdot \mathit{inj}(\boldsymbol{B})(\mathit{index}(\boldsymbol{h})(n)) \mid n \in \mathit{index}(\boldsymbol{A})\}$.

```
(6) (KIF$function coproduct-tuple)
    (= (KIF$source coproduct-tuple) tuple-set-morphism)
    (= (KIF$target coproduct-tuple) SET.FTN$function)
    (forall (?h (tuple-set-morphism ?h))
        (and (= (SET.FTN$source (coproduct-tuple ?h)) (set.col.coprd$index (source ?h)))
             (= (SET.FTN$target (coproduct-tuple ?h)) set.ftn$function)
             (forall (?n ((set.col.coprd$index (source ?h)) ?n))
                 (and (= (set.ftn$source ((coproduct-tuple ?h) ?n))
```

```
                        ((source ?h) ?n))
                    (= (set.ftn$target ((coproduct-tuple ?h) ?n))
                       (set.col.coprd$coproduct (target ?h)))
                    (= ((coproduct-tuple ?h) ?n)
                       (set.ftn$composition
                           [(transform ?h) ?n)
                            ((set.col.coprd$injection (target ?h)) ((index ?h) ?n))])))))
```

```
(7) (SET.FTN$function coproduct)
     (= (SET.FTN$source coproduct) tuple-set-morphism)
     (= (SET.FTN$target coproduct) set.ftn$function)
     (forall (?h (tuple-set-morphism ?h))
         (and (= (set.ftn$source (coproduct ?h)) (set.col.coprd$coproduct (source ?h)))
              (= (set.ftn$target (coproduct ?d)) (set.col.coprd$coproduct (target ?h)))
              (= (coproduct ?h) ((cotupling (source ?h)) (coproduct-tuple ?h))))))
```

- For any morphism of tuple sets $\boldsymbol{h} = \langle \mathit{index}(\boldsymbol{h}), \mathit{tfm}(\boldsymbol{h}) \rangle : \boldsymbol{A} \to \boldsymbol{B}$ the coproduct function is the vertical source for an *indication* quartet $\mathit{indic}(\boldsymbol{h})$.
  - The commutativity (preservation of indication)

    $$\mathit{coprd}(\boldsymbol{h}) \cdot \mathit{indic}(\boldsymbol{B}) = \mathit{indic}(\boldsymbol{A}) \cdot \mathit{index}(\boldsymbol{h}),$$

    is a property of the coproduct of tuple sets. This is obvious from the pointwise definition of the coproduct function.

```
(8) (SET.FTN$function indication)
     (= (SET.FTN$source indication) tuple-set-morphism)
     (= (SET.FTN$target indication) set.qtt$quartet)
     (forall (?h (tuple-set-morphism ?h))
         (and (= (set.qtt$horizontal-source (indication ?h))
                 (set.col.coprd$indication (source ?h)))
              (= (set.qtt$horizontal-target (indication ?h))
                 (set.col.coprd$index (target ?h)))
              (= (set.qtt$vertical-source (indication ?h)) (coproduct ?h))
              (= (set.qtt$vertical-target (indication ?h)) (index ?h)))))
```

Based Coproducts

set.col.art

Discrete based diagrams are in bijective correspondence with tuple subsets. Many of the discrete diagrams used in the IFF Model Theory Ontology are based. Hence, in this section we identify diagrams with arities.

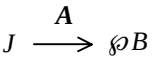


Figure 2: Arity

- A discrete based *diagram* (of sets) is the appropriate base diagram for a based coproduct. We identify each discrete based diagram with an arity – an indexed collection of subsets of a particular base set. We use either the generic term '`diagram`' or the specific term '`arity`' to denote the *arity* collection. A discrete based diagram is the special case of a general diagram of discrete shape that is based.

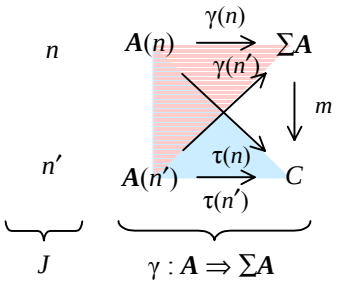


Diagram 4: Coproduct

- An *arity* or *tuple of subsets* $\boldsymbol{A} = \langle \mathit{index}(\boldsymbol{A}), \mathit{base}(\boldsymbol{A}), \mathit{ftn}(\boldsymbol{A}) \rangle$ consists of an *index* (source) set $J = \mathit{index}(\boldsymbol{A})$, a *base* set $B = \mathit{base}(\boldsymbol{p})$ and is a *function* $A : J \to \wp B$; in other notation, $\boldsymbol{A} = \{A_j \mid j \in J, A_j \subseteq B\}$. Since an arity is a set function, the collection of all the arities form a class.

```
(1) (SET$class arity)
    (SET$class tuple-subset)
    (= tuple-subset arity)

(2) (SET.FTN$function index)
    (= (SET.FTN$source index) arity)
    (= (SET.FTN$target index) set$set)

(3) (SET.FTN$function base)
    (= (SET.FTN$source base) arity)
    (= (SET.FTN$target base) set$set)

(4) (SET.FTN$function function)
    (= (SET.FTN$source function) arity)
    (= (SET.FTN$target function) set.ftn$function)
    (= (SET.FTN$composition [function set.ftn$source]) index)
    (= (SET.FTN$composition [function set.ftn$target])
       (SET.FTN$composition [base set$power]))
```

- Artities are determined by their arity-base-function triples. This fact is represented by an *injection* assertion for the function $\mathit{ftn} : \mathsf{arity} \to \mathsf{ftn}$.

```
(5) (SET.FTN$function inclusion)
    (= (SET.FTN$source inclusion) arity)
    (= (SET.FTN$target inclusion) set.ftn$function)
    (= (SET.FTN$composition [inclusion set.ftn$source]) index)
    (= (SET.FTN$composition [inclusion set.ftn$target])
       (SET.FTN$composition [base set$power]))

    (SET.FTN$injection function)
```

- A *cocone* (Diagram 4) consists of an arity $\boldsymbol{A}$, an *opvertex* C, and a collection τ of *component* functions indexed by the nodes in the index set of the arity (= diagram). The cocone is situated under the arity.

```
(6) (SET$class cocone)

(7) (SET.FTN$function cocone-arity)
    (= (SET.FTN$source cocone-arity) cocone)
    (= (SET.FTN$target cocone-arity) arity)

(8) (SET.FTN$function opvertex)
    (= (SET.FTN$source opvertex) cocone)
    (= (SET.FTN$target opvertex) set$set)

(9) (KIF$function component)
    (= (KIF$source component) cocone)
    (= (KIF$target component) SET.FTN$function)
    (forall (?s (cocone ?s))
```

```
(and (= (SET.FTN$source (component ?s)) (index (cocone-arity ?s)))
     (= (SET.FTN$target (component ?s)) set.ftn$function)
     (forall (?n ((index (cocone-arity ?s)) ?n))
         (and (= (set.ftn$source ((component ?s) ?n))
                 ((function (cocone-arity ?s)) ?n))
              (= (set.ftn$target ((component ?s) ?n))
                 (opvertex ?s))))))
```

- The KIF function '`colimiting-cocone`' maps an arity (discrete based diagram) $\boldsymbol{A}$ to its coproduct (colimiting coproduct cocone) (Diagram 5)

$$\begin{aligned} \mathit{coprd}(\boldsymbol{A}) &= \textstyle\sum_{n \in \mathit{index}(\boldsymbol{A})} \boldsymbol{A}(n) \\ &= \{(n, x) \mid n \in \mathit{index}(\boldsymbol{A}), x \in \boldsymbol{A}(n)\}. \end{aligned}$$

A colimiting coproduct cocone is the special case of a general colimiting cocone over a coproduct diagram (discrete based diagram or arity).

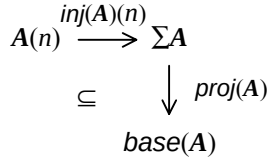


Diagram 5: Coproduct

For any arity $\boldsymbol{A}$ and any index $n \in \mathit{index}(\boldsymbol{A})$ there is an *injection* function

$$\mathit{inj}(\boldsymbol{A})(n) : \boldsymbol{A}(n) \to \mathit{coprd}(\boldsymbol{A})$$

defined by $\mathit{inj}(\boldsymbol{A})(n)(x) = (n, x)$ for all indices $n \in \mathit{index}(\boldsymbol{A})$ and all elements $x \in \boldsymbol{A}(n)$. Obviously, the injections are injective. They commute with projection and inclusion.

```
(10) (SET.FTN$function colimiting-cocone)
     (= (SET.FTN$source colimiting-cocone) arity)
     (= (SET.FTN$target colimiting-cocone) cocone)
     (SET.FTN$restriction colimiting-cocone set.col$colimiting-cocone)

(11) (SET.FTN$function colimit)
     (SET.FTN$function coproduct)
     (= coproduct colimit)
     (= (SET.FTN$source colimit) arity)
     (= (SET.FTN$target colimit) set$set)
     (forall (?a (arity ?a))
        (= (colimit ?a) (opvertex (colimiting-cocone ?a))))

(12) (SET.FTN$function injection)
     (= (SET.FTN$source injection) arity)
     (= (SET.FTN$target injection) set.ftn$function)
     (forall (?a (arity ?a)
         (= (injection ?a) (component (colimiting-cocone ?a))))
```

- There is a *comediator* function (Diagram 1) to the opvertex of a coproduct cocone over a coproduct diagram (discrete based diagram or arity) from the coproduct of the arity. This is the unique function that commutes with the component functions of the cocone. We have also introduced a "convenience term" cotupling. With an arity parameter, this maps a tuple of set functions, which form a coproduct cocone with the arity, to their comediator (or *cotupling*) function.

```
(13) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) cocone)
     (= (SET.FTN$target comediator) set.ftn$function)
     (forall (?s (cocone ?s))
     (= (SET.FTN$composition [comediator set.ftn$source])
        (SET.FTN$composition [cocone-arity colimit]))
     (= (SET.FTN$composition [comediator set.ftn$target]) opvertex)
     (SET.FTN$restriction comediator set.col$comediator)

(14) (KIF$function cotupling-cocone)
     (KIF$source cotupling-cocone) arity)
     (KIF$target cotupling-cocone) SET.FTN$partial-function)
     (forall (?a (arity ?a))
         (and (= (SET.FTN$source (cotupling-cocone ?a))
                 (SET.FTN$power [(index ?a) set.ftn$function]))
              (= (SET.FTN$target (cotupling-cocone ?a)) cocone)
              (forall (?f ((SET.FTN$power [(index ?a) set.ftn$function]) ?f))
                  (<=> ((SET.FTN$domain (cotupling-cocone ?a)) ?f)
                       (and (forall (?n ((index ?a) ?n))
                                    (= (set.ftn$source (?f ?n)) ((function ?a) ?n)))
                            (exists (?c (set$set ?c))
```

```
                                (forall (?n ((index ?a) ?n))
                                    (= (set.ftn$target (?f ?n)) ?c))))))))
    (SET.FTN$restriction cotupling-cocone set.col$cotupling-cocone)

(15) (KIF$function cotupling)
     (= (KIF$source cotupling) arity)
     (= (KIF$target cotupling) SET.FTN$partial-function)
     (forall (?a (arity ?a))
         (and (= (SET.FTN$source (cotupling ?a))
                 (SET.FTN$power [(index ?a) set.ftn$function]))
              (= (SET.FTN$target (cotupling ?a)) set.ftn$function)
              (= (SET.FTN$domain (cotupling ?a))
                 (SET.FTN$domain (cotupling-cocone ?a)))
              (forall ?f ((SET.FTN$domain (cotupling ?a)) ?f))
                  (= ((cotupling ?a) ?f)
                     (comediator ((cotupling-cocone ?a) ?f))))))
     (SET.FTN$restriction cotupling set.col$cotupling)
```

○ There is an *indication* function (special for coproducts) (Diagram 6)

$indic(\boldsymbol{A}) : coprd(\boldsymbol{A}) \to index(\boldsymbol{A})$

from the coproduct to the arity index, which is defined by $indic(\boldsymbol{A})(n, x) = n$.

The indication function is the cotupling of the collection of constant index functions

$\{\Delta_{\boldsymbol{A}}(n) : \boldsymbol{A}(n) \to index(\boldsymbol{A}) \mid n \in index(\boldsymbol{A})\}$.

There is a *projection* function (special for coproducts over arities) (Diagram 6)

$proj(\boldsymbol{A}) : coprd(\boldsymbol{A}) \to base(\boldsymbol{A})$

from the coproduct to the arity base, which is defined by $proj(\boldsymbol{A})(n, x) = x$.

The projection function is the cotupling of the collection of inclusion functions

$\{incl_{\boldsymbol{A}}(n) : \boldsymbol{A}(n) \to base(\boldsymbol{A}) \mid n \in index(\boldsymbol{A})\}$.

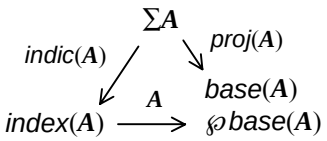


Diagram 6: Colimit, and Indication and Projection Functions of an Arity

```
(16) (KIF$function constant-index)
     (= (KIF$source constant-index) arity)
     (= (KIF$target constant-index) SET.FTN$function)
     (forall (?a (arity ?a))
         (and (= (SET.FTN$source (constant-index ?a)) (index ?a))
              (= (SET.FTN$target (constant-index ?a)) set.ftn$function)
              (forall (?n ((index ?a) ?n))
                  (and (= (set.ftn$source ((constant-index ?a) ?n)) ((function ?a) ?n))
                       (= (set.ftn$target ((constant-index ?a) ?n)) (index ?a))
                       (= ((constant-index ?a) ?n)
                          ((set.ftn$constant [((function ?a) ?n) (index ?a)]) ?n))))))

(17) (SET.FTN$function indication)
     (= (SET.FTN$source indication) arity)
     (= (SET.FTN$target indication) set.ftn$function)
     (forall (?a (arity ?a)
         (and (= (set.ftn$source (indication ?a)) (coproduct ?a))
              (= (set.ftn$target (indication ?a)) (index ?a))
              (= (indication ?a) ((cotupling ?a) (constant-index ?a)))))

(18) (KIF$function inclusion)
     (= (KIF$source inclusion) arity)
     (= (KIF$target inclusion) SET.FTN$function)
     (forall (?a (arity ?a))
         (and (= (SET.FTN$source (inclusion ?a)) (index ?a))
              (= (SET.FTN$target (inclusion ?a)) set.ftn$function)
              (forall (?n ((index ?a) ?n))
                  (and (= (set.ftn$source ((inclusion ?a) ?n)) ((function ?a) ?n))
                       (= (set.ftn$target ((inclusion ?a) ?n)) (base ?a))
                       (= ((inclusion ?a) ?n)
                          (set.ftn$inclusion [((function ?a) ?n) (base ?a)])))))))

(19) (SET.FTN$function projection)
     (= (SET.FTN$source projection) arity)
     (= (SET.FTN$target projection) set.ftn$function)
```

```
(forall (?a (arity ?a))
    (and (= (set.ftn$source (projection ?a)) (coproduct ?a))
         (= (set.ftn$target (projection ?a)) (base ?a))
         (= (projection ?a) ((cotupling ?a) (inclusion ?a)))))
```

Based Coproduct Morphisms

`set.col.art.mor`

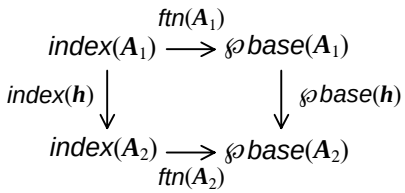


Diagram 7: Arity Morphism

- A *morphism of arities* $\boldsymbol{h} = \langle index(\boldsymbol{h}), base(\boldsymbol{h})\rangle : \boldsymbol{A}_1 \to \boldsymbol{A}_2$ (Diagram 7) consists of
  - an index function $index(\boldsymbol{h}) : index(\boldsymbol{A}_1) \to index(\boldsymbol{A}_2)$ and
  - a base function $base(\boldsymbol{h}) : base(\boldsymbol{A}_1) \to base(\boldsymbol{A}_2)$

  for which Diagram 1 commutes:

  $index(\boldsymbol{h}) \cdot ftn(\boldsymbol{A}_2) = ftn(\boldsymbol{A}_1) \cdot \wp base(\boldsymbol{h})$.

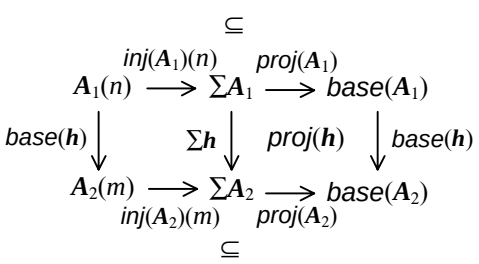
  Pointwise this state that $\wp base(\boldsymbol{h})(ftn(\boldsymbol{A}_1)(n)) = ftn(\boldsymbol{A}_2)(index(\boldsymbol{h})(n))$ for every source index $n \in index(\boldsymbol{A}_1)$.

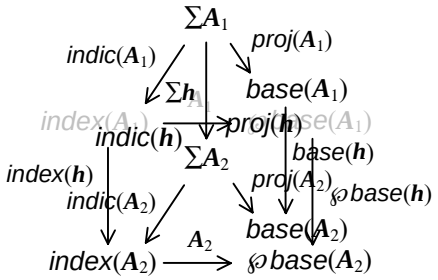
  This implies that for any pair (n, x), where $n \in index(\boldsymbol{A}_1)$ is a source index and $x \in ftn(\boldsymbol{A}_1)(n)$ is an element in the n^{th} arity $ftn(\boldsymbol{A}_1)(n) \subseteq base(\boldsymbol{A}_1)$, the pair $(m, y) = (index(\boldsymbol{h})(n), base(\boldsymbol{h})(x))$ consists of a target index $m \in index(\boldsymbol{A}_2)$ and an element $y \in ftn(\boldsymbol{A}_2)(m)$ in the m^{th} arity $ftn(\boldsymbol{A}_2)(m) \subseteq base(\boldsymbol{A}_2)$.

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

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) arity-morphism)
    (= (SET.FTN$target source) set.col.art$arity)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) arity-morphism)
    (= (SET.FTN$target target) set.col.art$arity)

(4) (SET.FTN$function index)
    (= (SET.FTN$source index) arity-morphism)
    (= (SET.FTN$target index) set.ftn$function)
    (= (SET.FTN$composition [index set.ftn$source])
       (SET.FTN$composition [source set.col.art$index]))
    (= (SET.FTN$composition [index set.ftn$target])
       (SET.FTN$composition [target set.col.art$index]))

(5) (SET.FTN$function base)
    (= (SET.FTN$source base) arity-morphism)
    (= (SET.FTN$target base) set.ftn$function)
    (= (SET.FTN$composition [base set.ftn$source])
       (SET.FTN$composition [source set.col.art$base]))
    (= (SET.FTN$composition [base set.ftn$target])
       (SET.FTN$composition [target set.col.art$base]))

(6) (forall (?h (arity-morphism ?h))
        (= (set.ftn$composition [(index ?h) (set.col.art$function (target ?h))])
           (set.ftn$composition
               [(set.col.art$function (source ?h)) (set.ftn$power (base ?h))])))
```

- Associated with any morphism of arities $\boldsymbol{h}$ is a *quartet* $qtt(\boldsymbol{h})$.

```
(7) (SET.FTN$function quartet)
    (= (SET.FTN$source quartet) arity-morphism)
    (= (SET.FTN$target quartet) set.qtt$quartet)
    (= (SET.FTN$composition [quartet set.qtt$horizontal-source])
       (SET.FTN$composition [source set.col.art$function]))
    (= (SET.FTN$composition [quartet set.qtt$horizontal-target])
       (SET.FTN$composition [target set.col.art$function]))
    (= (SET.FTN$composition [quartet set.qtt$vertical-source]) index)
    (= (SET.FTN$composition [quartet set.qtt$vertical-target])
       (SET.FTN$composition [base set.ftn$power]))
```





○ Any morphism of arities $\boldsymbol{h} = \langle \mathit{index}(\boldsymbol{h}), \mathit{base}(\boldsymbol{h}) \rangle : \boldsymbol{A}_1 \to \boldsymbol{A}_2$ defines a *coproduct* function

$\mathit{coprd}(\boldsymbol{h}) = \sum \boldsymbol{h} : \mathit{coprd}(\boldsymbol{A}_1) = \sum \boldsymbol{A}_1 \to \sum \boldsymbol{A}_2 = \mathit{coprd}(\boldsymbol{A}_2)$.

The pointwise definition is:

$\mathit{coprd}(\boldsymbol{h})((n, x)) = (\mathit{index}(\boldsymbol{h})(n), \mathit{base}(\boldsymbol{h})(x))$

for any index $n \in \mathit{arity}(\boldsymbol{A}_1)$ and element $x \in \boldsymbol{A}_1(n)$. This is well defined since $\boldsymbol{h}$ preserves index arity. The coproduct function is the cotupling of the injection of base restrictions:

$\{\mathit{base}(\boldsymbol{h}) \cdot \mathit{inj}(\boldsymbol{A}_2)(\mathit{index}(\boldsymbol{h})(n)) \mid n \in \mathit{index}(\boldsymbol{A})\}$.

```
(8) (KIF$function base-restriction)
    (= (KIF$source base-restriction) arity-morphism)
    (= (KIF$target base-restriction) SET.FTN$function)
    (forall (?h (arity-morphism ?h))
        (and (= (SET.FTN$source (base-restriction ?h)) (set.col.art$index (source ?h)))
             (= (SET.FTN$target (base-restriction ?h)) set.ftn$function)
             (forall (?n ((set.col.art$index (source ?h)) ?n))
                 (and (= (set.ftn$source ((base-restriction ?h) ?n))
                         ((set.col.art$function (source ?h)) ?n))
                      (= (set.ftn$target ((base-restriction ?h) ?n))
                         ((set.col.art$function (target ?h)) ((index ?h) ?n)))
                      (set.ftn$restriction ((base-restriction ?h) ?n) (base ?h)))))))

(9) (KIF$function coproduct-tuple)
    (= (KIF$source coproduct-tuple) arity-morphism)
    (= (KIF$target coproduct-tuple) SET.FTN$function)
    (forall (?h (arity-morphism ?h))
        (and (= (SET.FTN$source (coproduct-tuple ?h)) (set.col.art$index (source ?h)))
             (= (SET.FTN$target (coproduct-tuple ?h)) set.ftn$function)
             (forall (?n ((set.col.art$index (source ?h)) ?n))
                 (and (= (set.ftn$source ((coproduct-tuple ?h) ?n))
                         ((set.col.art$function (source ?h)) ?n))
                      (= (set.ftn$target ((coproduct-tuple ?h) ?n))
                         (set.col.art$coproduct (target ?h)))
                      (= ((coproduct-tuple ?h) ?n)
                         (set.ftn$composition
                             [(base-restriction ?h) ?n)
                              ((set.col.art$injection (target ?h)) ((index ?h) ?n))])))))))

(10) (SET.FTN$function coproduct)
     (= (SET.FTN$source coproduct) arity-morphism)
     (= (SET.FTN$target coproduct) set.ftn$function)
     (forall (?h (arity-morphism ?h))
         (and (= (set.ftn$source (coproduct ?h)) (set.col.art$coproduct (source ?h)))
              (= (set.ftn$target (coproduct ?d)) (set.col.art$coproduct (target ?h)))
              (= (coproduct ?h) ((cotupling (source ?h)) (coproduct-tuple ?h))))))
```

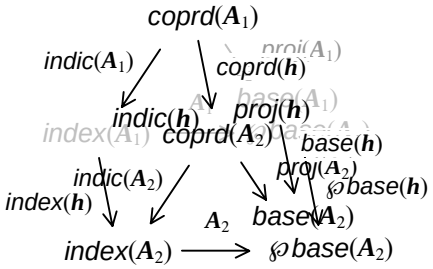


Figure 1: Coproduct of the Diagram of an Arity Morphism

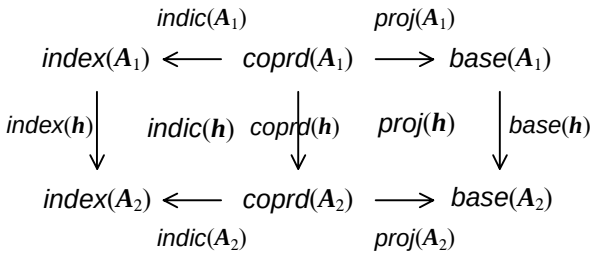


Figure 2: Spangraph Morphism of a Arity Morphism

- For any morphism of arities $\boldsymbol{h} = \langle \mathit{arity}(\boldsymbol{h}), \mathit{base}(\boldsymbol{h}) \rangle : \boldsymbol{A}_1 \to \boldsymbol{A}_2$ the coproduct function is the vertical source for two quartets: an *indication* quartet $\mathit{indic}(\boldsymbol{h})$ and a *projection* quartet $\mathit{proj}(\boldsymbol{h})$.
  - The commutativity (preservation of indication)

    $$\mathit{coprd}(\boldsymbol{h}) \cdot \mathit{indic}(\boldsymbol{A}_2) = \mathit{indic}(\boldsymbol{A}_1) \cdot \mathit{index}(\boldsymbol{h}),$$

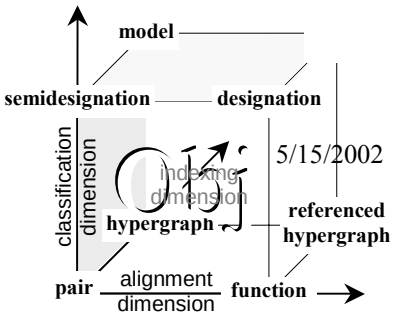
    is a property of the coproduct of arities. This is obvious from the pointwise definition of the coproduct function. This is provable using coproduct cotupling.
  - The commutativity (preservation of projection)

    $$\mathit{coprd}(\boldsymbol{h}) \cdot \mathit{proj}(\boldsymbol{A}_2) = \mathit{proj}(\boldsymbol{A}_1) \cdot \mathit{base}(\boldsymbol{f}),$$

    is a property of the coproduct of arities. This is obvious from the pointwise definition of the coproduct function. This is provable using coproduct cotupling.

  By the preservation of index arity, $\wp\mathit{base}(\boldsymbol{h})(\boldsymbol{A}_1(n)) = \boldsymbol{A}_2(\mathit{index}(\boldsymbol{h})(n))$ for every source index $n \in \mathit{index}(\boldsymbol{A}_1)$, the index quartet is a fibration: for any index $n \in \mathit{index}(\boldsymbol{A}_1)$ and any element $y \in \boldsymbol{A}_2(\mathit{index}(\boldsymbol{h})(n))$ there is an element $x \in \boldsymbol{A}_1(n)$ such that $\mathit{base}(\boldsymbol{h})(x) = y$.

```
(11) (SET.FTN$function indication)
     (= (SET.FTN$source indication) arity-morphism)
     (= (SET.FTN$target indication) set.qtt$fibration)
     (forall (?h (arity-morphism ?h))
         (and (= (set.qtt$horizontal-source (indication ?h))
                 (set.col.art$indication (source ?h)))
              (= (set.qtt$horizontal-target (indication ?h))
                 (set.col.art$index (target ?h)))
              (= (set.qtt$vertical-source (indication ?h)) (coproduct ?h))
              (= (set.qtt$vertical-target (indication ?h)) (index ?h))))

(12) (SET.FTN$function projection)
     (= (SET.FTN$source projection) arity-morphism)
     (= (SET.FTN$target projection) set.qtt$quartet)
     (forall (?h (arity-morphism ?h))
         (and (= (set.qtt$horizontal-source (projection ?h))
                 (set.col.art$projection (source ?h)))
              (= (set.qtt$horizontal-target (projection ?h))
                 (set.col.art$projection (target ?h)))
              (= (set.qtt$vertical-source (projection ?h)) (coproduct ?h))
              (= (set.qtt$vertical-target (projection ?h)) (base ?h))))
```



Set Pairs

`set.pr`

- A *set pair* $A = \langle A_1, A_2 \rangle = \langle \mathit{set1}(A), \mathit{set2}(A) \rangle$ is two sets.

```
(1) (SET$class pair)

(2) (SET.FTN$function set1)
    (= (SET.FTN$source set1) pair)
    (= (SET.FTN$target set1) set$set)

(3) (SET.FTN$function set2)
    (= (SET.FTN$source set2) pair)
    (= (SET.FTN$target set2) set$set)
```

> Set pairs form the object class of the $\triangle$Set category.

- Any set A defines a *diagonal* set pair $\mathit{diag}(A) = \langle A, A \rangle$.

```
(4) (SET.FTN$function diagonal)
    (= (SET.FTN$source diagonal) set$set)
    (= (SET.FTN$target diagonal) pair)
    (= (SET.FTN$composition [diagonal set1]) (SET.FTN$identity set$set))
    (= (SET.FTN$composition [diagonal set2]) (SET.FTN$identity set$set))
```

- Any set A defines a *terminal* set pair $\mathit{term}(A) = \langle A, 1 \rangle$.

```
(5) (SET.FTN$function terminal)
    (= (SET.FTN$source terminal) set$set)
    (= (SET.FTN$target terminal) pair)
    (= (SET.FTN$composition [terminal set1]) (SET.FTN$identity set$set))
    (= (SET.FTN$composition [terminal set2])
       (SET.FTN$constant [set$set set$set] set.lim$terminal))
```

- Set pairs are used to define tuple sets analogous to how functions define signature sets. For any pair $A = \langle A_1, A_2 \rangle$, a *tuple* $t : \mathit{arity}(t) \to A_2$ of A is a tuple of A_2 elements, where the arity of a tuple (its source) is a subset $\mathit{arity}(t) \subseteq A_1$. Let $\mathit{tuple}(A)$ denote the class of all tuples of A. This is the disjoint union of all the exponent sets for a subset of the first set and the second set $\mathit{tuple}(A) = \sum_{X \in \wp A1} \mathit{exp}(X, A_2)$. Since sets are closed under the exponent, power and coproduct operators, this is a set; that is, tuple sets are small. The tuple inclusion maps the set of tuples of a pair into the class of functions.

```
(4) (SET.FTN$function tuple)
    (= (SET.FTN$source tuple) pair)
    (= (SET.FTN$target tuple) set$set)
    (forall (?a (pair ?a))
        (and (SET$subclass (tuple ?a) set.ftn$function)
             (forall (?t (set.ftn$function ?t))
                 (<=> ((tuple ?a) ?t)
                      (and (= (set.ftn$target ?t) (set2 ?a))
                           (set$subset (set.ftn$source ?t) (set1 ?a)))))))

(5) (KIF$function tuple-inclusion)
    (= (KIF$source tuple-inclusion) pair)
    (= (KIF$target tuple-inclusion) SET.FTN$function)
    (forall (?a (pair ?a))
        (and (= (SET.FTN$source (tuple-inclusion ?a)) (tuple ?a))
             (= (SET.FTN$source (tuple-inclusion ?a)) set.ftn$function)
             (= (tuple-inclusion ?a)
                (SET.FTN$inclusion [(tuple ?a) set.ftn$function])))))
```

- One tuple $t_1 \in \mathit{tuple}(A)$ is more *specialized* (more specific) that another tuple $t_2 \in \mathit{tuple}(A)$, in other words t_2 is more *generalized* (more generic) that t_1, symbolized by $t_1 \leq t_2$, when $\mathit{arity}(t_1) \supseteq \mathit{arity}(t_2)$ and, as a function, t_2 is a restriction of t_1: $t_2(x) = t_1(x)$ for all $x \in \mathit{arity}(t_2)$. The *tuple order* $\mathit{tuple\text{-}ord}(A) = \langle \mathit{tuple}(A), \leq \rangle$ consists of the set of tuples with this specialization-generalization order.

```
(6) (SET.FTN$function tuple-order)
    (= (SET.FTN$source tuple-order) pair)
    (= (SET.FTN$target tuple-order) ord$partial-order)
    (forall (?a (pair ?a))
        (and (= (ord$set (tuple-order ?a)) (tuple ?a))
             (forall (?t1 ((tuple ?a) ?t1)
                      ?t2 ((tuple ?a) ?t2))
                 (<=> ((tuple-order ?a) ?t1 ?t2)
                      (and (set$subset (set.ftn$arity ?t2) (set.ftn$arity ?t1))
                           (set.ftn$restriction ?t2 ?t1)))))))
```

The tuple function associated with any pair is the image of the object function of the tuple functor applied to the pair:

$$tuple : \triangle \mathrm{Set} \to \mathrm{Set}.$$

- We rename the arity function, relating the source and target to the base pair. The *tuple arity* function for pairs $\#_A = tuple\text{-}arity : tuple(A) \to \wp set1(A)$ is compatible with ordinary tuple arity. Unlike the signature set defined later, there is no function inverse to arity. Thus, when you apply arity you lose information – the tuple set is more important than the power of the first component.

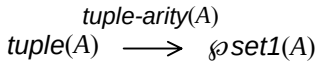


Figure 1: Arity

```
(7) (SET.FTN$function tuple-arity)
    (= (SET.FTN$source tuple-arity) pair)
    (= (SET.FTN$target tuple-arity) set.ftn$function)
    (= (SET.FTN$composition [tuple-arity set.ftn$source]) tuple)
    (= (SET.FTN$composition [tuple-arity set.ftn$target])
       (SET.FTN$composition [set1 set$power]))
    (forall (?a (pair ?a))
        (= (tuple-arity ?a)
           (set.ftn$composition [(tuple-inclusion ?a) set.ftn$arity])))
```

- Any set pair $A = \langle A_1, A_2 \rangle = \langle set1(A), set2(A) \rangle$ defines a *hypergraph* (Figure 3), whose reference pair is the pair A itself and whose tuple function is the identity function $id : tuple(A) \to tuple(A)$ on the set of tuples of A. This hypergraph has the first component set A_1 as it set of names, the second component set A_2 as it set of nodes, and the tuple set $tuple(A)$ as its set of edges. Clearly, the reference pair of the hypergraph of a pair is itself.

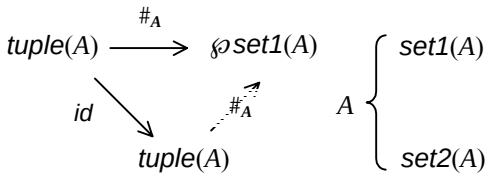


Figure 3: Hypergraph of a pair

```
(8) (SET.FTN$function hypergraph)
    (= (SET.FTN$source hypergraph) pair)
    (= (SET.FTN$target hypergraph) hgph$hypergraph)
    (forall (?a (pair ?a))
        (and (= (hgph$reference (hypergraph ?a)) ?a)
             (= (hgph$tuple (hypergraph ?a)) (set.ftn$identity (tuple ?a)))))
```

- For any set pair $A = \langle A_1, A_2 \rangle$ the coproduct *arity* $\#_A = arity(A) : tuple(A) \to \wp set1(A)$ is compatible with ordinary tuple arity. Unlike the signature set defined later, there is no function inverse to arity. Thus, when you apply arity you lose information – the tuple set is more important than the power of the first component.

```
(9) (SET.FTN$function arity)
    (= (SET.FTN$source arity) pair)
    (= (SET.FTN$target arity) set.col.art$arity)
    (= (SET.FTN$composition [arity set.col.art$index]) tuple)
    (= (SET.FTN$composition [arity set.col.art$base]) set1)
    (= (SET.FTN$composition [arity set.col.art$function]) tuple-arity)
```

○ In studies of natural language, the syntactic position of the argument of a verb correlates significantly with the meaning of the argument as a participant in either the action of the verb or the state indicated by the verb. These semantic relations between the verb and its arguments are called thematic roles or case relations. In frame systems, they are represented by slots in the frame for the corresponding verb. In the IFF Model Theory Ontology verbs correspond to relations. But relation types denote signatures or tuples in general. Hence, we can use this notion in the general discussion of tuples.

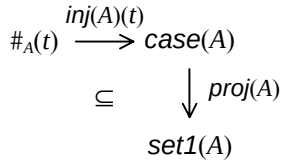


Diagram 3: Case

Any set pair A has a set of *thematic roles* or *case relations*

$$\mathit{case}(A) = \sum \mathit{arity}(A)$$
$$= \sum_{t \in \mathit{tuple}(A)} \mathit{arity}(A)(t)$$
$$= \{(t, x) \mid t \in \mathit{tuple}(A), x \in \mathit{arity}(A)(t)\},$$

which is the coproduct of its arity. In terms of frames, elements of $\mathit{case}(A)$ can be regarded as frame-slot pairs. The set $\mathit{case}(A)$ is rather large; this large cardinality will be handled properly with hypergraphs by adding a denotation or indexing function. In addition, any set pair A and any tuple $t \in \mathit{tuple}(A)$ define a case *injection* function:

$$\mathit{inj}(A)(t) : \#_A(t) = \mathit{arity}(A)(t) \to \mathit{case}(A).$$

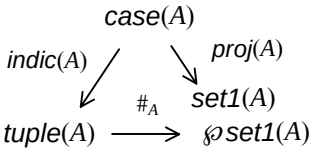


Figure 2: Coproduct of the Diagram of a Set Pair

This is defined by $\mathit{inj}(A)(t)(x) = (t, x)$ for all tuples (nexus) $t \in \mathit{tuple}(A)$ and all names (prehensions) $x \in \mathit{arity}(A)(t)$. Obviously, the injections are injective.

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

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

○ Any set pair A defines case *indication* and *projection* functions based on its arity:

$$\mathit{indic}(A) : \mathit{case}(A) \to \mathit{tuple}(A),$$
$$\mathit{proj}(A) : \mathit{case}(A) \to \mathit{set1}(A).$$

These are defined by

$$\mathit{indic}(A)((t, x)) = t \text{ and } \mathit{proj}(A)((t, x)) = x$$

for all tuples (nexus) $t \in \mathit{tuple}(A)$ and all names (prehensions) $x \in \mathit{arity}(A)(t)$. The injections commute (Diagram 3) with projection and inclusion.

```
(12) (SET.FTN$function indication)
     (= (SET.FTN$source indication) pair)
     (= (SET.FTN$target indication) set.ftn$function)
     (= indication (SET.FTN$composition [arity set.col.art$indication]))

(13) (SET.FTN$function projection)
     (= (SET.FTN$source projection) pair)
     (= (SET.FTN$target projection) set.ftn$function)
     (= projection (SET.FTN$composition [arity set.col.art$projection]))
```

○ Any set pair A defines a *comediator* function:

$$\tilde{*}_A = \mathit{comed}(A) : \mathit{case}(A) \to \mathit{set2}(A).$$

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

$$\mathit{comed}(A)((t, x)) = t(x)$$

for all tuples $t \in tuple(A)$ and all names $x \in arity(A)(t)$. Abstractly, it is the comediator – hence the name – of the coproduct cotuple (cocone) consisting of the collection of tuples regarded as functions,

$\{t : arity(A)(t) \to set2(A) \mid t \in tuple(A)\}$.

The comediator function commutes (Diagram 4) with the injection function and the tuple itself. If the indexed names in $case(A)$ are viewed as roles (prehensions), the comediator is a reference function from roles to objects (actualities).

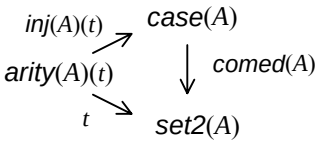


Diagram 4: Comediator

```
(14) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) pair)
     (= (SET.FTN$target comediator) set.ftn$function)
     (forall (?a (pair ?a))
         (and (= (set.ftn$source (comediator ?a)) (case ?a))
              (= (set.ftn$target (comediator ?a)) (set2 ?a))
              (= (comediator ?a)
                 ((set.col.art$cotupling (arity ?a)) (tuple-inclusion ?a)))))
```

$$set1(A) \quad set2(A) \quad \Rightarrow \quad case(A) \xrightarrow{comed(A)} set2(A)$$

The comediator function associated with any set pair is the image of the object function of the quartet functor applied to the set pair:

$qtt : \triangle\mathrm{Set} \to \square\mathrm{Set}$.

nexus prehensions actualities

$$tuple(A) \xleftarrow{indic(A)} case(A) \xrightarrow{comed(A)} set2(A)$$

Figure 2: Spangraph of a Set Pair

- Associated with any set pair A is a spangraph

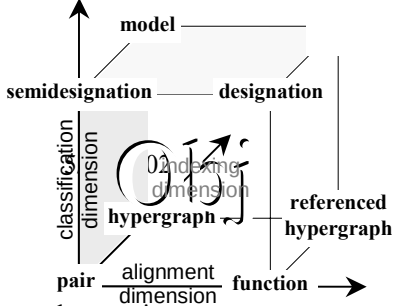
  $sgph(A) = \langle 1^{st}_{sgph(A)}, 2^{nd}_{sgph(A)} \rangle$

  consisting of the two functions $1^{st}_{sgph(A)}(A) = indic(A)$ and $2^{nd}_{sgph(A)}(A) = refer(A)$.

```
(15) (SET.FTN$function spangraph)
     (= (SET.FTN$source spangraph) pair)
     (= (SET.FTN$target spangraph) sgph$spangraph)
     (forall (?a (pair ?a))
         (and (= (sgph$set1 (spangraph ?a)) (tuple ?a))
              (= (sgph$set2 (spangraph ?a)) (set2 ?a))
              (= (sgph$vertex (spangraph ?a)) (case ?a))
              (= (sgph$first (spangraph ?a)) (indication ?a))
              (= (sgph$second (spangraph ?a)) (comediator ?a))))
```

The spangraph associated with any set pair is the image of the object function of the spangraph functor applied to the set pair:

$sgph : \triangle\mathrm{Set} \to \mathrm{Spangraph}$.



Set Functions

`set.ftn`

- For any set A, a *tuple* t of A-elements is a collection of (possibly duplicate) A-elements that are indexed. The indexing elements of t form a set (of indices). More precisely, a tuple of type $\mathit{type}(t)$ indexed by the set $\mathit{arity}(t)$ is a map (function) $t : \mathit{arity}(t) \to \mathit{type}(t)$. We use the exponential notation $t \in \mathit{type}(t)^{\mathit{arity}(t)}$ for tuples.

```
(1) (SET$class tuple)
    (= tuple function)

(2) (SET.FTN$function arity)
    (= (SET.FTN$source arity) tuple)
    (= (SET.FTN$target arity) set$set)
    (= arity source)

(3) (SET.FTN$function type)
    (= (SET.FTN$source type) tuple)
    (= (SET.FTN$target type) set$set)
    (= type target)
```

- The fiber of tuples $t \in \mathit{tuple}(A)$ of the same type A is defined.

```
(4) (SET.FTN$function type-fiber)
    (= (SET.FTN$source type-fiber) set)
    (= (SET.FTN$target type-fiber) (SET$power tuple))
    (= type-fiber (SET.FTN$fiber type))
```

- A function $f : A \to \wp B$ that maps to a power set is called a *set-valued* function with *base B*.

```
(5) (SET.FTN$class set-valued)
    (SET$subclass set-valued function)

(6) (SET.FTN$function base)
    (= (SET.FTN$source base) function)
    (= (SET.FTN$target base) set$set)
    (forall (?f (set-valued ?f))
        (= (target ?f) (set$power (base ?f))))
```

-

- Any set-valued function $f : A \to \wp B$ that maps to a power set, there is an *induced* classification

  $\mathit{induc}(f) = \langle B, A, \models_f \rangle$,

  whose extent function is f. This passage, between set-valued functions and classifications, is bijective.

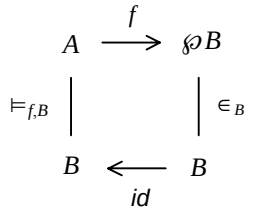


Figure 3: Induced Classification

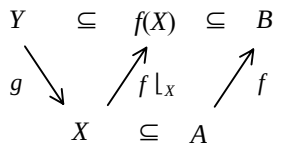
```
(7) (SET.FTN$function induction)
    (= (SET.FTN$source induction) set-valued)
    (= (SET.FTN$target induction) cls$classification)
    (forall (?f (set-valued ?f))
        (and (= (cls$instance (induction ?f)) (source ?f))
             (= (cls$type (induction ?f)) (base ?f))
             (forall (?b ((base ?f) ?b) ?a ((source ?f) ?a))
                 (<=> ((induction ?f) ?a ?b)
                      ((?f ?a) ?b)))))

    (forall (?f (set-valued ?f))
        (= (cls$extent (induction ?f)) ?f))

    (forall (?c (cls$classification ?a))
        (= (induction (cls$extent ?a)) ?a))
```

- Any function $f : A \to B$ can be regarded as a *classification* $\mathit{cla}(f) = \langle A, B, f \rangle$, where

$a \models_{cla(f)} b$ iff $f(a) = b$.

```
(1) (SET.FTN$function classification)
    (= (SET.FTN$source classification) function)
    (= (SET.FTN$target classification) cls$classification)
    (= (SET.FTN$composition [classification cls$instance]) source)
    (= (SET.FTN$composition [classification cls$type]) target)
    (forall (?a ((source ?f) ?a) ?b ((source ?f) ?b))
        (<=> ((classification ?f) ?a ?b)
             (= (?f ?a) ?b)))
```

- A *section* $g : B \to A$ of a function $f : A \to B$ is a left inverse $g \cdot f = id_B$ (on the whole target set B); a *local section* $g : Y \to X$ where $X \subseteq A$ and $Y \subseteq B$ is a left inverse on part of the target set $g \cdot f \lfloor_X = incl_{Y, f(C)}$. Local sections are injections. The classification of local *section*s of a function $cla(f) = \langle A, B, f \rangle$ is the power classification $\wp cla(f) = \langle \wp A, \wp B, \models \rangle$, whose instances are subsets of A, whose types are subsets of B, and whose incidence is the substitution of source elements into target elements: an subset instance $X \subseteq A$ has an subset type $Y \subseteq B$, denoted $X \models_{\wp cla(f)} Y$, when there exists a function $g : Y \to X$ such that $g(y) \models_{cla(f)} y$ for all $y \in Y$. By the definition of the function classification this means that $f(g(y)) = x$ for all $y \in Y$; or that $g \cdot f \lfloor_X = incl_{Y, f(X)}$; that is, that g is a local section of f.




```
(2) (SET.FTN$function section)
    (= (SET.FTN$source section) function)
    (= (SET.FTN$target section) cls$classification)
    (= (SET.FTN$composition [section cls$instance])
       (SET.FTN$composition [source set$power]))
    (= (SET.FTN$composition [section cls$type])
       (SET.FTN$composition [target set$power]))
    (= section (SET.FTN$composition [classification cls$power]))
```

- Associated with each function $f : A \to B$ is a *vertical identity* quartet $id(f)$.

```
(3) (SET.FTN$function vertical-identity)
    (= (SET.FTN$source vertical-identity) function)
    (= (SET.FTN$target vertical-identity) set.qtt$quartet)
    (= (SET.FTN$composition [vertical-identity set.qtt$horizontal-source])
       (SET.FTN$composition [source identity)]))
    (= (SET.FTN$composition [vertical-identity set.qtt$horizontal-target])
       (SET.FTN$composition [target identity)]))
    (= (SET.FTN$composition [vertical-identity set.qtt$vertical-source])
       (SET.FTN$identity function))
    (= (SET.FTN$composition [vertical-identity set.qtt$vertical-target])
       (SET.FTN$identity function))
```

- Associated with each function $f : A \to B$ is a *semi-identity* quartet $sid(f)$.

```
(4) (SET.FTN$function semi-identity)
    (= (SET.FTN$source semi-identity) function)
    (= (SET.FTN$target semi-identity) set.qtt$quartet)
    (= (SET.FTN$composition [semi-identity set.qtt$horizontal-source])
       (SET.FTN$identity function))
    (= (SET.FTN$composition [semi-identity set.qtt$horizontal-target])
       (SET.FTN$composition [target identity)]))
    (= (SET.FTN$composition [semi-identity set.qtt$vertical-source])
       (SET.FTN$identity function))
    (= (SET.FTN$composition [semi-identity set.qtt$vertical-target])
       (SET.FTN$composition [target identity)]))
```

- Any function $f : A \to B$ has an associated set pair $pr(f) = \langle A, B \rangle$.

```
(4) (SET.FTN$function pair)
    (= (SET.FTN$source pair) function)
    (= (SET.FTN$target pair) set.pr$pair)
    (= (SET.FTN$composition [pair set.pr$set1]) source)
    (= (SET.FTN$composition [pair set.pr$set2]) target)
```

- The *designation* function maps any function $f : A \to B$ to its associated identity designation

$dsgn(f) = \langle f, f \rangle : cls(A) \rightrightarrows cls(B)$.

```
(5) (SET.FTN$function designation)
    (= (SET.FTN$source designation) function)
    (= (SET.FTN$target designation) dsgn$designation)
    (= (SET.FTN$composition [designation dsgn$sign])
       (SET.FTN$composition [source set$classification]))
    (= (SET.FTN$composition [designation dsgn$object])
       (SET.FTN$composition [target set$classification]))
    (= (SET.FTN$composition [designation dsgn$instance]) (SET.FTN$identity function))
    (= (SET.FTN$composition [designation dsgn$type]) (SET.FTN$identity function))
```

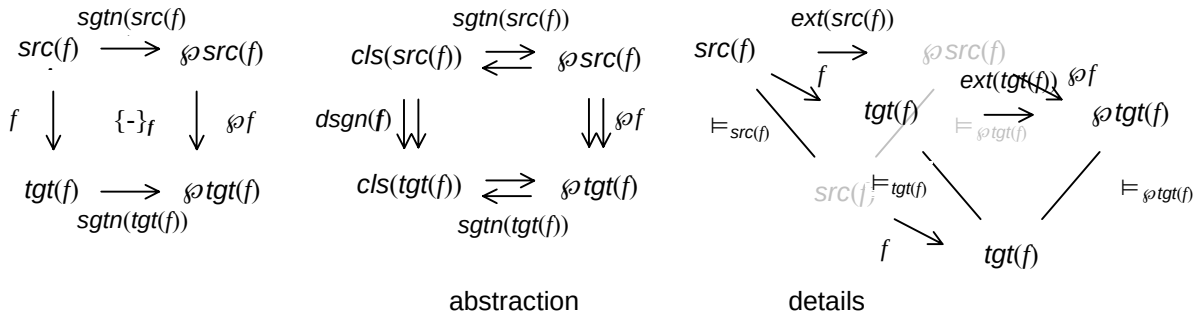


Figure 1: Singleton Function square

- The extent infomorphism of the identity classification associated with a set has the singleton function for that set as its type function. The singleton functions associated with the source and target of a function $f : A \to B$, define a quartet $\{\text{-}\}_f$.

```
(6) (SET.FTN$function singleton)
    (= (SET.FTN$source singleton) function)
    (= (SET.FTN$target singleton) set.qtt$quartet)
    (= (SET.FTN$composition [singleton set.qtt$horizontal-source])
       (SET.FTN$identity function))
    (= (SET.FTN$composition [singleton set.qtt$horizontal-target]) power)
    (= (SET.FTN$composition [singleton set.qtt$vertical-source])
       (SET.FTN$composition [source set$singleton]))
    (= (SET.FTN$composition [singleton set.qtt$vertical-target])
       (SET.FTN$composition [target set$singleton]))
```

- The singleton quartet of a function is the type aspect of the morphism of the designation of the function.

```
(forall (?f (function ?f))
    (= (singleton ?f)
       (dsgn.mor$type (dsgn$eta (designation ?f)))))
```

- The *inclusion* function maps the extent of the subset relation, the class of all pairs sets that are ordered by subset, to the corresponding inclusion functions.

```
(7) (SET.FTN$function inclusion)
    (= (SET.FTN$source inclusion) (REL$extent subset))
    (= (SET.FTN$target inclusion) function)
    (restriction inclusion SET.FTN$inclusion)
```

- The *power inclusion* function maps the power of a set, the set of all subsets, to the corresponding inclusion functions.

```
(8) (SET.FTN$function power-inclusion)
    (= (SET.FTN$source power-inclusion) set$set)
    (= (SET.FTN$target power-inclusion) function)
    (forall (?a (set$set ?a))
        (and (= (source (power-inclusion ?a)) (set$power ?a))
             (= (target (power-inclusion ?a)) (set.pr$tuple (set.pr$diagonal ?a)))
```

```
        (forall (?b (set$subset ?b ?a))
            (= ((power-inclusion ?a) ?b) (inclusion [?b ?a])))))
```

○ For any function $f : A \to B$, a *signature* $t : \mathit{src}(t) \to B$ of f is a tuple of B elements, where the source is a subset $\mathit{src}(t) \subseteq \mathit{src}(f)$ and t is a restriction of f to $\mathit{src}(t)$:

$t = \mathit{incl}_{\mathit{src}(t),\, \mathit{src}(f)} \cdot f$.

Let $\mathit{signature}(f)$ denote the class of all signatures of f. This is a subset of the tuple set for the pair of the function $\mathit{signature}(f) \subseteq \mathit{tuple}(\mathit{pair}(f))$ – this inclusion is represented by a *tuple* function. Since sets are closed under the exponent, power and coproduct operators, this is a set; that is, signature sets are small. The signature inclusion maps the set of signatures of a function into the class of functions.

```
(7) (SET.FTN$function signature)
    (= (SET.FTN$source signature) function)
    (= (SET.FTN$target signature) set$set)
    (forall (?f (function ?f))
        (and (SET$subclass (signature ?f) (set.pr$tuple (pair ?f)))
             (forall (?t (function ?t))
                 (<=> ((signature ?f) ?t)
                      (and (= (target ?t) (target ?f))
                           (set$subset (source ?t) (source ?f))
                           (restriction ?t ?f))))))

(8) (SET.FTN$function signature-tuple)
    (= (SET.FTN$source signature-tuple) function)
    (= (SET.FTN$target signature-tuple) function)
    (forall (?f (function ?f))
        (and (= (set.ftn$source (signature-tuple ?f)) (signature ?f))
             (= (set.ftn$target (signature-tuple ?f)) (set.pr$tuple (pair ?f)))
             (= (signature-tuple ?f)
                (set.ftn$inclusion [(signature ?f) (set.pr$tuple (pair ?f))])))))

(9) (KIF$function signature-inclusion)
    (= (KIF$source signature-inclusion) function)
    (= (KIF$target signature-inclusion) SET.FTN$function)
    (forall (?f (function ?f))
        (= (signature-inclusion ?f)
           (SET.FTN$composition
               [(signature-tuple ?f) (set.pr$tuple-inclusion (pair ?f))]))))
```

o One signature $t_1 \in \mathit{tuple}(A)$ is more specialized (more specific) than another signature $t_2 \in \mathit{tuple}(A)$, symbolized by $t_1 \leq t_2$, when $\mathit{arity}(t_1) \supseteq \mathit{arity}(t_2)$ and, as a function, t_2 is a restriction of t_1: $t_2(x) = t_1(x)$ for all $x \in \mathit{arity}(t_2)$. But, since arities are equivalent to signatures, we only need to used the arity inclusion. The *signature order* $\langle \mathit{sign}(A), \leq \rangle$ consists of the set of signatures with this order.

```
(9) (SET.FTN$function signature-order)
    (= (SET.FTN$source signature-order) function)
    (= (SET.FTN$target signature-order) ord$partial-order)
    (forall (?f (function ?f))
        (and (= (ord$set (signature-order ?f)) (signature ?a))
             (forall (?t1 ((signature ?f) ?t1)
                      ?t2 ((signature ?f) ?t2))
                 (<=> ((signature-order ?f) ?t1 ?t2)
                      (set$subset (set.ftn$arity ?t2) (set.ftn$arity ?t1)))))))
```

The signature function associated with any function is the image of the object function of the signature functor applied to the function:

$\mathit{signature} : \square\mathsf{Set} \to \mathsf{Set}$.

○ Any function $f : \mathit{src}(f) \to \mathit{tgt}(f)$ defines a *type language* $\mathit{lang}(f)$ (Figure 1), whose type function is the function f itself and whose signature function is the identity function $\mathit{id} : \mathit{sign}(f) \to \mathit{sign}(f)$ on the set of signatures of f. This lan-

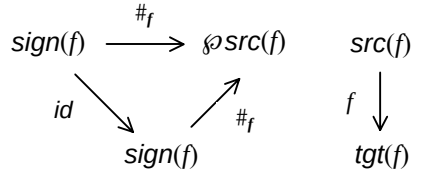


Figure 1: Hypergraph of a function

guage has the source set *src*(*f*) as it set of variables, the target set *tgt*(*f*) as it set of entity types, and the signature set *sign*(*f*) as its set of relation types. Clearly, the type function of the language of a function is itself: *type*(*lang*(*f*)) = *f*.

```
(10) (SET.FTN$function language)
     (= (SET.FTN$source language) function)
     (= (SET.FTN$target language) lang$language)
     (forall (?f (function ?f))
         (and (= (lang$type (language ?f)) ?f)
              (= (lang$signature (language ?f))
                 (set.ftn$identity (signature ?f)))))
```

- We rename the *signature arity* function, relating the source and target to the base function. Given the function $f : src(f) \to tgt(f)$, any signature is determined by its arity. This fact is realized by an *assign*ment function $sign\text{-}assign(f) : \wp src(f) \to sign(f)$ from the power of the source of *f* to the signature of *f*. The signature arity function is compatible (Figure 1) with tuple arity.

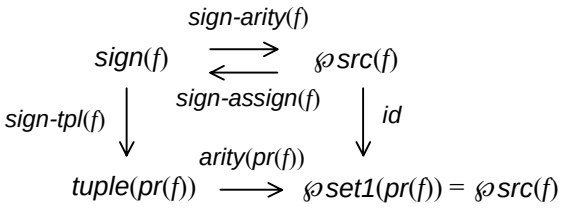


Figure 1: Assign & Arity

```
(11) (SET.FTN$function signature-arity)
     (= (SET.FTN$source signature-arity) function)
     (= (SET.FTN$target signature-arity) function)
     (= (SET.FTN$composition [signature-arity source]) signature)
     (= (SET.FTN$composition [signature-arity target])
        (SET.FTN$composition [source set$power]))
     (forall (?f (function ?f)
              ?t ((signature ?f) ?t))
         (= ((signature-arity ?f) ?t) (set$arity ?t)))

     (SET.FTN$function signature-assign)
     (= (SET.FTN$source signature-assign) function)
     (= (SET.FTN$target signature-assign) function)
     (= (SET.FTN$composition [signature-assign source])
        (SET.FTN$composition [source set$power]))
     (= (SET.FTN$composition [signature-assign target]) signature)
     (forall (?f (function ?f)
              ?X (set$subset ?X (source ?f)))
         (= ((signature-assign ?f) ?X)
            (set.ftn$composition [(inclusion [?X (source ?f)]) ?f])))

     (forall (?f (function ?f))
         (= (set.ftn$composition [(signature-tuple ?f) (set.pr$arity (pair ?f))])
            (signature-arity ?f)))
```

- The assignment function is the inverse of the arity function.

```
     (forall (?f (function ?f))
         (and (= (composition [(signature-arity ?f) (signature-assign ?f)])
                 (set.ftn$identity (signature ?f)))
              (= (composition [(signature-assign ?f) (signature-arity ?f)])
                 (set.ftn$identity (set$power (source ?f))))))
```

-

○ For any function $f : src(f) \to tgt(f)$ the coproduct *arity* $\#_f = arity(f) : sign(f) \to \wp src(f)$ is compatible with ordinary tuple arity.

```
(12) (SET.FTN$function arity)
     (= (SET.FTN$source arity) function)
     (= (SET.FTN$target arity) set.col.art$arity)
     (= (SET.FTN$composition [arity set.col.art$index]) signature)
     (= (SET.FTN$composition [arity set.col.art$base]) source)
     (= (SET.FTN$composition [arity set.col.art$function]) signature-arity)
```

○ Any function $f : src(f) \to tgt(f)$ has a set of *thematic roles* or *case relations*

$$case(f) = \sum arity(f)$$
$$= \sum_{t \in signature(f)} arity(f)(t)$$
$$= \{(t, x) \mid t \in sign(f), x \in arity(f)(t)\},$$

which is the coproduct of its arity. In terms of frames, elements of $case(f)$ can be regarded as frame-slot pairs. The set $case(f)$ is rather large; this large cardinality will be handled properly with hypergraphs by adding a denotation or indexing function. For any function $f : src(f) \to tgt(f)$ and any signature $t \in sign(f)$ there is a case *injection* function

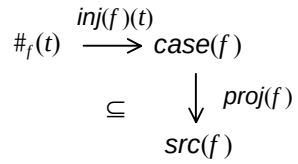


Diagram 3: Coproduct

$$inj(f)(t) : \#_f(t) = arity(f)(t) \to case(f),$$

defined by $inj(f)(t)(x) = (t, x)$ for all signatures (nexus) $t \in sign(f)$ and all names (prehensions) $x \in arity(f)(t)$. Obviously, the injections are injective.

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

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

○ Any function $f : src(f) \to tgt(f)$ defines case *indication* (*nexus*) and case *projection* (*prehension*) functions based on its arity:

$$indic(f) : case(f) \to sign(f),$$
$$proj(f) : case(f) \to src(f).$$

These are defined by

$$indic(f)((t, x)) = t \text{ and } proj(f)((t, x)) = x$$

for all signatures (nexus) $t \in sign(f)$ and all names (prehensions) $x \in arity(f)(t)$. The injections commute (Diagram 3) with projection and inclusion.

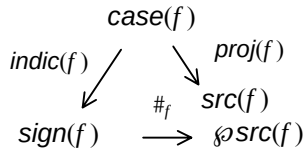


Figure 2: Coproduct and the index and projection functions

The fiber-index function $fib(indic(f)) : sign(f) \to \wp case(f)$ is defined by

$$fib(indic(f))(t) = \{(t, x) \mid x \in arity(f)(t)\} \subseteq case(f)$$

for all signatures $t \in sign(f)$. And of course the collection $\{fib(indic(f))(t) \mid t \in sign(f)\}$ is a partition of $case(f)$. These fibers are the images of the injection functions

$$fib(indic(f))(t) = inj(f)(t)[arity(f)(t)] \subseteq case(f)$$

for all signatures $t \in sign(f)$.

The projection function is locally bijective: for signature $t \in sign(f)$ the restriction of the projection function on each fiber $fib(indic(f))(t)$ is a bijection:

$$proj(f) : fib(indic(f))(t) \cong arity(f)(t).$$

This fact is important, since it implies that only one substitution function of instances into variables is possible. The projection function is the cotupling of the arity inclusion functions $\{incl_{arity(f)(t),\, src(f)} \mid t \in sign(f)\}$. To prove this use the class function

$arity(f)$. $pow\text{-}incl(src(f)).incl$: $sign(f) \to \wp src(f) \to tuple(\langle src(f), src(f)\rangle) \to function$

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

(16) (SET.FTN$function projection)
     (= (SET.FTN$source projection) function)
     (= (SET.FTN$target projection) function)
     (= projection (SET.FTN$composition [arity set.col.art$projection]))
```

- Any function $f : src(f) \to tgt(f)$ defines a *comediator* function:

  $\tilde{*}_f = comed(f) : case(f) \to tgt(f)$.

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

  $comed(f)((t, x)) = t(x)$

  for all signatures $t \in sign(f)$ and all names $x \in arity(f)(t)$. Abstractly, it is the comediator – hence the name – of the coproduct cotuple (cocone) consisting of the collection of signatures regarded as functions,

  $\{t : arity(f)(t) \to tgt(f) \mid t \in sign(f)\}$.

  The comediator function commutes (Diagram 4) with the case injection function and the signature itself. Since all the signatures factor as the composition of their arity inclusion with the original (reference) function f, the comediator function can simply be defined as the composition of projection with the original (reference) function. If the indexed names in $case(A)$ are viewed as roles (prehensions), the comediator is a reference function from roles to objects (actualities).

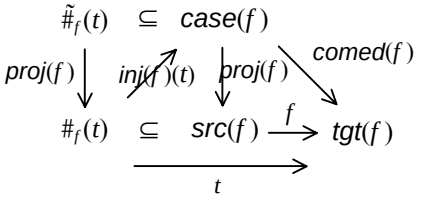


Diagram 4: Comediator

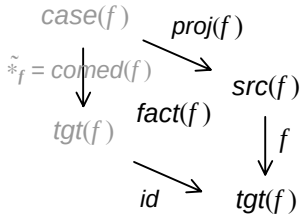
  The main reason for defining the comediator function for set pairs was to replace a set pair with a function, and by extension to provide object function of a functor from set semisquares to set squares. A reason for defining the comediator function for functions is to separate the arities of signatures. Of course, one could do this again. However, this gives nothing new, since the comediator of the comediator is isomorphic the first comediator.

```
(17) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) function)
     (= (SET.FTN$target comediator) function)
     (forall (?f (function ?f))
         (and (= (set.ftn$source (comediator ?f)) (case ?f))
              (= (set.ftn$target (comediator ?f)) (target ?f))
              (= (comediator ?f)
                 ((set.col.art$cotupling (arity ?f)) (signature-inclusion ?f)))))

     (forall (?f (function ?f))
         (= (reference ?f) (set.ftn$composition [(projection ?f) ?f])))
```

- The function comediator function can also be factored as the composition of the projection function and the original function

  $comed(f) = proj(f) \cdot f$.

  The fact can be represented as a *factorizarion* quartet.




```
(18) (SET.FTN$function factorization)
     (= (SET.FTN$source factorization) function)
     (= (SET.FTN$target factorization) set.qtt$quartet)
     (forall (?f (function ?f))
         (and (= (set.qtt$horizontal-source (factorization ?f)) (comediator ?f))
              (= (set.qtt$horizontal-target (factorization ?f)) ?f)
              (= (set.qtt$vertical-source (factorization ?g)) (projection ?f))
              (= (set.qtt$vertical-target (factorization ?f))
```

```
                (identity (target ?g)))))
```

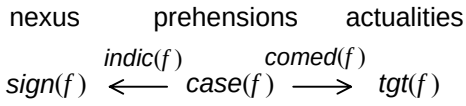


Figure 1: Spangraph of a Function

- Associated with any function $f : src(f) \to tgt(f)$ is a spangraph
  $sgph(f) = \langle 1^{st}_{sgph(f)}, 2^{nd}_{sgph(f)} \rangle$
  consisting of the two functions $1^{st}_{sgph(f)}(f) = indic(f)$ and $2^{nd}_{sgph(f)}(f) = comed(f)$.

```
(19) (SET.FTN$function spangraph)
     (= (SET.FTN$source spangraph) function)
     (= (SET.FTN$target spangraph) sgph$spangraph)
     (forall (?f (function ?f))
         (and (= (sgph$set1 (spangraph ?f)) (signature ?f))
              (= (sgph$set2 (spangraph ?f)) (target ?f))
              (= (sgph$vertex (spangraph ?f)) (case ?f))
              (= (sgph$first (spangraph ?f)) (indication ?f))
              (= (sgph$second (spangraph ?f)) (comediator ?f))))
```

> The spangraph associated with any function is the image of the object function of the spangraph functor applied to the function:
>
> $sgph$: horiz(□Set) → Spangraph.

- For any function $f : A \to B$, a *substitution* $h : dom(h) \to cod(h)$ of f is a surjective function in the morphic power $\mathsf{F}A$ of the source that respects f: $f(h(a)) = f(a)$ for all $a \in dom(h)$. Let $subst(f)$ denote the class of all substitutions of f. By the axioms for the set namespace, this is a set; that is, substitutions are small. We rename the domain and codomain.

```
(20) (SET.FTN$function substitution)
     (= (SET.FTN$source substitution) function)
     (= (SET.FTN$target substitution) set$set)
     (forall (?f (function ?f) ?h (function ?h))
         (<=> ((substitution ?f) ?h)
              (and (surjection ?h)
                   ((morphic-power (source ?f)) ?h))))

(21) (SET.FTN$function domain)
     (= (SET.FTN$source domain) function)
     (= (SET.FTN$target domain) function)
     (= (SET.FTN$composition [domain set.ftn$source]) substitution)
     (= (SET.FTN$composition [domain set.ftn$target])
        (SET.FTN$composition [source set$power))
     (forall (?f (function ?f) ?h (function ?h))
         (= ((domain ?f) ?h) ((set$domain (source ?f)) ?h))

(22) (SET.FTN$function codomain)
     (= (SET.FTN$source codomain) function)
     (= (SET.FTN$target codomain) function)
     (= (SET.FTN$composition [codomain set.ftn$source]) substitution)
     (= (SET.FTN$composition [codomain set.ftn$target])
        (SET.FTN$composition [source set$power))
     (forall (?f (function ?f) ?h (function ?h))
         (= ((codomain ?f) ?h) ((set$codomain (source ?f)) ?h))
```

Set Invertible-Pairs

`set.invpr`

- Sets are related through invertible pairs. An *invertible pair* $\boldsymbol{f} = \langle inv(\boldsymbol{f}), dir(\boldsymbol{f}) \rangle : A \cong B$ from set A to set B is a pair of oppositely directed functions, a function $dir(\boldsymbol{f}) : typ(\boldsymbol{A}) \to typ(\boldsymbol{B})$ in the direct direction and a function $inv(\boldsymbol{f}) : B \to inst(\boldsymbol{A})$ in the inverse direction, which are inverse to each other: they satisfy the *fundamental property*:

  $inv(\boldsymbol{f})(b) =_{\boldsymbol{A}} a$ iff $b =_{\boldsymbol{B}} dir(\boldsymbol{f})(a)$

  for all elements $b \in B$ and $a \in A$. This is the fundamental property of infomorphisms between identity classifications. This can equivalently and pointlessly be expressed in the usual fashion:

  $typ(\boldsymbol{f}) \cdot inst(\boldsymbol{f}) = id(A)$ and $inst(\boldsymbol{f}) \cdot typ(\boldsymbol{f}) = id(B)$.

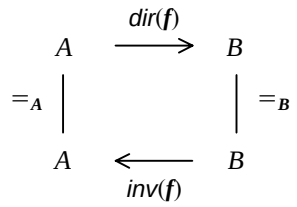


Figure 3: Invertible Pair

```
(1) (SET$class invertible-pair)

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) invertible-pair)
    (= (SET.FTN$target source) set$set)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) invertible-pair)
    (= (SET.FTN$target target) set$set)

(4) (SET.FTN$function inverse)
    (= (SET.FTN$source inverse) invertible-pair)
    (= (SET.FTN$target inverse) set.ftn$function)
    (= (SET.FTN$composition inverse set.ftn$source) target)
    (= (SET.FTN$composition inverse set.ftn$target) source)

(5) (SET.FTN$function direct)
    (= (SET.FTN$source direct) invertible-pair)
    (= (SET.FTN$target direct) set.ftn$function)
    (= (SET.FTN$composition direct set.ftn$source) source)
    (= (SET.FTN$composition direct set.ftn$target) target)

(6) (forall (?f (invertible-pair ?f))
        (and (= (set.ftn$composition [(direct ?f) (inverse ?f)])
                (set.ftn$identity (source ?f)))
             (= (set.ftn$composition [(inverse ?f) (direct ?f)])
                (set.ftn$identity (target ?f)))))
```

- Any set can be embedded as an identity classification. Can any function be embedded as an "identity infomorphism" between the identity classifications of its source and target? The answer is no. To see why, we need to understand the nature of an infomorphism between two identity classifications. Let A and B be any two sets and let $\boldsymbol{f} : cls(A) \rightleftarrows cls(B)$ be an infomorphism between their identity classifications. Let $a \in A = typ(cls(A))$ be any element of A, regarded as a type in the source classification. Since $typ(\boldsymbol{f})(a) = typ(\boldsymbol{f})(a)$ and since the incidence of source and target classifications is identity, by the fundamental property of infomorphisms $inst(\boldsymbol{f})(typ(\boldsymbol{f})(a)) = a$. Hence, $typ(\boldsymbol{f}) \cdot inst(\boldsymbol{f}) = id(A)$, since $a \in A$ was arbitrary. In a similar and dual fashion we can show that $inst(\boldsymbol{f}) \cdot typ(\boldsymbol{f}) = id(B)$. Thus, $\boldsymbol{f}$ consists of a pair of inverse functions. In the converse direction, any pair of inverse functions is such an infomorphism.

```
(7) (SET.FTN$function infomorphism)
    (= (SET.FTN$source infomorphism) inverse-pair)
    (= (SET.FTN$target infomorphism) cls.info$infomorphism)
    (= (SET.FTN$composition [infomorphism cls.info$source])
       (SET.FTN$composition [source set$classification]))
    (= (SET.FTN$composition [infomorphism cls.info$target])
       (SET.FTN$composition [target set$classification]))
    (= (SET.FTN$composition [infomorphism cls.info$instance]) inverse)
    (= (SET.FTN$composition [infomorphism cls.info$instance]) direct)
```

```
(SET.FTN$bijection infomorphism)
```

- Two invertible pairs are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable invertible pairs $\boldsymbol{f}_1 : A \to A'$ and $\boldsymbol{f}_2 : A' \to A''$ is defined in terms of the horizontal composition of their direct and inverse functions.

```
(8) (SET.LIM.PBK$opspan composable-opspan)
    (= (class1 composable-opspan) invertible-pair)
    (= (class2 composable-opspan) invertible-pair)
    (= (opvertex composable-opspan) set$set)
    (= (first composable-opspan) target)
    (= (second composable-opspan) source)

(9) (REL$relation composable)
    (= (REL$class1 composable) invertible-pair)
    (= (REL$class2 composable) invertible-pair)
    (= (REL$extent composable) (SET.LIM.PBK$pullback composable-opspan))

(10) (SET.FTN$function composition)
     (= (SET.FTN$source composition) (SET.LIM.PBK$pullback composable-opspan))
     (= (SET.FTN$target composition) invertible-pair)
     (forall (?f1 (invertible-pair ?f1) ?f2 (invertible-pair ?f2) (composable ?f1 ?f2))
         (and (= (source (composition [?f1 ?f2])) (source ?f1))
              (= (target (composition [?f1 ?f2])) (target ?f2))
              (= (direct (composition [?f1 ?f2]))
                 (set.ftn$composition [(direct ?f1) (direct ?f2)]))
              (= (inverse (composition [?f1 ?f2]))
                 (set.ftn$composition [(inverse ?f2) (inverse ?f1)])))))
```

- Composition satisfies the usual *associative law*.

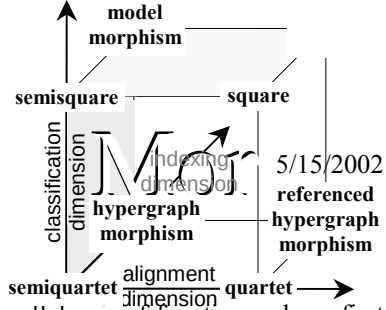
```
(forall (?f1 (invertible-pair ?f1)
         ?f2 (invertible-pair ?f2)
         ?f3 (invertible-pair ?f3)
         (composable ?f1 ?f2) (composable ?f2 ?f3))
    (= (composition [?f1 (composition [?f2 ?f3])])
       (composition [(composition [?f1 ?f2]) ?f3])))
```

- For any set *A*, there is an *identity* invertible pair.

```
(11) (SET.FTN$function identity)
     (= (SET.FTN$source identity) set$set)
     (= (SET.FTN$target identity) invertible-pair)
     (forall (?a (set$set ?a))
         (and (= (source (identity ?a)) ?a)
              (= (target (identity ?a)) ?a)
              (= (direct (identity ?a)) (set.ftn$identity ?a))
              (= (inverse (identity ?a)) (set.ftn$identity ?a))))
```

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

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



Set Semiquartets

`set.sqtt`

Set pairs are related through set semiquartets. A set semiquartet is a parallel pair of functions whose first component is invertible. The classes of set pairs and set semiquartets form the category △Set called the

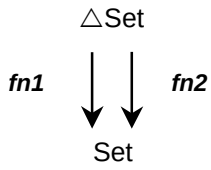


Diagram 1: The component functors

semiquartet of Set. The tuple functor – consisting of the tuple sets and tuple functions defined below – is the main reason for introducing this category.

- A *semiquartet*

  $\alpha = \langle fn1(\alpha), fn2(\alpha) \rangle : src(\alpha) \to tgt(\alpha)$

  from set pair src(α) to set pair tgt(α) with components *fn1*(α) and *fn2*(α), consists of a diagram (Figure 1) of these two functions, where the first function is a bijection and hence part of an invertible pair, and the two functions suitably match at source and target sets

  $src(fn1(\alpha)) = set1(src(\alpha))$,

  $tgt(fn1(\alpha)) = set1(tgt(\alpha))$,

  $src(fn2(\alpha)) = set2(src(\alpha))$, and

  $tgt(fn2(\alpha)) = set2(tgt(\alpha))$.

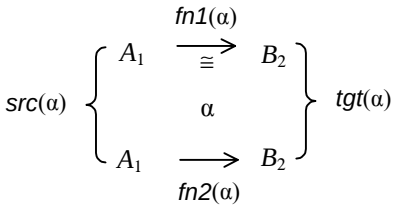


Figure 1: Semiquartet

```
(1) (SET$class semiquartet)

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

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) semiquartet)
    (= (SET.FTN$target target) set.pr$pair)

(4) (SET.FTN$function function1)
    (= (SET.FTN$source function1) semiquartet)
    (= (SET.FTN$target function1) set.ftn$bijection)
    (= (SET.FTN$composition [function1 set.ftn$source])
       (SET.FTN$composition [source set.pr$set1]))
    (= (SET.FTN$composition [function1 set.ftn$target])
       (SET.FTN$composition [target set.pr$set1]))

(5) (SET.FTN$function function2)
    (= (SET.FTN$source function2) semiquartet)
    (= (SET.FTN$target function2) set.ftn$function)
    (= (SET.FTN$composition [function2 set.ftn$source])
       (SET.FTN$composition [source set.pr$set2]))
    (= (SET.FTN$composition [function2 set.ftn$target])
       (SET.FTN$composition [target set.pr$set2]))
```

Semiquartets form the morphism class of the △Set category.

- Semiquartets are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable semiquartets is defined in terms of the composition of their component functions.

```
(6) (SET.LIM.PBK$opspan composable-opspan)
    (= (SET.LIM.PBK$class1 composable-opspan) semiquartet)
    (= (SET.LIM.PBK$class2 composable-opspan) semiquartet)
    (= (SET.LIM.PBK$opvertex composable-opspan) set.pr$pair)
    (= (SET.LIM.PBK$first composable-opspan) target)
    (= (SET.LIM.PBK$second composable-opspan) source)

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

(8) (SET.FTN$function composition)
    (= (SET.FTN$source composition)
       (SET.LIM.PBK$pullback composable-opspan))
    (= (SET.FTN$target composition) semiquartet)
    (forall (?p1 (semiquartet ?p1) ?p2 (semiquartet ?p2) (composable ?p1 ?p2))
        (and (= (source (composition [?p1 ?p2])) (source ?p1))
             (= (target (composition [?p1 ?p2])) (target ?p2))
             (= (function1 (composition [?p1 ?p2]))
                (set.ftn$composition [(function1 ?p1) (function1 ?p2)]))
             (= (function2 (composition [?p1 ?p2]))
                (set.ftn$composition [(function2 ?p1) (function2 ?p2)])))))
```

- Composition satisfies the usual *associative law*.

```
(forall (?p1 (semiquartet ?p1) ?p2 (semiquartet ?p2) ?p3 (semiquartet ?p3)
         (composable ?p1 ?p2) (composable ?p2 ?p3))
    (= (composition [?p1 (composition [?p2 ?p3])])
       (composition [(composition [?p1 ?p2]) ?p3])))
```

- For any function, regarded as a vertical morphism, there is an *identity* semiquartet.

```
(9) (SET.FTN$function identity)
    (= (SET.FTN$source identity) set.pr$pair)
    (= (SET.FTN$target identity) semiquartet)
    (forall (?a (set.pr$pair ?a))
        (and (= (source (identity ?a)) ?a)
             (= (target (identity ?a)) ?a)
             (= (function1 (identity ?a)) (set.ftn$identity (set.pr$set1 ?a)))
             (= (function2 (identity ?a)) (set.ftn$identity (set.pr$set2 ?a)))))
```

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

```
(forall (?p (semiquartet ?p))
    (and (= (composition [(identity (source ?p)) ?p]) ?p)
         (= (composition [?p (identity (target ?p))]) ?p)))
```

> The category △Set has set pairs as its objects, semiquartets as its morphisms, composition of semiquartets as its composition function and identity semiquartets as its identity function.

- Associated with a semiquartet α is an function of *tuples*

  $tuple(\alpha) : tuple(src(\alpha)) \to tuple(tgt(\alpha))$.

One use for set pairs is for indexing: the arity (source) is a set of indices, the target is a set of objects, and the tuple represents the referencing of objects by the indices in the arity. Here we point out some properties of semiquartets, interpreted as morphisms of indexes.

A tuple $t \in tuple(src(\alpha))$ is a function $t : arity(t) \to set2(src(\alpha))$. Letting $arity(t) = \{x_1, x_2, \ldots, x_n\}$ with $t(x_n) = t_n$, we can think of a tuple as a vector $t = (t_1, t_2, \ldots, t_n)$. The natural number indexing here is just a device to display the tuple in linear form. It is not

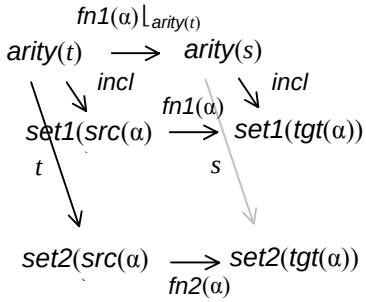


Diagram 1: Tuple Function

necessary in the abstract reality. Next, let us make the following definitions:

$\mathit{arity}(s) = \wp(\mathit{fn1}(\alpha))(\mathit{arity}(t))$,

$s = \mathit{tuple}(\alpha)(t)$, and

$\mathit{fn1}(\alpha)\lfloor_{\mathit{arity}(t)}$ is the restriction of $\mathit{fn1}(\alpha)$ to $\mathit{arity}(t)$.

Thus, the tuple $s \in \mathit{tuple}(\mathit{tgt}(\alpha))$, a function $s : \mathit{arity}(s) \to \mathit{set2}(\mathit{tgt}(\alpha))$, is the image of the tuple t along the function $\mathit{tuple}(\alpha) : \mathit{tuple}(\mathit{src}(\alpha)) \to \mathit{tuple}(\mathit{tgt}(\alpha))$; that is, $\mathit{tuple}(\alpha)(t) = s$. Therefore, we end up with Diagram 1. In particular, the back commuting rectangle expresses the following constraint between tuples:

$t \cdot \mathit{fn2}(\alpha) = \mathit{fn1}(\alpha)\lfloor_{\mathit{arity}(t)} \cdot s$.

In the frame interpretation mentioned above, and elaborated later, we view a tuple $t = (t_1, t_2, \ldots, t_n)$ to be much like a frame with the elements in $\mathit{arity}(t)$ functioning as frame slot names (attribute names) and elements in $\mathit{set2}(\mathit{src}(\alpha))$ functioning as frame slot fillers (attribute values). For the mapping by $\mathit{tuple}(\alpha)$ from t to s there is one case to discuss. Since the complete function $\mathit{fn1}(\alpha)$ is a bijection, the restriction $\mathit{fn1}(\alpha)\lfloor_{\mathit{arity}(t)}$ is also a bijection. It must map distinct slot names $x \neq x'$ with $x, x' \in \mathit{arity}(t)$ onto distinct slot names $y, y' \in \mathit{arity}(s)$: $\mathit{fn1}(\alpha)(x) = y$ and $\mathit{fn1}(\alpha)(x') = y'$.

– Suppose the tuple s fills the two slots y and y' with the same object $s(y) = s(y') \in \mathit{set2}(\mathit{tgt}(\alpha))$. The above tuple constraint indicates two ways that this can occur.
  * The tuple t fills the two slots with the same object: $t(x) = t(x') \in \mathit{set2}(\mathit{src}(\alpha))$.
  * The tuple t fills the two slots with different objects in $\mathit{set2}(\mathit{src}(\alpha))$, but the function $\mathit{fn2}(\alpha)$ maps these to the same object:

    $\mathit{fn2}(\alpha)(t(x)) = \mathit{fn2}(\alpha)(t(x')) \in \mathit{set2}(\mathit{tgt}(\alpha))$.

```
(10) (SET.FTN$function tuple)
     (= (SET.FTN$source tuple) semiquartet)
     (= (SET.FTN$target tuple) set.ftn$function)
     (= (SET.FTN$composition [tuple set.ftn$source])
        (SET.FTN$composition [source set.pr$tuple]))
     (= (SET.FTN$composition [tuple set.ftn$target])
        (SET.FTN$composition [target set.pr$tuple]))
     (forall (?p (semiquartet ?p)
              ?t ((set.pr$tuple (source ?p)) ?t))
         (and (= (set.pr$arity ((tuple ?p) ?t))
                 ((set.ftn$power (function1 ?p)) (set.pr$arity ?t)))
              (forall (?x ((set.pr$arity ((tuple ?p) ?t)) ?x))
                  (= (((tuple ?p) ?t) ?x)
                     ((function1 ?p) (?t ((set.ftn$inverse (function1 ?p)) ?x)))))))
```

The tuple function associated with any semiquartet is the image of the morphism function of the tuple functor applied to the semiquartet:

$\mathit{tuple} : \triangle\mathsf{Set} \to \mathsf{Set}$.

○ The tuple function allows us to define an *tuple arity* set quartet (Figure 1) $\mathit{tuple\text{-}arity}(\alpha)$ for any semiquartet α. This defines an arity functor $\mathit{tuple\text{-}arity} : \triangle\mathsf{Set} \to \mathit{horiz}(\square\mathsf{Set})$. But a functor whose

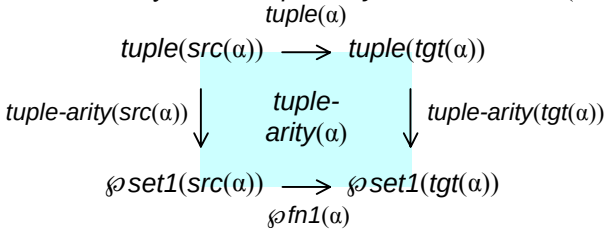


Figure 1: The tuple-arity quartet

quartet α. This defines an arity functor *tuple-arity* : △Set → *horiz*(□Set). But a functor whose target is *horiz*(□Set) splits into two Set-functors with a natural transformation between them.

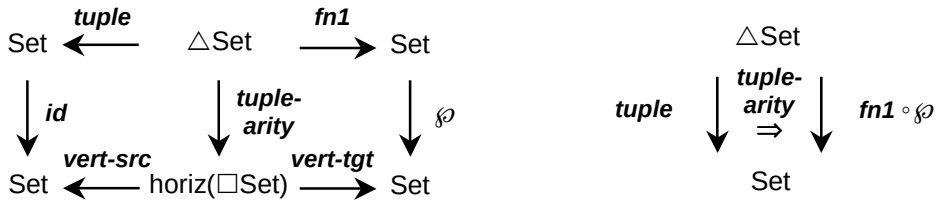


Figure 1: Tuple and Tuple Arity Functors and Natural Isomorphisms

```
(11) (SET.FTN$function tuple-arity)
     (= (SET.FTN$source tuple-arity) semiquartet)
     (= (SET.FTN$target tuple-arity) set.qtt$quartet)
     (= (SET.FTN$composition [tuple-arity set.qtt$horizontal-source])
        (SET.FTN$composition [source set.pr$tuple-arity]))
     (= (SET.FTN$composition [tuple-arity set.qtt$horizontal-target])
        (SET.FTN$composition [target set.pr$tuple-arity]))
     (= (SET.FTN$composition [tuple-arity set.qtt$vertical-source]) tuple)
     (= (SET.FTN$composition [tuple-arity set.qtt$vertical-target])
        (SET.FTN$composition [function1 set.ftn$power]))
```

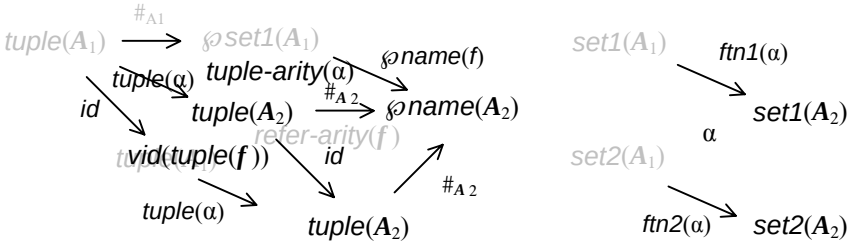


Figure 3 Hypergraph morphism of a set semiquartet

○ Any set semiquartet α from set pair $\boldsymbol{A}_1$ = src(α) to set pair $\boldsymbol{A}_2$ = tgt(α) with components *fn1*(α) and *fn2*(α), defines a *hypergraph morphism* (Figure 3), whose reference semiquartet is α itself and whose tuple quartet is the vertical identity quartet of the tuple function *tuple*(α) : *tuple*($\boldsymbol{A}_1$) → *tuple*($\boldsymbol{A}_2$). This hypergraph morphism has the first component bijection *fn1*(α) as its name function, the second component function *fn2*(α) as its node function, and the tuple function *tuple*(α) as its edge fuction. Clearly, the semiquartet of the hypergraph of a semiquartet is itself.

```
(12) (SET.FTN$function hypergraph-morphism)
     (= (SET.FTN$source hypergraph-morphism) semiquartet)
     (= (SET.FTN$target hypergraph-morphism) hgph.mor$hypergraph-morphism)
     (forall (?p (semiquartet ?p))
         (and (= (hgph.mor$reference (hypergraph-morphism ?p)) ?p)
              (= (hgph.mor$tuple (hypergraph-morphism ?p))
                 (set.ftn$vertical-identity (tuple ?p)))))
```

○ For any set semiquartet α from set pair $\boldsymbol{A}_1$ = src(α) to set pair $\boldsymbol{A}_2$ = tgt(α) with components *fn1*(α) and *fn2*(α), the coproduct *arity morphism* *arity-mor*(α) : *arity*($\boldsymbol{A}_1$) → *arity*($\boldsymbol{A}_2$) is compatible with the tuple arity quartet – the quartet of the coproduct arity morphism is the tuple arity quartet.

```
(13) (SET.FTN$function arity-morphism)
     (= (SET.FTN$source arity-morphism) semiquartet)
     (= (SET.FTN$target arity-morphism) set.col.art.mor$arity)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$source])
        (SET.FTN$composition [source set.col.art$arity]))
```

```
(= (SET.FTN$composition [arity-morphism set.col.art.mor$target])
   (SET.FTN$composition [target set.col.art$arity]))
(= (SET.FTN$composition [arity-morphism set.col.art.mor$index]) tuple)
(= (SET.FTN$composition [arity-morphism set.col.art.mor$base]) function1)

(= (SET.FTN$composition [arity-morphism set.col.art.mor$quartet]) tuple-arity)
```

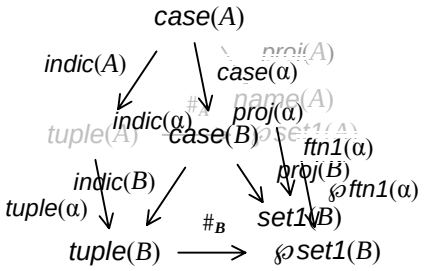


Figure 1: Coproduct of the Arity of a Semiquartet

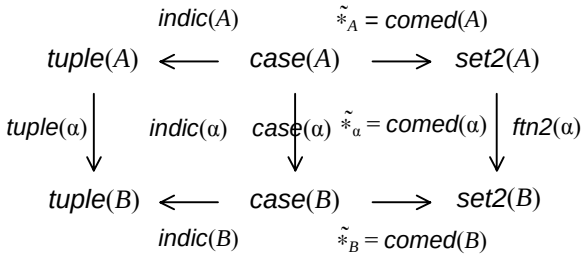


Figure 2: Spangraph Morphism of a Semiquartet

- Any semiquartet α, from set pair $\mathsf{src}(\alpha) = A = \langle A_1, A_2 \rangle = \langle \mathit{set1}(A), \mathit{set2}(A) \rangle$ to set pair $\mathsf{tgt}(\alpha) = B = \langle B_1, B_2 \rangle$ with function components $\mathit{fn1}(\alpha) : A_1 \cong B_1$ and $\mathit{fn2}(\alpha) : A_2 \to B_2$, defines a *case* or *role* function $\mathit{case}(\alpha) : \mathit{case}(A) = \sum \mathit{arity}(A) \to \sum \mathit{arity}(B) = \mathit{case}(B)$. The pointwise definition is:

  $$\mathit{case}(\alpha)((t, x)) = (\mathit{tuple}(\alpha)(t), \mathit{fn1}(\alpha)(x))$$

  for any tuple $t \in \mathit{tuple}(A)$ and any name $x \in \mathit{arity}(A)(t)$. This is well defined since α preserves tuple arity. The abstract definition is in terms of the coproduct of the arity morphism.

```
(14) (SET.FTN$function case)
     (= (SET.FTN$source case) semiquartet)
     (= (SET.FTN$target case) set.ftn$function)
     (forall (?p (semiquartet ?p))
         (and (= (set.ftn$source (case ?p)) (set.pr$case (source ?p)))
              (= (set.ftn$target (case ?p)) (set.pr$case (target ?p)))
              (= (case ?p)
                 (set.col.art.mor$coproduct (arity-morphism ?p)))))
```

- The case function is the vertical source for two quartets: an *indication* quartet $\mathit{indic}(\alpha)$ and a *projection* quartet $\mathit{proj}(\alpha)$.
  - The commutativity $\mathit{case}(\alpha) \cdot \mathit{indic}(B) = \mathit{indic}(A) \cdot \mathit{tuple}(\alpha)$, a property of the coproduct of arities (preservation of index), is obvious from the pointwise definition of the case function.
  - The commutativity $\mathit{case}(\alpha) \cdot \mathit{proj}(B) = \mathit{proj}(A) \cdot \mathit{name}(\alpha)$, a property of the coproduct of arities (preservation of projection), is obvious from the pointwise definition of the case function.

  Even though the first component function $\mathit{ftn1}(\alpha) : \mathit{set1}(A) \to \mathit{set1}(B)$ and its power $\wp\mathit{ftn1}(\alpha)$ are bijections, the case function $\mathit{case}(\alpha) : \mathit{case}(A) \to \mathit{case}(B)$ is not necessarily a bijection since the tuple function $\mathit{tuple}(\alpha) : \mathit{tuple}(A) \to \mathit{tuple}(B)$ need not be bijective (and this in turn since the second component function $\mathit{ftn2}(\alpha) : \mathit{set2}(A) \to \mathit{set2}(B)$ need not be bijective). Abstractly by the preservation of tuple arity and concretely by the bijective nature of $\mathit{ftn1}(\alpha)$, the index quartet is a fibration: for any tuple $t \in \mathit{tuple}(A)$ and any name $y \in \mathit{arity}(B)(\mathit{tuple}(\alpha)(t)) = \mathit{fn1}(\alpha)\lfloor_{\mathit{arity}(t)}$ there is a name $x \in \mathit{arity}(A)(t)$ such that $\mathit{ftn1}(\alpha)(x) = y$.

```
(15) (SET.FTN$function indication)
     (= (SET.FTN$source indication) semiquartet)
     (= (SET.FTN$target indication) set.qtt$fibration)
     (forall (?p (semiquartet ?p))
         (and (= (set.qtt$horizontal-source (indication ?p)) (set.pr$indication (source ?p)))
              (= (set.qtt$horizontal-target (indication ?p)) (set.pr$indication (target ?p)))
              (= (set.qtt$vertical-source (indication ?p)) (case ?p))
              (= (set.qtt$vertical-target (indication ?p)) (tuple ?p))))

(16) (SET.FTN$function projection)
     (= (SET.FTN$source projection) semiquartet)
     (= (SET.FTN$target projection) set.qtt$quartet)
```

```
(forall (?p (semiquartet ?p))
    (and (= (set.qtt$horizontal-source (projection ?p)) (set.pr$projection (source ?p)))
         (= (set.qtt$horizontal-target (projection ?p)) (set.pr$projection (target ?p)))
         (= (set.qtt$vertical-source (projection ?p)) (case ?p))
         (= (set.qtt$vertical-target (projection ?p)) (function1 ?p))))
```

- Any semiquartet α, from set pair $\mathsf{src}(\alpha) = A = \langle A_1, A_2 \rangle = \langle \mathit{set1}(A), \mathit{set2}(A) \rangle$ to set pair $\mathsf{tgt}(\alpha) = B = \langle B_1, B_2 \rangle$ with function components $\mathit{fn1}(\alpha) : A_1 \cong B_1$ and $\mathit{fn2}(\alpha) : A_2 \to B_2$, defines a *comediator* quartet $\tilde{*}_\alpha = \mathit{comed}(\alpha)$.
  - The commutativity $\mathit{case}(\alpha) \cdot \mathit{comed}(B) = \mathit{comed}(A) \cdot \mathit{ftn2}(\alpha)$ holds by a property of the coproduct of arities (preservation of cotupling).

  Pointwsie, commutativity states that

  $$\mathit{tuple}(\alpha)(t)(\mathit{ftn1}(\alpha)(x)) = \mathit{ftn2}(\alpha)(t(x))$$

  for any tuple $t \in \mathit{tuple}(A)$ and any name $x \in \mathit{arity}(A)(t)$, which is true by preservation of tuple.

```
(17) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) semiquartet)
     (= (SET.FTN$target comediator) set.qtt$quartet)
     (forall (?p (semiquartet ?p))
         (and (= (set.qtt$horizontal-source (comediator ?p)) (set.pr$comediator (source ?p)))
              (= (set.qtt$horizontal-target (comediator ?p)) (set.pr$comediator (target ?p)))
              (= (set.qtt$vertical-source (comediator ?p)) (case ?p))
              (= (set.qtt$vertical-target (comediator ?p)) (function2 ?p))))
```

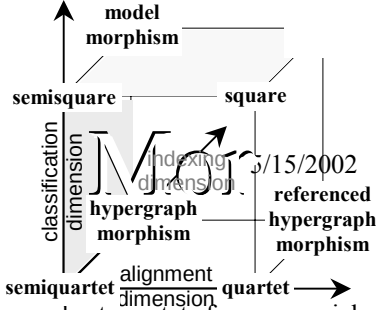
- Associated with any semiquartet α is a spangraph morphism

  $$\mathit{sgph}(\alpha) = \langle \mathit{1}^{st}{}_{\mathit{sgph}(\alpha)}, \mathit{2}^{nd}{}_{\mathit{sgph}(\alpha)} \rangle : \mathit{sgph}(A) \to \mathit{sgph}(B)$$

  with the two quartets (the 1^{st} is a fibration) $\mathit{1}^{st}{}_{\mathit{sgph}(\alpha)}(\alpha) = \mathit{indic}(\alpha)$ and $\mathit{2}^{nd}{}_{\mathit{sgph}(\alpha)}(\alpha) = \mathit{comed}(\alpha)$.

```
(18) (SET.FTN$function spangraph-morphism)
     (= (SET.FTN$source spangraph-morphism) semiquartet)
     (= (SET.FTN$target spangraph-morphism) sgph.mor$spangraph-morphism)
     (forall (?p (semiquartet ?p))
         (and (= (sgph.mor$source (spangraph-morphism ?p)) (set.pr$spangraph (source ?p)))
              (= (sgph.mor$target (spangraph-morphism ?p)) (set.pr$spangraph (target ?p)))
              (= (sgph.mor$first (spangraph-morphism ?p)) (indication ?p))
              (= (sgph.mor$second (spangraph-morphism ?p)) (comediator ?p))))
```

> The spangraph morphism associated with any semiquartet is the image of the morphism function of the spangraph functor applied to the semiquartet:
>
> $$\mathit{sgph} : \triangle\mathsf{Set} \to \mathsf{Spangraph}.$$



Set Quartets

`set.qtt`

Functions are related through set quartets. The classes of sets, set functions, and set quartets form a special kind of double category ☐Set called the *quartet* of Set.

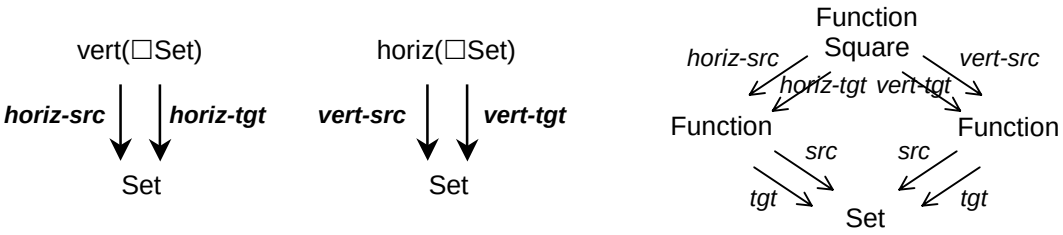


Diagram 1: The Categories and Functors implicit within ☐Set the Quartet of Set

- A *set quartet* (*set function square*) α,
  horizontally from function *horiz-src*(α) to function *horiz-tgt*(α)
  α = ⟨*vert-src*(α), *vert-tgt*(α)⟩ : *horiz-src*(α) → *horiz-tgt*(α), and
  vertically from function *vert-src*(α) to function *vert-tgt*(α),
  α = ⟨*horiz-src*(α), *horiz-tgt*(α)⟩ : *vert-src*(α) → *vert-tgt*(α),
  consists of a commutative diagram (Figure 1) of these four functions,
  vert-src(α) · *horiz-tgt*(α) = *horiz-src*(α) · *vert-tgt*(α),
  which suitably match at source and target sets
  src(*vert-src*(α)) = *src*(*horiz-src*(α)),
  tgt(*vert-tgt*(α)) = *tgt*(*horiz-tgt*(α)),
  composable(*vert-src*(α), *horiz-tgt*(α)), and
  composable(*horiz-src*(α), *vert-tgt*(α)).

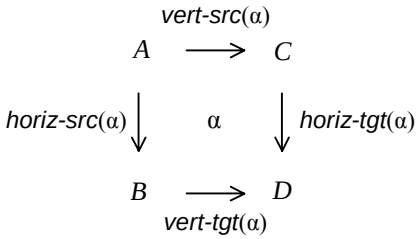


Figure 1: Set Quartet

Commutatively means that reference is preserved: the referent of the image is the image of the referent.

```
(1) (SET$class quartet)

(2) (SET.FTN$function horizontal-source)
    (= (SET.FTN$source horizontal-source) quartet)
    (= (SET.FTN$target horizontal-source) set.ftn$function)

(3) (SET.FTN$function horizontal-target)
    (= (SET.FTN$source horizontal-target) quartet)
    (= (SET.FTN$target horizontal-target) set.ftn$function)

(4) (SET.FTN$function vertical-source)
    (= (SET.FTN$source vertical-source) quartet)
    (= (SET.FTN$target vertical-source) set.ftn$function)

(5) (SET.FTN$function vertical-target)
    (= (SET.FTN$source vertical-target) quartet)
    (= (SET.FTN$target vertical-target) set.ftn$function)

    (= (SET.FTN$composition [vertical-source set.ftn$source])
       (SET.FTN$composition [horizontal-source set.ftn$source]))
    (= (SET.FTN$composition [vertical-target set.ftn$target])
       (SET.FTN$composition [horizontal-target set.ftn$target]))
    (= (SET.FTN$composition [vertical-source set.ftn$target])
       (SET.FTN$composition [horizontal-target set.ftn$source]))
    (= (SET.FTN$composition [horizontal-source set.ftn$target])
       (SET.FTN$composition [vertical-target set.ftn$source]))
```

```
(6) (forall (?a (quartet ?a))
        (= (set.ftn$composition [(vertical-source ?a) (horizontal-target ?a)])
           (set.ftn$composition [(horizontal-source ?a) (vertical-target ?a)])))
```

Set quartets form the square (cell) class of the double category ☐Set.

- A quartet α is *fibration* when the mediator (pullback pairing) of ⟨*horiz-src*(α), *vert-src*(α)⟩ is with respect to the pullback of the opspan ⟨*vert-tgt*(α), *horiz-tgt*(α)⟩ is surjective: for any element $a \in$ *src*(*vert-tgt*(α)) = *tgt*(*horiz-src*(α)) and any element $y \in$ *src*(*horiz-tgt*(α)) = *tgt*(*vert-src*(α)), if *vert-tgt*(α)(*a*)= *horiz-tgt*(α)(*y*) then there is an element $x \in$ *src*(*vert-src*(α)) = *src*(*horiz-src*(α)) with *horiz-src*(α)(*x*) = *a* and *vert-src*(α)(*a*) = *y*.

```
(7) (SET.FTN$function fibration-opspan)
    (= (SET.FTN$source fibration-opspan) quartet)
    (= (SET.FTN$target fibration-opspan) set.lim.pbk$opspan)
    (= (SET.FTN$composition [fibration-opspan set.lim.pbk$set1])
       (SET.FTN$composition [horizontal-source set.ftn$target]))
    (= (SET.FTN$composition [fibration-opspan set.lim.pbk$set2])
       (SET.FTN$composition [horizontal-target set.ftn$source]))
    (= (SET.FTN$composition [fibration-opspan set.lim.pbk$opvertex])
       (SET.FTN$composition [horizontal-target set.ftn$target]))
    (= (SET.FTN$composition [fibration-opspan set.lim.pbk$opfirst]) vertical-target)
    (= (SET.FTN$composition [fibration-opspan set.lim.pbk$opsecond]) horizontal-target)

(8) (SET.FTN$function fibration-cone)
    (= (SET.FTN$source fibration-cone) quartet)
    (= (SET.FTN$target fibration-cone) set.lim.pbk$cone)
    (= (SET.FTN$composition [fibration-cone set.lim.pbk$cone-diagram]) fibration-opspan)
    (= (SET.FTN$composition [fibration-cone set.lim.pbk$vertex])
      (SET.FTN$composition [horizontal-source set.ftn$source]))
    (= (SET.FTN$composition [fibration-cone set.lim.pbk$first]) horizontal-source)
    (= (SET.FTN$composition [fibration-cone set.lim.pbk$second]) vertical-source)

(9) (SET$class fibration)
    (SET$subclass fibration quartet)
    (forall (?a (quartet ?a))
        (<=> (fibration ?a)
             (set.ftn$surjection (set.lim.pbk$mediator (fibration-cone ?a)))))
```

- Associated with a fibration α is a *fiber* set quartet *fib*(α) with
  horizontal source *horiz-src*(*fib*(α)) = *fib*(*horiz-src*(α)),
  horizontal target *horiz-tgt*(*fib*(α)) = *fib*(*horiz-tgt*(α)),
  vertical source *vert-src*(*fib*(α)) = *vert-tgt*(α), and
  vertical target *vert-tgt*(*fib*(α)) = ℘*vert-src*(α).

  Without the fibration property, we only have the inequality
  fib(*horiz-src*(α)) · ℘*vert-src*(α) ≤ *vert-tgt*(α) · *fib*(*horiz-tgt*(α)).

  A fibration property is precisely what is needed for equality here. The fiber operation here is like the fiber operation for functions – it flips the direction; in this case, it flips the vertical direction of the quartet.

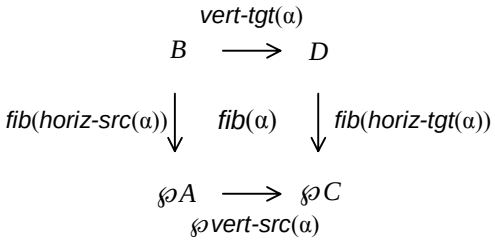


Figure 1: Fiber Function Square

```
(10) (SET.FTN$function fiber)
     (= (SET.FTN$source fiber) fibration)
     (= (SET.FTN$target fiber) quartet)
     (= (SET.FTN$composition [fiber horizontal-source])
        (SET.FTN$composition [horizontal-source set.ftn$fiber]))
     (= (SET.FTN$composition [fiber horizontal-target])
        (SET.FTN$composition [horizontal-target set.ftn$fiber]))
     (= (SET.FTN$composition [fiber vertical-source]) vertical-target)
     (= (SET.FTN$composition [fiber vertical-target])
        (SET.FTN$composition [vertical-source set.ftn$power]))
```

- Two quartets are *horizontally composable* when the horizontal target of the first is equal to the horizontal source of the second. The *horizontal composition* of two horizontally composable quartets is defined in terms of the composition of their vertical source and vertical target functions.

```
(11) (SET.LIM.PBK$opspan horizontally-composable-opspan)
     (= (SET.LIM.PBK$class1 horizontally-composable-opspan) quartet)
     (= (SET.LIM.PBK$class2 horizontally-composable-opspan) quartet)
     (= (SET.LIM.PBK$opvertex horizontally-composable-opspan) set.ftn$function)
     (= (SET.LIM.PBK$first horizontally-composable-opspan) horizontal-target)
     (= (SET.LIM.PBK$second horizontally-composable-opspan) horizontal-source)

(12) (REL$relation horizontally-composable)
     (= (REL$class1 horizontally-composable) quartet)
     (= (REL$class2 horizontally-composable) quartet)
     (= (REL$extent horizontally-composable)
        (SET.LIM.PBK$pullback horizontally-composable-opspan))

(13) (SET.FTN$function horizontal-composition)
     (= (SET.FTN$source horizontal-composition)
        (SET.LIM.PBK$pullback horizontally-composable-opspan))
     (= (SET.FTN$target horizontal-composition) quartet)
     (forall (?a1 (quartet ?a1) ?a2 (quartet ?a2)
             (horizontally-composable ?a1 ?a2))
         (and (= (horizontal-source (horizontal-composition [?a1 ?a2]))
                 (horizontal-source ?a1))
              (= (horizontal-target (horizontal-composition [?a1 ?a2]))
                 (horizontal-target ?a2))
              (= (vertical-source (horizontal-composition [?a1 ?a2]))
                 (set.ftn$composition [(vertical-source ?a1) (vertical-source ?a2)]))
              (= (vertical-target (horizontal-composition [?a1 ?a2]))
                 (set.ftn$composition [(vertical-target ?a1) (vertical-target ?a2)])))))
```

- Horizontal composition satisfies the usual *associative law*.

```
(forall (?a1 (quartet ?a1)
         ?a2 (quartet ?a2)
         ?a3 (quartet ?a3)
         (horizontally-composable ?a1 ?a2)
         (horizontally-composable ?a2 ?a3))
    (= (horizontal-composition [?a1 (horizontal-composition [?a2 ?a3])])
       (horizontal-composition [(horizontal-composition [?a1 ?a2]) ?a3])))
```

- For any function, regarded as a vertical morphism, there is a *horizontal identity* quartet.

```
(14) (SET.FTN$function horizontal-identity)
     (= (SET.FTN$source horizontal-identity) set.ftn$function)
     (= (SET.FTN$target horizontal-identity) quartet)
     (forall (?f (set.ftn$function ?f))
         (and (= (horizontal-source (horizontal-identity ?f)) ?f)
              (= (horizontal-target (horizontal-identity ?f)) ?f)
              (= (vertical-source (horizontal-identity ?f))
                 (set.ftn$identity (set.ftn$source ?f)))
              (= (vertical-target (horizontal-identity ?f))
                 (set.ftn$identity (set.ftn$target ?f))))))
```

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

```
(forall (?a (quartet ?a))
    (and (= (horizontal-composition
                [(horizontal-identity (horizontal-source ?a)) ?a]) ?a)
         (= (horizontal-composition
                [?a (horizontal-identity (horizontal-target ?a))]) ?a)))
```

The horizontal category horiz(□Set) has functions as its objects, quartets as its morphisms, horizontal composition as its composition function and horizontal identity as its identity function.

- Two quartets are *vertically composable* when the vertical target of the first is equal to the vertical source of the second. The *vertical composition* of two vertically composable quartets takes the vertical source of the first and the vertical target of the second.

```
(15) (SET.LIM.PBK$opspan vertically-composable-opspan)
     (= (class1 vertically-composable-opspan) quartet)
     (= (class2 vertically-composable-opspan) quartet)
     (= (opvertex vertically-composable-opspan) set.ftn$function)
     (= (first vertically-composable-opspan) vertical-target)
     (= (second vertically-composable-opspan) vertical-source)

(16) (REL$relation vertically-composable)
     (= (REL$class1 vertically-composable) quartet)
     (= (REL$class2 vertically-composable) quartet)
     (= (REL$extent vertically-composable)
        (SET.LIM.PBK$pullback vertically-composable-opspan))

(17) (SET.FTN$function vertical-composition)
     (= (SET.FTN$source vertical-composition)
        (SET.LIM.PBK$pullback vertically-composable-opspan))
     (= (SET.FTN$target vertical-composition) quartet)
     (forall (?a1 (quartet ?a1) ?a2 (quartet ?a2)
             (vertically-composable ?a1 ?a2))
         (and (= (horizontal-source (vertical-composition [?a1 ?a2]))
                 (set.ftn$composition
                     [(horizontal-source ?a1) (horizontal-source ?a2)]))
              (= (horizontal-target (vertical-composition [?a1 ?a2]))
                 (set.ftn$composition
                     [(horizontal-target ?f1) (horizontal-target ?f2)]))
              (= (vertical-source (vertical-composition [?a1 ?a2]))
                 (vertical-source ?a1))
              (= (vertical-target (vertical-composition [?a1 ?a2]))
                 (vertical-target ?a2))))
```

- Vertical composition satisfies the usual *associative law*.

```
(forall (?a1 (quartet ?a1)
         ?a2 (quartet ?a2)
         ?a3 (quartet ?a3)
         (vertically-composable ?a1 ?a2)
         (vertically-composable ?a2 ?a3))
    (= (vertical-composition [?a1 (vertical-composition [?a2 ?a3])])
       (vertical-composition [(vertical-composition [?a1 ?a2]) ?a3])))
```

- For any function, regarded as a horizontal morphism, there is a *vertical identity* quartet.

```
(18) (SET.FTN$function vertical-identity)
     (= (SET.FTN$source vertical-identity) set.ftn$function)
     (= (SET.FTN$target vertical-identity) quartet)
     (forall (?f (set.ftn$function ?f))
         (and (= (vertical-source (vertical-identity ?f)) ?f)
              (= (vertical-target (vertical-identity ?f)) ?f)
              (= (horizontal-source (vertical-identity ?f))
                 (set.ftn$identity (set.ftn$source ?f)))
              (= (horizontal-target (vertical-identity ?f))
                 (set.ftn$identity (set.ftn$target ?f))))))
```

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

```
(forall (?a (quartet ?a))
    (and (= (vertical-composition
                [(vertical-identity (vertical-source ?a)) ?a]) ?a)
         (= (vertical-composition
                [?a (vertical-identity (vertical-target ?a))]) ?a)))
```

The vertical category vert(□Set) has functions as its objects, quartets as its morphisms, vertical composition as its composition function and vertical identity as its identity function.

- Associated with a quartet α from function f_1 = horiz-src(α) to function f_2 = horiz-tgt(α) with vertical components *vert-src*(α) and *vert-tgt*(α), is an function of *signatures*

  $sign(\alpha) : sign(f_1) \to sign(f_2)$,

  which is defined to the composite:

  $sign(\alpha) = sign\text{-}arity(f_1) \cdot \wp\, vert\text{-}src(\alpha) \cdot sign\text{-}assign(f_2)$.

  Clearly, this signature operator is oriented along the horizontal dimension. Also, this function does not necessarily preserve arity (up to isomorphism), since the function $\wp\, vert\text{-}src(\alpha)$, although surjective in its elementwise restriction $vert\text{-}src(\alpha)\lfloor_{arity(t)}$ for each signature $t \in sign(f_1)$, is not necessarily bijective. Note however, the variable function of the type language of a model, which is the vertical source of the reference quartet for that language, is a bijection. Thus the signature function for these special type languages does preserve arity (up to isomorphism).

The signature function associated with any quartet is the image of the morphism function of the signature functor applied to the quartet:

$sign : horiz(\square \mathrm{Set}) \to \mathrm{Set}$.

One use for functions is as reference: the source is a set of names, the target is a set of objects, and the function represents the referencing of an object by a name. We give a frame example later that makes use of this paradigm. Here we point out some properties of quartets, interpreted as morphisms of references.

A signature $t \in sign(f_1)$ is a function $t : arity(t) \to tgt(f_1)$. Letting $arity(t) = \{x_1, x_2, \ldots, x_n\}$ with $t(x_n) = t_n$, we can think of a signature as a tuple $t = (t_1, t_2, \ldots, t_n)$. The natural number indexing here is just a device to display the signature function in linear form. It is not necessary in the abstract reality. Next, let us make the following definitions:

$arity(s) = \wp(vert\text{-}src(\alpha))(arity(t))$,

$s = sign(\alpha)(t)$, and

$vert\text{-}src(\alpha)\lfloor_{arity(t)}$ is the restriction of $vert\text{-}src(\alpha)$ to $arity(t)$.

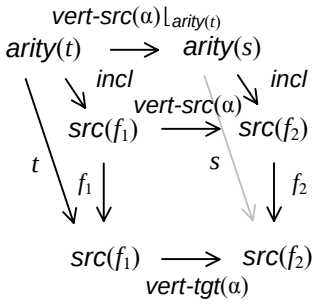


Diagram 1: Signature Function

Thus, the signature $s \in sign(f_2)$, a function $s : arity(s) \to tgt(f_2)$, is the image of the signature t along the function $sign(\alpha) : sign(f_1) \to sign(f_2)$; that is, $sign(\alpha)(t) = s$. We know that $t = incl_{arity(t),\, src(f1)} \cdot f_1$ and $s = incl_{arity(s),\, src(f2)} \cdot f_2$. Therefore, we end up with the commutative Diagram 1. In particular, the back commuting rectangle expresses the following constraint between signatures:

$t \cdot vert\text{-}tgt(\alpha) = vert\text{-}src(\alpha)\lfloor_{arity(t)} \cdot s$.

In the frame interpretation mentioned above, and elaborated later, we view a signature $t = (t_1, t_2, \ldots, t_n)$ to be much like a frame with the elements in $arity(t)$ functioning as frame slot names (attribute names) and elements in $tgt(f_1)$ functioning as frame slot fillers (attribute values). For the mapping by $sign(\alpha)$ from t to s there are several cases to discuss.

– Suppose the restriction $vert\text{-}src(\alpha)\lfloor_{arity(t)}$ maps two distinct slot names $x \neq x'$ with $x, x' \in arity(t)$ onto the same slot name $y \in arity(s)$: $vert\text{-}src(\alpha)(x) = y$ and $vert\text{-}src(\alpha)(x') = y$. The above signature constraint indicates two ways that this can occur.

  * The signature t fills the two slots with the same object: $t(x) = t(x') \in tgt(f_1)$.

  * The signature t fills the two slots with different objects in $tgt(f_1)$, but the function $vert\text{-}tgt(\alpha)$ maps these to the same object:

    $vert\text{-}tgt(\alpha)(t(x)) = vert\text{-}tgt(\alpha)(t(x')) \in tgt(f_2)$.

– Suppose the restriction $\textit{vert-src}(\alpha)\lfloor_{\textit{arity}(t)}$ maps two distinct slot names $x \neq x'$ with $x, x' \in \textit{arity}(t)$ onto two distinct slot names $y \neq y'$ with $y, y' \in \textit{arity}(s)$: $\textit{vert-src}(\alpha)(x) = y$ and $\textit{vert-src}(\alpha)(x') = y'$. Also suppose that the signature t maps the two slot names x and x' to the same slot filler: $t(x) = t(x')$. The above signature constraint indicates only one way that this can occur.

* The signature s fills the two slots y and y' with the same object in $\textit{tgt}(f_2)$.

```
(19) (SET.FTN$function signature)
     (= (SET.FTN$source signature) quartet)
     (= (SET.FTN$target signature) set.ftn$function)
     (= (SET.FTN$composition [signature set.ftn$source])
        (SET.FTN$composition [horizontal-source set.ftn$signature]))
     (= (SET.FTN$composition [signature set.ftn$target])
        (SET.FTN$composition [horizontal-target set.ftn$signature]))
     (forall (?a (quartet ?a))
         (= (signature ?a)
            (set.ftn$composition
                [(set.ftn$arity (horizontal-source ?a))
                 (set.ftn$composition
                     [(set.ftn$power (vertical-source ?a))
                      (set.ftn$assign (horizontal-target ?a))])])))
```

○

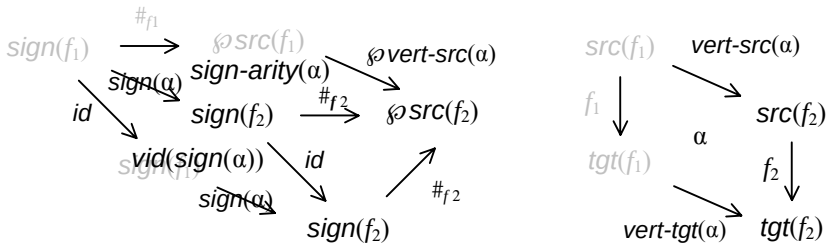


Figure 3: Type language morphism of a quartet

○ Any set quartet α from function $f_1 = \text{horiz-src}(\alpha)$ to function $f_2 = \text{horiz-tgt}(\alpha)$ with vertical components $\textit{vert-src}(\alpha)$ and $\textit{vert-tgt}(\alpha)$, defines a *type language morphism* (Figure 3), whose type quartet is α itself and whose signature quartet is the vertical identity quartet of the signature function $\textit{sign}(\alpha) : \textit{sign}(f_1) \to \textit{sign}(f_2)$. This language morphism has the vertical source $\textit{vert-src}(\alpha)$ as its variable function, the vertical target $\textit{vert-tgt}(\alpha)$ as its entity type function, and the signature function $\textit{sign}(\alpha)$ as its relation type fuction. Clearly, the type quartet of the language morphism of a quartet is itself.

```
(20) (SET.FTN$function language-morphism)
     (= (SET.FTN$source language-morphism) quartet)
     (= (SET.FTN$target language-morphism) lang.mor$language-morphism)
     (forall (?p (quartet ?p))
         (and (= (lang.mor$type (language-morphism ?p)) ?p)
              (= (lang.mor$signature (language-morphism ?p))
                 (set.ftn$vertical-identity (signature ?p)))))
```

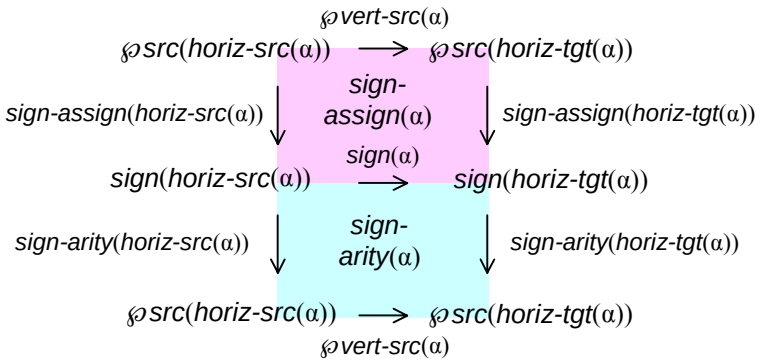


Figure 1: The assign and arity quartets

- The signature notion allows us to define functions that map a quartet α to two quartets (Figure 1), *sign-assign*(α) and *sign-arity*(α), that are vertical inverses of each other.

```
(21) (SET.FTN$function signature-assign)
     (= (SET.FTN$source signature-assign) quartet)
     (= (SET.FTN$target signature-assign) quartet)
     (= (SET.FTN$composition [signature-assign horizontal-source])
        (SET.FTN$composition [horizontal-source set.ftn$signature-assign]))
     (= (SET.FTN$composition [signature-assign horizontal-target])
        (SET.FTN$composition [horizontal-target set.ftn$signature-assign]))
     (= (SET.FTN$composition [signature-assign vertical-source])
        (SET.FTN$composition [vertical-source set.ftn$power]))
     (= (SET.FTN$composition [signature-assign vertical-target]) signature)

(22) (SET.FTN$function signature-arity)
     (= (SET.FTN$source signature-arity) quartet)
     (= (SET.FTN$target signature-arity) quartet)
     (= (SET.FTN$composition [signature-arity horizontal-source])
        (SET.FTN$composition [horizontal-source set.ftn$signature-arity]))
     (= (SET.FTN$composition [signature-arity horizontal-target])
        (SET.FTN$composition [horizontal-target set.ftn$signature-arity]))
     (= (SET.FTN$composition [signature-arity vertical-source]) signature)
     (= (SET.FTN$composition [signature-arity vertical-target])
        (SET.FTN$composition [vertical-source set.ftn$power]))

     (forall (?a (quartet ?a))
        (and (= (vertical-composition [(signature-arity ?a) (signature-assign ?a)])
                (set.ftn$vertical-identity (signature ?a)))
             (= (vertical-composition [(signature-assign ?a) (signature-arity ?a)])
                (set.ftn$vertical-identity (set.ftn$power (vertical-source ?a))))))
```

- Each defines a functor *sign-arity*, *sign-assign* : △Set → *horiz*(□Set). But any functor whose target is *horiz*(□Set) splits into two Set-functors with a natural transformation between them.

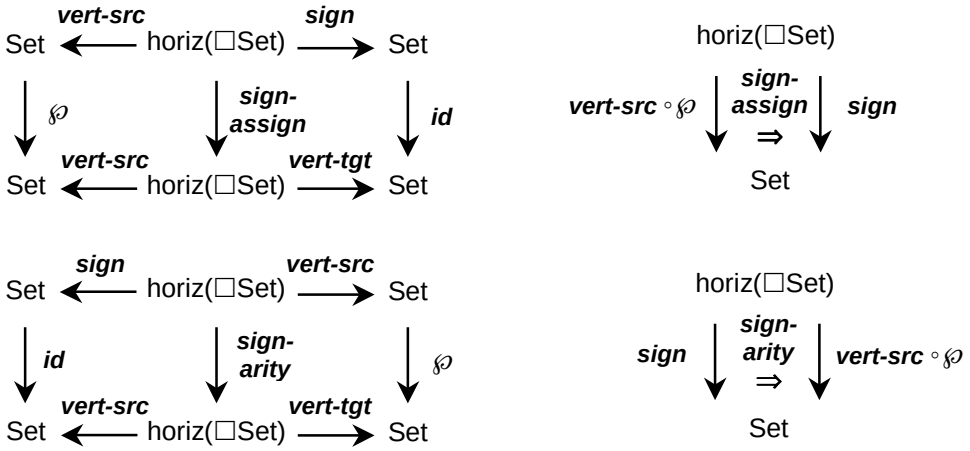


Figure 1: Signature, Assign and Arity Functors and Natural Isomorphisms

○ For any quartet $\alpha = \langle \mathit{vert\text{-}src}(\alpha), \mathit{vert\text{-}tgt}(\alpha) \rangle : \mathit{horiz\text{-}src}(\alpha) \to \mathit{horiz\text{-}tgt}(\alpha)$, from function $f_1 = \mathit{horiz}\text{-src}(\alpha)$ to function $f_2 = \mathit{horiz}\text{-tgt}(\alpha)$ with vertical components $\mathit{vert\text{-}src}(\alpha)$ and $\mathit{vert\text{-}tgt}(\alpha)$, the coproduct *arity morphism* $\mathit{arity\text{-}mor}(\alpha) : \mathit{arity}(f_1) \to \mathit{arity}(f_2)$ is compatible with the signature arity quartet – the quartet of the coproduct arity morphism is the signature arity quartet.

```
(23) (SET.FTN$function arity-morphism)
     (= (SET.FTN$source arity-morphism) quartet)
     (= (SET.FTN$target arity-morphism) set.col.art.mor$arity)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$source])
        (SET.FTN$composition [horizontal-source set.col.art$arity]))
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$target])
        (SET.FTN$composition [horizontal-target set.col.art$arity]))
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$index]) signature)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$base]) vertical-source)

     (= (SET.FTN$composition [arity-morphism set.col.art.mor$quartet]) signature-arity)
```

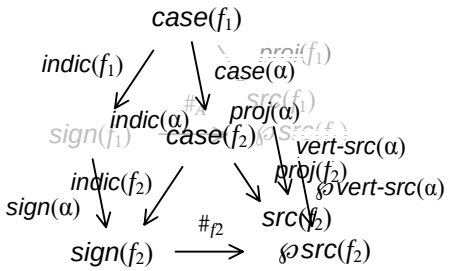


Figure 1: Coproduct of the Arity of a Quartet

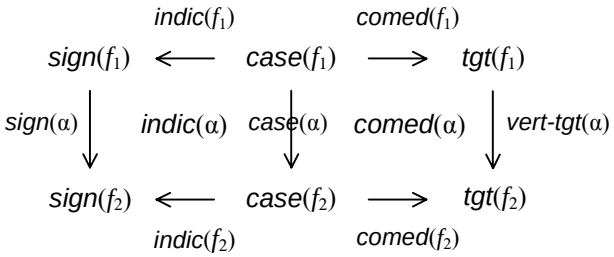


Figure 2: Spangraph Morphism of a Quartet

○ Any quartet $\alpha = \langle \mathit{vert\text{-}src}(\alpha), \mathit{vert\text{-}tgt}(\alpha) \rangle : \mathit{horiz\text{-}src}(\alpha) \to \mathit{horiz\text{-}tgt}(\alpha)$ from function $f_1 = \mathit{horiz}\text{-src}(\alpha)$ to function $f_2 = \mathit{horiz}\text{-tgt}(\alpha)$ with vertical components $\mathit{vert\text{-}src}(\alpha)$ and $\mathit{vert\text{-}tgt}(\alpha)$, defines a *case* or *role* function $\mathit{case}(\alpha) : \mathit{case}(f_1) = \sum \mathit{arity}(f_1) \to \sum \mathit{arity}(f_2) = \mathit{case}(f_2)$. The pointwise definition is:

$$\mathit{case}(\alpha)((t, x)) = (\mathit{sign}(\alpha)(t), \mathit{vert\text{-}src}(\alpha)(x))$$

for any signature $t \in \mathit{sign}(A)$ and any name $x \in \mathit{arity}(f_1)(t)$. This is well defined, since α preserves signature arity. The abstract definition is in terms of the coproduct of the arity morphism.

```
(24) (SET.FTN$function case)
     (= (SET.FTN$source case) quartet)
     (= (SET.FTN$target case) set.ftn$function)
     (forall (?p (quartet ?p))
```

```
(and (= (set.ftn$source (case ?p)) (set.ftn$case (horizontal-source ?p)))
     (= (set.ftn$target (case ?p)) (set.ftn$case (horizontal-target ?p)))
     (= (case ?p)
        (set.col.art.mor$coproduct (arity-morphism ?p)))))
```

- The case function is the vertical source for two quartets: an *indication* quartet ***indic***(α) and a *projection* quartet ***proj***(α).
  - The commutativity ***case***(α) · ***indic***(*B*) = ***indic***(*A*) · ***sign***(α), a property of the coproduct of arities (preservation of index), is obvious from the pointwise definition of the case function.
  - The commutativity ***case***(α) · ***proj***(*B*) = ***proj***(*A*) · ***vert-tgt***(α), a property of the coproduct of arities (preservation of projection), is obvious from the pointwise definition of the case function.

  Abstractly by the preservation of signature arity, the index quartet is a fibration: for any signature $t \in \mathit{sign}(f_1)$ and any name $y \in \mathit{arity}(f_2)(\mathit{sign}(\alpha)(t)) = \wp\mathit{vert\text{-}src}(\alpha)(\mathit{arity}(f_1)(t))$ there is a name $x \in \mathit{arity}(f_1)(t)$ such that $\mathit{vert\text{-}src}(\alpha)(x) = y$.

```
(25) (SET.FTN$function indication)
     (= (SET.FTN$source indication) quartet)
     (= (SET.FTN$target indication) fibration)
     (forall (?p (quartet ?p))
         (and (= (horizontal-source (indication ?p))
                 (set.ftn$indication (horizontal-source ?p)))
              (= (horizontal-target (indication ?p))
                 (set.ftn$indication (horizontal-target ?p)))
              (= (vertical-source (indication ?p)) (case ?p))
              (= (vertical-target (indication ?p)) (signature ?p))))

(26) (SET.FTN$function projection)
     (= (SET.FTN$source projection) quartet)
     (= (SET.FTN$target projection) quartet)
     (forall (?p (quartet ?p))
         (and (= (horizontal-source (projection ?p))
                 (set.ftn$projection (horizontal-source ?p)))
              (= (horizontal-target (projection ?p))
                 (set.pr$projection (horizontal-target ?p)))
              (= (vertical-source (projection ?p)) (case ?p))
              (= (vertical-target (projection ?p)) (vertical-source ?p))))
```

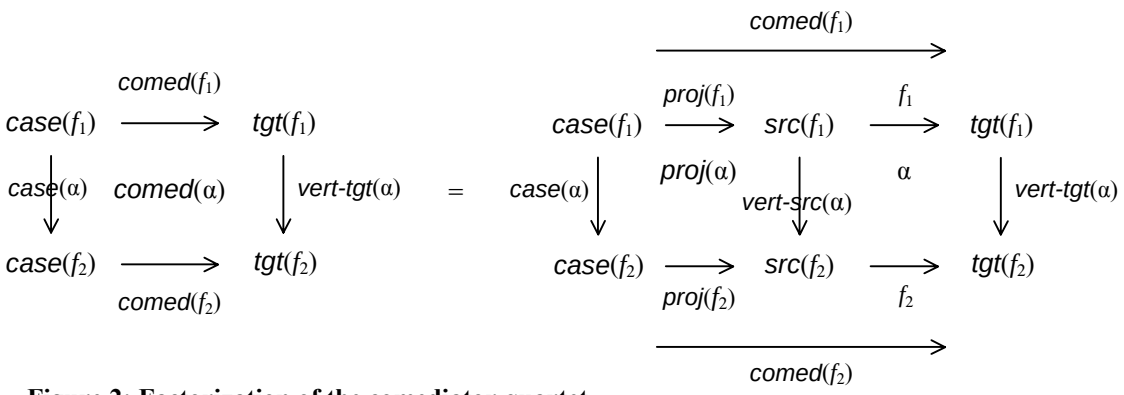


Figure 2: Factorization of the comediator quartet

- Any quartet $\alpha = \langle \mathit{vert\text{-}src}(\alpha), \mathit{vert\text{-}tgt}(\alpha) \rangle : \mathit{horiz\text{-}src}(\alpha) \to \mathit{horiz\text{-}tgt}(\alpha)$, from function $f_1 = \mathit{horiz}\text{-}\mathrm{src}(\alpha)$ to function $f_2 = \mathit{horiz}\text{-}\mathrm{tgt}(\alpha)$ with vertical components ***vert-src***(α) and ***vert-tgt***(α), defines a *comediator* quartet $\tilde{*}_\alpha = \mathit{comed}(\alpha)$.
  - The commutativity ***case***(α) · ***comed***(f_2) = ***comed***(f_1) · ***vert-tgt***(α) holds by a property of the coproduct of arities (preservation of cotupling).

  Pointwise, commutativity states that

  $\mathit{sign}(\alpha)(t)(\mathit{vert\text{-}src}(\alpha)(x)) = \mathit{vert\text{-}tgt}(\alpha)(t(x))$

  for any signature $t \in \mathit{sign}(f_1)$ and any name $x \in \mathit{arity}(f_1)(t)$, which is true by preservation of signature.

```
(27) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) quartet)
     (= (SET.FTN$target comediator) quartet)
     (forall (?p (quartet ?p))
         (and (= (horizontal-source (comediator ?p))
                 (set.ftn$comediator (horizontal-source ?p)))
              (= (horizontal-target (comediator ?p))
                 (set.ftn$comediator (horizontal-target ?p)))
              (= (vertical-source (comediator ?p)) (case ?p))
              (= (vertical-target (comediator ?p)) (vertical-target ?p))))
```

- Associated with any quartet α is a spangraph morphism

  $sgph(\alpha) = \langle 1^{st}_{sgph(\alpha)}, 2^{nd}_{sgph(\alpha)} \rangle : sgph(f_1) \to sgph(f_2)$

  with the two quartets (the 1st is a fibration) $1^{st}_{sgph(\alpha)}(\alpha) = indic(\alpha)$ and $2^{nd}_{sgph(\alpha)}(\alpha) = comed(\alpha)$.

```
(28) (SET.FTN$function spangraph-morphism)
     (= (SET.FTN$source spangraph-morphism) quartet)
     (= (SET.FTN$target spangraph-morphism) sgph.mor$spangraph-morphism)
     (forall (?p (quartet ?p))
         (and (= (sgph.mor$source (spangraph-morphism ?p))
                 (set.ftn$spangraph (horizontal-source ?p)))
              (= (sgph.mor$target (spangraph-morphism ?p))
                 (set.ftn$spangraph (horizontal-target ?p)))
              (= (sgph.mor$first (spangraph-morphism ?p)) (indication ?p))
              (= (sgph.mor$second (spangraph-morphism ?p)) (comediator ?p))))
```

> The spangraph morphism associated with any quartet is the image of the morphism function of the spangraph functor applied to the quartet:
>
> $sgph$: horiz(□Set) → Spangraph.

Special Set Quartets

`set.qtt.spl`

- A quartet α is *special* when the vertical source function is a bijection. Special set quartets need their own namespace. The terminology for special set quartets includes at least vertical-source, vertical-target, horizontal-source, horizontal-target, etc. Since horizontal identities are special and special set quartets are closed under horizontal composition, special type language morphisms also form a horizontal subcategory of quartets. And composition and identity functions should be included in the special namespace.

```
(1) (SET$class special)
    (SET$subclass special set.qtt$quartet)

(2) (SET.FTN$function horizontal-source)
    (= (SET.FTN$source horizontal-source) special)
    (= (SET.FTN$target horizontal-source) set.ftn$function)
    (SET.FTN$restriction horizontal-source set.qtt$horizontal-source)

(3) (SET.FTN$function horizontal-target)
    (= (SET.FTN$source horizontal-target) special)
    (= (SET.FTN$target horizontal-target) set.ftn$function)
    (SET.FTN$restriction horizontal-target set.qtt$horizontal-target)

(4) (SET.FTN$function vertical-source)
    (= (SET.FTN$source vertical-source) special)
    (= (SET.FTN$target vertical-source) set.ftn$function)
    (SET.FTN$restriction vertical-source set.qtt$vertical-source)

(5) (SET.FTN$function vertical-target)
    (= (SET.FTN$source vertical-target) special)
    (= (SET.FTN$target vertical-target) set.ftn$function)
    (SET.FTN$restriction vertical-target set.qtt$vertical-target)

    (forall (?h (set.qtt$quartet ?h))
        (<=> (special ?h) (bijection (vertical-source ?h))))
```

-

-

- A special quartet α has an associated (underlying) semiquartet *sqtt*(α) between the set pairs underlying its horizontal source and target functions, since the vertical source function is bijective.

```
(6) (SET.FTN$function semiquartet)
    (= (SET.FTN$source semiquartet) special)
    (= (SET.FTN$target semiquartet) seth.sqtt$semiquartet)
```

...

Function Pairs

`set.ftn.pr`

Set pairs are also related through function pairs. A function pair is a pair of functions. The classes of set pairs and function pairs form the category $\mathsf{Set}^{\mathbf{2}}$ called the *square* of Set. The category $\triangle\mathsf{Set}$ is a subcategory: $\triangle\mathsf{Set} \subseteq \mathsf{Set}^{\mathbf{2}}$. The category $\mathsf{Set}^{\mathbf{2}}$ underlies the category of classifications and designations.

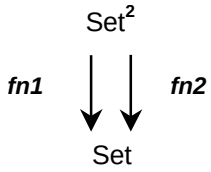


Diagram 1: The component functors

- A *function pair* (set pair morphism)

  $\alpha = \langle fn1(\alpha), fn2(\alpha) \rangle : src(\alpha) \Rightarrow tgt(\alpha)$

  from set pair src(α) to set pair tgt(α) with components *fn1*(α) and *fn2*(α), consists of a diagram (Figure 1) of these two functions, where the two functions suitably match at source and target sets

  $src(fn1(\alpha)) = set1(src(\alpha))$,

  $tgt(fn1(\alpha)) = set1(tgt(\alpha))$,

  $src(fn2(\alpha)) = set2(src(\alpha))$, and

  $tgt(fn2(\alpha)) = set2(tgt(\alpha))$.

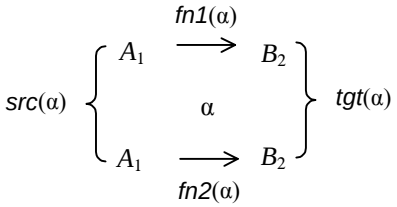


Figure 1: Function Pair

```
(1) (SET$class pair)

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

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) pair)
    (= (SET.FTN$target target) set.pr$pair)

(4) (SET.FTN$function function1)
    (= (SET.FTN$source function1) pair)
    (= (SET.FTN$target function1) set.ftn$function)
    (= (SET.FTN$composition [function1 set.ftn$source])
       (SET.FTN$composition [source set.pr$set1]))
    (= (SET.FTN$composition [function1 set.ftn$target])
       (SET.FTN$composition [target set.pr$set1]))

(5) (SET.FTN$function function2)
    (= (SET.FTN$source function2) pair)
    (= (SET.FTN$target function2) set.ftn$function)
    (= (SET.FTN$composition [function2 set.ftn$source])
       (SET.FTN$composition [source set.pr$set2]))
    (= (SET.FTN$composition [function2 set.ftn$target])
       (SET.FTN$composition [target set.pr$set2]))
```

Function pairs form the morphism class of the $\mathsf{Set}^{\mathbf{2}}$ category.

- Function pairs are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable function pairs is defined in terms of the composition of their component functions.

```
(6) (SET.LIM.PBK$opspan composable-opspan)
```

```
      (= (SET.LIM.PBK$class1 composable-opspan) pair)
      (= (SET.LIM.PBK$class2 composable-opspan) pair)
      (= (SET.LIM.PBK$opvertex composable-opspan) set.pr$pair)
      (= (SET.LIM.PBK$first composable-opspan) target)
      (= (SET.LIM.PBK$second composable-opspan) source)

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

(8) (SET.FTN$function composition)
    (= (SET.FTN$source composition)
       (SET.LIM.PBK$pullback composable-opspan))
    (= (SET.FTN$target composition) pair)
    (forall (?p1 (pair ?p1) ?p2 (pair ?p2) (composable ?p1 ?p2))
        (and (= (source (composition [?p1 ?p2])) (source ?p1))
             (= (target (composition [?p1 ?p2])) (target ?p2))
             (= (function1 (composition [?p1 ?p2]))
                (set.ftn$composition [(function1 ?p1) (function1 ?p2)]))
             (= (function2 (composition [?p1 ?p2]))
                (set.ftn$composition [(function2 ?p1) (function2 ?p2)])))))
```

- Composition satisfies the usual *associative law*.

```
(forall (?p1 (pair ?p1) ?p2 (pair ?p2) ?p3 (pair ?p3)
         (composable ?p1 ?p2) (composable ?p2 ?p3))
    (= (composition [?p1 (composition [?p2 ?p3])])
       (composition [(composition [?p1 ?p2]) ?p3])))
```

- For any function, regarded as a vertical morphism, there is an *identity* pair.

```
(9) (SET.FTN$function identity)
    (= (SET.FTN$source identity) set.pr$pair)
    (= (SET.FTN$target identity) pair)
    (forall (?a (set.pr$pair ?a))
        (and (= (source (identity ?a)) ?a)
             (= (target (identity ?a)) ?a)
             (= (function1 (identity ?a)) (set.ftn$identity (set.pr$set1 ?a)))
             (= (function2 (identity ?a)) (set.ftn$identity (set.pr$set2 ?a)))))
```

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

```
(forall (?p (pair ?p))
    (and (= (composition [(identity (source ?p)) ?p]) ?p)
         (= (composition [?p (identity (target ?p))]) ?p)))
```

> The category $\mathsf{Set}^{\mathbf{2}}$ has set pairs as its objects, function pairs as its morphisms, composition of function pairs as its composition and identity pairs as identities.

- Given a classification $\boldsymbol{A}$ and a function pair $\alpha = \langle \mathit{fn1}(\alpha), \mathit{fn2}(\alpha) \rangle : A \rightrightarrows \mathit{pr}(\boldsymbol{A})$ whose target is the underlying set pair of the classification there is an *isotaxy* $\mathit{iso}((\alpha, \boldsymbol{A})) = \langle \mathit{fn1}(\alpha), \mathit{fn2}(\alpha) \rangle : \mathit{src}(\boldsymbol{p}) \rightrightarrows \mathit{tgt}(\boldsymbol{p})$ with that classification as its target and that function pair underlying it.

```
(10 (SET.LIM.PBK$opspan design-opspan)
     (= (class1 design-opspan) pair)
     (= (class2 design-opspan) cls$classification)
     (= (opvertex design-opspan) set.pr$pair)
     (= (first design-opspan) target)
     (= (second design-opspan) cls$pair)

(11) (REL$relation designable)
     (= (REL$class1 designable) pair)
     (= (REL$class2 designable) cls$classification)
     (= (REL$extent designable) (SET.LIM.PBK$pullback design-opspan))

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

```
(= (SET.FTN$source isotaxy) (SET.LIM.PBK$pullback design-opspan))
(= (SET.FTN$target isotaxy) cls.dsgn$isotaxy)
(forall (?p (pair ?p) ?a (cls$classification ?a) (designable ?p ?a))
    (and (= (cls.dsgn$target (isotaxy [?p ?a])) ?a)
         (= (cls.dsgn$pair (isotaxy [?p ?a])) ?p)
         (= (cls$instance (cls.dsgn$source (isotaxy [?p ?a])))
            (set.pr$set2 (source ?p1)))
         (= (cls$type (cls.dsgn$source (isotaxy [?p ?a])))
            (set.pr$set1 (source ?p)))
         (forall (?x2 ((set.pr$set2 (source ?p)) ?x2)
                  ?x1 ((set.pr$set1 (source ?p)) ?x1))
             (<=> ((cls.dsgn$source (isotaxy [?p ?a])) ?x2 ?x1)
                  (?a ((function2 ?p) ?x2) ((function1 ?p) ?x1))))))
```

The Namespace of Diagrams

dgm

- A (Set-) *diagram D* is essentially a large graph morphism $\boldsymbol{D} : \boldsymbol{G} \rightarrow |\mathsf{Set}|$ from a small *shape* graph $\boldsymbol{G}$ into the large graph |Set| of sets and functions. The object (node) function of this graph morphism $obj(\boldsymbol{D})$ is here called *set*, and the morphism (edge) function of this graph morphism $mor(\boldsymbol{D})$ is here called *function*. Diagrams are determined by their *shape*, *set* and *function*. Set diagrams correspond to Set functors, and hence all diagrams form a KIF collection. Note that the set function of a diagram is a SET tuple set (a special case of a KIF tuple class).

```
(1) (KIF$collection diagram)

(2) (KIF$function shape)
    (= (KIF$source shape) diagram)
    (= (KIF$target shape) gph$graph)

(3) (KIF$function set)
    (= (KIF$source set) diagram)
    (= (KIF$target set) SET.FTN$function)
    (forall (?d (diagram ?d))
        (and (= (SET.FTN$source (set ?d)) (gph$node (shape ?d)))
             (= (SET.FTN$target (set ?d)) set$set)))

(4) (KIF$function function)
    (= (KIF$source function) diagram)
    (= (KIF$target function) SET.FTN$function)
    (forall (?d (diagram ?d))
        (and (= (SET.FTN$source (function ?d)) (gph$edge (shape ?d)))
             (= (SET.FTN$target (function ?d)) set.ftn$function)))
             (= (SET.FTN$composition [(function ?d) set.ftn$source])
                (SET.FTN$composition [(gph$source (shape ?d)) (set ?d)]))
             (= (SET.FTN$composition [(function ?d) set.ftn$target])
                (SET.FTN$composition [(gph$target (shape ?d)) (set ?d)])))))

(5) (forall (?d1 (diagram ?d1) ?d2 (diagram ?d2))
        (=> (and (= (shape ?d1) (shape ?d2))
                 (= (set ?d1) (set ?d2)))
                 (= (function ?d1) (function ?d2)))
            (= ?d1 ?d2)))
```

- Any diagram has an associated *tuple set* defined by its set component.

```
(6) (KIF$function tuple-class)
    (= (KIF$source tuple-class) diagram)
    (= (KIF$target tuple-class) set.ftn$tuple-set)
    (forall (?d (diagram ?d))
        (and (= (set.ftn$arity (tuple-set ?d)) (gph$node (shape ?d)))
             (= (tuple-set ?d) (set ?d))))
```

  - A diagram *D* is *discrete* when it is the diagram induced by a tuple set *A*.

```
(7) (KIF$collection discrete)
    (KIF$subcollection discrete diagram)
    (forall (?d (diagram ?d))
        (<=> (discrete ?d)
             (exists (?a (set.ftn$tuple-class ?a))
                 (= ?d (set.ftn$diagram ?a)))))
```