The IFF Lower Classification (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.

Need new notions axiomatized here!

- ○ A *hemidesignation* (`cls.hdsgn`) is like a semidesignation – it has a (source) set and a (target) classification, except that the type component is (like the instance component) only a set pair not a set function. A tuple classification for hemidesignations can still be defined – just use type tuples (not signatures) and instance tuples, with classification as in semidesignations.

$$\begin{tikzcd} set(\boldsymbol{p}) \arrow[d, dotted, "\boldsymbol{p}"'] \\ cls(\boldsymbol{p}) \end{tikzcd}$$

Figure: 1 Hemidesignation – abstract

$$\begin{array}{c} \overbrace{set(\boldsymbol{p}) \quad typ(cls(\boldsymbol{p}))}^{typ(\boldsymbol{p})} \\ = \qquad\qquad \Big|\; \models_{cls(\boldsymbol{p})} \\ \underbrace{set(\boldsymbol{p}) \quad inst(cls(\boldsymbol{p}))}_{inst(\boldsymbol{p})} \end{array}$$

Figure 2: Hemidesignation – details

- ○ A *classification span* (`cls.spn`) has three classifications (vertex, first = classification1, second = classification2) and two designations (source = infmorphism1 and target = infomorphism2). A classification span has both a type aspect and an instance aspect. The type aspect is the set span of types, and the insrtance aspect is the set span of instances.
- ○ A *classification hypergraph* (`cls.hgph`) has a reference hemidesignation whose source is the set of names (which could be thought of as an identity classification) and whose target is the node classification, and a signature designation whose source is the relation classification and whose target is the tuple classification of the reference hemidesignation. A classification hypergraph has both type and instance aspects. The type aspect is the set hypergraph of types, and the instance aspect is the set hypergraph of instances.

The Namespace of Classifications

Classifications

`cls`

- A classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ (see Figure 1) consists of a set of instances $\mathit{inst}(\boldsymbol{A})$, a set of types $\mathit{typ}(\boldsymbol{A})$, and an incidence or classification relation between instances and types $\models_{\boldsymbol{A}}$. A classification is the same thing as a binary relation, except that the names of the coordinate domains have been changed to reflect the intended applications: "object$_1$" to "instance" and "object$_2$" to "type". The incidence function maps a classification to the corresponding binary relation.

$$\begin{array}{c} \mathit{typ}(\boldsymbol{A}) \\ \Big| \; \models_{\boldsymbol{A}} \\ \mathit{inst}(\boldsymbol{A}) \end{array}$$

Figure 1: Classification

  The following is a KIF representation for the elements of a classification. Classifications are specified by declaration and population. The elements in the KIF representation are useful for the declaration and population of a classification. The term '`classification`' specified in axiom (1) represents the object aspect of the category Classification – it allows one to *declare* classifications. The terms '`instance`' and '`type`' specified in axioms (2–3) and the term '`incidence`' specified in axiom (4) resolve classifications into their parts, thus allowing one to *populate* classifications. Axiom (5) states that classifications are determined by their (instance, type, incidence) triple.

```
(1) (SET$class classification)
    (= classification rel$relation)

(2) (SET.FTN$function instance)
    (= (SET.FTN$source instance) classification)
    (= (SET.FTN$target instance) set$set)
    (= instance rel$object1)

(3) (SET.FTN$function type)
    (= (SET.FTN$source type) classification)
    (= (SET.FTN$target type) set$set)
    (= type rel$object2)

(4) (SET.FTN$function incidence)
    (= (SET.FTN$source incidence) classification)
    (= (SET.FTN$target incidence) set$relation)
    (= incidence rel$extent)
```

- To quote (Barwise and Seligman, 1997), "in any classification, we think of the types as classifying the instances, but it is often useful to think of the instances as classifying the types." For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, the *opposite* or *dual* of $\boldsymbol{A}$ is the classification $\boldsymbol{A}^{\perp} = \langle \mathit{typ}(\boldsymbol{A}), \mathit{inst}(\boldsymbol{A}), \models_{\boldsymbol{A}}^{\perp} \rangle$ whose instances are types of $\boldsymbol{A}$ and whose types are instances of $\boldsymbol{A}$, and whose incidence is: $t \models^{\perp} i$ when $i \models t$. Axiom (6) specifies the opposite operator on classifications.

```
(6) (SET.FTN$function opposite)
    (= (SET.FTN$source opposite) classification)
    (= (SET.FTN$target opposite) classification)
    (= opposite rel$opposite)
```

- Associated with any classification is a function that produces the *intent* of an instance and a function that produces the *extent* of a type, both within the context of the classification. The intent of an instance $i \in \mathit{inst}(\boldsymbol{A})$ in a classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ is defined by

  $$\mathit{intent}_{\boldsymbol{A}}(i) = \{t \in \mathit{typ}(\boldsymbol{A}) \mid i \models_{\boldsymbol{A}} t\}.$$

  Dually, the extent of a type $t \in \mathit{typ}(\boldsymbol{A})$ in a classification $\boldsymbol{A}$ defined by

  $$\mathit{extent}_{\boldsymbol{A}}(t) = \{i \in \mathit{inst}(\boldsymbol{A}) \mid i \models_{\boldsymbol{A}} t\}.$$

  The axiom for extent shows that the relative instantiation-predication represented by '`incidence`' is compatible with the absolute KIF instantiation-predication. Axiom (7) specifies the intent function in terms of the direct fiber function '`rel$fiber12`' and axiom (8) specifies the extent function in terms of the inverse fiber function '`rel$fiber21`'.

```
(7) (SET.FTN$function intent)
    (= (SET.FTN$source intent) classification)
    (= (SET.FTN$target intent) set.ftn$function)
    (= (SET.FTN$composition [intent set.ftn$source]) instance)
    (= (SET.FTN$composition [intent set.ftn$target])
       (SET.FTN$composition [type set$power]))
    (= intent rel$fiber12)

(8) (SET.FTN$function extent)
    (= (SET.FTN$source extent) classification)
    (= (SET.FTN$target extent) set.ftn$set-valued)
    (= (SET.FTN$composition [extent set.ftn$source]) type)
    (= (SET.FTN$composition [extent set.ftn$base]) instance)
    (= (SET.FTN$composition [extent set.ftn$induction])
       (SET.FTN$identity classification))
```

- For any classification $\boldsymbol{A} = \langle inst(\boldsymbol{A}), typ(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$, two instances $i_1, i_2 \in inst(\boldsymbol{A})$ are *indistinguishable* in $\boldsymbol{A}$ (Barwise and Seligman, 1997), written $i_1 \sim_{\boldsymbol{A}} i_2$, when $intent_{\boldsymbol{A}}(i_1) = intent_{\boldsymbol{A}}(i_2)$. Two types $t_1, t_2 \in typ(\boldsymbol{A})$ are *coextensive* in $\boldsymbol{A}$, written $t_1 \sim_{\boldsymbol{A}} t_2$, when $extent_{\boldsymbol{A}}(t_1) = extent_{\boldsymbol{A}}(t_2)$. A classification A is *separated* when there are no distinct indistinguishable instances, and *extensional* when there are no distinct coextensive types.

  The term '`indistinguishable`' of axiom (9) is specified in terms of the first component equivalence relation '`equivalence1`' associated with the binary incidence relation, and the term '`coextensive`' of axiom (10) is specified in terms of the second component equivalence relation '`equivalence2`' associated with the binary incidence relation. These terms represent the Information Flow notions of type *coextension* and instance *indistinguishability*, respectively. The terms '`extensional`' and '`separated`', that are specified in axioms (11–12) in terms of SET equalizers, represent the Information Flow notions of classification *extensionality* and *separateness*, respectively.

```
(9) (SET.FTN$function indistinguishable)
    (= (SET.FTN$source indistinguishable) classification)
    (= (SET.FTN$target indistinguishable) rel$equivalence-relation)
    (= (SET.FTN$composition indistinguishable rel$object) instance)
    (= indistinguishable rel$equivalence1)

(10) (SET.FTN$function coextensive)
     (= (SET.FTN$source coextensive) classification)
     (= (SET.FTN$target coextensive) rel$relation)
     (= (SET.FTN$composition coextensive rel$object) type)
     (= coextensive rel$equivalence2)

(11) (SET.LIM.EQU$parallel-pair separated-parallel-pair)
     (= (SET.LIM.EQU$source separated-parallel-pair) classification)
     (= (SET.LIM.EQU$target separated-parallel-pair) rel$endorelation)
     (= (SET.LIM.EQU$function1 separated-parallel-pair) indistinguishable)
     (= (SET.LIM.EQU$function2 separated-parallel-pair)
        (SET.FTN$composition instance rel$identity))

(12) (SET$class separated)
     (SET$subclass separated classification)
     (= separated (SET.LIM.EQU$equalizer separated-parallel-pair))

(13) (SET.LIM.EQU$parallel-pair extensional-parallel-pair)
     (= (SET.LIM.EQU$source extensional-parallel-pair) classification)
     (= (SET.LIM.EQU$target extensional-parallel-pair) rel$relation)
     (= (SET.LIM.EQU$function1 extensional-parallel-pair) coextensive)
     (= (SET.LIM.EQU$function2 extensional-parallel-pair)
        (SET.FTN$composition type rel$identity))

(14) (SET$class extensional)
     (SET$subclass extensional classification)
     (= extensional (SET.LIM.EQU$equalizer extensional-parallel-pair))
```

- Any classification has an underlying *set pair*.

```
(1) (SET.FTN$function pair)
    (= (SET.FTN$source pair) classification)
    (= (SET.FTN$target pair) set.pr$pair)
    (= (SET.FTN$composition [pair set.pr$set1]) type)
    (= (SET.FTN$composition [pair set.pr$set2]) instance)
```

- For any classification $\boldsymbol{A} = \langle inst(\boldsymbol{A}), typ(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$, whose instances and types are regarded as names or indices, there is an associated *power* classification $\wp\boldsymbol{A} = \langle \wp inst(\boldsymbol{A}), \wp typ(\boldsymbol{A}), \models\rangle$, whose instances are subsets of $\boldsymbol{A}$-instances $inst(arity(\boldsymbol{A})) = \wp inst(\boldsymbol{A})$, whose types are subsets of $\boldsymbol{A}$-types $typ(arity(\boldsymbol{A})) = \wp typ(\boldsymbol{A})$, and whose incidence is the substitution of constants into variables: an arity instance (instance arity) $C \subseteq inst(\boldsymbol{A})$ has an arity type (type arity) $X \subseteq typ(\boldsymbol{A})$, denoted $C \models_{arity(\boldsymbol{A})} X$, when there exists a function $c : X \rightarrow C$ such that $c(x) \models_{\boldsymbol{A}} x$ for all $x \in X$. Note that some of the constants in C may not get "used" in this notion of substitution; that is, the function c is not necessarily a surjection. Therefore, this is a more flexible notion of substitution than the usual. A classification incidence $C \models_{\wp\boldsymbol{A}} X$ is *exact* when the function $c : X \rightarrow C$ is a surjection. Note that, if $C_1 \models_{\wp\boldsymbol{A}} X$ and $C_1 \subseteq C_2$ then $C_2 \models_{\wp\boldsymbol{A}} X$ also.

  When the classification is the identity classification $\boldsymbol{A} = cls(A)$ for some set A, the arity classification $sup(A) = \wp cls(A) = \langle \wp A, \wp A, \supseteq\rangle$ is called the *superset* classification.

```
(1) (SET.FTN$function power)
    (= (SET.FTN$source power) classification)
    (= (SET.FTN$target power) classification)
    (forall (?a (classification ?a))
        (and (= (instance (power ?a)) (set$power (instance ?a)))
             (= (type (power ?a)) (set$power (type ?a)))
             (forall (?C (set$subset ?C (instance ?a))
                      ?X (set$subset ?X (type ?a)))
                 (<=> ((power ?a) ?C ?X)
                      (exists (?c (set.ftn$function ?c))
                          (and (= (set.ftn$source ?c) ?X)
                               (= (set.ftn$target ?c) ?C))
                               (forall (?x (?X ?x))
                                   (?a (?c ?x) ?x))))))))

(2) (SET.FTN$function superset)
    (= (SET.FTN$source superset) set$set)
    (= (SET.FTN$target superset) classification)
    (= superset (SET.FTN$composition [set$classification power]))

    (forall (?a (set$set ?a) ?c (set$subset ?c ?a) ?x (set$subset ?x ?a))
        (<=> ((superset ?a) ?c ?x) (set$subset ?x ?c)))
```

> The power classification associated with any classification is the image of the object function of the power functor applied to the classification:
>
> $\wp$: Classification $\rightarrow$ Classification.

- These goes in a product namespace `cls.dsgn.prd2` for designations: For any two classifications $\boldsymbol{A}_1 = \langle inst(\boldsymbol{A}_1), typ(\boldsymbol{A}_1), \models_{\boldsymbol{A}1}\rangle$ and $\boldsymbol{A}_2 = \langle inst(\boldsymbol{A}_2), typ(\boldsymbol{A}_2), \models_{\boldsymbol{A}2}\rangle$, there is a full product classification $\boldsymbol{A}_1 \otimes \boldsymbol{A}_2$ and two projection designations $\pi_1(\boldsymbol{A}_1, \boldsymbol{A}_2) = \langle \pi_1, \pi_1\rangle : \boldsymbol{A}_1 \otimes \boldsymbol{A}_2 \rightrightarrows \boldsymbol{A}_1$ and $\pi_2(\boldsymbol{A}_1, \boldsymbol{A}_2) = \langle \pi_2, \pi_2\rangle : \boldsymbol{A}_1 \otimes \boldsymbol{A}_2 \rightrightarrows \boldsymbol{A}_2$. These form a binary product in the category Designation.

```
(1) (SET.FTN$function binary-product)
    (= (SET.FTN$source binary-product) pair)
    (= (SET.FTN$target binary-product) cls$classification)
    (= (SET.FTN$composition [binary-product cls$instance])
       (SET.FTN$composition [instance set.lim.prd2$binary-product])

(2) (SET.FTN$function projection1)
    (= (SET.FTN$source projection1) pair)
    (= (SET.FTN$target projection1) cls.dsgn$designation)
```

```
    (= (SET.FTN$composition [projection1 cls.dsgn$source]) binary-product)
    (= (SET.FTN$composition [projection1 cls.dsgn$target]) classification1)
    (= (SET.FTN$composition [projection1 cls.dsgn$instance])
       (SET.FTN$composition [instance set.lim.prd2$projection1]))
    (= (SET.FTN$composition [projection1 cls.dsgn$type])
       (SET.FTN$composition [type set.lim.prd2$projection1])
```

- As a special case, for any two set V and any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, there is a semi-product classification $V\bullet\boldsymbol{A} = \mathit{cls}(V)\otimes\boldsymbol{A}$ and two a semi-projection designation $\pi(V, \boldsymbol{A}) = \langle \pi_2, \pi_2 \rangle : \mathit{cls}(V)\otimes\boldsymbol{A} \rightrightarrows \boldsymbol{A}$.

$$\begin{tikzcd} V\times \mathit{typ}(\boldsymbol{A}) \arrow[r, "\pi_2"] & \mathit{typ}(\boldsymbol{A}) \\ V\times \mathit{inst}(\boldsymbol{A}) \arrow[u, dash, "\models_{V\bullet\boldsymbol{A}}"] \arrow[r, "\pi_2"'] & \mathit{inst}(\boldsymbol{A}) \arrow[u, dash, "\models_{\boldsymbol{A}}"'] \end{tikzcd}$$

Figure 1: Semi-projection Designation

```
(4) (SET.FTN$function semi-projection)
    (= (SET.FTN$source semi-projection)
       (SET.LIM.PRD2$binary-product [set$set cls$classification]))
    (= (SET.FTN$target projector1) cls.dsgn$designation)
    (forall (?v (set$set ?v) ?a (cls$classification ?a))
        (and (= (cls.dsgn$source (semi-projection [?v ?a]))
                (binary-product (pair [(set$classification ?v) ?a])))
             (= (cls.dsgn$target (semi-projection [?v ?a])) ?a)
             (= (semi-projection [?v ?a])
                (projection2 (pair [(set$classification ?v) ?a])))))
```

Order Classifications

`cls.ord`

- A classification is identical to a binary relation. However, from a category-theoretic standpoint, the context of classifications is very different from the context of relations, since their morphisms are very different. An order classification is identical to an order relation or bimodule.

$$\begin{tikzcd} \mathit{typ}(\boldsymbol{A}) \\ \mathit{inst}(\boldsymbol{A}) \arrow[u, dash, "\models_{\boldsymbol{A}}"'] \end{tikzcd}$$

Figure 1: Order Classification

An *order classification* $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ consists of a preorder of instances $\mathit{inst}(\boldsymbol{A}) = \langle \mathit{inst}(\boldsymbol{A}), \leq_{\mathit{inst}} \rangle$, a preorder of types $\mathit{typ}(\boldsymbol{A}) = \langle \mathit{typ}(\boldsymbol{A}), \leq_{\mathit{typ}} \rangle$, and a set of incidence or classification $\models_{\boldsymbol{A}}$ identified with the extent set of a bimodule. An order classification is order-closed at instances and types:

if $i_2 \leq_{\mathit{inst}} i_1$ and $i_1 \models_{\boldsymbol{A}} t$ then $i_2 \models_{\boldsymbol{A}} t$ for all instances $i_2, i_1 \in \mathit{inst}(\boldsymbol{A})$ and all types $t \in \mathit{typ}(\boldsymbol{A})$, and

if $i \models_{\boldsymbol{A}} t_1$ and $t_1 \leq_{\boldsymbol{Q}} t_2$ then $i \models_{\boldsymbol{A}} t_2$ for all instances $i \in \mathit{inst}(\boldsymbol{A})$ and all types $t_1, t_2 \in \mathit{typ}(\boldsymbol{A})$.

```
(1) (KIF$collection order-classification)

(2) (KIF$function instance)
    (= (KIF$source instance) order-classification)
    (= (KIF$target instance) ord$preorder)

(3) (KIF$function type)
    (= (KIF$source type) order-classification)
    (= (KIF$target type) ord$preorder)

(4) (KIF$function classification)
    (= (KIF$source classification) order-classification)
    (= (KIF$target classification) cls$classification)
    (= (SET.FTN$composition [classification cls$instance])
       (SET.FTN$composition [instance ord$set]))
    (= (SET.FTN$composition [classification cls$type])
       (SET.FTN$composition [type ord$set]))

(5) (forall (?a (order-classification ?a)
             ?i2 ((ord$set (instance ?a)) ?i2)
             ?i1 ((ord$set (instance ?a)) ?i1)
             ?t ((ord$set (type ?a)) ?t))
        (=> (and ((instance ?a) ?i2 ?i1) ((classification ?a) ?i1 ?t))
            ((classification ?a) ?i2 ?t)))

(6) (forall (?a (order-classification ?a)
             ?i ((ord$set (instance ?a)) ?i)
```

```
          ?t1 ((ord$set (type ?a)) ?t1)
          ?t2 ((ord$set (type ?a)) ?t2))
     (=> (and ((classification ?a) ?i ?t1) ((type ?a) ?t1 ?t2))
         ((classification ?a) ?i ?t2)))
```

Infomorphisms

`cls.info`

$$\begin{tikzcd} \mathit{typ}(\boldsymbol{A}) \arrow[r, "\mathit{ext}_{\boldsymbol{A}}"] \arrow[d, dash, "\models_{\boldsymbol{A}}"'] & \wp\mathit{inst}(\boldsymbol{A}) \arrow[d, dash, "\in_A"] \\ \mathit{inst}(\boldsymbol{A}) & \mathit{inst}(\boldsymbol{A}) \arrow[l, "\mathit{id}"] \end{tikzcd}$$

Figure 3: Extent Infomorphism

- It is a standard fact in Information Flow that from any classification $\boldsymbol{A}$ there is a unique canonical *extent* infomorphism

  $\eta_{\boldsymbol{A}} : \boldsymbol{A} \rightleftarrows \wp\mathit{inst}(\boldsymbol{A})$

  from $\boldsymbol{A}$ to the instance power classification $\wp\mathit{inst}(\boldsymbol{A}) = \langle \mathit{inst}(\boldsymbol{A}), \wp\mathit{inst}(\boldsymbol{A}), \in_{\boldsymbol{A}} \rangle$, whose instance function is the identity function on the instance set $\mathit{inst}(\boldsymbol{A})$, and whose type function is the extent function $\mathit{ext}_{\boldsymbol{A}} : \mathit{typ}(\boldsymbol{A}) \rightarrow \wp\mathit{inst}(\boldsymbol{A})$.

```
(17) (SET.FTN$function eta)
     (= (SET.FTN$source eta) cls$classification)
     (= (SET.FTN$target eta) infomorphism)
     (= (SET.FTN$composition eta source)
        (SET$identity cls$classification))
     (= (SET.FTN$composition eta target)
        (SET.FTN$composition cls$instance cls$instance-power))
     (= (SET.FTN$composition eta instance)
        (SET.FTN$composition cls$instance set.ftn$identity))
     (= (SET.FTN$composition eta type) cls$extent)
```

> This canonical infomorphism is the $\boldsymbol{A}$-th component of the unit of the functorial adjunction
>
> $\mathit{inst} \dashv \wp^{\mathrm{op}}$
>
> between the underlying (contravariant) instance functor and the instance power functor.

This fact is important in foundations because it corresponds to existence of power objects in the topos

$$\begin{tikzcd} B \arrow[r, "f"] \arrow[d, dash, "\models"'] \arrow[dr, dash] & \wp(A) \arrow[d, dash, "\in_A"] \\ A & A \arrow[l, "id_A"] \end{tikzcd} \qquad \begin{tikzcd} \models \arrow[r, tail] \arrow[d] \arrow[dr, phantom, "\ulcorner"] & B\times A \arrow[d, "f\times id_A"] \\ \in_A \arrow[r, tail] & \wp(A)\times A \end{tikzcd}$$

Set. Here is the KIF formulation of this at the basic level of sets, relations and functions.

```
(forall (?r (rel$relation ?r))
    (exists-unique (?f (set.ftn$function ?f))
        (and (= (set.ftn$source ?f) (rel$object2 ?r))
             (= (set.ftn$target ?f) (set$power (rel$object1 ?r)))
             (<=> ((rel$extent ?r) [?i ?t]) ((?f ?t) ?i)))))
```

Here is the KIF formulation of this at the level of classifications and infomorphisms.

```
(forall (?c (cls$classification ?c))
    (exists-unique (?f (infomorphism ?f))
        (and (= (source ?f) ?c)
             (= (target ?f) (cls$power (cls$instance ?c)))
             (= (instance ?f) (set.ftn$identity (cls$instance ?c))))))
```

- For any infomorphism $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$, whose instance and type functions are regarded as name or index functions, there is associated *power* infomorphism $\wp\boldsymbol{f} : \wp\boldsymbol{A} \rightleftarrows \wp\boldsymbol{B}$, whose instance function is the direct-image or power of the $\boldsymbol{f}$-instance function $\mathit{inst}(\wp\boldsymbol{f}) = \wp\mathit{inst}(\boldsymbol{f})$, and whose type function is the direct-image or power of the $\boldsymbol{f}$-type function $\mathit{typ}(\wp\boldsymbol{f}) = \wp\mathit{typ}(\boldsymbol{f})$. The fundamental property of infomor-

$$\begin{tikzcd} \wp\mathit{typ}(\boldsymbol{A}) \arrow[r, "\wp\mathit{typ}(\boldsymbol{f})"] \arrow[d, dash, "\models_{\mathit{arity}(\boldsymbol{A})}"'] & \wp\mathit{typ}(\boldsymbol{B}) \arrow[d, dash, "\models_{\mathit{arity}(\boldsymbol{B})}"] \\ \wp\mathit{inst}(\boldsymbol{A}) & \wp\mathit{inst}(\boldsymbol{B}) \arrow[l, "\wp\mathit{inst}(\boldsymbol{f})"] \end{tikzcd}$$

Figure 1: Arity Infomorphism

phisms means that for any power instance $C \subseteq \mathit{inst}(\boldsymbol{B})$ and any power type $X \subseteq \mathit{typ}(\boldsymbol{A})$,

$$\wp\mathit{inst}(\boldsymbol{f})(C) \vDash_{\wp\boldsymbol{A}} X \text{ iff } C \vDash_{\wp\boldsymbol{B}} \wp\mathit{inst}(\boldsymbol{f})(X);$$

that is, there exists a function $a : X \rightarrow \wp\mathit{inst}(\boldsymbol{f})(C)$ such that $a(x) \vDash_{\boldsymbol{A}} x$ for all $x \in X$ <u>iff</u> there exists a function $b : \wp\mathit{typ}(\boldsymbol{f})(X) \rightarrow C$ such that $b(x) \vDash_{\boldsymbol{B}} y$ for all $y \in \wp\mathit{typ}(\boldsymbol{f})(X)$. See Figure 1 where the function $\mathit{inst}(\boldsymbol{f})\lfloor_C$ is the restriction of the instance function $\mathit{inst}(\boldsymbol{f})$ to the subset C, and dually the function $\mathit{typ}(\boldsymbol{f})\lfloor_X$ is the restriction of the type function $\mathit{typ}(\boldsymbol{f})$ to the subset X. The <u>if</u> direction is obvious – just define a to be the composition $a = \mathit{typ}(\boldsymbol{f})\lfloor_X \cdot b \cdot \mathit{inst}(\boldsymbol{f})\lfloor_C$ and then use the fundamental property for each $x \in X$. For the only if direction, let $f : C \rightarrow \wp\mathit{inst}(\boldsymbol{f})(C)$ be a left inverse to $\mathit{inst}(\boldsymbol{f})\lfloor_C$, and let $g : \wp\mathit{typ}(\boldsymbol{f})(X) \rightarrow X$ be a left inverse to $\mathit{typ}(\boldsymbol{f})\lfloor_X$. These functions exist, since $\mathit{inst}(\boldsymbol{f})\lfloor_C$ and $\mathit{typ}(\boldsymbol{f})\lfloor_X$ are surjections. Then define b to be the composition $b = g \cdot a \cdot f$.

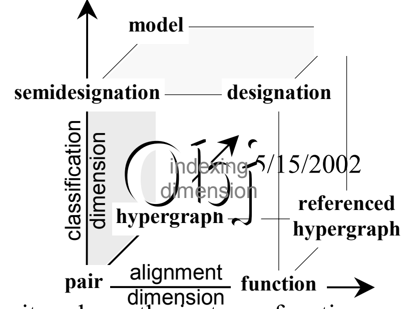
$$\begin{tikzcd} X \arrow[r, "\mathit{typ}(\boldsymbol{f})\lfloor_X"] \arrow[d, "a"'] & \wp\mathit{typ}(\boldsymbol{f})(X) \arrow[d, "b"] \\ \wp\mathit{inst}(\boldsymbol{f})(C) & C \arrow[l, "\mathit{inst}(\boldsymbol{f})\lfloor_C"] \end{tikzcd}$$

Figure 1: Substitution Incidence

```
(1) (SET.FTN$function power)
    (= (SET.FTN$source power) infomorphism)
    (= (SET.FTN$target power) infomorphism)
    (forall (?f (infomorphism ?f))
        (and (= (source (power ?f)) (cls$power (source ?f)))
             (= (target (power ?f)) (cls$power (target ?f)))
             (= (instance (power ?f)) (set.ftn$power (instance ?f)))
             (= (type (power ?f)) (set.ftn$power (type ?f)))))
```

The power infomorphism associated with any infomorphism is the image of the morphism function of the power functor applied to the infomorphism:

$\wp$: Classification $\rightarrow$ Classification.



Semidesignations

`cls.sdsgn`

○ A semidesignation is a weakened or generalized form of a designation it replaces the instance function of a desgination with a pair of sets and replaces the source classification of a desgination with the identity classification of a set.

$$\begin{array}{c} \mathit{set}(\boldsymbol{p}) \\ \boldsymbol{p}\ \big\downarrow \\ \mathit{cls}(\boldsymbol{p}) \end{array}$$

Figure: 1 Semidesignation – abstract

$$\begin{array}{ccc} A_1 = \mathit{set}(\boldsymbol{p}) & \xrightarrow{\mathit{typ}(\boldsymbol{p})} & \mathit{typ}(\mathit{cls}(\boldsymbol{p})) \\ = & & \Big| \models_{\mathit{cls}(\boldsymbol{p})} \\ A_1 & & A_2 = \mathit{inst}(\mathit{cls}(\boldsymbol{p})) \end{array}$$

Figure 2: Semidesignation – details

A *semidesignation* $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$, consists of

– a *set* $\mathit{set}(\boldsymbol{p}) = A_1$,
(corresponds to the source classification of a designation)

– a *classification* $\mathit{cls}(\boldsymbol{p})$,
(corresponds to the target classification of a designation)

– an *instance* set pair $\mathit{inst}(\boldsymbol{p}) = \langle A_1, A_2\rangle$, and
(corresponds to the instance function of a designation)

– a *type* reference function $\mathit{typ}(\boldsymbol{p}) : \mathit{set}(\boldsymbol{p}) \rightarrow \mathit{typ}(\mathit{cls}(\boldsymbol{p}))$,

where the first component set of the instance pair is the set

$\mathit{set1}(\mathit{inst}(\boldsymbol{p})) = A_1 = \mathit{set}(\boldsymbol{p})$,

and the second component set of the instance pair is the instance set of the classification

$\mathit{set2}(\mathit{inst}(\boldsymbol{p})) = A_2 = \mathit{inst}(\mathit{cls}(\boldsymbol{p}))$.

We view the type reference function as a mapping of variables (type signs) to object types.

```
(1) (SET$class semidesignation)

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

(3) (SET.FTN$function classification)
    (= (SET.FTN$source classification) semidesignation)
    (= (SET.FTN$target classification) cls$classification)

(4) (SET.FTN$function instance)
    (= (SET.FTN$source instance) semidesignation)
    (= (SET.FTN$target instance) set.pr$pair)
    (= (SET.FTN$composition [instance set.pr$set1]) set)
    (= (SET.FTN$composition [instance set.pr$set2])
       (SET.FTN$composition [classification cls$instance]))

(5) (SET.FTN$function type)
    (= (SET.FTN$source type) semidesignation)
    (= (SET.FTN$target type) set.ftn$function)
    (= (SET.FTN$composition [type set.ftn$source]) set)
    (= (SET.FTN$composition [type set.ftn$target])
       (SET.FTN$composition [classification cls$type]))
```

The class of semidesignations forms the object class of the category ▲Classification.

- For any semidesignation $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : set(\boldsymbol{p}) \rightleftharpoons cls(\boldsymbol{p})$ there is a *signature* classification

  $sign(\boldsymbol{p}) = \langle tuple(inst(\boldsymbol{p})), sign(typ(\boldsymbol{p})), \models\rangle$.

$$\begin{array}{c} sign(typ(\boldsymbol{p})) \\ \models_{sign(\boldsymbol{p})} \Big| \\ tuple(\boldsymbol{p}) \end{array}$$

Figure 1: Signature Classification

The type set for the signature classification is $sign(typ(\boldsymbol{p}))$, the signature set of the type function $typ(\boldsymbol{p}) : set(\boldsymbol{p}) \rightarrow typ(cls(\boldsymbol{p}))$. The instance set for the signature classification is $tuple(\boldsymbol{p}) = tuple(inst(\boldsymbol{p}))$, the tuple set of the set pair $inst(\boldsymbol{p})$. Here instance tuples t are tuples of the instance set pair $inst(\boldsymbol{p})$ and type signatures τ are signatures of the type function $typ(\boldsymbol{p})$,

– $t : arity(t) \rightarrow inst(cls(\boldsymbol{p}))$ with $arity(t) \subseteq set(\boldsymbol{p})$,

– $\tau : arity(\tau) \rightarrow typ(cls(\boldsymbol{p}))$ with $arity(\tau) \subseteq set(\boldsymbol{p})$.

We define an intuitive but reasonably flexible notion of classification: for any tuple $t \in tuple(\boldsymbol{p})$ and any signature $\tau \in sign(typ(\boldsymbol{p}))$, we say that t has type τ, symbolized $t \models_{sign(\boldsymbol{p})} \tau$, when $arity(t) \supseteq arity(\tau)$ and $t(x) \models_{cls(\boldsymbol{p})} \tau(x)$ for all type signs $x \in arity(\tau)$. To explain this more concretely, let us write the arity of the type signature as $arity(\tau) = \{x_1, x_2, \ldots, x_m\}$. In these terms the instance tuple t and the type signature τ can be displayed as

$$\tau = \{\tau(x) \mid x \in arity(\tau)\} = (\tau(x_1), \tau(x_2), \ldots, \tau(x_m)),$$

$$t = \{t(x) \mid x \in arity(\tau)\} = (t(x_1), t(x_2), \ldots, t(x_m), \ldots),$$

$$\begin{tikzcd}
arity(\sigma) \arrow[d, "h"'] \arrow[rrr, "\sigma"] & & & typ(cls(\boldsymbol{p})) \\
arity(\tau) \arrow[rrru, "\tau"] \subseteq arity(t) \subseteq set(\boldsymbol{p}) \arrow[rrr, "typ(\boldsymbol{p})"] & & & typ(cls(\boldsymbol{p})) \arrow[d, dash, "\models_{cls(\boldsymbol{p})}"] \\
arity(t) \arrow[rrr, "t"] & & & inst(cls(\boldsymbol{p}))
\end{tikzcd}$$

and the classification $t \models \tau$ means that with $t(x_i) \models \tau(x_i)$ for all $1 \leq i \leq m$. It is important to note that a priori the tuples of a semidesignation are untyped. In particular, a component of a tuple, which is an element of $inst(cls(\boldsymbol{p}))$, is not required a priori to have any particular type. Such a tuple component is constrained to have a type only when its ambient tuple is constrained to have some signature type, and the type constraint only comes through that signature.

Previously it was thought that tuples should be typed (See the old comments below). This is now thought to be a bad approach, since amonst other things, free logics need untyped tuples. We need to change (simplify) the ensuing development.

However, there is no instance function, and there is some question about the correct set of instances for $sign(\boldsymbol{p})$. Since these should be tuples, for brevity let us call these $tuple(\boldsymbol{p}) = inst(sign(\boldsymbol{p}))$. Since we only have a instance set pair instead of an instance set function, we cannot use signatures. However, we do have a set of tuples for the set pair $inst(\boldsymbol{p})$. Hence, let us choose the set $tuple(\boldsymbol{p}) = tuple(inst(\boldsymbol{p}))$ as a first cut.

This means that the tuple t is compatible with the type reference function $typ(\boldsymbol{p}) : set(\boldsymbol{p}) \rightarrow typ(cls(\boldsymbol{p}))$, at least on the set $arity(\tau)$. However, consider the sets:

$$set(\boldsymbol{p}) \supseteq arity(t) \supseteq arity(\tau).$$

Since the tuple t is define on the larger arity set $arity(t)$ a question remains: should the tuple t be compatible with the type reference function $typ(\boldsymbol{p})$ on the elements in the set difference $arity(t)–arity(\tau)$? It would be strange if these were not true. Indeed, if any element $x \in arity(t)$ is to be useful in a model that has $\boldsymbol{p}$ as its reference semidesignation, then it should participate in a classification incidence as just describe. But then the tuple t would be compatible with the type function $typ(\boldsymbol{p})$ at that element. Hence we assume this as an *admission requirement* for membership in the set of instance tuples $tuple(\boldsymbol{p})$. That is, we do not use the entire set $tuple(inst(\boldsymbol{p}))$. Here is the new definition.

$$tuple(\boldsymbol{p}) = \{t \in tuple(inst(\boldsymbol{p})) \mid t(x) \models_{cls(\boldsymbol{p})} typ(\boldsymbol{p})(x) \text{ for all } x \in arity(t)\}.$$

The incidence in the signature classification is order closed at instances and types: for any two tuples $t_1, t_2 \in tuple(\boldsymbol{p})$ and any two signatures $\tau_1, \tau_2 \in sign(typ(\boldsymbol{p}))$, if tuple t_2 is more specific than tuple t_1

($t_2 \leq t_1$), tuple t_1 has signature type τ_1 ($t_1 \vDash \tau_1$), and signature τ_1 is more specific than signature τ_2 ($\tau_1 \leq \tau_2$), then tuple t_2 has type τ_2 ($t_2 \vDash \tau_2$). That is, the classification is closed under instance and type order. The *signature order classification* $\langle sign(typ(\boldsymbol{p})), \leq\rangle$ consists of the classification of signatures with tuple order on instances and signature order on types.

```
(6) (SET.FTN$function tuple)
    (= (SET.FTN$source tuple) semidesignation)
    (= (SET.FTN$target tuple) set$set)
    (forall (?p (semidesignation ?p))
        (and (set$subset (tuple ?p) (set.pr$tuple (instance ?p)))
             (forall (?t ((set.pr$tuple (instance ?p)) ?t)
                 (<=> ((tuple ?p) ?t)
                      (forall (?x ((set.ftn$arity ?t) ?x))
                          ((classification ?p) (?t ?x) ((type ?p) ?x))))))

(7) (SET.FTN$function tuple-order)
    (= (SET.FTN$source tuple-order) semidesignation)
    (= (SET.FTN$target tuple-order) ord$partial-order)
    (forall (?p (semidesignation ?p))
        (and (= (ord$set (tuple-order ?p)) (tuple ?p))
             (forall (?t1 ((tuple ?p) ?t1)
                      ?t2 ((tuple ?p) ?t2))
                 (<=> ((tuple-order ?p) ?t1 ?t2)
                      ((set.pr$tuple-order (instance ?p)) ?t1 ?t2)))))

(8) (SET.FTN$function signature)
    (= (SET.FTN$source signature) semidesignation)
    (= (SET.FTN$target signature) cls$classification)
    (= (SET.FTN$composition [signature cls$instance]) tuple)
    (= (SET.FTN$composition [signature cls$type])
       (SET.FTN$composition [type set.ftn$signature]))
    (forall (?p (semidesignation ?p)
             ?t ((tuple ?p) ?t)
             ?tau ((set.ftn$signature (type ?p)) ?tau))
        (<=> ((signature ?p) ?t ?tau)
             (set$subset (set$arity ?tau) (set$arity ?t))))

(9) (SET.FTN$function signature-order)
    (= (SET.FTN$source signature-order) function)
    (= (SET.FTN$target signature-order) cls$order-classificcation)
    (= (SET.FTN$composition [signature-order cls.ord$instance]) tuple-order)
    (= (SET.FTN$composition [signature-order cls.ord$instance])
       (SET.FTN$composition [type set.ftn$signature-order]))
    (= (SET.FTN$composition [signature-order cls.ord$classification]) signature)
```

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

$$sign : \blacktriangle\text{Classification} \rightarrow \text{Classification}.$$

- We also define a strict (traditional) signature classification $\bullet sign(\boldsymbol{p})$, where tuples are incident on signatures only when they share the same arity.

```
(10) (SET.FTN$function strict-signature)
     (= (SET.FTN$source strict-signature) semidesignation)
     (= (SET.FTN$target strict-signature) cls$classification)
     (= (SET.FTN$composition [strict-signature cls$instance]) tuple)
     (= (SET.FTN$composition [strict-signature cls$type])
        (SET.FTN$composition [type set.ftn$signature]))
     (forall (?p (semidesignation ?p)
              ?t ((tuple ?p) ?t)
              ?tau ((set.ftn$signature (type ?p)) ?tau))
         (<=> ((strict-signature ?p) ?t ?tau)
              (= (set$arity ?tau) (set$arity ?t))))
```

- Since the strict signature classification is a subclassification of the (full) signature classification, for any type signature $\tau \in$ *sign*(*typ*($\boldsymbol{p}$)) there is an inclusion function from its •*sign*($\boldsymbol{p}$)-extent to its *sign*($\boldsymbol{p}$)-extent

  $$incl(\boldsymbol{p})(\tau) : ext(\bullet sign(\boldsymbol{p}))(\tau) \rightarrow ext(sign(\boldsymbol{p}))(\tau).$$

```
(11) (KIF$function inclusion)
     (= (KIF$source inclusion) semidesignation)
     (= (KIF$target inclusion) SET.FTN$function)
     (forall (?p (semidesignation ?p))
          (= (SET.FTN$source (inclusion ?p)) (set.ftn$signature (type ?p)))
          (= (SET.FTN$target (inclusion ?p)) set.ftn$function)
          (= (SET.FTN$composition [(inclusion ?p) set.ftn$source])
             (cls$extent (strict-signature ?p)))
          (= (SET.FTN$composition [(inclusion ?p) set.ftn$target])
             (cls$extent (signature ?p))))
```

- For any type signature $\tau \in$ *sign*(*typ*($\boldsymbol{p}$)), there is a surjective, idempotent trim function from its *sign*($\boldsymbol{p}$)-extent to its •*sign*($\boldsymbol{p}$)-extent

  $$trim(\boldsymbol{p})(\tau) : ext(sign(\boldsymbol{p}))(\tau) \rightarrow ext(\bullet sign(\boldsymbol{p}))(\tau).$$

  This function trims each instance tuple $t \in$ *tuple*($\boldsymbol{p}$) returning only the part of t that is essential in the classification $t \vDash_{sign(\boldsymbol{p})} \tau$. Thus, *trim*($\boldsymbol{p}$)($\tau$) restricts t to the subset *arity*(τ) $\subseteq$ *arity*(t). The trim function is left adjoint right inverse to the inclusion function. The trim function can be defined by these properties:

  [**increasing:**] $id \leq trim(\boldsymbol{p})(\tau) \cdot incl(\boldsymbol{p})(\tau)$, and

  [**surjective:**] $id = incl(\boldsymbol{p})(\tau) \cdot trim(\boldsymbol{p})(\tau)$.

```
(12) (KIF$function trim)
     (= (KIF$source trim) semidesignation)
     (= (KIF$target trim) SET.FTN$function)
     (forall (?p (semidesignation ?p))
          (and (= (SET.FTN$source (trim ?p)) (set.ftn$signature (type ?p)))
               (= (SET.FTN$target (trim ?p)) set.ftn$function)
               (= (SET.FTN$composition [(trim ?p) set.ftn$source])
                  (cls$extent (signature ?p)))
               (= (SET.FTN$composition [(trim ?p) set.ftn$target])
                  (cls$extent (strict-signature ?p))
               (forall (?tau ((cls$type (signature ?p)) ?tau))
                   (and (= (set.ftn$composition [((inclusion ?p) ?tau) ((trim ?p) ?tau)])
                           (set.ftn$identity (cls$type (strict-signature ?p))))
                        (forall (?t ((cls$instance (signature ?p)) ?t))
                            ((tuple-order ?p) ?t (((inclusion ?p) ?tau) (((trim ?p) ?tau) ?t))))))))
```

- For any semidesignation $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : set(\boldsymbol{p}) \rightleftharpoons cls(\boldsymbol{p})$, there is a(n instance) *tuple arity* function *tuple-arity*($\boldsymbol{p}$) : *tuple*($\boldsymbol{p}$) $\rightarrow \wp$*set*($\boldsymbol{p}$) on *instance* tuples. This is the composition of the inclusion of the tuple set into the instance tuple set followed by the arity function for the set pair tuples.

```
(13) (SET.FTN$function tuple-arity)
     (= (SET.FTN$source tuple-arity) semidesignation)
     (= (SET.FTN$target tuple-arity) set.ftn$function)
     (forall (?p (semidesignation ?p))
         (and (= (set.ftn$source (tuple-arity ?p)) (tuple ?p))
              (= (set.ftn$target (tuple-arity ?p)) (set$power (set ?p)))
              (= (tuple-arity ?p)
                 (set.ftn$composition
                     [(set.ftn$inclusion [(tuple ?p) (set.pr$tuple (instance ?p))])
                      (set.pr$arity (instance ?p))]))))
```

- Because of the admission requirement on tuples, there is also a *tuple assign* function inverse to tuple arity. This can be defined as the restriction of the type function to the *typ*($\boldsymbol{p}$) given arity.

```
(14) (SET.FTN$function tuple-assign)
     (= (SET.FTN$source tuple-assign) semidesignation)
     (= (SET.FTN$target tuple-assign) set.ftn$function)
     (forall (?p (semidesignation ?p))
         (and (= (set.ftn$source (tuple-assign ?p)) (set$power (set ?p)))
              (= (set.ftn$target (tuple-assign ?p)) (tuple ?p))))
```

- For any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p}) \rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ and any element $x \in \mathit{set}(\boldsymbol{p})$, there is a tuple projection function

  $\pi_{\boldsymbol{p}}(x) : \mathit{tuple}(\boldsymbol{p}) \rightarrow \mathit{tuple}(\boldsymbol{p})$.

  For any tuple $t \in \mathit{tuple}(\boldsymbol{p})$,

  – if $x \notin \mathit{arity}(t)$, then $\pi_{\boldsymbol{p}}(x)(t) = t$,

  – if $x \in \mathit{arity}(t)$, then $\mathit{arity}(\pi_{\boldsymbol{p}}(x)(t)) = \mathit{arity}(t)-\{x\}$ and $\pi_{\boldsymbol{p}}(x)(t)$ is a restriction of t.

```
(13) (KIF$function projection)
     (= (KIF$source projection) semidesignation)
     (= (KIF$target projection) SET.FTN$function)
     (forall (?p (semidesignation ?p))
         (and (= (SET.FTN$source (projection ?p)) (set ?p))
              (= (SET.FTN$target (projection ?p)) set.ftn$function)
              (= (SET.FTN$composition [(projection ?p) set.ftn$source]) tuple)
              (= (SET.FTN$composition [(projection ?p) set.ftn$target]) tuple)
              (forall (?x ((set ?p) ?x))
                  (and (= (set.ftn$composition
                                [((projection ?p) ?x) (tuple-arity ?p)])
                          (set.ftn$composition
                                [(tuple-arity ?p) ((set.ftn$deletion (set ?p)) ?x)]))
                       (forall (?t ((tuple ?p ?t))
                           (set.ftn$restriction (((projection ?p) ?x) ?t) ?t))))))
```

- For any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p}) \rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ and any element $x \in \mathit{set}(\boldsymbol{p})$, there are existential and universal functions

  $\exists_{\boldsymbol{p},x} : \wp\mathit{tuple}(\boldsymbol{p}) \rightarrow \wp\mathit{tuple}(\boldsymbol{p})$ and $\forall_{\boldsymbol{p},x} : \wp\mathit{tuple}(\boldsymbol{p}) \rightarrow \wp\mathit{tuple}(\boldsymbol{p})$,

  where

  $\exists_{\boldsymbol{p},x}(X) = \{t \in \mathit{tuple}(\boldsymbol{p}) \mid \exists s \in \mathit{tuple}(\boldsymbol{p})$ such that $\pi_{\boldsymbol{p}}(x)(s) = t$ and $s \in X\}$

  $\forall_{\boldsymbol{p},x}(X) = \{t \in \mathit{tuple}(\boldsymbol{p}) \mid \forall s \in \mathit{tuple}(\boldsymbol{p})$ such that if $\pi_{\boldsymbol{p}}(x)(s) = t$ then $s \in X\}$

  for any subset of tuples $X \in \wp\mathit{tuple}(\boldsymbol{p})$. These function are defined in terms of the corresponding set function operations with respect to projection.

```
(14) (KIF$function existential)
     (= (KIF$source existential) semidesignation)
     (= (KIF$target existential) SET.FTN$function)
     (forall (?p (semidesignation ?p))
         (and (= (SET.FTN$source (existential ?p)) (set ?p))
              (= (SET.FTN$target (existential ?p)) set.ftn$function)
              (= (SET.FTN$composition [(existential ?p) set.ftn$source])
                 (SET.FTN$composition [tuple set$power]))
              (= (SET.FTN$composition [(existential ?p) set.ftn$target])
                 (SET.FTN$composition [tuple set$power]))
              (= existential
                (SET.FTN$composition [(projection ?p) set.ftn$existential]))

(15) (KIF$function universal)
     (= (KIF$source universal) semidesignation)
     (= (KIF$target universal) SET.FTN$function)
     (forall (?p (semidesignation ?p))
         (and (= (SET.FTN$source (universal ?p)) (set ?p))
              (= (SET.FTN$target (universal ?p)) set.ftn$function)
              (= (SET.FTN$composition [(universal ?p) set.ftn$source])
                 (SET.FTN$composition [tuple set$power]))
              (= (SET.FTN$composition [(universal ?p) set.ftn$target])
                 (SET.FTN$composition [tuple set$power]))
              (= universal
                (SET.FTN$composition [(projection ?p) set.ftn$universal]))
```

- For any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p}) \rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ there is a *signature arity* designation

  $\mathit{sign\text{-}arity}(\boldsymbol{p})$

$$\begin{tikzcd}
\mathit{sign}(\mathit{typ}(\boldsymbol{p})) \arrow[r, "\mathit{arity}(\mathit{typ}(\boldsymbol{p}))"] \arrow[d, dash, "\models_{\mathit{sign}(\boldsymbol{p})}"'] \arrow[dr, phantom, "\mathit{sign\text{-}arity}(\boldsymbol{p})"] & \wp\mathit{set}(\boldsymbol{p}) \arrow[d, dash, "\models_{\mathit{sup}(\mathit{set}(\boldsymbol{p}))}"] \\
\mathit{tuple}(\boldsymbol{p}) \arrow[r, "\mathit{tuple\text{-}arity}(\boldsymbol{p})"'] & \wp\mathit{set}(\boldsymbol{p})
\end{tikzcd}$$

Figure 1: Arity designation

$= \langle \mathit{tuple\text{-}arity}(\boldsymbol{p}), \mathit{arity}(\mathit{typ}(\boldsymbol{p}))\rangle : \mathit{sign}(\boldsymbol{p}) \rightrightarrows \mathit{sup}(\mathit{set}(\boldsymbol{p}))$,

whose source is the signature classification of $\boldsymbol{p}$, whose target is the power of the classification of $\mathit{set}(\boldsymbol{p})$, the superset classification over $\mathit{set}(\boldsymbol{p})$, whose instance function is the tuple arity function of $\boldsymbol{p}$, and whose type function is arity function for the type function of $\boldsymbol{p}$.

```
(12) (SET.FTN$function signature-arity)
     (= (SET.FTN$source signature-arity) semidesignation)
     (= (SET.FTN$target signature-arity) dsgn$designation)
     (= (SET.FTN$composition [signature-arity cls.dsgn$source]) signature)
     (= (SET.FTN$composition [signature-arity cls.dsgn$target])
        (SET.FTN$composition [set cls$superset]))
     (= (SET.FTN$composition [signature-arity cls.dsgn$instance]) instance-arity)
     (= (SET.FTN$composition [signature-arity cls.dsgn$type])
        (SET.FTN$composition [type set.ftn$arity]))
```

○ Any semidesignation

$\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \rightarrowtail \mathit{cls}(\boldsymbol{p})$

defines a *model* (Figure 3), whose reference semidesignation is itself and whose signature designation is the identity designation $\mathit{id} : \mathit{sign}(\boldsymbol{p}) \rightrightarrows \mathit{sign}(\boldsymbol{p})$ at the signature classification of $\boldsymbol{p}$. This model has $\mathit{set}(\boldsymbol{p})$ as its set of variables, $\mathit{inst}(\mathit{cls}(\boldsymbol{p}))$ as its universe, $\mathit{typ}(\mathit{cls}(\boldsymbol{p}))$ as its set of entity types, $\mathit{tuple}(\boldsymbol{p})$ as its set of tuples, and $\mathit{sign}(\mathit{typ}(\boldsymbol{p}))$ as its set of relation types. Clearly, the semidesignation of the model of a semidesignation is itself.

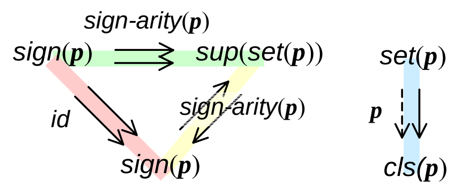


Figure 3: Model of a semidesignation

```
(13) (SET.FTN$function model)
     (= (SET.FTN$source model) semidesignation)
     (= (SET.FTN$target model) mod$model)
     (forall (?p (semidesignation ?p))
         (and (= (mod$reference (model ?p)) ?p)
              (= (mod$signature (model ?p)) (cls.dsgn$identity (signature ?p)))))
```

○ Any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \rightarrowtail \mathit{cls}(\boldsymbol{p})$ defines a coproduct *instance arity*.

```
(14) (SET.FTN$function instance-arity)
     (= (SET.FTN$source instance-arity) semidesignation)
     (= (SET.FTN$target instance-arity) set.col.art$arity)
     (forall (?p (semidesignation ?p))
         (and (= (set.col.art$index (instance-arity ?p)) (tuple ?p))
              (= (set.col.art$base (instance-arity ?p)) (set ?p))
              (= (set.col.art$function (instance-arity ?p)) (tuple-arity ?p))))
```

○ Any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \rightarrowtail \mathit{cls}(\boldsymbol{p})$ has a set of *instance cases* or *thematic roles*

$\mathit{inst\text{-}case}(\boldsymbol{p}) = \sum \mathit{inst\text{-}arity}(\boldsymbol{p})$

$= \sum_{t \in \mathit{tuple}(\boldsymbol{p})} \mathit{inst\text{-}arity}(\boldsymbol{p})(t)$

$= \{(t, x) \mid t \in \mathit{tuple}(\boldsymbol{p}), x \in \mathit{inst\text{-}arity}(\boldsymbol{p})(t)\}$,

$$\begin{tikzcd}
\#_{\boldsymbol{p}}(t) \arrow[r, "\mathit{inst\text{-}inj}(\boldsymbol{p})(t)"] \arrow[dr, phantom, "\subseteq"] & \mathit{inst\text{-}case}(\boldsymbol{p}) \arrow[d, "\mathit{inst\text{-}proj}(\boldsymbol{p})"] \\
 & \mathit{set}(\boldsymbol{p})
\end{tikzcd}$$

Diagram 3: Coproduct

which is the coproduct of its arity. In terms of frames, elements of $\mathit{inst\text{-}case}(\boldsymbol{p})$ can be regarded as frame-slot (nexus-prehension) pairs. Although set-theoretically small, the set $\mathit{inst\text{-}case}(\boldsymbol{p})$ is practically rather large; this large cardinality will be handled properly in the extension from semidesignations to models by adding a denotation or indexing function called the signature of an abstract tuple (relation instance).

In addition, any semidesignation $\boldsymbol{p}$ and any tuple $t \in \mathit{tuple}(\boldsymbol{p})$ define an *instance injection* function:

$\mathit{inst\text{-}inj}(\boldsymbol{p})(t) : \#_{\boldsymbol{p}}(t) = \mathit{arity}(\boldsymbol{p})(t) \rightarrow \mathit{inst\text{-}case}(\boldsymbol{p})$.

This is defined by $\mathit{inst\text{-}inj}(\boldsymbol{p})(t)(x) = (t, x)$ for all tuples (instance nexus) $t \in \mathit{tuple}(\boldsymbol{p})$ and all names (instance prehensions) $x \in \mathit{inst\text{-}arity}(\boldsymbol{p})(t)$. Obviously, the injections are injective. They commute (Diagram 3) with projection and inclusion.

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

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

- Any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ defines *instance indication* (*nexus*) and *instance projection* (*prehension*) functions based on its instance arity:

  $\mathit{inst\text{-}indic}(\boldsymbol{p}) : \mathit{inst\text{-}case}(\boldsymbol{p}) \rightarrow \mathit{tuple}(\boldsymbol{p})$,

  $\mathit{inst\text{-}proj}(\boldsymbol{p}) : \mathit{inst\text{-}case}(\boldsymbol{p}) \rightarrow \mathit{set}(\boldsymbol{p})$.

  They are defined by

  $\mathit{inst\text{-}indic}(\boldsymbol{p})((t, x)) = t$ and $\mathit{inst\text{-}proj}(\boldsymbol{p})((t, x)) = x$

  for all tuples (instance nexus) $t \in \mathit{tuple}(\boldsymbol{p})$ and all names (instance prehensions) $x \in \mathit{inst\text{-}arity}(\boldsymbol{p})(t)$.

$$\begin{tikzcd} & \mathit{inst\text{-}case}(\boldsymbol{p}) \arrow[dl, "\mathit{inst\text{-}indic}(\boldsymbol{p})"'] \arrow[dr, "\mathit{inst\text{-}proj}(\boldsymbol{p})"] & \\ \mathit{tuple}(\boldsymbol{p}) \arrow[rr, "\#_{\boldsymbol{p}}"] & & \mathit{set}(\boldsymbol{p}) \quad \wp\mathit{set}(\boldsymbol{p}) \end{tikzcd}$$

Figure 2: The indication and projection functions of a semidesignation

  The fiber-indication function $\mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p})) : \mathit{tuple}(\boldsymbol{p}) \rightarrow \wp\mathit{inst\text{-}case}(\boldsymbol{p})$ is defined by

  $\mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p}))(t) = \{(t, x) \mid x \in \mathit{inst\text{-}arity}(\boldsymbol{p})(t)\} \subseteq \mathit{inst\text{-}case}(\boldsymbol{p})$

  for all tuples $t \in \mathit{tuple}(\boldsymbol{p})$. And of course the collection $\{\mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p}))(t) \mid t \in \mathit{tuple}(\boldsymbol{p})\}$ is a partition of $\mathit{inst\text{-}case}(\boldsymbol{p})$. These fibers are the images of the injection functions

  $\mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p}))(t) = \mathit{inst\text{-}inj}(\boldsymbol{p})(t)[\mathit{inst\text{-}arity}(\boldsymbol{p})(t)] \subseteq \mathit{inst\text{-}case}(\boldsymbol{p})$

  for all tuples $t \in \mathit{tuple}(\boldsymbol{p})$.

  The instance projection function is locally bijective: for tuple $t \in \mathit{tuple}(\boldsymbol{p})$ the restriction of the instance projection function on each fiber $\mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p}))(t)$ is a bijection:

  $\mathit{inst\text{-}proj}(\boldsymbol{p}) : \mathit{fib}(\mathit{inst\text{-}indic}(\boldsymbol{p}))(t) \cong \mathit{inst\text{-}arity}(\boldsymbol{p})(t)$.

  This fact is important, since it implies that only one substitution function of instance names into type names (variables) is possible. The instance projection function is the cotupling of the arity inclusion functions $\{\mathit{incl}_{\text{inst-arity}(\boldsymbol{p})(t),\, \mathit{set}(\boldsymbol{p})} \mid t \in \mathit{tuple}(\boldsymbol{p})\}$.

```
(17) (SET.FTN$function instance-indication)
     (= (SET.FTN$source instance-indication) semidesignation)
     (= (SET.FTN$target instance-indication) set.ftn$function)
     (= instance-indication
        (SET.FTN$composition [instance-arity set.col.art$indication]))

(18) (SET.FTN$function instance-projection)
     (= (SET.FTN$source instance-projection) semidesignation)
     (= (SET.FTN$target instance-projection) set.ftn$function)
     (= instance-projection
        (SET.FTN$composition [instance-arity set.col.art$projection]))
```

- Any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ defines an *instance comediator* function:

  $\tilde{*}_{\boldsymbol{p}} = \mathit{inst\text{-}comed}(\boldsymbol{p}) : \mathit{inst\text{-}case}(\boldsymbol{p}) \rightarrow \mathit{inst}(\mathit{cls}(\boldsymbol{p}))$.

  This function is the slot-filler function for frames. It is defined by

  $\mathit{inst\text{-}comed}(\boldsymbol{p})((t, x)) = t(x)$

$$\begin{tikzcd} & \mathit{inst\text{-}case}(\boldsymbol{p}) \arrow[d, "\tilde{*}_{\boldsymbol{p}}"] \\ \mathit{inst\text{-}arity}(\boldsymbol{p})(t) \arrow[ur, "\mathit{inst\text{-}inj}(\boldsymbol{p})(t)"] \arrow[r, "t"'] & \mathit{inst}(\mathit{cls}(\boldsymbol{p})) \end{tikzcd}$$

Diagram 4: Reference

  for all tuples $t \in \mathit{tuple}(\boldsymbol{p})$ and all names $x \in \mathit{inst\text{-}arity}(\boldsymbol{p})(t)$. The comediator commutes (Diagram 4) with the injection function and the tuple itself. If the indexed names in $\mathit{inst\text{-}case}(\boldsymbol{p})$ are regarded to be roles, the comediator is a reference function from roles to referenced objects.

```
(19) (SET.FTN$function instance-comediator)
     (= (SET.FTN$source instance-comediator) semidesignation)
```

```
      (= (SET.FTN$target instance-comediator) set.ftn$function)
      (forall (?p (semidesignation ?p))
          (and (= (set.ftn$source (instance-comediator ?p)) (instance-case ?p))
               (= (set.ftn$target (instance-comediator ?p))
                  (cls$instance (classification ?p)))
               (= (instance-comediator ?p)
                  ((set.col.art$cotupling (instance-arity ?p))
                   (SET.FTN$inclusion [(tuple ?p) set.ftn$tuple])))))
```

○ There is a *case* classification $case(\boldsymbol{p}) = \langle inst\text{-}case(\boldsymbol{p}), case(typ(\boldsymbol{p})), \models_{case(\boldsymbol{p})}\rangle$, whose instances (t, x) are tuple-name pairs for tuples $t \in tuple(\boldsymbol{p})$ and names $x \in inst\text{-}arity(\boldsymbol{p})(t) \subseteq set(\boldsymbol{p})$, whose types (τ, x) are signature-name pairs for signatures $\tau \in sign(typ(\boldsymbol{p}))$ and names $x \in arity(typ(\boldsymbol{p}))(\tau) \subseteq set(\boldsymbol{p})$, and whose classification is defined by

$$(t, x) \models_{case(\boldsymbol{p})} (\tau, x') \text{ when } t \models_{sign(\boldsymbol{p})} \tau \text{ and } x = x'.$$

```
(20) (SET.FTN$function case)
     (= (SET.FTN$source case) semidesignation)
     (= (SET.FTN$target case) cls$classification)
     (= (SET.FTN$composition [case cls$instance]) instance-case)
     (= (SET.FTN$composition [case cls$type])
        (SET.FTN$composition [type set.ftn$case]))
     (forall (?p (semidesignation ?p)
              ?t ?x ((instance-case ?p) [?t ?x])
              ?tau ?x1 ((set.ftn$case (type ?p)) [?tau ?x1]))
         (<=> ((case ?p) [?t ?x] [?tau ?x1])
              (and ((signature ?p) ?t ?tau)
                   (= ?x ?x1))))
```

○ Any semidesignation $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : set(\boldsymbol{p}) \looparrowright cls(\boldsymbol{p})$ has an associated *indication* designation

$indic(\boldsymbol{p})$

$$= \langle inst\text{-}indic(\boldsymbol{p}), indic(typ(\boldsymbol{p}))\rangle : case(\boldsymbol{p}) \rightrightarrows sign(\boldsymbol{p}),$$

whose source is the case classification of $\boldsymbol{p}$, whose target is the signature classification of $\boldsymbol{p}$, whose instance function is the instance indication function, and whose type function is indication function for the type function.

$$\begin{tikzcd}
case(typ(\boldsymbol{p})) \arrow[r, "indic(typ(\boldsymbol{p}))"] & sign(typ(\boldsymbol{p})) \\
inst\text{-}case(\boldsymbol{p}) \arrow[u, dash, "\models_{case(\boldsymbol{p})}"] \arrow[r, "inst\text{-}indic(\boldsymbol{p})"'] \arrow[ur, phantom, "indic(\boldsymbol{p})"] & sign(\boldsymbol{p}) \arrow[u, dash, "\models_{sign(\boldsymbol{p})}"']
\end{tikzcd}$$

Figure 1: Indication designation

```
(21) (SET.FTN$function indication)
     (= (SET.FTN$source indication) semidesignation)
     (= (SET.FTN$target indication) cls.dsgn$designation)
     (= (SET.FTN$composition [indication cls.dsgn$source]) case)
     (= (SET.FTN$composition [indication cls.dsgn$target]) signature)
     (= (SET.FTN$composition [indication cls.dsgn$instance]) instance-indication)
     (= (SET.FTN$composition [indication cls.dsgn$type])
        (SET.FTN$composition [type set.ftn$indication]))
```

○ Any semidesignation $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : set(\boldsymbol{p}) \looparrowright cls(\boldsymbol{p})$ has an associated *projection* designation

$$proj(\boldsymbol{p}) = \langle inst\text{-}proj(\boldsymbol{p}), proj(typ(\boldsymbol{p}))\rangle : case(\boldsymbol{p}) \rightrightarrows cls(set(\boldsymbol{p})),$$

whose source is the case classification of $\boldsymbol{p}$, whose target is the identity classification of the set of $\boldsymbol{p}$, whose instance function is the instance projection function, and whose type function is projection for the type function. The designation requirement that classification be preserved is evident from the definitions of the case classification, the instance projection function and the projection for the type function.

$$\begin{tikzcd}
case(typ(\boldsymbol{p})) \arrow[r, "proj(typ(\boldsymbol{p}))"] & set(\boldsymbol{p}) \\
inst\text{-}case(\boldsymbol{p}) \arrow[u, dash, "\models_{case(\boldsymbol{p})}"] \arrow[r, "inst\text{-}proj(\boldsymbol{p})"'] \arrow[ur, phantom, "proj(\boldsymbol{p})"] & set(\boldsymbol{p}) \arrow[u, dash, "=_{set(\boldsymbol{p})}"']
\end{tikzcd}$$

Figure 1: Projection designation

```
(22) (SET.FTN$function projection)
     (= (SET.FTN$source projection) semidesignation)
     (= (SET.FTN$target projection) cls.dsgn$designation)
     (= (SET.FTN$composition [projection cls.dsgn$source]) case)
     (= (SET.FTN$composition [projection cls.dsgn$target])
        (SET.FTN$composition [set set$classification]))
```

```
    (= (SET.FTN$composition [projection cls.dsgn$instance]) instance-projection)
    (= (SET.FTN$composition [projection cls.dsgn$type])
       (SET.FTN$composition [type set.ftn$projection]))
```

- Any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ has an associated *comediator* designation

  $*_{\boldsymbol{p}} = \mathit{comed}(\boldsymbol{p})$

  $= \langle \mathit{inst\text{-}comed}(\boldsymbol{p}), \mathit{comed}(\mathit{typ}(\boldsymbol{p}))\rangle : \mathit{case}(\boldsymbol{p}) \rightrightarrows \mathit{cls}(\boldsymbol{p})$,

  whose source is the case classification of $\boldsymbol{p}$, whose target is the classification of $\boldsymbol{p}$, whose instance function is the instance comediator function, and whose type function is the comediator function for the type function.

$$\begin{tikzcd}
\mathit{case}(\mathit{typ}(\boldsymbol{p})) \arrow[r, "\mathit{comed}(\mathit{typ}(\boldsymbol{p}))"] \arrow[d, dash, "\models_{\mathit{case}(\boldsymbol{p})}"'] \arrow[dr, phantom, "\mathit{comed}(\boldsymbol{p})"] & \mathit{typ}(\mathit{cls}(\boldsymbol{p})) \arrow[d, dash, "\models_{\mathit{cls}(\boldsymbol{p})}"] \\
\mathit{inst\text{-}case}(\boldsymbol{p}) \arrow[r, "\mathit{inst\text{-}comed}(\boldsymbol{p})"'] & \mathit{inst}(\mathit{cls}(\boldsymbol{p}))
\end{tikzcd}$$

Figure 1: Comediator designation

```
(23) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) semidesignation)
     (= (SET.FTN$target comediator) cls.dsgn$designation)
     (= (SET.FTN$composition [comediator cls.dsgn$source]) case)
     (= (SET.FTN$composition [comediator cls.dsgn$target]) classification)
     (= (SET.FTN$composition [comediator cls.dsgn$instance]) instance-comediator)
     (= (SET.FTN$composition [comediator cls.dsgn$type])
        (SET.FTN$composition [type set.ftn$comediator]))
```

$$\begin{tikzcd}
\mathit{set}(\boldsymbol{p}) \arrow[d, "\boldsymbol{p}"'] \\
\mathit{cls}(\boldsymbol{p})
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
\mathit{sign}(\boldsymbol{p}) & \mathit{case}(\boldsymbol{p}) \arrow[l, shift left, "\mathit{indic}(\boldsymbol{p})"'] \arrow[l, shift right] \arrow[r, shift left, "\mathit{comed}(\boldsymbol{p})"] \arrow[r, shift right] \arrow[d, shift left] \arrow[d, shift right, "\mathit{proj}(\boldsymbol{p})"'] & \mathit{cls}(\boldsymbol{p}) \\
& \mathit{set}(\boldsymbol{p}) &
\end{tikzcd}$$

Figure 1: Semisdesignation to Spanmodel - abstract version

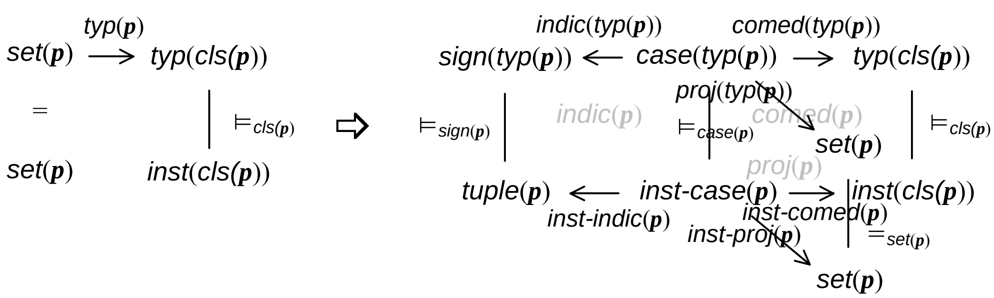


Figure 1: Semidesignation to Spanmodel - detailed version

- Associated with any semidesignation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$ is a spanmodel $\mathit{spnmod}(\boldsymbol{p})$, whose vertex (prehension) classification is the case (role) classification of $\boldsymbol{p}$,
  - whose first designation (actuality) is the comediator designation $1^{st}_{\mathit{spnmod}(\boldsymbol{p})} = \mathit{comed}(\boldsymbol{p})$ with target classification being the classification of $\boldsymbol{p}$,
  - whose second designation (naming) is the projection designation $2^{nd}_{\mathit{spnmod}(\boldsymbol{p})} = \mathit{proj}(\boldsymbol{p})$ with target classification being the set identity classification of $\boldsymbol{p}$, and
  - whose third designation (nexus) is the indication designation $3^{rd}_{\mathit{spnmod}(\boldsymbol{p})} = \mathit{indic}(\boldsymbol{p})$ with target classification being the signature classification of $\boldsymbol{p}$.

```
(24) (SET.FTN$function spanmodel)
     (= (SET.FTN$source spanmodel) semidesignation)
     (= (SET.FTN$target spanmodel) smod$spanmodel)
     (forall (?p (semidesignation ?p))
         (and (= (smod$vertex (spanmodel ?p)) (case ?p))
```

```
(= (smod$first (spanmodel ?p)) (comediator ?p))
(= (smod$classification1 (spanmodel ?p)) (classification ?p))
(= (smod$second (spanmodel ?p)) (projection ?p))
(= (smod$classification2 (spanmodel ?p)) (set$classification (set ?p)))
(= (smod$third (spanmodel ?p)) (indication ?p))
(= (smod$classification3 (spanmodel ?p)) (signature ?p))))
```

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

$$\mathit{spnmod} : \blacktriangle\mathrm{Classification} \rightarrow \mathrm{SpanModel}.$$

In summary, any semidesignation $\boldsymbol{p} = \langle set(\boldsymbol{p}), cls(\boldsymbol{p}), inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle$ with

name set $set(\boldsymbol{p})$,
object classification $cls(\boldsymbol{p})$,
instance set pair $inst(\boldsymbol{p})$,
type reference function $typ(\boldsymbol{p}) : set(\boldsymbol{p}) \rightarrow typ(cls(\boldsymbol{p}))$,
signature classification $sign(\boldsymbol{p}) = \langle tuple(\boldsymbol{p}), sign(typ(\boldsymbol{p})), \models\rangle$,
instance arity function $\#_{\boldsymbol{p}} = inst\text{-}arity(\boldsymbol{p}) : tuple(\boldsymbol{p}) \rightarrow \wp set(\boldsymbol{p})$,

---

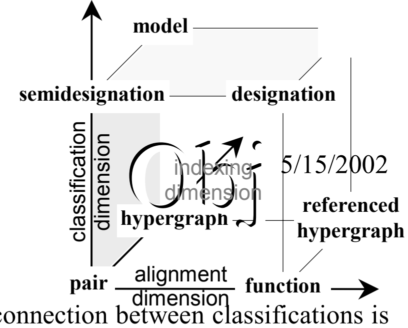
instance case set $inst\text{-}case(\boldsymbol{p})$,
instance index function $inst\text{-}index(\boldsymbol{p}) : inst\text{-}case(\boldsymbol{p}) \rightarrow tuple(\boldsymbol{p})$,
instance projection function $inst\text{-}proj(\boldsymbol{p}) : inst\text{-}case(\boldsymbol{p}) \rightarrow set(\boldsymbol{p})$, and
instance reference function $\tilde{*}_{\boldsymbol{p}} = inst\text{-}refer(\boldsymbol{p}) : inst\text{-}case(\boldsymbol{p}) \rightarrow inst(cls(\boldsymbol{p}))$,

defines the following designations

$\#_{\boldsymbol{p}} = arity(\boldsymbol{p}) = \langle inst\text{-}arity(\boldsymbol{p}), arity(typ(\boldsymbol{p}))\rangle : signature(\boldsymbol{p}) \rightrightarrows sup(set(\boldsymbol{p}))$,

---

$index(\boldsymbol{p}) = \langle inst\text{-}index(\boldsymbol{p}), index(typ(\boldsymbol{p}))\rangle : case(\boldsymbol{p}) \rightrightarrows sign(\boldsymbol{p})$,
$*_{\boldsymbol{p}} = refer(\boldsymbol{p}) = \langle inst\text{-}refer(\boldsymbol{p}), refer(typ(\boldsymbol{p}))\rangle : case(\boldsymbol{p}) \rightrightarrows cls(\boldsymbol{p})$,and
$proj(\boldsymbol{p}) = \langle inst\text{-}proj(\boldsymbol{p}), proj(typ(\boldsymbol{p}))\rangle : case(\boldsymbol{p}) \rightrightarrows cls(set(\boldsymbol{p}))$.



Designations

`cls.dsgn`

Although infomorphisms are of main interest, another morphism-like connection between classifications is also of interest.

$$\begin{array}{c} src(\boldsymbol{p}) \\ \boldsymbol{p}\ \downdownarrows \\ tgt(\boldsymbol{p}) \end{array}$$

Figure 1: Designation -abstract

$$\begin{array}{ccc} typ(src(\boldsymbol{p})) & \xrightarrow{typ(\boldsymbol{p})} & typ(tgt(\boldsymbol{p})) \\ \vDash_{src(\boldsymbol{p})} \Big| & & \Big| \vDash_{tgt(\boldsymbol{p})} \\ inst(src(\boldsymbol{p})) & \xrightarrow[inst(\boldsymbol{p})]{} & inst(tgt(\boldsymbol{p})) \end{array}$$

Figure 1: Designation - details

○

○ A *designation* $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : src(\boldsymbol{p}) \rightrightarrows tgt(\boldsymbol{p})$, of a target classification $tgt(\boldsymbol{p})$ of *objects* by a source classification $src(\boldsymbol{p})$ of *signs*, consists of a covariant pair of reference functions,

– a *type* function $typ(\boldsymbol{p}) : typ(src(\boldsymbol{p})) \rightarrow typ(tgt(\boldsymbol{p}))$ mapping type signs to object types, and

– an *instance* function $inst(\boldsymbol{p}) : inst(sign(\boldsymbol{p})) \rightarrow inst(obj(\boldsymbol{p}))$ mapping instance signs to object instances.

Since signs denote objects, these functions are required to *preserve classification* – to map sign classification to instance classification – by satisfying the following implication:

$c \vDash_{src(\boldsymbol{p})} x$ implies $inst(\boldsymbol{p})(c) \vDash_{tgt(\boldsymbol{p})} typ(\boldsymbol{p})(x)$

for each instance sign $c \in inst(src(\boldsymbol{p}))$ and each type sign $x \in typ(src(\boldsymbol{p}))$.

```
(1) (SET$class designation)

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) designation)
    (= (SET.FTN$target source) cls$classification)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) designation)
    (= (SET.FTN$target target) cls$classification)

(4) (SET.FTN$function instance)
    (SET.FTN$function function2)
    (= function2 instance)
    (= (SET.FTN$source instance) designation)
    (= (SET.FTN$target instance) set.ftn$function)
    (= (SET.FTN$composition [instance set.ftn$source])
       (SET.FTN$composition [source cls$instance]))
    (= (SET.FTN$composition [instance set.ftn$target])
       (SET.FTN$composition [target cls$instance]))

(5) (SET.FTN$function type)
    (SET.FTN$function function1)
    (= function1 type)
    (= (SET.FTN$source type) designation)
    (= (SET.FTN$target type) set.ftn$function)
    (= (SET.FTN$composition [type set.ftn$source])
       (SET.FTN$composition [source cls$type]))
    (= (SET.FTN$composition [type set.ftn$target])
       (SET.FTN$composition [target cls$type]))

(6) (forall (?p (designation ?p)
             ?c ((cls$instance (source ?p)) ?c)
             ?x ((cls$type (source ?p)) ?x))
        (=> ((source ?p) ?c ?x)
            ((target ?p) ((instance ?p) ?c) ((type ?p) ?x)))))
```

Designations form the morphism class of the Designation category.

- Two designations are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable designations $\boldsymbol{p}_1 : \boldsymbol{A} \rightarrow \boldsymbol{A}'$ and $\boldsymbol{p}_2 : \boldsymbol{A}' \rightarrow \boldsymbol{A}''$ is defined in terms of the composition of their instance and type functions. The commitment to compose is based upon the agreement to identify the objects of the first designation with the signs of the second designation.

```
(7) (SET.LIM.PBK$opspan composable-opspan)
    (= (class1 composable-opspan) designation)
    (= (class2 composable-opspan) designation)
    (= (opvertex composable-opspan) cls$classification)
    (= (first composable-opspan) target)
    (= (second composable-opspan) source)

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

(9) (SET.FTN$function composition)
    (= (SET.FTN$source composition) (SET.LIM.PBK$pullback composable-opspan))
    (= (SET.FTN$target composition) designation)
    (forall (?p1 (designation ?p1) ?p2 (designation ?p2)
            (composable ?p1 ?p2))
        (and (= (source (composition [?p1 ?p2])) (source ?p1))
             (= (target (composition [?p1 ?p2])) (target ?p2))
             (= (instance (composition [?p1 ?p2]))
                (set.ftn$composition [(instance ?p1) (instance ?p2)]))
             (= (type (composition [?p1 ?p2]))
                (set.ftn$composition [(type ?p1) (type ?p2)]))))
```

- Composition satisfies the usual *associative law*.

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

- For any classification $\boldsymbol{A}$, there is an *identity* designation.

```
(10) (SET.FTN$function identity)
     (= (SET.FTN$source identity) cls$classification)
     (= (SET.FTN$target identity) designation)
     (forall (?a (cls$classification ?a))
         (and (= (source (identity ?a)) ?a)
              (= (target (identity ?a)) ?a)
              (= (instance (identity ?a))
                 (set.ftn$identity (cls$instance ?a)))
              (= (type (identity ?a))
                 (set.ftn$identity (cls$type ?a)))))
```

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

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

The category Designation has classifications as its objects and designations as its morphisms.

-

- Any designation has an underlying *function pair*.

```
(11) (SET.FTN$function pair)
     (= (SET.FTN$source pair) designation)
     (= (SET.FTN$target pair) set.ftn.pr$pair)
     (= (SET.FTN$composition [pair set.ftn.pr$function1]) function1)
     (= (SET.FTN$composition [pair set.ftn.pr$function2]) function2)
```

- A classification $\boldsymbol{A}$ is part of a classification $\boldsymbol{B}$ when they share the same instance and type sets and the incidence of $\boldsymbol{A}$ implies the incidence of $\boldsymbol{B}$. This situation is represent by an *inclusion* designation.

```
(12) (SET$class inclusion)
     (SET$subclass inclusion designation)
     (forall (?p (designation ?p))
         (<=> (inclusion ?p)
              (= (cls$instance (source ?p)) (cls$instance (target ?p)))
              (= (cls$type (source ?p)) (cls$type (target ?p)))))
```

Satisfaction

- Associated with any designation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{src}(\boldsymbol{p}) \rightrightarrows \mathit{tgt}(\boldsymbol{p})$ is a *power* designation $\wp\boldsymbol{p} = \langle \wp\mathit{inst}(\boldsymbol{p}), \wp\mathit{typ}(\boldsymbol{p})\rangle : \wp\mathit{src}(\boldsymbol{p}) \rightrightarrows \wp\mathit{tgt}(\boldsymbol{p})$.

  To verify that this is a designation, let $A \models_{\wp\mathit{src}(\boldsymbol{p})} X$ for some subset of instances $A \subseteq \mathit{inst}(\mathit{src}(\boldsymbol{p}))$ and some subset of types $X \subseteq \mathit{typ}(\mathit{src}(\boldsymbol{p}))$. Then there exists a function $s : X \rightarrow A$ such that $s(x) \models_{\mathit{src}(\boldsymbol{p})} x$ for all $x \in X$. For convenience of notation, define $B = \wp\mathit{inst}(\boldsymbol{p})(A) \subseteq \mathit{inst}(\mathit{tgt}(\boldsymbol{p}))$ and $Y = \wp\mathit{typ}(\boldsymbol{p})(X) \subseteq \mathit{typ}(\mathit{tgt}(\boldsymbol{p}))$. To preserve classification we need to show that $B \models_{\wp\mathit{tgt}(\boldsymbol{p})} Y$. The restriction functions $\wp\mathit{inst}(\boldsymbol{p})\lfloor_A : A \rightarrow B$ and $\wp\mathit{typ}(\boldsymbol{p})\lfloor_X : X \rightarrow Y$ are surjections; in particular the restricted type function has a left inverse injection $m : Y \rightarrow X$. This means that $\wp\mathit{typ}(\boldsymbol{p})(m(y)) = y$ for every $y \in Y$. Define the substitution function $t = m \cdot s \cdot \wp\mathit{inst}(\boldsymbol{p})\lfloor_A : Y \rightarrow B$. Now $s(m(y)) \models_{\mathit{src}(\boldsymbol{p})} m(y)$ for all $y \in Y$. Preservation of classification by $\boldsymbol{p}$ implies that $t(y) = \wp\mathit{inst}(\boldsymbol{p})(s(m(y))) \models_{\mathit{tgt}(\boldsymbol{p})} y$ for all $y \in Y$.

$$
\begin{tikzcd}
X \arrow[r, "\wp\mathit{typ}(\boldsymbol{p})\lfloor_X"] \arrow[d, "s"'] & Y \arrow[d, "t"] \\
A \arrow[r, "\wp\mathit{inst}(\boldsymbol{p})\lfloor_A"'] & B
\end{tikzcd}
$$

  Here we discuss an important special case of the power operator. Let $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \boldsymbol{A} \rightrightarrows \mathit{cls}(A)$ be a designation whose source is some classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$, and whose target is the identity classification for some set A. Then from the argument above, if $A \models_{\wp\mathit{src}(\boldsymbol{p})} X$ via a substitution function $s : X \rightarrow A$, then $t(y) = \wp\mathit{inst}(\boldsymbol{p})(s(m(y))) = y$ for all $y \in Y = \wp\mathit{typ}(\boldsymbol{p})(X) \subseteq \mathit{typ}(\mathit{tgt}(\boldsymbol{p}))$. If the restriction functions $\wp\mathit{inst}(\boldsymbol{p})\lfloor_A : A \rightarrow B$ and $\wp\mathit{typ}(\boldsymbol{p})\lfloor_X : X \rightarrow Y$ are not only surjections, but actually bijections, then there can be only one such substitution s.

```
(11) (SET.FTN$function power)
     (= (SET.FTN$source power) designation)
     (= (SET.FTN$target power) designation)
     (= (SET.FTN$composition [power source]) (SET.FTN$composition [source cls$power]))
     (= (SET.FTN$composition [power target]) (SET.FTN$composition [target cls$power]))
     (= (SET.FTN$composition [power instance])
        (SET.FTN$composition [instance set.ftn$power]))
     (= (SET.FTN$composition [power type])
        (SET.FTN$composition [type set.ftn$power]))
```

- Associated with any designation $\boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{src}(\boldsymbol{p}) \rightrightarrows \mathit{tgt}(\boldsymbol{p})$ is an *instance power* designation $\wp\mathit{inst}(\boldsymbol{p}) = \langle \mathit{inst}(\boldsymbol{p}), \wp\mathit{inst}(\boldsymbol{p})\rangle : \wp\mathit{inst}(\mathit{src}(\boldsymbol{p})) \rightrightarrows \wp\mathit{inst}(\mathit{tgt}(\boldsymbol{p}))$.

```
(11) (SET.FTN$function instance-power)
     (= (SET.FTN$source instance-power) designation)
     (= (SET.FTN$target instance-power) designation)
     (= (SET.FTN$composition [instance-power source])
        (SET.FTN$composition [source cls$instance-power]))
     (= (SET.FTN$composition [instance-power target])
```

```
            (SET.FTN$composition [target cls$instance-power]))
        (= (SET.FTN$composition [instance-power instance]) instance)
        (= (SET.FTN$composition [instance-power type])
            (SET.FTN$composition [instance set.ftn$power]))
```

○ Infomorphisms are exactly related to extent infomorphisms: they commute with the extent infomorphisms of source and target and the instance power infomorphism. However, since designations only require an implication in their constraint, they are not exactly related to extent infomorphisms. The fact that designations preserve classifications can be expressed pointwise in terms of extents as:

$$c \in ext_{src(\boldsymbol{p})}(x) \text{ implies } inst(\boldsymbol{p})(c) \in ext_{tgt(\boldsymbol{p})}(typ(\boldsymbol{p})(x))$$

for each source instance $a \in inst(src(\boldsymbol{p}))$ and each source type $a \in typ(src(\boldsymbol{p}))$. So, the type constraint for designations can be expressed pointlessly in terms of extents as:

1. $ext_{src(\boldsymbol{p})} \leq typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})} \cdot inst(\boldsymbol{p})^{-1}$ or equivalently
2. $ext_{src(\boldsymbol{p})} \cdot \wp inst(\boldsymbol{p}) \leq typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})}$.

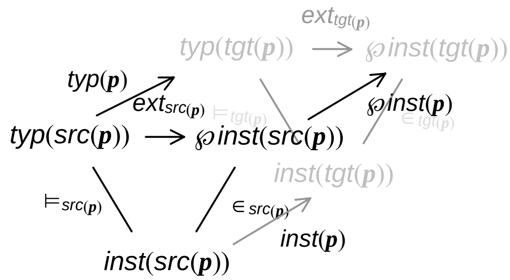


Figure 4: Designations and Extent

Here are the images of all four functions when applied to a source type $x \in typ(source(\boldsymbol{p}))$.

$$[ext_{src(\boldsymbol{p})}](x) = \{a \in \wp inst(src(\boldsymbol{p})) \mid a \vDash_{src(\boldsymbol{p})} x\}$$

$$[typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})} \cdot inst(\boldsymbol{p})^{-1}](x) = \{a \in \wp inst(src(\boldsymbol{p})) \mid inst(\boldsymbol{p})(a) \vDash_{tgt(\boldsymbol{p})} typ(\boldsymbol{p})(x)\}$$

$$[ext_{src(\boldsymbol{p})} \cdot \wp inst(\boldsymbol{p})](x) = \{b \in inst(tgt(\boldsymbol{p})) \mid b = inst(\boldsymbol{p})(a) \text{ for some } a \vDash_{src(\boldsymbol{p})} x\}$$

$$[typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})}](x) = \{b \in inst(obj(\boldsymbol{p})) \mid b \vDash_{tgt(\boldsymbol{p})} typ(\boldsymbol{p})(x)\}$$

Assume that instance and type sets of the target classification are the underlying sets of partial orders. Also, assume the classification is closed with respect to both orders. For such a target-ordered designation $\boldsymbol{p} = \langle inst(\boldsymbol{p}), typ(\boldsymbol{p})\rangle : src(\boldsymbol{p}) \rightrightarrows |tgt(\boldsymbol{p})|$ and any source type $x \in typ(src(\boldsymbol{p}))$,

– we say that $\boldsymbol{p}$ *represents* x when inequality 1 is an equality at x,

$$ext_{src(\boldsymbol{p})} \leq typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})} \cdot inst(\boldsymbol{p})^{-1}$$

– and we say that $\boldsymbol{p}$ *satisfies* x when inequality 2 is an equality at x, after closing under order.

$$\downarrow_{inst(tgt(\boldsymbol{p}))}(ext_{src(\boldsymbol{p})} \cdot \wp inst(\boldsymbol{p})) = typ(\boldsymbol{p}) \cdot ext_{tgt(\boldsymbol{p})}.$$

A designation represents a source type when source instances have that type iff their target instance images have the target image type. Designation satisfaction will be used to define model-theoretic satisfaction.

```
(12) (SET.FTN$function represents)
     (= (SET.FTN$source represents) designation)
     (= (SET.FTN$target represents) set$set)
     (forall (?p (designation ?p))
         (and (set$subset (represents ?p) (cls$type (source ?p)))
              (forall (?x ((cls$type (source ?p)) ?x))
                  (<=> ((represents ?p) ?x)
                       (= ((cls$extent (source ?p)) ?x)
                          ((set.ftn$inverse-image (instance ?p))
                               ((cls$extent (cls.ord$classification (target ?p)))
                                    ((type ?p) ?x))))))))

(13) (SET.FTN$function satisfies)
     (= (SET.FTN$source satisfies) designation)
     (= (SET.FTN$target satisfies) set$set)
     (forall (?p (designation ?p))
         (and (set$subset (satisfies ?p) (cls$type (source ?p)))
              (forall (?x ((cls$type (source ?p)) ?x))
                  (<=> ((satisfies ?p) ?x)
                       (= ((ord$down (cls.ord$type (target ?p)))
                              ((set.ftn$power (instance ?p))
                                  ((cls$extent (source ?p)) ?x)))
```

```
                                    ((cls$extent (cls.ord$classification (target ?p)))
                                        ((type ?p) ?x)))))))
```

- We extend the first notion to all type signs.
  - When inequality 1 is an equality (for all type signs), we call the designation an *isotaxy* (neologism for *equal arrangement*). Equivalently, an isotaxy $\boldsymbol{p}$ satisfies the logical equivalence:

    $a \vDash_{src(\boldsymbol{p})} x$ iff $inst(\boldsymbol{p})(a) \vDash_{tgt(\boldsymbol{p})} typ(\boldsymbol{p})(x)$

    for each source instance $a \in inst(src(\boldsymbol{p}))$ and each source type $x \in typ(src(\boldsymbol{p}))$. Clearly, a designation is an isotaxy when it represents all type signs.

```
(14) (SET$class isotaxy)
     (SET$subclass isotaxy designation)
     (forall (?p (designation ?p))
         (<=> (isotaxy ?p)
              (= (cls$extent (source ?p))
                 (set.ftn$composition
                     [(set.ftn$composition [(type ?p) (cls$extent (target ?p))])
                      (set.ftn$inverse-image (instance ?p))])))))
```

- For any designation $\boldsymbol{p} : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ the *inverse image* classification $\boldsymbol{A}^{@}$ makes $\boldsymbol{p}$ into an isotaxy. The source is a subclassification of the inverse image $\boldsymbol{A} \subseteq \boldsymbol{A}^{@}$ – the inclusion $\eta_{\boldsymbol{p}} : \boldsymbol{A} \subseteq \boldsymbol{A}^{@}$ is called the *induction*.

```
(15) (SET.FTN$function inverse-image)
     (= (SET.FTN$source inverse-image) designation)
     (= (SET.FTN$target inverse-image) cls$classification)
     (= (SET.FTN$composition [inverse-image cls$instance]) instance)
     (= (SET.FTN$composition [inverse-image cls$type]) type)
     (forall (?p (designation ?p)
              ?x2 ((cls$instance (source ?p)) ?x2)
              ?x1 ((cls$type (source ?p)) ?x1))
         (<=> ((inverse-image ?p) ?x2 ?x1)
              ((target ?p) ((instance ?p) ?x2) ((type ?p) ?x1))))

     (forall (?p (designation ?p)
              ?x2 ((cls$instance (source ?p)) ?x2)
              ?x1 ((cls$type (source ?p)) ?x1))
         (=> ((source ?p) ?x2 ?x1)
             ((inverse-image ?p) ?x2 ?x1)))

(16) (SET.FTN$function induction)
     (= (SET.FTN$source induction) designation)
     (= (SET.FTN$target induction) inclusion)
     (forall (?p (designation ?p)
         (and (= (source (induction ?p)) (source ?p))
              (= (target (induction ?p)) (inverse-image ?p))))
```

- For any isotaxy the inverse image is the source and the induction is the identity at the source.

```
     (forall (?p (isotaxy ?p))
         (= (inverse-image ?p) (source ?p)))
```

- Designation can be factored: For any designation $\boldsymbol{p} : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ the underlying function pair $pr(\boldsymbol{p}) : pr(\boldsymbol{A}) \rightrightarrows pr(\boldsymbol{B})$ and the target classification $\boldsymbol{B}$ are designable. The isotaxy $iso(\boldsymbol{p}) = \tilde{\boldsymbol{p}} = iso((pr(\boldsymbol{p}), \boldsymbol{B}))$ is called the *isotaxation* of $\boldsymbol{p}$. Any designation $\boldsymbol{p} : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ factors through its inverse image via its induction and its isotaxation

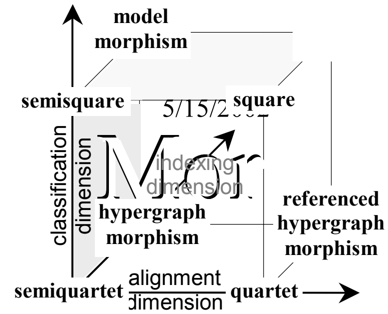
  $$\begin{tikzcd} \boldsymbol{A} \arrow[r, "\eta_{\boldsymbol{p}}"] \arrow[dr, "\boldsymbol{p}"'] & \boldsymbol{A}^{@} \arrow[d, "\tilde{\boldsymbol{p}}"] \\ & \boldsymbol{B} \end{tikzcd}$$

  $pr(\boldsymbol{p}) = \eta_{\boldsymbol{p}} \cdot \tilde{\boldsymbol{p}}$.

```
(17) (SET.FTN$function isotaxation)
     (= (SET.FTN$source isotaxation) designation)
     (= (SET.FTN$target isotaxation) isotaxy)
     (= (SET.FTN$composition [isotaxation source]) inverse-iamge)
     (= (SET.FTN$composition [isotaxation target]) target)
     (forall (?p (designation ?p))
         (= (isotaxation ?p) (set.ftn.pr$isotaxy [(pair ?p) (target ?p)])))

     (forall (?p (designation ?p))
```

```
(= ?p (composition [(induction ?p) (isotaxation ?p)])))
```



Classification Semisquares

`cls.ssqr`

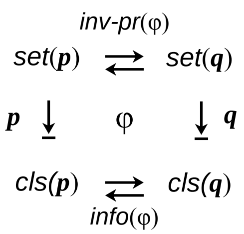


Figure 1: Classification Semisquare – abstract

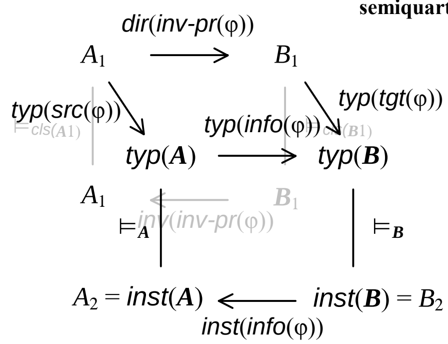


Figure 1: Classification Semisquare – details

- A *classification semisquare* φ = ⟨*inv-pr*(φ), *info*(φ)⟩ : *src*(φ) → *tgt*(φ) (Figure 1) from semidesignation *src*(φ) to semidesignation *tgt*(φ), consists of
  - an invertible pair *inv-pr*(φ) : *set*(*src*(φ)) ≅ *set*(*tgt*(φ)) and
  - an infomorphism *info*(φ) : *cls(src*(φ)) ⇄ *cls(tgt*(φ)),

  which preserve type reference

  dir(*inv-pr*(φ)) · *typ*(*tgt*(φ)) = *typ*(*src*(φ)) · *typ*(*info*(φ)).

  This means that the type component of φ forms a quartet. The instance component forms a semiquartet – it needs no constraint.

```
(1) (SET$class semisquare)

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) semisquare)
    (= (SET.FTN$target source) cls.sdsgn$semidesignation)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) semisquare)
    (= (SET.FTN$target target) cls.sdsgn$semidesignation)

(4) (SET.FTN$function invertible-pair)
    (= (SET.FTN$source invertible-pair) semisquare)
    (= (SET.FTN$target invertible-pair) set.invpr$invertible-pair)

(5) (SET.FTN$function infomorphism)
    (= (SET.FTN$source infomorphism) semisquare)
    (= (SET.FTN$target infomorphism) cls.info$infomorphism)
```

> Classification semisquares form the morphism class of the category ▲Classification.

- Two classification semisquares are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable classification semisquares is defined in terms of the composition of their invertible pairs and infomorphisms.

```
(6) (SET.LIM.PBK$opspan composable-opspan)
    (= (SET.LIM.PBK$class1 composable-opspan) semisquare)
    (= (SET.LIM.PBK$class2 composable-opspan) semisquare)
    (= (SET.LIM.PBK$opvertex composable-opspan) cls.sdsgn$semidesignation)
    (= (SET.LIM.PBK$first composable-opspan) target)
    (= (SET.LIM.PBK$second composable-opspan) source)
```

```
(7) (REL$relation composable)
    (= (REL$class1 composable) semisquare)
    (= (REL$class2 composable) semisquare)
    (= (REL$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) semisquare)
    (forall (?h1 (semisquare ?h1) ?h2 (semisquare ?h2) (composable ?h1 ?h2))
        (and (= (source (composition [?h1 ?h2])) (source ?h1))
             (= (target (composition [?h1 ?h2])) (target ?h2))
             (= (invertible-pair (composition [?h1 ?h2]))
                (set.invpr$composition [(invertible-pair ?h1) (invertible-pair ?h2)]))
             (= (infomorphism (composition [?h1 ?h2]))
                (cls.info$composition [(infomorphism ?h1) (infomorphism ?h2)]))))
```

- Composition satisfies the usual *associative law*.

```
(forall (?h1 (semisquare ?h1) ?h2 (semisquare ?h2) ?h3 (semisquare ?h3)
        (composable ?h1 ?h2) (composable ?h2 ?h3))
    (= (composition [?h1 (composition [?h2 ?h3])])
       (composition [(composition [?h1 ?h2]) ?h3])))
```

- For any semidesignation, there is an *identity* semisquare.

```
(9) (SET.FTN$function identity)
    (= (SET.FTN$source identity) cls.sdsgn$semidesignation)
    (= (SET.FTN$target identity) semisquare)
    (forall (?p (cls.sdsgn$semidesignation ?p))
        (and (= (source (identity ?p)) ?p)
             (= (target (identity ?p)) ?p)
             (= (invertible-pair (identity ?p))
                (set.ftn$identity (cls.sdsgn$set ?p)))
             (= (infomorphism (identity ?p))
                (cls.info$identity (cls.sdsgn$classification ?p)))))
```

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

```
(forall (?h (semisquare ?h))
    (and (= (composition [(identity (source ?h)) ?h]) ?h)
         (= (composition [?h (identity (target ?h))]) ?h)))
```

> The category ▲Classification has semidesignations as its objects, classification semisquares as its morphisms, semisquare composition as its composition function and identity as its identity function.

$$\text{Invertible-Pair} \xleftarrow{\;\mathit{set\text{-}pr}\;} \blacktriangle\text{Classification} \xrightarrow{\;\mathit{cla\text{-}info}\;} \text{Classification}$$

Figure 1: The pair-infomorphism span of categories and functors

- A classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$ also has the components:
  - an instance semiquartet $\mathit{inst}(\varphi)$ with
    $\mathit{src}(\mathit{inst}(\varphi)) = \mathit{inst}(\mathit{tgt}(\varphi))$, $\mathit{tgt}(\mathit{inst}(\varphi)) = \mathit{inst}(\mathit{src}(\varphi))$,
    $\mathit{ftn1}(\mathit{inst}(\varphi))) = \mathit{inv}(\mathit{inv\text{-}pr}(\varphi))$ and $\mathit{ftn2}(\mathit{inst}(\varphi))) = \mathit{inst}(\mathit{info}(\varphi))$; and
  - a type quartet $\mathit{typ}(\varphi) : \mathit{cls}(\mathit{src}(\varphi)) \rightleftarrows \mathit{cls}(\mathit{tgt}(\varphi))$,
    $\mathit{horiz\text{-}src}(\mathit{typ}(\varphi)) = \mathit{typ}(\mathit{src}(\varphi))$, $\mathit{horiz\text{-}tgt}(\mathit{typ}(\varphi)) = \mathit{typ}(\mathit{tgt}(\varphi))$,
    $\mathit{vert\text{-}src}(\mathit{typ}(\varphi)) = \mathit{dir}(\mathit{inv\text{-}pr}(\varphi))$, and $\mathit{vert\text{-}tgt}(\mathit{typ}(\varphi)) = \mathit{typ}(\mathit{info}(\varphi))$.

```
(10) (SET.FTN$function instance)
     (= (SET.FTN$source instance) semisquare)
     (= (SET.FTN$target instance) set.sqtt$semiquartet)
     (= (SET.FTN$composition [instance set.sqtt$source])
        (SET.FTN$composition [target cls.sdsgn$instance]))
     (= (SET.FTN$composition [instance set.sqtt$target])
        (SET.FTN$composition [source cls.sdsgn$instance]))
     (= (SET.FTN$composition [instance set.sqtt$function1])
        (SET.FTN$composition [invertible-pair set.invpr$inverse]))
     (= (SET.FTN$composition [instance set.sqtt$function2])
        (SET.FTN$composition [infomorphism cls.info$instance]))

(11) (SET.FTN$function type)
     (= (SET.FTN$source type) semisquare)
     (= (SET.FTN$target type) set.qtt$quartet)
     (= (SET.FTN$composition [type set.qtt$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$type]))
     (= (SET.FTN$composition [type set.qtt$horizontal-target])
        (SET.FTN$composition [target cls.sdsgn$type]))
     (= (SET.FTN$composition [type set.qtt$vertical-source])
        (SET.FTN$composition [invertible-pair set.invpr$direct]))
     (= (SET.FTN$composition [type set.qtt$vertical-target])
        (SET.FTN$composition [infomorphism cls.info$type]))
```

$$\triangle\mathrm{Set}^{op} \xleftarrow{\;inst\;} \blacktriangle\mathrm{Classification} \xrightarrow{\;typ\;} \mathrm{horiz}(\square\mathrm{Set})$$

Figure 1: The type-instance designation span of categories and functors

- The major work in this section will be to construct a spanmodel morphism from any classification square $\varphi = \langle inv\text{-}pr(\varphi), info(\varphi)\rangle : src(\varphi) \rightarrow tgt(\varphi)$. In order to do this we need to define the following components.
  - $sign(\varphi)$ – signature infomorphism,
  - $inst\text{-}indic(\varphi)$ – instance indication fibration,
  - $inst\text{-}proj(\varphi)$ – instance projection quartet,
  - $inst\text{-}comed(\varphi)$ – instance comediator quartet,
  - $typ\text{-}indic(\varphi) = indic(typ(\varphi))$ – type indication fibration (already defined),
  - $typ\text{-}proj(\varphi) = proj(typ(\varphi))$ – type projection quartet (already defined),
  - $typ\text{-}comed(\varphi) = comed(typ(\varphi))$ – type comediator quartet (already defined),
  - $case(\varphi)$ – case infomorphism,
  - $indic(\varphi)$ – indication square,
  - $proj(\varphi)$ – projection square,
  - $comed(\varphi)$ – comediator square, and
  - $spansqr(\varphi)$ – spansquare from $spandsgn(\boldsymbol{p})$ to $spandsgn(\boldsymbol{q})$.
- Associated with a classification semisquare φ, is a *signature* infomorphism (Figure 1)

  $sign(\varphi) = \langle tuple(\varphi), sign(typ(\varphi))\rangle : sign(src(\varphi)) \rightleftarrows sign(tgt(\varphi))$,

$$\begin{tikzcd}
sign(typ(src(\varphi))) \arrow[r, "sign(typ(\varphi))"] \arrow[d, dash, "\models_{sign(src(\varphi))}"'] & sign(typ(tgt(\varphi))) \arrow[d, dash, "\models_{sign(tgt(\varphi))}"] \\
tuple(src(\varphi)) & tuple(tgt(\varphi)) \arrow[l, "tuple(\varphi)"]
\end{tikzcd}$$

Figure 1: Signature Infomorphism

where the tuple function $tuple(\varphi) : tuple(tgt(\varphi)) \rightarrow tuple(src(\varphi))$ is the restriction of the tuple function of the instance semisquare $tuple(inst(\varphi)) : tuple(inst(tgt(\varphi))) \rightarrow typ(inst(src(\varphi))$ to admissible tuples; that is, tuples that are compatible with the respective type function. We need to verify two things.

– To verify that the tuple function maps admissible tuples to admissible tuples, let t be a tuple of the instance set pair $inst(tgt(\varphi))$, $t \in tuple(inst(tgt(\varphi)))$,

  * $t : arity(t) \rightarrow inst(cls(tgt(\varphi))) = inst(\boldsymbol{B}) = B_2$ with $arity(t) \subseteq set(tgt(\varphi)) = B_1$,

  and let s be the tuple of the instance set pair $inst(src(\varphi))$, $s \in tuple(inst(src(\varphi)))$,

  * $s = tuple(\varphi)(t) : arity(s) \rightarrow inst(cls(src(\varphi))) = inst(\boldsymbol{A}) = A_2$ with $arity(s) \subseteq set(src(\varphi)) = A_1$

  which is the image of t under the tuple function. Then we know that

  $arity(s) = \wp inv(inv\text{-}pr(\varphi))(arity(t))$, and

  $t \cdot inst(info(\varphi)) = inv(inv\text{-}pr(\varphi))\lfloor_{arity(t)} \cdot s$,

  where $inv(inv\text{-}pr(\varphi))\lfloor_{arity(t)}$ is the restriction of $inv(inv\text{-}pr(\varphi))$ to $arity(t)$.

  The string of logical equivalences

  $s(x) \vDash_{src(\varphi)} typ(src(\varphi))(x)$ for all $x \in set(src(\varphi))$

  iff $inst(info(\varphi))(t(y)) \vDash_{src(\varphi)} typ(src(\varphi))(x)$ where $y = inv(inv\text{-}pr(\varphi))(x)$

  iff $t(y) \vDash_{tgt(\varphi)} typ(info(\varphi))(typ(src(\varphi))(x))$ or $x = dir(inv\text{-}pr(\varphi))(y)$

  iff $t(y) \vDash_{tgt(\varphi)} typ(tgt(\varphi))(y)$ for all $y \in set(tgt(\varphi))$

  show that s is admissible, $s \in tuple(src(\varphi))$, iff t is admissible, $t \in tuple(tgt(\varphi))$.

○ To verify that this is an infomorphism, let t be a tuple of the instance set pair $inst(tgt(\varphi))$ and let σ be a signature of the type function $typ(src(\varphi)) : set(src(\varphi)) \rightarrow typ(cls(src(\varphi)))$,

  * $t : arity(t) \rightarrow inst(cls(tgt(\varphi))) = inst(\boldsymbol{B}) = B_2$ with $arity(t) \subseteq set(tgt(\varphi)) = B_1$,
  * $\sigma : arity(\sigma) \rightarrow typ(cls(src(\varphi))) = typ(\boldsymbol{A})$ with $arity(\sigma) \subseteq set(src(\varphi)) = A_1$.

  For ease of notation, make the following definitions:

  * $s = tuple(inst(\varphi))(t) : arity(s) \rightarrow inst(cls(src(\varphi))) = inst(\boldsymbol{A}) = A_2$ with $arity(s) \subseteq set(src(\varphi)) = A_1$,
  * $\tau = sign(typ(\varphi))(\sigma) : arity(\tau) \rightarrow typ(cls(tgt(\varphi))) = typ(\boldsymbol{B})$ with $arity(\tau) \subseteq set(tgt(\varphi)) = B_1$.

  We need to show that the following fundamental property of infomorphisms holds:

  $s \vDash_{sign(src(\varphi))} \sigma$ iff $t \vDash_{sign(tgt(\varphi))} \tau$.

  We know that

  $arity(s) = \wp inv(inv\text{-}pr(\varphi))(arity(t))$, and

  $t \cdot inst(info(\varphi)) = inv(inv\text{-}pr(\varphi))\lfloor_{arity(t)} \cdot s$,

  where $inv(inv\text{-}pr(\varphi))\lfloor_{arity(t)}$ is the restriction of $inv(inv\text{-}pr(\varphi))$ to $arity(t)$.

  We also know that

  $arity(\tau) = \wp dir(inv\text{-}pr(\varphi))(arity(\sigma))$, and

  $\sigma \cdot typ(info(\varphi)) = dir(inv\text{-}pr(\varphi))\lfloor_{arity(\sigma)} \cdot \tau$,

  where $dir(inv\text{-}pr(\varphi))\lfloor_{arity(\sigma)}$ is the restriction of $dir(inv\text{-}pr(\varphi))$ to $arity(\sigma)$.

  Now $s \vDash_{sign(src(\varphi))} \sigma$

  iff $arity(s) \supseteq arity(\sigma)$ and $s(x) \vDash_{cls(src(\varphi))} \sigma(x)$ for all elements $x \in arity(\sigma) \subseteq set(src(\varphi))$

  iff $\wp inv(inv\text{-}pr(\varphi))(arity(t)) \supseteq arity(\sigma)$ and $inst(info(\varphi))(t(y)) \vDash \sigma(x)$ where $y = inv(inv\text{-}pr(\varphi))(x)$

  iff $arity(t) \supseteq \wp dir(inv\text{-}pr(\varphi))(arity(\sigma))$ and $t(y) \vDash typ(info(\varphi))(\sigma(x))$ or $x = dir(inv\text{-}pr(\varphi))(y)$

  iff $arity(t) \supseteq \wp dir(inv\text{-}pr(\varphi))(arity(\sigma))$

    and $t(y) \vDash \tau(y)$ for all elements $y \in arity(\tau) \subseteq set(tgt(\varphi))$

iff $t \vDash_{\mathrm{sign}(\mathrm{tgt}(\varphi))} \tau$.

```
(12) (SET.FTN$function tuple)
     (= (SET.FTN$source tuple) semisquare)
     (= (SET.FTN$target tuple) set.ftn$function)
     (= (SET.FTN$composition [tuple set.ftn$source])
        (SET.FTN$composition [target cls.sdsgn$tuple]))
     (= (SET.FTN$composition [tuple set.ftn$target])
        (SET.FTN$composition [source cls.sdsgn$tuple]))
     (forall (?h (semisquare ?h))
         (set.ftn$restriction (tuple ?h) (set.sqtt$tuple (instance ?h))))

(13) (SET.FTN$function signature)
     (= (SET.FTN$source signature) semisquare)
     (= (SET.FTN$target signature) cls.info$infomorphism)
     (= (SET.FTN$composition [signature cls.info$source])
        (SET.FTN$composition [source cls.sdsgn$signature]))
     (= (SET.FTN$composition [signature cls.info$target])
        (SET.FTN$composition [target cls.sdsgn$signature]))
     (= (SET.FTN$composition [signature cls.info$instance]) tuple)
     (= (SET.FTN$composition [signature cls.info$type])
        (SET.FTN$composition [type set.qtt$signature]))
```

> The signature infomorphism associated with any classification semisquare is the image of the morphism function of the signature functor applied to the classification semisquare:
>
> $$\mathit{sign} : \blacktriangle\mathrm{Classification} \rightarrow \mathrm{Classification}.$$

$$\begin{tikzcd}
\blacktriangle\mathrm{Classification} \arrow[r, "\mathit{sign}"] \arrow[d, "\mathit{inst}"'] \arrow[dr, "\mathit{tuple}"'] & \mathrm{Classification} \arrow[d, "\mathit{inst}"] \\
\triangle\mathrm{Set}^{\mathrm{op}} \arrow[r, "\mathit{tuple}^{\mathrm{op}}"'] & \mathrm{Set}^{\mathrm{op}}
\end{tikzcd}
\qquad
\begin{tikzcd}
\blacktriangle\mathrm{Classification} \arrow[r, "\mathit{sign}"] \arrow[d, "\mathit{typ}"'] & \mathrm{Classification} \arrow[d, "\mathit{typ}"] \\
\mathit{horiz}(\Box\mathrm{Set}) \arrow[r, "\mathit{sign}"'] & \mathrm{Set}
\end{tikzcd}$$

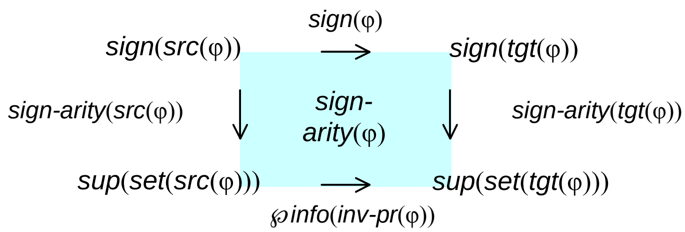


Figure 1: The signature arity classification square – abstract

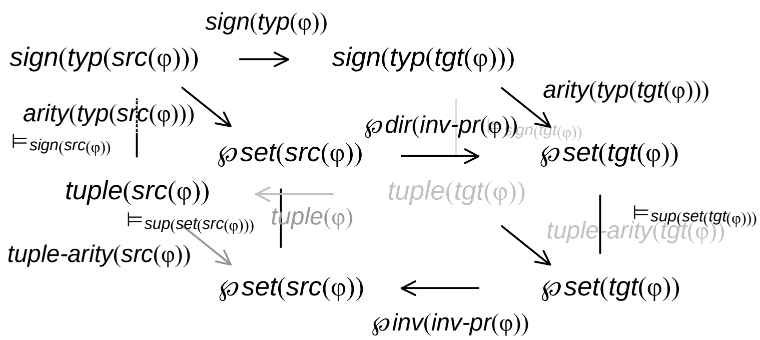


Figure 1: The signature arity classification square – details

- The signature infomorphism allows us to define a signature arity classification square (Figure 1) *sign-arity*(φ). This defines a functor from ▲Classification to the horizontal category *horiz*(■Classification). Since the instance quartet of this signature square is not exactly the same as the tuple arity quartet of the instance semiquartet, due to the admissible restriction on the tuple function, we define this first.

```
(13) (SET.FTN$function tuple-arity)
     (= (SET.FTN$source tuple-arity) semisquare)
     (= (SET.FTN$target tuple-arity) set.qtt$quartet)
     (= (SET.FTN$composition [tuple-arity set.qtt$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$tuple-arity]))
     (= (SET.FTN$composition [tuple-arity set.qtt$horizontal-target])
        (SET.FTN$composition [target cls.sdsgn$tuple-arity]))
     (= (SET.FTN$composition [tuple-arity set.qtt$vertical-source]) tuple)
     (= (SET.FTN$composition [tuple-arity set.qtt$vertical-target])
        (SET.FTN$composition
            [(SET.FTN$composition [invertible-pair set.invpr$inverse]) set.ftn$power]))

(14) (SET.FTN$function signature-arity)
     (= (SET.FTN$source signature-arity) semisquare)
     (= (SET.FTN$target signature-arity) cls.sqr$square)
     (= (SET.FTN$composition [signature-arity cls.sqr$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$signature-arity]))
     (= (SET.FTN$composition [signature-arity cls.sqr$horizontal-target])
        (SET.FTN$composition [target cls.sdsgn$signature-arity]))
     (= (SET.FTN$composition [signature-arity cls.sqr$vertical-source]) signature)
     (= (SET.FTN$composition [signature-arity cls.sqr$vertical-target])
        (SET.FTN$composition
            [(SET.FTN$composition [invertible-pair set.invpr$infomorphism]) cls.info$power]))
     (= (SET.FTN$composition [signature-arity cls.sqr$instance]) tuple-arity)
     (= (SET.FTN$composition [signature-arity cls.sqr$type])
        (SET.FTN$composition [type set.qtt$signature-arity]))
```

> The signature-arity classification square associated with any classification semisquare is the image of the morphism function of the signature arity functor applied to the classification semisquare:
>
> *sign-arity* : ▲Classification → *horiz*(■Classification).

- Figure 2 displays several commutting diagrams connected with the instance, type and signature-arity functors.

$$\begin{tikzcd}
\text{Classification} \arrow[d, "id"] & \blacktriangle\text{Classification} \arrow[l, "\textit{sign}"'] \arrow[r, "\textit{inv-pr}"] \arrow[d, "\textit{sign-arity}"] & \text{Invertible-Pair} \arrow[dr, "\textit{info}"] & \\
\text{Classification} & \text{horiz}(\blacksquare\text{Classification}) \arrow[l, "\textit{vert-src}"] \arrow[r, "\textit{vert-tgt}"'] & \text{Classification} & \text{Classification} \arrow[l, "\wp"]
\end{tikzcd}$$

Figure 1: Signature and Signature-Arity Func-

$$\begin{tikzcd}
\square\text{Set}^{op} \arrow[d, "\textit{tuple-arity}^{op}"] & \blacktriangle\text{Classification} \arrow[l, "\textit{inst}"'] \arrow[r, "\textit{typ}"] \arrow[d, "\textit{sign-arity}"] & \square\text{Set} \arrow[d, "\textit{sign-arity}"] \\
\square\text{Set}^{op} & \text{horiz}(\blacksquare\text{Classification}) \arrow[l, "\textit{inst}"'] \arrow[r, "\textit{typ}"] & \square\text{Set}
\end{tikzcd}$$

Figure 2: Instance, Type and Signature-arity Functors

-
-

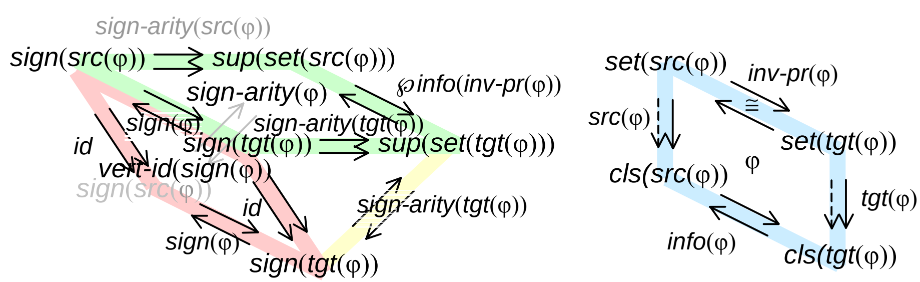


Figure 1: Model morphism of a classification semisquare

- Any classification semisquare

  $\varphi = \langle \textit{inv-pr}(\varphi), \textit{info}(\varphi)\rangle : \textit{src}(\varphi) \rightarrow \textit{tgt}(\varphi)$

  defines a *model morphism*(Figure 3), whose reference semisquare is itself and whose signature square is the vertical identity square *vert-id* at *sign*(φ) the signature classification of φ. This model morphism has *inv-pr*(φ) as its bijection of variables, *inst*(*info*(φ)) as its map between universes of the source and target models, *typ*(*info*(φ)) as its entity type function, *tuple*(φ) as its map between tuples of the source and target models, and *sign*(*typ*(φ)) as its relation type function. Clearly, the semisquare of the model morphism of a semisquare is itself.

```
(14) (SET.FTN$function model-morphism)
     (= (SET.FTN$source model-morphism) semisquare)
     (= (SET.FTN$target model-morphism) mod.mor$model-morphism)
     (forall (?h (semisquare ?h))
```

```
          (and (= (mod.mor$source (model-morphism ?h)) (cls.sdsgn$model (source ?h)))
               (= (mod.mor$target (model-morphism ?h)) (cls.sdsgn$model (target ?h)))
               (= (mod.mor$reference (model-morphism ?h)) ?h)
               (= (mod.mor$signature (model-morphism ?h))
                  (cls.sqr$vertical-identity (signature ?h)))))
```

○ Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$ defines a coproduct *instance arity* morphism. We cannot use the coproduct arity morphism of the instance semiquartet, since the tuple functions are not exactly the same. The underlying quartet is tuple arity.

```
(15) (SET.FTN$function instance-arity)
     (= (SET.FTN$source instance-arity) semisquare)
     (= (SET.FTN$target instance-arity) set.col.art.mor$arity-morphism)
     (forall (?h (semisquare ?h))
         (and (= (set.col.art.mor$source (instance-arity ?h))
                 (cls.ssqr$instance-arity (source ?h)))
              (= (set.col.art.mor$target (instance-arity ?h))
                 (cls.ssqr$instance-arity (target ?h)))
              (= (set.col.art.mor$index (instance-arity ?h)) (tuple ?h))
              (= (set.col.art.mor$base (instance-arity ?h))
                 (set.invpr$inverse (invertible-pair ?h)))))

     (forall (?h (semisquare ?h))
          (= (set.col.art.mor$quartet (instance-arity ?h)) tuple-arity))
```

○

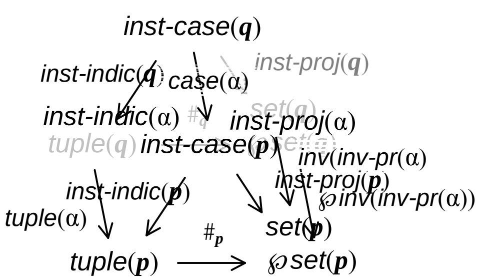


Figure 1: Coproduct of the instance arity of a semisquare

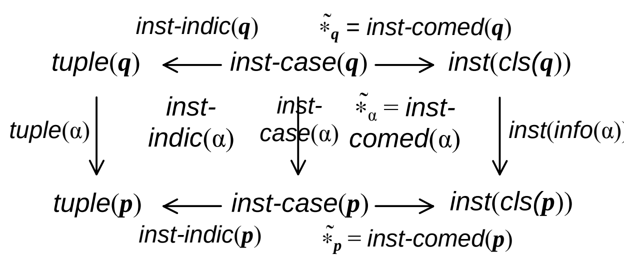


Figure 2: Instance spangraph morphism of a semisquare

○ Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$,

from semidesignation $\mathit{src}(\varphi) = \boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$

to semidesignation $\mathit{tgt}(\varphi) = \boldsymbol{q} = \langle \mathit{inst}(\boldsymbol{q}), \mathit{typ}(\boldsymbol{q})\rangle : \mathit{set}(\boldsymbol{q}) \looparrowright \mathit{cls}(\boldsymbol{q})$

with components $\mathit{inv\text{-}pr}(\varphi) : \mathit{set}(\boldsymbol{p}) \cong \mathit{set}(\boldsymbol{q})$ and $\mathit{info}(\varphi) : \mathit{cls}(\boldsymbol{p}) \rightarrow \mathit{cls}(\boldsymbol{q})$,

defines a *instance case* or *instance thematic role* function (in the opposite direction)

$\mathit{inst\text{-}case}(\varphi) : \mathit{inst\text{-}case}(\boldsymbol{q}) = \sum \mathit{arity}(\boldsymbol{q}) \rightarrow \sum \mathit{arity}(\boldsymbol{p}) = \mathit{inst\text{-}case}(\boldsymbol{p})$.

The pointwise definition is: $\mathit{inst\text{-}case}(\varphi)((t, y)) = (\mathit{tuple}(\varphi)(t), \mathit{inv}(\mathit{inv\text{-}pr}(\varphi))(y))$ for any tuple $t \in \mathit{tuple}(\boldsymbol{q})$ and any name $y \in \mathit{arity}(\boldsymbol{q})(t)$. This is well defined since α preserves tuple arity.

```
(16) (SET.FTN$function instance-case)
     (= (SET.FTN$source instance-case) semisquare)
     (= (SET.FTN$target instance-case) set.ftn$function)
     (forall (?h (semisquare ?h))
         (and (= (set.ftn$source (instance-case ?h))
                 (cls.sdsgn$instance-case (target ?h)))
              (= (set.ftn$target (instance-case ?h))
                 (cls.sdsgn$instance-case (source ?h)))
              (= (instance-case ?p)
                 (set.col.art.mor$coproduct (instance-arity ?h)))))
```

○ The instance case function is the vertical source for two quartets:

an *instance indication* quartet

$\mathit{inst\text{-}indic}(\varphi) = \langle \mathit{inst\text{-}case}(\varphi), \mathit{tuple}(\varphi)\rangle : \mathit{inst\text{-}indic}(\boldsymbol{q}) \rightarrow \mathit{inst\text{-}indic}(\boldsymbol{p})$, and

an *instance projection* quartet

$\mathit{inst\text{-}proj}(\alpha) = \langle \mathit{inst\text{-}case}(\varphi), \mathit{inv}(\mathit{inv\text{-}pr}(\alpha))\rangle : \mathit{inst\text{-}proj}(\boldsymbol{q}) \rightarrow \mathit{inst\text{-}proj}(\boldsymbol{p})$.

– The commutativity $\mathit{inst\text{-}case}(\alpha) \cdot \mathit{inst\text{-}indic}(\boldsymbol{p}) = \mathit{inst\text{-}indic}(\boldsymbol{q}) \cdot \mathit{tuple}(\alpha)$, a property of the coproduct of arities (preservation of index), is obvious from the pointwise definition of the case function.
– The commutativity $\mathit{inst\text{-}case}(\alpha) \cdot \mathit{inst\text{-}proj}(\boldsymbol{p}) = \mathit{inst\text{-}proj}(\boldsymbol{q}) \cdot \mathit{inv}(\mathit{inv\text{-}pr}(\alpha)$, a property of the coproduct of arities (preservation of projection), is obvious from the pointwise definition of the case function.

Even though the inverse function of the invertible pair $\mathit{inv}(\mathit{inv\text{-}pr}(\alpha)) : \mathit{set}(\boldsymbol{q}) \rightarrow \mathit{set}(\boldsymbol{p})$ and its power $\wp\mathit{inv}(\mathit{inv\text{-}pr}(\alpha))$ are bijections, the instance case function $\mathit{inst\text{-}case}(\alpha) : \mathit{inst\text{-}case}(\boldsymbol{q}) \rightarrow \mathit{inst\text{-}case}(\boldsymbol{p})$ is not necessarily a bijection since the tuple function $\mathit{tuple}(\alpha) : \mathit{tuple}(\boldsymbol{q}) \rightarrow \mathit{tuple}(\boldsymbol{p})$ need not be bijective (and this in turn since the instance function of the infomorphism $\mathit{inst}(\mathit{info}(\alpha)) : \mathit{inst}(\mathit{cls}(\boldsymbol{q})) \rightarrow \mathit{inst}(\mathit{cls}(\boldsymbol{p}))$ need not be bijective). Abstractly by the preservation of tuple arity and concretely by the bijective nature of $\mathit{inv\text{-}pr}(\alpha)$, the indication quartet is a fibration: for any tuple $t \in \mathit{tuple}(\boldsymbol{q})$ and any name $x \in \mathit{arity}(\boldsymbol{p})(\mathit{tuple}(\alpha)(t)) = \mathit{inv}(\mathit{inv\text{-}pr}(\alpha))\lfloor_{\mathit{arity}(t)}$ there is a name $x \in \mathit{arity}(\boldsymbol{q})(t)$ such that $\mathit{inv}(\mathit{inv\text{-}pr}(\alpha))(\alpha)(x) = y$.

```
(17) (SET.FTN$function instance-indication)
     (= (SET.FTN$source instance-indication) semisquare)
     (= (SET.FTN$target instance-indication) set.qtt$fibration)
     (forall (?h (semisquare ?h))
         (and (= (set.qtt$horizontal-source (instance-indication ?h))
                 (cls.sdsgn$instance-indication (target ?h)))
              (= (set.qtt$horizontal-target (instance-indication ?h))
                 (cls.sdsgn$instance-indication (source ?h)))
              (= (set.qtt$vertical-source (instance-indication ?h)) (instance-case ?h))
              (= (set.qtt$vertical-target (instance-indication ?h)) (tuple ?h))))

(18) (SET.FTN$function instance-projection)
     (= (SET.FTN$source instance-projection) semisquare)
     (= (SET.FTN$target instance-projection) set.qtt$quartet)
     (forall (?h (semisquare ?h))
         (and (= (set.qtt$horizontal-source (instance-projection ?h))
                 (cls.sdsgn$instance-projection (target ?h)))
              (= (set.qtt$horizontal-target (instance-projection ?h))
                 (cls.sdsgn$instance-projection (source ?h)))
              (= (set.qtt$vertical-source (instance-projection ?p)) (instance-case ?p))
              (= (set.qtt$vertical-target (instance-projection ?p))
                 (set.invpr$inverse (invertible-pair ?h)))))
```

○ Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$,

from semidesignation $\mathit{src}(\varphi) = \boldsymbol{p} = \langle \mathit{inst}(\boldsymbol{p}), \mathit{typ}(\boldsymbol{p})\rangle : \mathit{set}(\boldsymbol{p}) \looparrowright \mathit{cls}(\boldsymbol{p})$

to semidesignation $\mathit{tgt}(\varphi) = \boldsymbol{q} = \langle \mathit{inst}(\boldsymbol{q}), \mathit{typ}(\boldsymbol{q})\rangle : \mathit{set}(\boldsymbol{q}) \looparrowright \mathit{cls}(\boldsymbol{q})$

with components $\mathit{inv\text{-}pr}(\varphi) : \mathit{set}(\boldsymbol{p}) \cong \mathit{set}(\boldsymbol{q})$ and $\mathit{info}(\varphi) : \mathit{cls}(\boldsymbol{p}) \rightarrow \mathit{cls}(\boldsymbol{q})$,

has an *instance comediator* quartet

$\mathit{inst\text{-}comed}(\varphi) = \langle \mathit{inst\text{-}case}(\varphi), \mathit{inst}(\mathit{info}(\varphi))\rangle : \mathit{inst\text{-}comed}(\boldsymbol{q}) \rightarrow \mathit{inst\text{-}comed}(\boldsymbol{p})$.

```
(19) (SET.FTN$function instance-comediator)
     (= (SET.FTN$source instance-comediator) semisquare)
     (= (SET.FTN$target instance-comediator) set.qtt$quartet)
     (forall (?h (semisquare ?h))
         (and (= (set.qtt$horizontal-source (instance-comediator ?h))
                 (cls.sdsgn$instance-comediator (target ?h)))
              (= (set.qtt$horizontal-target (instance-comediator ?h))
                 (cls.sdsgn$instance-comediator (source ?h)))
              (= (set.qtt$vertical-source (instance-comediator ?h)) (instance-case ?h))
              (= (set.qtt$vertical-target (instance-comediator ?h))
                 (cls.info$instance (infomorphism ?h)))))
```

- There is a *case* infomorphism

  $$\mathit{case}(\varphi) = \langle \mathit{inst\text{-}case}(\varphi), \mathit{case}(\mathit{typ}(\boldsymbol{p}))\rangle : \mathit{case}(\boldsymbol{p}) \rightleftarrows \mathit{case}(\boldsymbol{q}),$$

  whose instance function is the instance case function

  $$\mathit{inst\text{-}case}(\varphi) : \mathit{inst\text{-}case}(\boldsymbol{q}) \rightarrow \mathit{inst\text{-}case}(\boldsymbol{p})$$

  and whose type function is the case function

  $$\mathit{case}(\mathit{typ}(\boldsymbol{p})) : \mathit{case}(\mathit{typ}(\boldsymbol{p})) \rightarrow \mathit{case}(\mathit{typ}(\boldsymbol{q}))$$

  of the type quartet of φ. The fundamental condition for infomorphisms is expressed as follows.

  $$\mathit{inst\text{-}case}(\varphi)(t, y) \vDash_{\mathit{case}(\boldsymbol{p})} (\tau, x) \text{ iff } (t, y) \vDash_{\mathit{case}(\boldsymbol{p})} \mathit{case}(\mathit{typ}(\varphi))(\tau, x)$$

  $$\begin{tikzcd} \mathit{case}(\mathit{typ}(\boldsymbol{p})) \arrow[r, "\mathit{case}(\mathit{typ}(\varphi))"] \arrow[d, dash, "\vDash_{\mathit{case}(\boldsymbol{p})}"'] & \mathit{case}(\mathit{typ}(\boldsymbol{q})) \arrow[d, dash, "\vDash_{\mathit{case}(\boldsymbol{q})}"] \\ \mathit{inst\text{-}case}(\boldsymbol{p}) & \mathit{inst\text{-}case}(\boldsymbol{p}) \arrow[l, "\mathit{inst\text{-}case}(\varphi)"] \end{tikzcd}$$

  Figure 1: Case Infomorphism

  for any tuple $t \in \mathit{tuple}(\boldsymbol{q})$ and name $y \in \mathit{inst\text{-}arity}(\boldsymbol{q})(t) \subseteq \mathit{set}(\boldsymbol{q})$ and any signature $\tau \in \mathit{sign}(\mathit{typ}(\boldsymbol{p}))$ and name $x \in \mathit{arity}(\mathit{typ}(\boldsymbol{p}))(\tau) \subseteq \mathit{set}(\boldsymbol{p})$.

```
(20) (SET.FTN$function case)
     (= (SET.FTN$source case) semisquare)
     (= (SET.FTN$target case) cls.info$infomorphism)
     (forall (?h (semisquare ?h))
         (and (= (cls.info$source (case ?h)) (cls.sdsgn$case (source ?h)))
              (= (cls.info$target (case ?h)) (cls.sdsgn$case (target ?h)))
              (= (cls.info$instance (case ?h)) (instance-case ?h))
              (= (cls.info$type (case ?h)) (set.qtt$case (type ?h)))))
```

- Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$, has an *indication* classification square $\mathit{indic}(\varphi) = \langle \mathit{case}(\varphi), \mathit{sign}(\varphi)\rangle : \mathit{indic}(\boldsymbol{p}) \rightarrow \mathit{indic}(\boldsymbol{q})$, with instance fibration $\mathit{inst\text{-}indic}(\varphi)$ and type fibration $\mathit{typ\text{-}indic}(\varphi) = \mathit{indic}(\mathit{typ}(\varphi))$.

```
(21) (SET.FTN$function indication)
     (= (SET.FTN$source indication) semisquare)
     (= (SET.FTN$target indication) cls.sqr$square)
     (= (SET.FTN$composition [indication cls.sqr$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$indication]))
     (= (SET.FTN$composition [indication cls.sqr$horizontal-target])
        (SET.FTN$composition [target cls.sdsgn$indication]))
     (= (SET.FTN$composition [indication cls.sqr$vertical-source]) case)
     (= (SET.FTN$composition [indication cls.sqr$vertical-target]) signature)
     (= (SET.FTN$composition [indication cls.sqr$instance]) instance-indication)
     (= (SET.FTN$composition [indication cls.sqr$type])
        (SET.FTN$composition [type set.qtt$indication]))
```

- Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$, has a *projection* classification square $\mathit{proj}(\varphi) = \langle \mathit{case}(\varphi), \mathit{inv\text{-}pr}(\varphi)\rangle : \mathit{proj}(\boldsymbol{p}) \rightarrow \mathit{proj}(\boldsymbol{q})$, with instance quartet $\mathit{inst\text{-}proj}(\varphi)$ and type quartet $\mathit{typ\text{-}proj}(\varphi) = \mathit{proj}(\mathit{typ}(\varphi))$.

```
(22) (SET.FTN$function projection)
     (= (SET.FTN$source projection) semisquare)
     (= (SET.FTN$target projection) cls.sqr$square)
     (= (SET.FTN$composition [projection cls.sqr$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$projection]))
     (= (SET.FTN$composition [projection cls.sqr$horizontal-target])
        (SET.FTN$composition [target cls.sdsgn$projection]))
     (= (SET.FTN$composition [projection cls.sqr$vertical-source]) case)
     (= (SET.FTN$composition [projection cls.sqr$vertical-target]) invertible-pair)
     (= (SET.FTN$composition [projection cls.sqr$instance]) instance-projection)
     (= (SET.FTN$composition [projection cls.sqr$type])
        (SET.FTN$composition [type set.qtt$projection]))
```

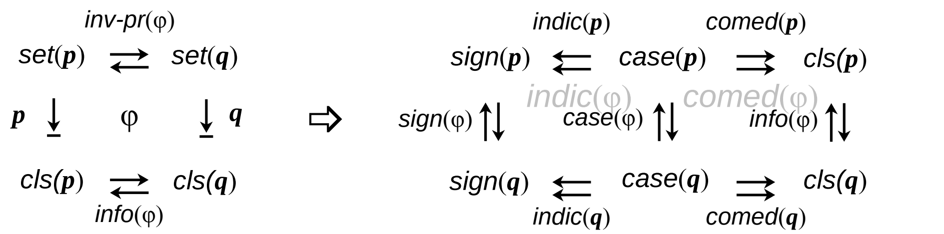
- Any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$, has a *comediator* classification square $\mathit{comed}(\varphi) = \langle \mathit{case}(\varphi), \mathit{info}(\varphi)\rangle : \mathit{comed}(\boldsymbol{p}) \rightarrow \mathit{comed}(\boldsymbol{q})$, with instance quartet $\mathit{inst\text{-}comed}(\varphi)$ and type quartet $\mathit{typ\text{-}comed}(\varphi) = \mathit{comed}(\mathit{typ}(\varphi))$.

```
(23) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) semisquare)
     (= (SET.FTN$target comediator) cls.sqr$square)
     (= (SET.FTN$composition [comediator cls.sqr$horizontal-source])
        (SET.FTN$composition [source cls.sdsgn$comediator]))
```

```
(= (SET.FTN$composition [comediator cls.sqr$horizontal-target])
   (SET.FTN$composition [target cls.sdsgn$comediator]))
(= (SET.FTN$composition [comediator cls.sqr$vertical-source]) case)
(= (SET.FTN$composition [comediator cls.sqr$vertical-target]) infomorphism)
(= (SET.FTN$composition [comediator cls.sqr$instance]) instance-comediator)
(= (SET.FTN$composition [comediator cls.sqr$type])
   (SET.FTN$composition [type set.qtt$comediator]))
```

○

○



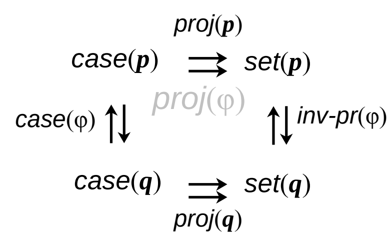
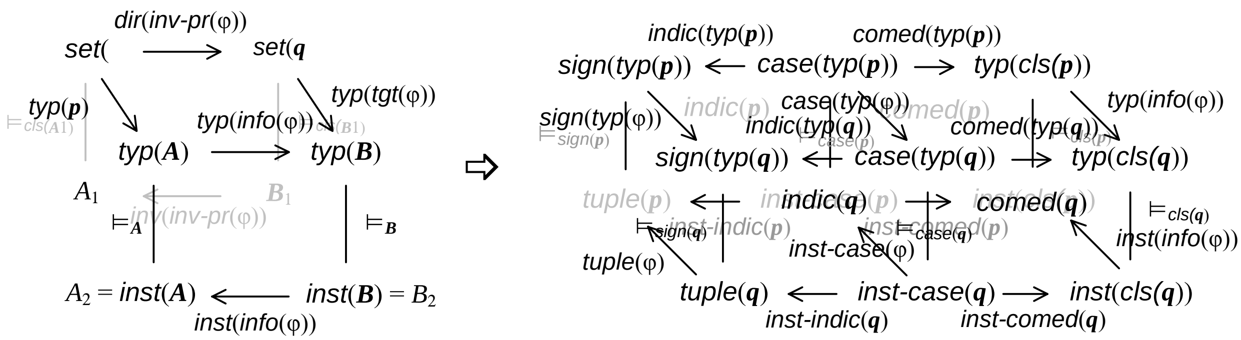


Figure 1: Semisquare to Spanmodel Morphism - abstract version



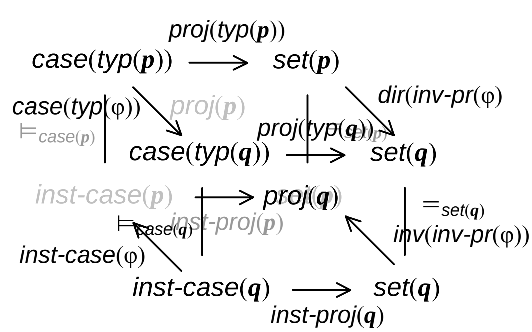


Figure 1: Semisquare to Spanmodel Morphism - detailed version

Associated with any classification semisquare $\varphi = \langle \mathit{inv\text{-}pr}(\varphi), \mathit{info}(\varphi) \rangle : \mathit{src}(\varphi) \rightarrow \mathit{tgt}(\varphi)$ is a spanmodel morphism (Figure 1)

$$\mathit{spnmod}(\varphi) = \langle 1^{st}_{\mathit{spnmod}(\varphi)}, 2^{nd}_{\mathit{spnmod}(\varphi)}, 3^{rd}_{\mathit{spnmod}(\varphi)} \rangle : \mathit{spnmod}(\mathit{src}(\varphi)) \rightarrow \mathit{spnmod}(\mathit{tgt}(\varphi)),$$

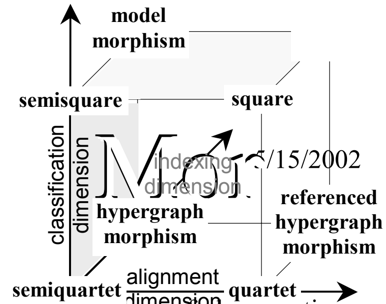
whose vertex (prehension) infomorphism is the case (role) infomorphism of φ,

- whose first classification square (actuality) is the comediator square $1^{st}_{\mathit{spnmod}(\varphi)} = \mathit{comed}(\varphi)$ with target infomorphism being the infomorphism of φ,
- whose second classification square (indexing) is the projection square $2^{nd}_{\mathit{spnmod}(\varphi)} = \mathit{proj}(\varphi)$ with target infomorphism being the invertible pair of φ, and
- whose third classification square (nexus) is the indication square $3^{rd}_{\mathit{spnmod}(\varphi)} = \mathit{indic}(\varphi)$ with target infomorphism being the signature infomorphism of φ.

```
(24) (SET.FTN$function spanmodel-morphism)
     (= (SET.FTN$source spanmodel-morphism) semiquartet)
     (= (SET.FTN$target spanmodel-morphism) smod.mor$spangraph-morphism)
     (forall (?h (semisquare ?h))
         (and (= (smod.mor$source (spanmodel-morphism ?h)) (cls.sdsgn$spanmodel (source ?h)))
              (= (smod.mor$target (spanmodel-morphism ?h)) (cls.sdsgn$spanmodel (target ?h)))
              (= (smod.mor$vertex (spanmodel-morphism ?h)) (case ?h))
              (= (smod.mor$first (spanmodel-morphism ?h)) (comediator ?h))
              (= (smod.mor$infomorphism1 (spanmodel-morphism ?h)) (infomorphism ?h))
              (= (smod.mor$second (spanmodel-morphism ?h)) (projection ?h))
              (= (smod.mor$infomorphism2 (spanmodel-morphism ?h))
                 (cls.info$infomorphism (invertible-pair ?h)))
              (= (smod.mor$third (spanmodel-morphism ?h)) (indication ?h))
              (= (smod.mor$infomorphism3 (spanmodel-morphism ?h)) (signature ?h))))
```

The spanmodel morphism associated with any semisquare is the image of the morphism function of the spanmodel functor applied to the semisquare:

$$\mathit{spnmod} : \blacktriangle\mathrm{Classification} \rightarrow \mathrm{SpanModel}.$$



Classification Squares

cls.sqr

Designations and infomorphisms are related through classification squares. The classes of classifications, infomorphisms, designations, and classification squares form the double category ■Classification.

$$\square\mathrm{Set}^{op} \xleftarrow{\;inst^{op}\;} \blacksquare\mathrm{Classification} \xrightarrow{\;typ\;} \square\mathrm{Set}$$

Figure 1: The type-instance designation span of double categories and functors

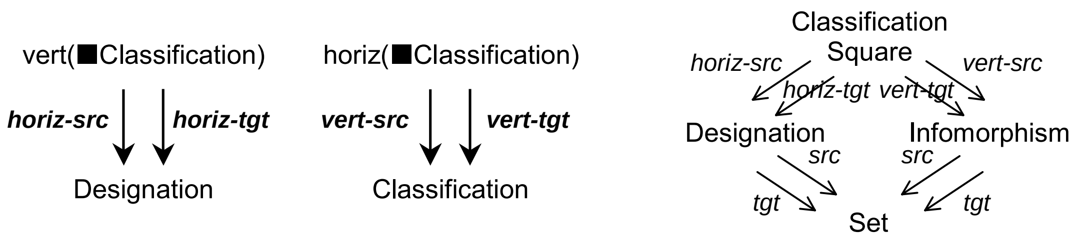


Diagram 1: The Categories and Functors implicit within the Double Category of Classification Squares

○ A *classification square* φ (Figures 1 & 2), horizontally from designation *horiz-src*(φ) to designation *horiz-tgt*(φ) and vertically from infomorphism *vert-src*(φ) to infomorphism *vert-tgt*(φ),

$\mathit{horiz\text{-}src}(\varphi) : \mathit{src}(\mathit{horiz\text{-}src}(\varphi)) \rightrightarrows \mathit{tgt}(\mathit{horiz\text{-}src}(\varphi))$,

$\mathit{horiz\text{-}tgt}(\varphi) : \mathit{src}(\mathit{horiz\text{-}tgt}(\varphi)) \rightrightarrows \mathit{tgt}(\mathit{horiz\text{-}tgt}(\varphi))$,

$\mathit{vert\text{-}src}(\varphi) : \mathit{src}(\mathit{vert\text{-}src}(\varphi)) \rightleftarrows \mathit{tgt}(\mathit{vert\text{-}src}(\varphi))$, and

$\mathit{vert\text{-}tgt}(\varphi) : \mathit{src}(\mathit{vert\text{-}tgt}(\varphi)) \rightleftarrows \mathit{tgt}(\mathit{vert\text{-}tgt}(\varphi))$.

These are required to be compatible in the sense that

$\mathit{src}(\mathit{horiz\text{-}src}(\varphi)) = \mathit{src}(\mathit{vert\text{-}src}(\varphi))$,

$\mathit{tgt}(\mathit{vert\text{-}tgt}(\varphi)) = \mathit{tgt}(\mathit{horiz\text{-}tgt}(\varphi))$,

$\mathit{tgt}(\mathit{vert\text{-}src}(\varphi)) = \mathit{src}(\mathit{horiz\text{-}tgt}(\varphi))$, and

$\mathit{tgt}(\mathit{horiz\text{-}src}(\varphi)) = \mathit{src}(\mathit{vert\text{-}tgt}(\varphi))$,

and to preserve instance reference

$\mathit{inst}(\mathit{vert\text{-}src}(\varphi)) \cdot \mathit{inst}(\mathit{horiz\text{-}src}(\varphi))$

$= \mathit{inst}(\mathit{horiz\text{-}tgt}(\varphi)) \cdot \mathit{inst}(\mathit{vert\text{-}tgt}(\varphi))$,

which means the instance component forms a quartet, and to preserve type reference

$\mathit{typ}(\mathit{vert\text{-}src}(\varphi)) \cdot \mathit{typ}(\mathit{horiz\text{-}tgt}(\varphi))$

$= \mathit{typ}(\mathit{horiz\text{-}src}(\varphi)) \cdot \mathit{typ}(\mathit{vert\text{-}tgt}(\varphi))$,

which means the type component forms a quartet.

$$\begin{tikzcd}
\boldsymbol{A}_1 \arrow[r, shift left, "\mathit{vert\text{-}src}(\varphi)"] \arrow[r, leftarrow, shift right] \arrow[d, shift right, "\mathit{horiz\text{-}src}(\varphi)"'] \arrow[d, shift left] \arrow[dr, phantom, "\varphi"] & \boldsymbol{B}_1 \arrow[d, shift left, "\mathit{horiz\text{-}tgt}(\varphi)"] \arrow[d, shift right] \\
\boldsymbol{A}_2 \arrow[r, shift left] \arrow[r, leftarrow, shift right, "\mathit{vert\text{-}tgt}(\varphi)"'] & \boldsymbol{B}_2
\end{tikzcd}$$

Figure 1: Classification square – abstract

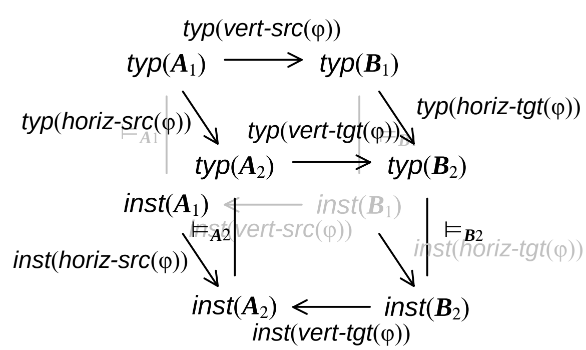


Figure 2: Classification Square – details

```
(1) (SET$class square)

(2) (SET.FTN$function horizontal-source)
    (= (SET.FTN$source horizontal-source) square)
    (= (SET.FTN$target horizontal-source) cls.dsgn$designation)

(3) (SET.FTN$function horizontal-target)
```

```
    (= (SET.FTN$source horizontal-target) square)
    (= (SET.FTN$target horizontal-target) cls.dsgn$designation)

(4) (SET.FTN$function vertical-source)
    (= (SET.FTN$source vertical-source) square)
    (= (SET.FTN$target vertical-source) cls.info$infomorphism)

(5) (SET.FTN$function vertical-target)
    (= (SET.FTN$source vertical-target) square)
    (= (SET.FTN$target vertical-target) cls.info$infomorphism)

(6) (SET.FTN$function instance)
    (= (SET.FTN$source instance) square)
    (= (SET.FTN$target instance) set.qtt$quartet)
    (= (SET.FTN$composition [instance set.qtt$horizontal-source])
       (SET.FTN$composition [horizontal-target cls.dsgn$instance]))
    (= (SET.FTN$composition [instance set.qtt$horizontal-target])
       (SET.FTN$composition [horizontal-source cls.dsgn$instance]))
    (= (SET.FTN$composition [instance set.qtt$vertical-source])
       (SET.FTN$composition [vertical-source cls.info$instance]))
    (= (SET.FTN$composition [instance set.qtt$vertical-target])
       (SET.FTN$composition [vertical-target cls.info$instance]))

(7) (SET.FTN$function type)
    (= (SET.FTN$source type) square)
    (= (SET.FTN$target type) set.qtt$quartet)
    (= (SET.FTN$composition [type set.qtt$horizontal-source])
       (SET.FTN$composition [horizontal-source cls.dsgn$type]))
    (= (SET.FTN$composition [type set.qtt$horizontal-target])
       (SET.FTN$composition [horizontal-target cls.dsgn$type]))
    (= (SET.FTN$composition [type set.qtt$vertical-source])
       (SET.FTN$composition [vertical-source cls.info$type]))
    (= (SET.FTN$composition [type set.qtt$vertical-target])
       (SET.FTN$composition [vertical-target cls.info$type]))
```

Classification squares form the square (cell) class of the double category ■Classification.

- Two classification squares 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 classification squares is defined in terms of the composition of their vertical source and vertical target infomorphisms.

```
(11) (SET.LIM.PBK$opspan horizontally-composable-opspan)
     (= (SET.LIM.PBK$class1 horizontally-composable-opspan) square)
     (= (SET.LIM.PBK$class2 horizontally-composable-opspan) square)
     (= (SET.LIM.PBK$opvertex horizontally-composable-opspan) cls.dsgn$designation)
     (= (SET.LIM.PBK$first horizontally-composable-opspan) horizontal-target)
     (= (SET.LIM.PBK$second horizontally-composable-opspan) horizontal-source)

(12) (REL$relation horizontally-composable)
     (= (REL$class1 horizontally-composable) square)
     (= (REL$class2 horizontally-composable) square)
     (= (REL$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) square)
     (forall (?h1 (square ?h1) ?h2 (square ?h2)
             (horizontally-composable ?h1 ?h2))
         (and (= (horizontal-source (horizontal-composition [?h1 ?h2]))
                 (horizontal-source ?h1))
              (= (horizontal-target (horizontal-composition [?h1 ?h2]))
                 (horizontal-target ?h2))
              (= (vertical-source (horizontal-composition [?h1 ?h2]))
                 (cls.info$composition [(vertical-source ?h1) (vertical-source ?h2)]))
```

```
                 (= (vertical-target (horizontal-composition [?h1 ?h2]))
                    (cls.info$composition [(vertical-target ?h1) (vertical-target ?h2)]))))
```

- Horizontal composition satisfies the usual *associative law*.

```
(forall (?h1 (square ?h1)
         ?h2 (square ?h2)
         ?h3 (square ?h3)
         (horizontally-composable ?h1 ?h2)
         (horizontally-composable ?h2 ?h3))
    (= (horizontal-composition [?h1 (horizontal-composition [?h2 ?h3])])
       (horizontal-composition [(horizontal-composition [?h1 ?h2]) ?h3])))
```

- For any designation, regarded as a vertical morphism, there is a *horizontal identity* square.

```
(14) (SET.FTN$function horizontal-identity)
     (= (SET.FTN$source horizontal-identity) cls.dsgn$designation)
     (= (SET.FTN$target horizontal-identity) square)
     (forall (?p (cls.dsgn$designation ?p))
         (and (= (horizontal-source (horizontal-identity ?p)) ?p)
              (= (horizontal-target (horizontal-identity ?p)) ?p)
              (= (vertical-source (horizontal-identity ?p))
                 (cls.info$identity (cls.dsgn$source ?p)))
              (= (vertical-target (horizontal-identity ?p))
                 (cls.info$identity (cls.dsgn$target ?p)))))
```

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

```
(forall (?h (square ?h))
    (and (= (horizontal-composition
                [(horizontal-identity (horizontal-source ?h)) ?h]) ?h)
         (= (horizontal-composition
                [?h (horizontal-identity (horizontal-target ?h))]) ?h)))
```

> The horizontal category horiz(■Classification) has designations as its objects, classification squares as its morphisms, horizontal composition as its composition function and horizontal identity as its identity function.

- Two classification squares 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 classification squares takes the vertical source of the first and the vertical target of the second.

```
(15) (SET.LIM.PBK$opspan vertically-composable-opspan)
     (= (class1 vertically-composable-opspan) square)
     (= (class2 vertically-composable-opspan) square)
     (= (opvertex vertically-composable-opspan) cls.info$infomorphism)
     (= (first vertically-composable-opspan) vertical-target)
     (= (second vertically-composable-opspan) vertical-source)

(16) (REL$relation vertically-composable)
     (= (REL$class1 vertically-composable) square)
     (= (REL$class2 vertically-composable) square)
     (= (REL$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) square)
     (forall (?h1 (square ?h1) ?h2 (square ?h2)
              (vertically-composable ?h1 ?h2))
         (and (= (horizontal-source (vertical-composition [?h1 ?h2]))
                 (cls.dsgn$composition
                     [(horizontal-source ?h1) (horizontal-source ?h2)]))
              (= (horizontal-target (vertical-composition [?h1 ?h2]))
                 (cls.dsgn$composition
                     [(horizontal-target ?f1) (horizontal-target ?f2)]))
              (= (vertical-source (vertical-composition [?h1 ?h2]))
                 (vertical-source ?h1))
```

```
            (= (vertical-target (vertical-composition [?h1 ?h2]))
               (vertical-target ?h2))))
```

- Vertical composition satisfies the usual *associative law*.

```
     (forall (?h1 (square ?h1)
              ?h2 (square ?h2)
              ?h3 (square ?h3)
              (vertically-composable ?h1 ?h2)
              (vertically-composable ?h2 ?h3))
         (= (vertical-composition [?h1 (vertical-composition [?h2 ?h3])])
            (vertical-composition [(vertical-composition [?h1 ?h2]) ?h3])))
```

- For any infomorphism, regarded as a horizontal morphism, there is a *vertical identity* square.

```
(18) (SET.FTN$function vertical-identity)
     (= (SET.FTN$source vertical-identity) cls.info$infomorphism)
     (= (SET.FTN$target vertical-identity) square)
     (forall (?f (cls.info$infomorphism ?f))
         (and (= (vertical-source (vertical-identity ?f)) ?f)
              (= (vertical-target (vertical-identity ?f)) ?f)
              (= (horizontal-source (vertical-identity ?f))
                 (cls.dsgn$identity (cls.info$source ?f)))
              (= (horizontal-target (vertical-identity ?f))
                 (cls.dsgn$identity (cls.info$target ?f)))))
```

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

```
     (forall (?h (square ?h))
         (and (= (vertical-composition
                     [(vertical-identity (vertical-source ?h)) ?h]) ?h)
              (= (vertical-composition
                     [?h (vertical-identity (vertical-target ?h))]) ?h)))
```

> The vertical category vert(■Classification) has infomorphisms as its objects, classification squares as its morphisms, vertical composition as its composition function and vertical identity as its identity function.