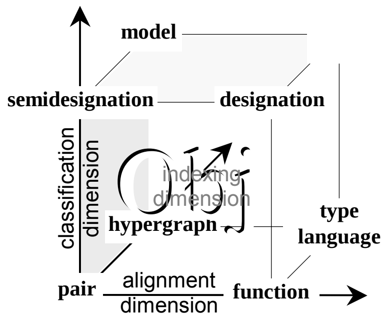


The Namespace of Type Languages

It has been the opinion of many that the best way to handle multivalent relations is with hypergraphs. According to Pat Hayes, ontology maps very well to the concept of hypergraphs. “Since an axiom can use more than two concepts or terms, in general the relevant structure [for an ontology] is a hypergraph, which is similar to a graph but allows ‘edges’ which link more than two nodes. The theory of hypergraphs is somewhat more complicated, and less thoroughly investigated, than that of graphs.” According to John Sowa, “two kinds of generalized graphs may be used to represent n-adic relations: hypergraphs and bipartite graphs.” The IFF Model Theory Ontology takes that advice sincerely to heart.

Table 1 lists the terminology of the namespace of type languages.

Table 1: Terminology for Type Languages

	Class	Function	Other
lang	language	reference signature relation-arity reference-assign reference-arity variable entity relation	
		model model-inclusion model-expression recursive substitution domain codomain substitution-opspan substitutable	
		arity case injection indication projection arity-morphism case-signature comediator factorization spangraph	
lang .mor	language-morphism simple special	source target reference signature relation-arity reference-assign reference-arity variable entity relation composition identity substitutable eta	composable-opspan composable
		pair instance-reference instance-arity instance-signature model-reference model-arity model-signature model link recursive kernel epimorphism monomorphism	
		arity-morphism case indication projection comediator spangraph-morphism epsilon	
lang .expr	kind connective quantifier	tuple-set set injection indication arity-tuple arity arity-fiber sentence signature expression	substitution-opspan substitutable primitive negative substitutive existential universal
		atom negation conjunction disjunction implication equivalence existential-quantification universal-quantification substitution	
lang .expr .mor		tuple-set-morphism function expression eta free collapse-tuple collapse mu	
lang .interp	interpretation special	source target morphism extension composition identity	composable-opspan composable
		model link	

Type Languages

`lang`

This section discusses first order type languages. This section axiomatizes the namespace for type languages and their morphisms. These form a category called Language.

The notion of a type language is like an aligned notion of hypergraph. *Entity* types are synonymous with (hypergraph) nodes; *relation* types are synonymous with (hyper)edges. Moreover, following tradition, the names in hypergraphs are called (logical) *variables*, and there is an explicit referencing (sorting) function from variables (names) to entity types (nodes). First, we give a concise mathematical definition (Figure 1), and then we discuss and formalize the various parts of this definition.

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}) \arrow[rr, "\#_L"] \arrow[dr, "\partial_L"'] & & \wp\mathit{var}(\boldsymbol{L}) & \mathit{var}(\boldsymbol{L}) \arrow[d, "*_L"] \\
& \mathit{sign}(\tau_L) \arrow[ur, dashed, "\mathit{refer\text{-}arity}(\boldsymbol{L})"] & & \mathit{ent}(\boldsymbol{L})
\end{tikzcd}$$

Figure 1: Type Language

- A first order *type language* $\boldsymbol{L} = \langle \mathit{refer}(\boldsymbol{L}), \mathit{sign}(\boldsymbol{L})\rangle$ (Figure 1) consists of
  - a *reference* function $*_L = \mathit{refer}(\boldsymbol{L})$
  - a *signature* function $\partial_L = \mathit{sign}(\boldsymbol{L})$

  that satisfy the pullback constraint

  $\mathit{tgt}(\mathit{sign}(\boldsymbol{L})) = \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$.

```
(1) (SET$class language)

(2) (SET.FTN$function reference)
    (= (SET.FTN$source reference) language)
    (= (SET.FTN$target reference) set.ftn$function)

(3) (SET.FTN$function signature)
    (= (SET.FTN$source signature) language)
    (= (SET.FTN$target signature) set.ftn$function)

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

- For convenience of theoretical presentation, we introduce additional type language terminology for the composition between the reference function and the signature assign and arity functions.
  - *reference-assign* function $\mathit{refer\text{-}assign}(\boldsymbol{L}) = \mathit{sign\text{-}assign}(*_L) : \wp\mathit{var}(\boldsymbol{L}) \rightarrow \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$,
  - *reference-arity* function $\mathit{refer\text{-}arity}(\boldsymbol{L}) = \mathit{sign\text{-}arity}(*_L) : \mathit{sign}(\mathit{refer}(\boldsymbol{L})) \rightarrow \wp\mathit{var}(\boldsymbol{L})$.

  As we know, these designations are inverse to each other:

  $\mathit{refer\text{-}assign}(\boldsymbol{L}) \bullet \mathit{refer\text{-}arity}(\boldsymbol{L}) = \mathit{id}_{\wp\mathit{var}(\boldsymbol{L})}$, and

  $\mathit{refer\text{-}arity}(\boldsymbol{L}) \bullet \mathit{refer\text{-}assign}(\boldsymbol{L}) = \mathit{id}_{\mathit{sign}(\mathit{refer}(\boldsymbol{L}))}$.

```
(5) (SET.FTN$function reference-assign)
    (= (SET.FTN$source reference-assign) language)
    (= (SET.FTN$target reference-assign) set.ftn$function)
    (= reference-assign (SET.FTN$composition [reference set.ftn$signature-assign]))

(6) (SET.FTN$function reference-arity)
    (= (SET.FTN$source reference-arity) language)
    (= (SET.FTN$target reference-arity) set.ftn$function)
    (= reference-arity (SET.FTN$composition [reference set.ftn$signature-arity]))
```

- The set function composition of the signature function with the reference arity function defines
  - an *relation-arity* function $\#_L = \mathit{rel\text{-}arity}(\boldsymbol{L}) = \mathit{sign}(\boldsymbol{L}) \cdot \mathit{refer\text{-}arity}(\boldsymbol{L})$

  which can be used in place of the signature function. The pullback constraint takes the form

  $\mathit{tgt}(\mathit{arity}(\boldsymbol{L})) = \wp\mathit{var}(\boldsymbol{L}) = \wp\mathit{src}(\mathit{refer}(\boldsymbol{L}))$.

  Since reference arity and reference assign are inverse functions, we can equivalently define the signature function in terms of the arity function.

```
(7) (SET.FTN$function relation-arity)
    (= (SET.FTN$source relation-arity) language)
```

```
        (= (SET.FTN$target relation-arity) set.ftn$function)
        (forall (?l (language ?l))
            (= (relation-arity ?l)
               (set.ftn$composition [(signature ?l) (reference-arity ?l)])))

    (8) (forall (?l (language ?l))
            (= (signature ?l)
               (set.ftn$composition [(relation-arity ?l) (reference-assign ?l)])))
```

○ For convenience of practical reference, we introduce additional type language terminology for the source and target components of these functions. The source of the signature function is called the set of *relation* types of $\boldsymbol{L}$ and denoted $\mathit{rel}(\boldsymbol{L})$. The target of the reference function is called the set of *entity* types of $\boldsymbol{L}$ and denoted $\mathit{ent}(\boldsymbol{L})$. The source of the reference function is called the set of *variables* of $\boldsymbol{L}$ and denoted $\mathit{var}(\boldsymbol{L})$. In summary, we provide terminology for the following sets:

- the set of *entity* types $\mathit{ent}(\boldsymbol{L}) = \mathit{tgt}(\mathit{refer}(\boldsymbol{L}))$,
- the set of *relation* types $\mathit{rel}(\boldsymbol{L}) = \mathit{src}(\mathit{sign}(\boldsymbol{L}))$, and
- the set of *variables* $\mathit{var}(\boldsymbol{L}) = \mathit{src}(\mathit{refer}(\boldsymbol{L}))$.

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

- *signature* function $\partial_{\boldsymbol{L}} = \mathit{sign}(\boldsymbol{L}) : \mathit{rel}(\boldsymbol{L}) \rightarrow \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$,
- *arity* function $\#_{\boldsymbol{L}} = \mathit{rel\text{-}arity}(\boldsymbol{L}) : \mathit{rel}(\boldsymbol{L}) \rightarrow \wp\mathit{var}(\boldsymbol{L})$, and
- *reference* function $*_{\boldsymbol{L}} = \mathit{refer}(\boldsymbol{L}) : \mathit{var}(\boldsymbol{L}) \rightarrow \mathit{ent}(\boldsymbol{L})$.

Therefore, a type language can be displayed as in Figure 1.

Variables refer to entity types – the reference (or sort) function maps a variable to its referent entity type. Logical variables reference participants in a relation, but don't really vary. We identify logical variables with the *dynamic names* in ACE (Attempto Controlled English) and the *coreference labels* in conceptual graphs, since they name things (nodes, entities, etc.). Entity types correspond to sorts in a many-sorted language. A reference function $\mathit{refer(L)} : \mathit{var}(\boldsymbol{L}) \rightarrow \mathit{ent}(\boldsymbol{L})$ defines sorted variables. For each entity type $\alpha \in \mathit{ent}(\boldsymbol{L})$, the fiber $\mathit{refer}(L)^{-1}(\alpha)$ represents the set of variables of type (sort) α .

We assume that each type language has an adequate, possibly denumerable, set of variables $\mathit{var}(\boldsymbol{L})$. This generalizes the usual case where sequences are used – the advantage for this generality is elimination of the dependency on sequences and natural numbers. One way to return to sequence-like structures is to define a set of indexed variables $\mathit{ind\text{-}var}(\boldsymbol{L}) = \mathit{index}\times\mathit{ent}(\boldsymbol{L})$ for some index set such as $\mathit{index} = \mathit{Natno}$, and to let the index reference function be defined as the projection $\mathit{refer}(\boldsymbol{L}) = \pi : \mathit{index}\times\mathit{ent}(\boldsymbol{L}) \rightarrow \mathit{ent}(\boldsymbol{L})$. Alternatively, if preservation of the original reference function is important, one can use the composed reference:

$$\pi \cdot *_{\boldsymbol{L}} : \mathit{ind\text{-}var}(\boldsymbol{L}) = \mathit{index}\times\mathit{var}(\boldsymbol{L}) \rightarrow \mathit{var}(\boldsymbol{L}) \rightarrow \mathit{ent}(\boldsymbol{L}).$$

The relation types in a type language are many-sorted, which means that the component entity types in signatures are named. For any relation type $\rho \in \mathit{rel}(\boldsymbol{L})$ the *signature* of ρ is a function $\partial_{\boldsymbol{L}}(\rho) : \mathit{arity}(\boldsymbol{L})(\rho) \rightarrow \mathit{ent}(\boldsymbol{L})$ that is a restriction of the reference function $\partial_{\boldsymbol{L}}(\rho)(x) = \mathit{refer}(\boldsymbol{L})(x)$ for all $x \in \mathit{arity}(\boldsymbol{L})(\rho)$. The set of variables $\mathit{arity}(\boldsymbol{L})(\rho)$ that reference the entity types in the signature of a relation type $\rho \in \mathit{rel}(\boldsymbol{L})$ is called its *arity*. The arity of relation types can overlap. The signature $\partial_{\boldsymbol{L}}(\rho)$ is a tuple of typed (sorted) variables with indexing set $\mathit{arity}(\boldsymbol{L})(\rho) \subseteq \mathit{var}(\boldsymbol{L})$. The number of components $|\mathit{arity}(\boldsymbol{L})(\rho)|$ in the arity of a relation type is called its *valence*. The signature and arity of a relation type are equivalent – they satisfy the property $\partial_{\boldsymbol{L}}(\rho) = \mathit{incl}_{\#\boldsymbol{L}(\rho),\, \mathit{var}(\boldsymbol{L})} \cdot *_{\boldsymbol{L}}$ for any relation type $\rho \in \mathit{rel}(\boldsymbol{L})$.

An external arity form for a relation type is $(\rho, x_1, x_2, \ldots x_n)$, where $\mathit{arity}(\boldsymbol{L})(\rho) = \{x_1, x_2, \ldots x_n\}$. An external signature form for a relation type is $(\rho, x_1 : \alpha_1, x_2 : \alpha_2, \ldots x_n : \alpha_n)$, where $\partial_{\boldsymbol{L}}(\rho) = (x_1 : \alpha_1, x_2 : \alpha_2, \ldots x_n : \alpha_n)$.

```
    (9) (SET.FTN$function entity)
        (= (SET.FTN$source entity) language)
        (= (SET.FTN$target entity) set$set)
        (= entity (SET.FTN$composition [reference set.ftn$target]))
```

```
(10) (SET.FTN$function relation)
     (= (SET.FTN$source relation) language)
     (= (SET.FTN$target relation) set$set)
     (= relation (SET.FTN$composition [signature set.ftn$source]))

(11) (SET.FTN$function variable)
     (= (SET.FTN$source variable) language)
     (= (SET.FTN$target variable) set$set)
     (= variable (SET.FTN$composition [reference set.ftn$source]))
```

○ Here is the specification for the fiber class of all models $\mathit{mod}(\boldsymbol{L}) \subseteq \mathsf{model}$ whose type language is $\boldsymbol{L}$.

```
(12) (SET.FTN$function model)
     (= (SET.FTN$source model) language)
     (= (SET.FTN$target model) (SET$power mod$model))
     (= model (SET.FTN$fiber mod$type))

(13) (KIF$function model-inclusion)
     (= (KIF$source model-inclusion) language)
     (= (KIF$target model-inclusion) (SET$exponent [power mod$model))
     (forall (?l (language ?l))
         (and (SET.FTN$source (model-inclusion ?l) (model ?l))
              (SET.FTN$target (model-inclusion ?l) mod$model)
              (= (model-inclusion ?l) (SET.FTN$inclusion [(model ?l) mod$model]))))
```

○ There is also a related *model expression* class function

$$\mathit{mod\text{-}expr}(\boldsymbol{L}) : \mathit{mod}(\boldsymbol{L}) \rightarrow \mathit{mod}(\mathit{expr}(\boldsymbol{L})),$$

which restricts the model expression class function

$$\mathit{expr} : \mathsf{model} \rightarrow \mathsf{model}$$

to fibers. Model expression commutes with inclusion.

```
(14) (KIF$function model-expression)
     (= (KIF$source model-expression) language)
     (= (KIF$target model-expression) SET.FTN$function)
     (forall (?l (language ?l))
         (and (SET.FTN$source (model-expression ?l) (model ?l))
              (SET.FTN$target (model-expression ?l) (model (expression ?l)))
              (forall (?a ((model ?l) ?a))
                  (= ((model-expression ?l) ?a)
                     (mod$expression ?a)))))
```

○ An arbitrary model $\boldsymbol{A} \in \mathit{expr}(\boldsymbol{L})$ is not necessary recursively defined in terms of the relation types in $\boldsymbol{L}$ in the following sense: Intuitively, a model $\boldsymbol{A} \in \mathit{expr}(\boldsymbol{L})$ is recursively defined when the expression extent function is recursively defined or equivalently the expression classification is recursively defined. Abstractly, a model in is recursively defined $\boldsymbol{A} \in \mathit{recur}(\boldsymbol{L})$ when it is in the image of $\mathit{mod\text{-}expr}(\boldsymbol{L})$: $\mathit{recur}(\boldsymbol{L}) = \mathit{image}(\mathit{mod\text{-}expr}(\boldsymbol{L}))$.

```
(15) (KIF$function recursive)
     (= (KIF$source recursive) language)
     (= (KIF$target recursive) SET$class)
     (forall (?l (language ?l))
         (and (SET$subclass (recursive ?l) (model (expression ?l)))
              (= (recursive ?l) (SET.FTN$image (model-expression ?l)))))
```

○ For any type language $\boldsymbol{L}$ a *substitution* $h : \mathit{src}(h) \rightarrow \mathit{tgt}(h)$ of $\boldsymbol{L}$ is a substitution of the reference function. This means that it is a surjective function between variable sets $\mathit{src}(h), \mathit{tgt}(h) \subseteq \mathit{var}(\boldsymbol{L})$ that respects the reference function. Let $\mathit{subtn}(\boldsymbol{L})$ denote the class of all substitutions of $\boldsymbol{L}$. For convenience of reference, we rename the source and target of substitutions.

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}) \arrow[dr, "\mathit{arity}(\boldsymbol{L})"'] & & \mathit{subtn}(\boldsymbol{L}) \arrow[dl, "\mathit{dom}(\boldsymbol{L})"] \\
& \wp\mathit{var}(\boldsymbol{L}) &
\end{tikzcd}$$

Figure 2: Substitution Opspan

```
(16) (SET.FTN$function substitution)
     (= (SET.FTN$source substitution) language)
     (= (SET.FTN$target substitution) set$set)
     (= substitution (SET.FTN$composition [reference set.ftn$substitution]))
```

```
(17) (SET.FTN$function domain)
     (= (SET.FTN$source domain) language)
     (= (SET.FTN$target domain) set.ftn$function)
     (= (SET.FTN$composition [domain set.ftn$source]) substitution)
     (= (SET.FTN$composition [domain set.ftn$target])
         (SET.FTN$composition [variable set$power]))
     (= domain (SET.FTN$composition [reference set.ftn$domain]))

(18) (SET.FTN$function codomain)
     (= (SET.FTN$source codomain) language)
     (= (SET.FTN$target codomain) set.ftn$function)
     (= (SET.FTN$composition [codomain set.ftn$source]) substitution)
     (= (SET.FTN$composition [codomain set.ftn$target])
        (SET.FTN$composition [variable set$power]))
     (= codomain (SET.FTN$composition [reference set.ftn$codomain]))
```

○ A pair $\langle\rho, h\rangle$ is *substitutable* (Figure 2) when $\rho \in \mathit{rel}(\boldsymbol{L})$ is a relation type and $h \in \mathit{subtn}(\boldsymbol{L})$ is a substitution whose domain is the arity of the relation type: $\mathit{arity}(\boldsymbol{L})(\rho) = \mathit{dom}(\boldsymbol{L})(h)$. Let $\mathit{subst}(\boldsymbol{L})$ denote the set of all substitutable pairs.

```
(19) (SET.FTN$function substitution-opspan)
     (= (SET.FTN$source substitution-opspan) language)
     (= (SET.FTN$target substitution-opspan) set.lim.pbk$opspan)
     (forall (?l (language ?l))
         (and (= (set.lim.pbk$set1 (substitution-opspan ?l) (relation ?l))
              (= (set.lim.pbk$set2 (substitution-opspan ?l) (substitution ?l))
              (= (set.lim.pbk$opvertex (substitution-opspan ?l)
                 (set$power (variable ?l)))
              (= (set.lim.pbk$opfirst (substitution-opspan ?l) (arity ?l))
              (= (set.lim.pbk$opsecond (substitution-opspan ?l) (domain ?l))))

(20) (SET.FTN$function substitutable)
     (= (SET.FTN$source substitutable) language)
     (= (SET.FTN$target substitutable) rel$relation)
     (forall (?l (language ?l))
         (= (substitutable ?l) (set.lim.pbk$relation (substitution-opspan ?l))))
```

Case and Spangraphs

- The terminology of case relations or thematic roles is introduced here for two reasons: to define the passage from type languages (referenced hypergraphs) to spangraphs; and for use in defining expressions.

  Any type language $\boldsymbol{L} = \langle \mathit{refer}(\boldsymbol{L}), \mathit{sign}(\boldsymbol{L})\rangle$ has a coproduct *arity*

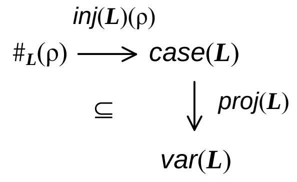
  $$\#_{\boldsymbol{L}} = \mathit{arity}(\boldsymbol{L}) : \mathit{rel}(\boldsymbol{L}) \rightarrow \wp\mathit{var}(\boldsymbol{L}).$$

```
(21) (SET.FTN$function arity)
     (= (SET.FTN$source arity) language)
     (= (SET.FTN$target arity) set.col.art$arity)
     (= (SET.FTN$composition [arity set.col.art$index]) relation)
     (= (SET.FTN$composition [arity set.col.art$base]) variable)
     (= (SET.FTN$composition [arity set.col.art$function]) relation-arity)
```

- Any type language $\boldsymbol{L}$ has a set of *cases* or *roles*

  $$\mathit{case}(\boldsymbol{L}) = \textstyle\sum\mathit{arity}(\boldsymbol{L}) = \sum_{\rho\in\mathit{rel}(\boldsymbol{L})} \mathit{arity}(\boldsymbol{L})(\rho) = \{(\rho, x) \mid \rho \in \mathit{rel}(\boldsymbol{L}), x \in \mathit{arity}(\boldsymbol{L})(\rho)\}$$

  that is the coproduct of its arity.



$$\begin{tikzcd} \#_{\boldsymbol{L}}(\rho) \arrow[r, "\mathit{inj}(\boldsymbol{L})(\rho)"] \arrow[dr, phantom, "\subseteq"] & \mathit{case}(\boldsymbol{L}) \arrow[d, "\mathit{proj}(\boldsymbol{L})"] \\ & \mathit{var}(\boldsymbol{L}) \end{tikzcd}$$

Diagram 1: Coproduct

  For any type language $\boldsymbol{L}$ and any relation type $\rho \in \mathit{rel}(\boldsymbol{L})$ there is a case *injection* function:

  $$\mathit{inj}(\boldsymbol{L})(\rho) : \#_{\boldsymbol{L}}(\rho) = \mathit{arity}_{\boldsymbol{L}}(\rho) \rightarrow \mathit{case}(\boldsymbol{L})$$

  defined by $\mathit{inj}(\boldsymbol{L})(\rho)(x) = (\rho, x)$ for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$ and all variables $x \in \mathit{arity}(\boldsymbol{L})(\rho)$. Obviously, the injections are injective. They commute (Diagram 3) with projection and inclusion.

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

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

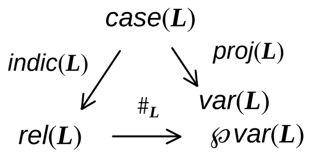
- Any type language $\boldsymbol{L}$ defines case *indication* and *projection* functions (Figure 3) based on its arity:

  $$\mathit{indic}(\boldsymbol{L}) : \mathit{case}(\boldsymbol{L}) \rightarrow \mathit{rel}(\boldsymbol{L}),$$
  $$\mathit{proj}(\boldsymbol{L}) : \mathit{case}(\boldsymbol{L}) \rightarrow \mathit{var}(\boldsymbol{L}).$$

  These are defined by

  $$\mathit{indic}(\boldsymbol{L})((\rho, x)) = \rho \text{ and } \mathit{proj}(\boldsymbol{L})((\rho, x)) = x$$

  for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$ and all variables $x \in \mathit{arity}(\boldsymbol{L})(\rho)$.



$$\begin{tikzcd} & \mathit{case}(\boldsymbol{L}) \arrow[dl, "\mathit{indic}(\boldsymbol{L})"'] \arrow[dr, "\mathit{proj}(\boldsymbol{L})"] & \\ \mathit{rel}(\boldsymbol{L}) \arrow[rr, "\#_{\boldsymbol{L}}"] & & \mathit{var}(\boldsymbol{L}) \quad \wp\mathit{var}(\boldsymbol{L}) \end{tikzcd}$$

Figure 3: Indication and projection

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

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

○ Any type language $\boldsymbol{L}$ defines a morphism (Figure 4) from its coproduct arity to the coproduct arity of its reference function:

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

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

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}) \arrow[r, "\mathit{arity}(\boldsymbol{L})"] \arrow[d, "\mathit{sign}(\boldsymbol{L})"'] & \wp\mathit{var}(\boldsymbol{L}) \arrow[d, "\mathit{id}"] \\
\mathit{tuple}(\mathit{refer}(\boldsymbol{L})) \arrow[r, "\mathit{arity}(\mathit{refer}(\boldsymbol{L}))"'] & \wp\mathit{var}(\boldsymbol{L})
\end{tikzcd}$$

Figure 4: Arity morphism

```
(26) (SET.FTN$function arity-morphism)
     (= (SET.FTN$source arity-morphism) language)
     (= (SET.FTN$target arity-morphism) set.col.art.mor$arity-morphism)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$source]) arity)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$target])
        (SET.FTN$composition [reference set.ftn$arity]))
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$index]) signature)
     (= (SET.FTN$composition [arity-morphism set.col.art.mor$base])
        (SET.FTN$composition [(SET.FTN$composition [variable set$power]) set.ftn$identity]))
```

○ Any type language $\boldsymbol{L}$ has a function (Figure 5)

$$\mathit{case\text{-}sign}(\boldsymbol{L}) : \mathit{case}(\boldsymbol{L}) \rightarrow \mathit{case}(\mathit{refer}(\boldsymbol{L})),$$

that satisfies the following constraints (Figure 2):

$$\mathit{case\text{-}sign}(\boldsymbol{L}) \cdot \mathit{index}(\mathit{refer}(\boldsymbol{L})) = \mathit{index}(\boldsymbol{L}) \cdot \mathit{sign}(\boldsymbol{L}),$$

$$\mathit{case\text{-}sign}(\boldsymbol{L}) \cdot \mathit{proj}(\mathit{refer}(\boldsymbol{L})) = \mathit{proj}(\boldsymbol{L}).$$

Pointwise, this is defined by

$$\mathit{case\text{-}sign}(\boldsymbol{L})((\rho, x)) = (\partial_{\boldsymbol{L}}(\rho), x)$$

for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$ and all variables $x \in \mathit{arity}(\boldsymbol{L})(\rho)$. Abstractly, this is defined to be the coproduct of the arity morphism of $\boldsymbol{L}$.

It is the mediating function for the signature function $\partial_{\boldsymbol{L}} = \mathit{sign}(\boldsymbol{L}) : \mathit{rel}(\boldsymbol{L}) \rightarrow \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$, regarded as a coproduct cocone over the reference function coproduct arity.

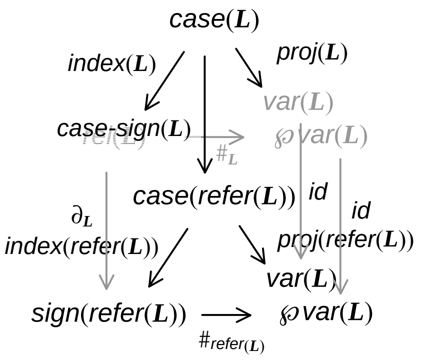


Figure 5: Case-tuple function

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

○ Any type language $\boldsymbol{L}$ defines a *comediator* function:

$$\tilde{\tau}_{\boldsymbol{L}} = \mathit{comed}(\boldsymbol{L}) : \mathit{case}(\boldsymbol{L}) \rightarrow \mathit{ent}(\boldsymbol{L}).$$

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

$$\mathit{comed}(\boldsymbol{L})((\rho, x)) = \partial_{\boldsymbol{L}}(\rho)(x)$$

for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$ and all variables $x \in \mathit{arity}(\boldsymbol{L})(\rho)$. Abstractly, it is the comediator – hence the name – of the coproduct cotuple (cocone) consisting of the collection of relation type signatures regarded as functions,

$$\{\partial_{\boldsymbol{L}}(\rho) : \mathit{arity}(\boldsymbol{L})(\rho) \rightarrow \mathit{ent}(\boldsymbol{L}) \mid \rho \in \mathit{rel}(\boldsymbol{L})\}.$$

However, a simpler definition is at hand: the language comediator can be defined (Diagram 2) as the composition of the case-signature function and the reference function comediator

$$\mathit{comed}(\boldsymbol{L}) = \mathit{case\text{-}sign}(\boldsymbol{L}) \cdot \mathit{comed}(\mathit{refer}(\boldsymbol{L})).$$

$$\begin{tikzcd}
\mathit{case}(\boldsymbol{L}) \arrow[dr, "\mathit{comed}(\boldsymbol{L})"] \arrow[dd, "\mathit{case\text{-}sign}(\boldsymbol{L})"'] & \\
& \mathit{ent}(\boldsymbol{L}) \\
\mathit{case}(\mathit{refer}(\boldsymbol{L})) \arrow[ur, "\mathit{comed}(\mathit{type}(\boldsymbol{L}))"'] &
\end{tikzcd}$$

Diagram 2: Comediator

The comediator function commutes with the injection and signature functions of any relation type – it is a generalized reference function for these signatures. If the indexed variables in $case(\boldsymbol{L})$ are viewed as roles (prehensions), then the comediator is a reference function from roles to objects (actualities).

```
(28) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) language)
     (= (SET.FTN$target comediator) set.ftn$function)
     (forall (?l (language ?l))
         (and (= (set.ftn$source (comediator ?l)) (case ?l))
              (= (set.ftn$target (comediator ?l)) (entity ?l))
              (= (comediator ?l)
                 (set.ftn$composition
                     [(case-signature ?l) (set.ftn$comediator (reference ?l))]))))
```

○ The language comediator function can also be factored as the composition of the projection function and the reference function

$$comed(\boldsymbol{L}) = proj(\boldsymbol{L}) \cdot refer(\boldsymbol{L}).$$

This fact can be represented (Figure 6) as a *factorization* quartet $fact(\boldsymbol{L})$,

horizontally from function $comed(\boldsymbol{L})$ to function $refer(\boldsymbol{L})$

$$fact(\boldsymbol{L}) = \langle proj(\boldsymbol{L}), id_{ent(\boldsymbol{L})}\rangle : comed(\boldsymbol{L}) \rightarrow refer(\boldsymbol{L}).$$

$$\begin{tikzcd}
case(\boldsymbol{L}) \arrow[r, "proj(\boldsymbol{L})"] \arrow[d, "comed(\boldsymbol{L})"'] \arrow[dr, phantom, "fact(\boldsymbol{L})"] & var(\boldsymbol{L}) \arrow[d, "refer(\boldsymbol{L})"] \\
ent(\boldsymbol{L}) \arrow[r, "id"'] & ent(\boldsymbol{L})
\end{tikzcd}$$

Figure 6: Factorization

```
(29) (SET.FTN$function factorization)
     (= (SET.FTN$source factorization) language)
     (= (SET.FTN$target factorization) set.qtt$quartet)
     (forall (?l (language ?l))
         (and (= (set.qtt$horizontal-source (factorization ?l)) (comediator ?l))
              (= (set.qtt$horizontal-target (factorization ?l)) (reference ?l))
              (= (set.qtt$vertical-source (factorization ?l)) (projection ?l))
              (= (set.qtt$vertical-target (factorization ?l))
                 (set.ftn$identity (entity ?l)))))
```

$$\begin{tikzcd}
rel(\boldsymbol{L}) \arrow[r, "\#_{\boldsymbol{L}}"] \arrow[dr, "\partial_{\boldsymbol{L}}"'] & \wp var(\boldsymbol{L}) \\
& tuple(\tau_{\boldsymbol{L}}) \arrow[u, "arity(*_{\boldsymbol{L}})"'] \\
& var(\boldsymbol{L}) \arrow[d, "*_{\boldsymbol{L}}"'] \\
& ent(\boldsymbol{L})
\end{tikzcd}
\qquad \Rightarrow \qquad
\begin{tikzcd}
\text{nexus} & \text{prehensions} & \text{actualities} \\
rel(\boldsymbol{L}) & case(\boldsymbol{L}) \arrow[l, "indic(\boldsymbol{L})"'] \arrow[r, "comed(\boldsymbol{L})"] \arrow[d, "proj(\boldsymbol{L})"'] & ent(\boldsymbol{L}) \\
& var(\boldsymbol{L}) \arrow[ur, "refer(\boldsymbol{L})"'] & \\
& \text{names} &
\end{tikzcd}$$

Diagram 3: From type language to spangraph

○ Associated with any type language $\boldsymbol{L}$ is a spangraph $spngph(\boldsymbol{L}) = \langle 1^{st}_{spngph(\boldsymbol{L})}, 2^{nd}_{spngph(\boldsymbol{L})}, 3^{rd}_{spngph(\boldsymbol{L})}\rangle$ (Diagram 3), whose vertex set is the case set of $\boldsymbol{L}$, and whose first (actualities), second (prehensions) and third (nexus) functions are

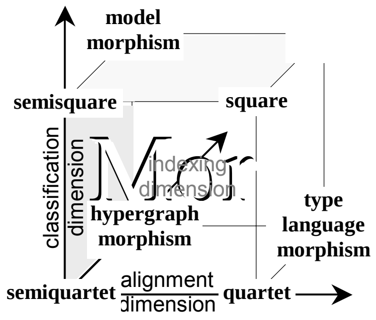
$$1^{st}_{spngph(\boldsymbol{L})} = comed(\boldsymbol{L}) : case(\boldsymbol{L}) \rightarrow ent(\boldsymbol{L}),$$

$$2^{nd}_{spngph(\boldsymbol{L})} = proj(\boldsymbol{L}) : case(\boldsymbol{L}) \rightarrow var(\boldsymbol{L}), \text{ and}$$

$$3^{st}_{spngph(\boldsymbol{L})} = indic(\boldsymbol{L}) : case(\boldsymbol{L}) \rightarrow rel(\boldsymbol{L}).$$

```
(30) (SET.FTN$function spangraph)
     (= (SET.FTN$source spangraph) language)
     (= (SET.FTN$target spangraph) sgph$graph)
     (forall (?l (language ?l))
         (and (= (sgph$vertex (spangraph ?l)) (case ?l))
              (= (sgph$first (spangraph ?l)) (comediator ?l))
```

```
                  (= (sgph$set1 (spangraph ?l)) (entity ?l))
                  (= (sgph$second (spangraph ?l)) (projection ?l))
                  (= (sgph$set2 (spangraph ?l)) (variable ?l))
                  (= (sgph$third (spangraph ?l)) (indication ?l))
                  (= (sgph$set3 (spangraph ?l)) (relation ?l))))
```



Type Language Morphisms

lang.mor

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

$$\begin{tikzcd}
var(\boldsymbol{L}_1) \arrow[r, "var(\boldsymbol{f})"] \arrow[d, "refer(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "refer(\boldsymbol{f})"] & var(\boldsymbol{L}_2) \arrow[d, "refer(\boldsymbol{L}_2)"] \\
ent(\boldsymbol{L}_1) \arrow[r, "ent(\boldsymbol{f})"'] & ent(\boldsymbol{L}_2)
\end{tikzcd}
\qquad
\begin{tikzcd}
rel(\boldsymbol{L}_1) \arrow[r, "rel(\boldsymbol{f})"] \arrow[d, "sign(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "sign(\boldsymbol{f})"] & rel(\boldsymbol{L}_2) \arrow[d, "sign(\boldsymbol{L}_2)"] \\
sign(refer(\boldsymbol{L}_1)) \arrow[r, "sign(refer(\boldsymbol{f}))"'] & sign(refer(\boldsymbol{L}_2))
\end{tikzcd}$$

Figure 7: Type Language Morphism

○ A *type language morphism* $\boldsymbol{f} = \langle refer(\boldsymbol{f}), sign(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ from type language $\boldsymbol{L}_1$ to type language $\boldsymbol{L}_2$ (Figure 7) is a two dimensional construction consisting of a reference quartet $refer(\boldsymbol{f})$ and a signature quartet $sign(\boldsymbol{f})$, where the signature of the reference quartet is the vertical target of the signature quartet

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

A type language morphism does not necessary have an associated (underlying) hypergraph morphism between the hypergraphs underlying its source and target, since the variable function (vertical source of the reference quartet) is not necessarily bijective.

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

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) language-morphism)
    (= (SET.FTN$target source) lang$language)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) language-morphism)
    (= (SET.FTN$target target) lang$language)

(4) (SET.FTN$function reference)
    (= (SET.FTN$source reference) language-morphism)
    (= (SET.FTN$target reference) set.qtt$quartet)
    (= (SET.FTN$composition [reference set.qtt$horizontal-source])
       (SET.FTN$composition [source lang$reference]))
    (= (SET.FTN$composition [reference set.qtt$horizontal-target])
       (SET.FTN$composition [target lang$reference]))

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

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

$$\begin{tikzcd}
\wp\mathit{var}(\boldsymbol{L}_1) \arrow[r, "\wp\mathit{var}(\boldsymbol{f})"] \arrow[d, "\mathit{refer\text{-}assign}(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "\mathit{refer\text{-}assign}(\boldsymbol{f})"] & \wp\mathit{var}(\boldsymbol{L}_2) \arrow[d, "\mathit{refer\text{-}assign}(\boldsymbol{L}_2)"] \\
\mathit{sign}(\mathit{refer}(\boldsymbol{L}_1)) \arrow[r, "\mathit{sign}(\mathit{refer}(\boldsymbol{f}))"'] & \mathit{sign}(\mathit{refer}(\boldsymbol{L}_2))
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{sign}(\mathit{refer}(\boldsymbol{L}_1)) \arrow[r, "\mathit{sign}(\mathit{refer}(\boldsymbol{f}))"] \arrow[d, "\mathit{refer\text{-}arity}(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "\mathit{refer\text{-}arity}(\boldsymbol{f})"] & \mathit{sign}(\mathit{refer}(\boldsymbol{L}_2)) \arrow[d, "\mathit{refer\text{-}arity}(\boldsymbol{L}_2)"] \\
\wp\mathit{var}(\boldsymbol{L}_1) \arrow[r, "\wp\mathit{var}(\boldsymbol{f})"'] & \wp\mathit{var}(\boldsymbol{L}_2)
\end{tikzcd}$$

Figure 8: The reference-assign and reference-arity quartets

○ For convenience of theoretical presentation, we introduce additional type language terminology for the composition between the reference function and the signature assign and arity functions. Associated with a type language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ is a *reference-assign* quartet *refer-assign*($\boldsymbol{f}$) and a *reference-arity* quartet *refer-arity*($\boldsymbol{f}$).

```
(7) (SET.FTN$function reference-assign)
    (= (SET.FTN$source reference-assign) language-morphism)
    (= (SET.FTN$target reference-assign) set.qtt$quartet)
    (= reference-assign (SET.FTN$composition [reference set.qtt$signature-assign]))
```

```
(8) (SET.FTN$function reference-arity)
    (= (SET.FTN$source reference-arity) language-morphism)
    (= (SET.FTN$target reference-arity) set.qtt$quartet)
    (= reference-arity (SET.FTN$composition [reference set.qtt$signature-arity]))
```

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}_1) \arrow[r, "\mathit{rel}(\boldsymbol{f})"] \arrow[d, "\mathit{rel\text{-}arity}(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "\mathit{rel\text{-}arity}(\boldsymbol{f})"] & \mathit{rel}(\boldsymbol{L}_2) \arrow[d, "\mathit{rel\text{-}arity}(\boldsymbol{L}_2)"] \\
\wp\mathit{var}(\boldsymbol{L}_1) \arrow[r, "\wp\mathit{var}(\boldsymbol{f})"'] & \wp\mathit{var}(\boldsymbol{L}_2)
\end{tikzcd}$$

Figure 9: The relation-arity quartet

○ The vertical composition of the signature quartet with the reference arity quartet defines a *relation-arity* quartet $\#_{\boldsymbol{L}}$ = *rel-arity*($\boldsymbol{L}$) = *sign*($\boldsymbol{L}$) • *refer-arity*($\boldsymbol{L}$) (Figure 9), which can be used in place of the signature quartet. The pullback constraint takes the form *vert-tgt*(*rel-arity*($\boldsymbol{f}$)) = $\wp$*vert-src*(*refer*($\boldsymbol{f}$)). Since reference arity and reference assign are (vertically) inverse quartets, we can equivalently define the signature function in terms of the relation-arity function.

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

```
(10) (forall (?f (language-morphism ?f))
         (= (signature ?f)
            (set.qtt$vertical-composition
                [(relation-arity ?f) (reference-assign ?f)])))
```

- For convenience of reference, we introduce additional type language terminology for the vertical source and target components of these quartets. The vertical source of the signature quartet is called the *relation* type function of $\boldsymbol{f}$ and denoted $\mathit{rel}(\boldsymbol{f})$. The vertical target of the reference quartet is called the *entity* type function of $\boldsymbol{f}$ and denoted $\mathit{ent}(\boldsymbol{f})$. The vertical source of the reference quartet is called the *variable* function of $\boldsymbol{f}$ and denoted $\mathit{var}(\boldsymbol{f})$. In summary, a type language morphism has the following component functions:
  - the *entity* type function $\mathit{ent}(\boldsymbol{f}) = \mathit{vert\text{-}tgt}(\mathit{refer}(\boldsymbol{f})) : \mathit{ent}(\boldsymbol{L}_1) \rightarrow \mathit{ent}(\boldsymbol{L}_2)$,
  - the *relation* type function $\mathit{rel}(\boldsymbol{f}) = \mathit{vert\text{-}src}(\mathit{sign}(\boldsymbol{f})) : \mathit{rel}(\boldsymbol{L}_1) \rightarrow \mathit{rel}(\boldsymbol{L}_2)$, and
  - the *variable* function $\mathit{var}(\boldsymbol{f}) = \mathit{vert\text{-}src}(\mathit{refer}(\boldsymbol{f})) : \mathit{var}(\boldsymbol{L}_1) \rightarrow \mathit{var}(\boldsymbol{L}_2)$.

  Therefore, a type language morphism can be displayed as in Figure 1. It is determined by the variable, entity type and relation type functions, and alternatively can be expressed and symbolized as

  $$\boldsymbol{f} = \langle \mathit{var}(\boldsymbol{f}), \mathit{ent}(\boldsymbol{f}), \mathit{rel}(\boldsymbol{f}) \rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2.$$

  The preservation of entity type reference, $*_{L1} \cdot \mathit{ent}(\boldsymbol{f}) = \mathit{var}(\boldsymbol{f}) \cdot *_{L2}$, means that the $\mathit{ent}(\boldsymbol{f})$-image of the $\boldsymbol{L}_1$-type of any variable $x \in \mathit{var}(\boldsymbol{L}_1)$ is the $\boldsymbol{L}_2$-type of the $\mathit{var}(\boldsymbol{f})$-image of the variable: $\mathit{ent}(f)(*_{L1}(x)) = *_{L2}(\mathit{var}(\boldsymbol{f})(x))$; or symbolically,

  if $x : \nu$ then $\mathit{var}(\boldsymbol{f})(x) : \mathit{ent}(f)(\nu)$.

  The preservation of relation type arity, $\#_{L1} \cdot \wp\mathit{var}(\boldsymbol{f}) = \mathit{rel}(\boldsymbol{f}) \cdot \#_{L2}$, means that the $\wp\mathit{var}(\boldsymbol{f})$-(direct)-image of the $\boldsymbol{L}_1$-arity of any relation type $\rho \in \mathit{rel}(\boldsymbol{L}_1)$ is the $\boldsymbol{L}_2$-arity of the $\mathit{rel}(\boldsymbol{f})$-image of the relation type: $\wp\mathit{var}(\boldsymbol{f})(\#_{L1}(\rho)) = \#_{L2}(\mathit{rel}(\boldsymbol{f})(\rho))$; or symbolically,

  if $(\rho, x_1, x_2, \ldots x_n)$ then $(\mathit{rel}(\boldsymbol{f})(\rho), \mathit{var}(\boldsymbol{f})(x_1), \mathit{var}(\boldsymbol{f})(x_2), \ldots \mathit{var}(\boldsymbol{f})(x_n))$.

  The preservation of relation type signature, $\partial_{L1} \cdot \mathit{sign}(\boldsymbol{f}) = \mathit{rel}(\boldsymbol{f}) \cdot \partial_{L2}$, means that the $\mathit{sign}(\boldsymbol{f})$-image of the $\boldsymbol{L}_1$-signature of any relation type $\rho \in \mathit{rel}(\boldsymbol{L}_1)$ is the $\boldsymbol{L}_2$-signature of the $\mathit{rel}(\boldsymbol{f})$-image of the relation type: $\mathit{sign}(\boldsymbol{f})(\partial_{L1}(\rho)) = \partial_{L2}(\mathit{rel}(\boldsymbol{f})(\rho))$; or symbolically,

  if $(\rho, x_1 : \alpha_1, x_2 : \alpha_2, \ldots x_n : \alpha_n)$ then $(\mathit{rel}(\boldsymbol{f})(\rho), \mathit{var}(\boldsymbol{f})(x_1) : \mathit{ent}(\boldsymbol{f})(\nu_1), \ldots \mathit{var}(\boldsymbol{f})(x_n) : \mathit{ent}(\boldsymbol{f})(\nu_n))$.

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

(12) (SET.FTN$function relation)
     (SET.FTN$function edge)
     (= edge relation)
     (= (SET.FTN$source relation) language-morphism)
     (= (SET.FTN$target relation) set.ftn$function)
     (= relation (SET.FTN$composition [signature set.qtt$vertical-source]))

(13) (SET.FTN$function variable)
     (SET.FTN$function name)
     (= name variable)
     (= (SET.FTN$source variable) language-morphism)
     (= (SET.FTN$target variable) set.ftn$function)
     (= variable (SET.FTN$composition [reference set.qtt$vertical-source]))
```

- Two type language morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable type language morphisms $\boldsymbol{f}_1 : \boldsymbol{L} \rightarrow \boldsymbol{L}'$ and $\boldsymbol{f}_2 : \boldsymbol{L}' \rightarrow \boldsymbol{L}''$ is defined in terms of the horizontal composition of their reference and signature quartets.

```
(14) (SET.LIM.PBK$opspan composable-opspan)
     (= (SET.LIM.PBK$class1 composable-opspan) language-morphism)
     (= (SET.LIM.PBK$class2 composable-opspan) language-morphism)
     (= (SET.LIM.PBK$opvertex composable-opspan) lang$language)
     (= (SET.LIM.PBK$first composable-opspan) target)
     (= (SET.LIM.PBK$second composable-opspan) source)

(15) (REL$relation composable)
```

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

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

- o Composition satisfies the usual *associative law*.

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

- o For any hypergraph ***L***, there is an *identity* type language morphism.

```
   (17) (SET.FTN$function identity)
        (= (SET.FTN$source identity) lang$language)
        (= (SET.FTN$target identity) language-morphism)
        (forall (?l (lang$language ?l))
            (and (= (source (identity ?l)) ?l)
                 (= (target (identity ?l)) ?l)
                 (= (reference (identity ?l))
                    (set.qtt$horizontal-identity (lang$reference ?l)))
                 (= (signature (identity ?l))
                    (set.qtt$horizontal-identity (lang$signature ?l)))))
```

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

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

- ○ A *simple* type language morphism is a type language morphism $\boldsymbol{f} = \langle id, id, rel(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ that is identity on variables and entity types. Since identities are simple and simple type language morphisms are closed under composition, simple type language morphisms form a subcategory of Language.

```
   (18) (SET$class simple)
        (SET$subclass simple language-morphism)
        (forall (?f (language-morphism ?f))
            (<=> (simple ?f)
                 (and (= (lang$variable (source ?f)) (lang$variable (target ?f)))
                      (= (variable ?f) (set.ftn$identity (lang$variable (source ?f))))
                      (= (lang$entity (source ?f)) (lang$entity (target ?f)))
                      (= (entity ?f) (set.ftn$identity (lang$entity (source ?f)))))))
```

- ○ A *special* type language morphism is a type language morphism that is bijective on variables. Obviously, a simple type language morphisms is special. Since identities are special and special type language morphisms are closed under composition, special type language morphisms also form a subcategory of Language.

```
   (19) (SET$class special)
        (SET$subclass special language-morphism)
        (forall (?f (language-morphism ?f))
            (<=> (special ?f) (bijection (variable ?f))))

        (SET$subclass simple special)
```

- ○ A simple type language morphism defines a function between substitutable sets.

```
(20) (SET.FTN$function substitutable)
     (= (SET.FTN$source substitutable) simple)
     (= (SET.FTN$target substitutable) set.ftn$function)
     (forall (?f (special ?f))
         (and (= (set.ftn$source (substitutable ?f))
                 (rel$extent (lang$substitutable (source ?f))))
              (= (set.ftn$target (substitutable ?f))
                 (rel$extent (lang$substitutable (target ?f))))
              (= (set.ftn$composition
                     [(substitutable ?f) (rel$first (lang$substitutable (target ?f)))])
                 (set.ftn$composition
                     [(rel$first (lang$substitutable (source ?f))) (relation ?f)]))
              (= (set.ftn$composition
                     [(substitutable ?f) (rel$second (lang$substitutable (target ?f)))])
                 (rel$second (lang$substitutable (source ?f))))))
```

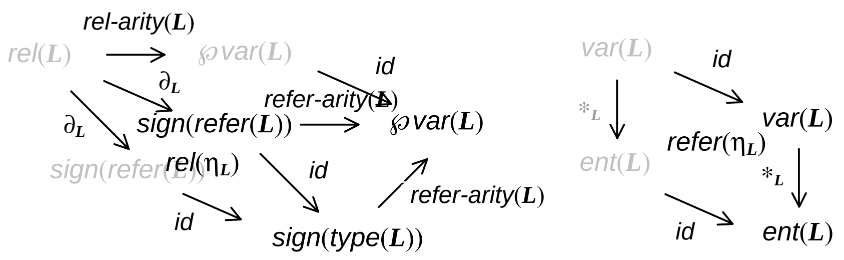


Figure 10: Eta Type Language Morphism

○ For every type language $\boldsymbol{L}$, there is an <u>simple</u> type language morphism

$$\eta_{\boldsymbol{L}} = \langle id_{var(\boldsymbol{L})}, id_{ent(\boldsymbol{L})}, sign(\boldsymbol{L})\rangle : \boldsymbol{L} \rightarrow lang(refer(\boldsymbol{L}))$$

from $\boldsymbol{L}$ to the type language of its reference function, where

$$refer(\eta_{\boldsymbol{L}}) = id_{type(\boldsymbol{L})} \text{ and } rel(\eta_{\boldsymbol{L}}) = sign(\boldsymbol{L}).$$

This type language morphism is the $\boldsymbol{L}$th component of a natural transformation

$$\eta : id_{\text{Language}} \Rightarrow refer \cdot lang.$$

```
(21) (SET.FTN$function eta)
     (= (SET.FTN$source eta) lang$language)
     (= (SET.FTN$target eta) language-morphism)
     (forall (?l (lang$language ?l))
         (and (= (source (eta ?l)) ?l)
              (= (target (eta ?l))
                 (set.ftn$language (lang$reference ?l)))
              (= (reference (eta ?l))
                 (set.qtt$horizontal-identity (lang$reference ?l)))
              (= (relation (eta ?l)) (lang$signature ?l))))

     (forall (?l (lang$language ?l))
         (simple (eta ?l)))
```

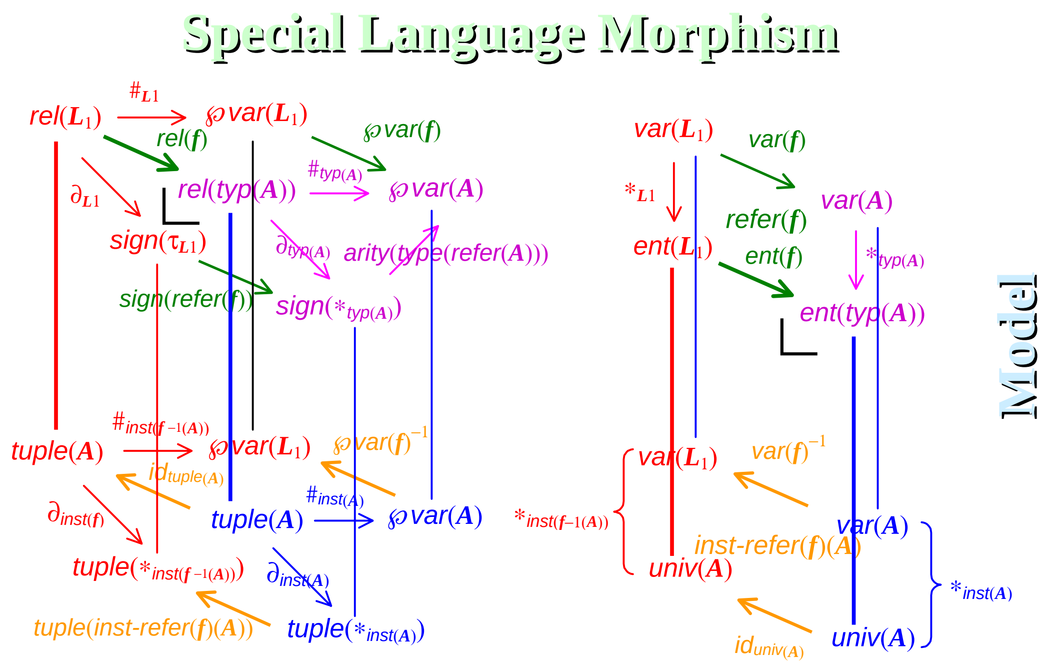


Figure 11: Model Fiber and Link model morphism

- Every special type language morphism

  $f : L_1 \rightarrow L_2$

  defines a model fiber (inverse image) operator

  $mod(f) : mod(L_2) \rightarrow mod(L_1)$

  from the model fiber over the target language to the model fiber over the source language. The details are illustrated in Figure 11, which is color coded for intelligibility.

  Conjecture: the model operation can be generalized to type language morphisms with surjective variable functions (but no linkage model morphism).

  Let $f = \langle refer(f), sign(f)\rangle = \langle var(f), ent(f), rel(f)\rangle : L_1 \rightarrow L_2$ be any special type language morphism and let $A = \langle refer(A), sign(A)\rangle$ be any model in the target fiber $A \in mod(L_2)$, so that f and A satisfy the matching constraint $tgt(f) = typ(A)$. We want to define an (inverse image) model in the source fiber

  $mod(f)(A) = f^{-1}(A) \in mod(L_1)$.

  As Figure 11 illustrates, the central components in this operator are two classification fiber (inverse image) operators, one for the entity classification and one for the relation classification.

  Before the main definitions for the model operator and the linking model morphism, there are some auxiliary definitions.

- There is a set *pair* $pr(f)(A) = *_{inst(f^{-1}(A))} = (var(L_1), univ(A))$.

```
(22) (KIF$function pair)
     (= (KIF$source pair) special)
     (= (KIF$target pair) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (pair ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (pair ?f)) set.pr$pair)
              (= (SET.FTN$composition [(pair ?f) set.pr$set1])
                 ((SET.FTN$constant [(lang$model (target ?f)) set$set])
                  (lang$variable (source ?f))))
              (= (SET.FTN$composition [(pair ?f) set.pr$set2])
                 (SET.FTN$composition [(lang$model-inclusion (target ?f)) mod$universe]))))
```

- There is an *instance reference* semiquartet $\mathit{inst\text{-}refer}(\boldsymbol{f})(\boldsymbol{A}) = *_{\mathit{inst}(\boldsymbol{f})}: *_{\mathit{inst}(\boldsymbol{A})} \rightarrow *_{\mathit{inst}(\boldsymbol{f}^{-1}(\boldsymbol{A}))}$.

```
(23) (KIF$function instance-reference)
     (= (KIF$source instance-reference) special)
     (= (KIF$target instance-reference) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (instance-reference ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (instance-reference ?f)) set.sqtt$semiquartet)
              (= (SET.FTN$composition [(instance-reference ?f) set.sqtt$source])
                 (SET.FTN$composition [(SET.FTN$composition
                     [(lang$model-inclusion (target ?f)) mod$instance]) hgph$reference])
              (= (SET.FTN$composition [(instance-reference ?f) set.sqtt$target]) (pair ?f))
              (= (SET.FTN$composition [(instance-reference ?f) set.sqtt$function1])
                 ((SET.FTN$constant [(lang$model (target ?f)) set.ftn$function])
                  (set.ftn$inverse (variable ?f))))
              (= (SET.FTN$composition [(instance-reference ?f) set.sqtt$function2])
                 (SET.FTN$composition [(SET.FTN$composition
                     [(lang$model-inclusion (target ?f)) mod$universe]) set.ftn$identity])))
```

- There is an *instance arity* quartet $\mathit{inst\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})$:
  - $\mathit{horiz\text{-}src}(\mathit{inst\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{edge\text{-}arity}(\mathit{inst}(\boldsymbol{A}))$
  - $\mathit{vert\text{-}src}(\mathit{inst\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{id}_{\mathit{tuple}(\boldsymbol{A})}$
  - $\mathit{vert\text{-}tgt}(\mathit{inst\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \wp \mathit{var}(\boldsymbol{f})^{-1}$

  Because of the quartet constraint, there is no need to specify the $\mathit{horiz\text{-}tgt}(\mathit{inst\text{-}arity}(\boldsymbol{f})(\boldsymbol{A}))$.

```
(24) (KIF$function instance-arity)
     (= (KIF$source instance-arity) special)
     (= (KIF$target instance-arity) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (instance-arity ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (instance-arity ?f)) set.qtt$quartet)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (set.qtt$horizontal-source ((instance-arity ?f) ?a))
                          (hgph$edge-arity (mod$instance ?a)))
                       (= (set.qtt$vertical-source ((instance-arity ?f) ?a))
                          (set.ftn$identity (mod$tuple ?a)))
                       (= (set.qtt$vertical-target ((instance-arity ?f) ?a))
                          (set.ftn$power (set.ftn$inverse (mod$variable ?f))))))))
```

- There is an *instance signature* quartet $\mathit{inst\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})$:
  - $\mathit{horiz\text{-}src}(\mathit{inst\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{sign}(\mathit{inst}(\boldsymbol{A}))$
  - $\mathit{vert\text{-}src}(\mathit{inst\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{id}_{\mathit{tuple}(\boldsymbol{A})}$
  - $\mathit{vert\text{-}tgt}(\mathit{inst\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{tuple}(\mathit{inst\text{-}refer}(\boldsymbol{f})(\boldsymbol{A}))$

```
(25) (KIF$function instance-signature)
     (= (KIF$source instance-signature) special)
     (= (KIF$target instance-signature) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (instance-signature ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (instance-signature ?f)) set.qtt$quartet)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (set.qtt$horizontal-source ((instance-signature ?f) ?a))
                          (hgph$signature (mod$instance ?a)))
                       (= (set.qtt$vertical-source ((instance-signature ?f) ?a))
                          (set.ftn$identity (mod$tuple ?a)))
                       (= (set.qtt$vertical-target ((instance-signature ?f) ?a))
                          (set.sqtt$tuple ((instance-reference ?f) ?a)))))))
```

- There is a *model reference* semidesignation

  $$\mathit{mod\text{-}refer}(\boldsymbol{f})(\boldsymbol{A}) = \langle \mathit{pr}(\boldsymbol{f})(\boldsymbol{A}), \mathit{refer}(\mathit{src}(\boldsymbol{f}))\rangle : \mathit{var}(\mathit{src}(\boldsymbol{f})) \looparrowright \mathit{typ\text{-}fib}(\mathit{ent}(\boldsymbol{f}))(\mathit{ent}(\boldsymbol{A})):$$

  - $\mathit{cls}(\mathit{mod\text{-}refer}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{typ\text{-}fib}(\mathit{ent}(\boldsymbol{f}))(\mathit{ent}(\boldsymbol{A}))$

    using the classification fiber (inverse image) function

    $\mathit{typ\text{-}fib}(\mathit{ent}(\boldsymbol{f})) : \mathit{typ\text{-}fib}(\mathit{ent}(\boldsymbol{L}_2)) \rightarrow \mathit{typ\text{-}fib}(\mathit{ent}(\boldsymbol{L}_1))$

of the entity (type) function $\mathit{ent}(\boldsymbol{f}) : \mathit{ent}(\boldsymbol{L}_1) \rightarrow \mathit{ent}(\boldsymbol{L}_2)$

- $\mathit{inst}(\mathit{mod\text{-}refer}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{pr}(\boldsymbol{f})(\boldsymbol{A})$
- $\mathit{typ}(\mathit{mod\text{-}refer}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{refer}(\mathit{src}(\boldsymbol{f}))$

```
(26) (KIF$function model-reference)
     (= (KIF$source model-reference) special)
     (= (KIF$target model-reference) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (model-reference ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (model-reference ?f)) cls.sdsgn$semidesignation)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (cls.sdsgn$set ((model-reference ?f) ?a))
                          (lang$variable (source ?f)))
                       (= (cls.sdsgn$classification ((model-reference ?f) ?a))
                          ((set.ftn$type-fiber (entity ?f)) (mod$entity ?a)))
                       (= (cls.sdsgn$instance ((model-reference ?f) ?a))
                          ((pair ?f) ?a))
                       (= (cls.sdsgn$type ((model-reference ?f) ?a))
                          (lang$reference (source ?f)))))))
```

○ There is a *model arity* designation

$\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A}) = \langle \#_{\mathit{inst}(\boldsymbol{A})} \cdot \wp \mathit{var}(\boldsymbol{f})^{-1}, \mathit{arity}(\boldsymbol{L}_1)\rangle : \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f}))(\mathit{rel}(\boldsymbol{A})) \rightrightarrows \mathit{sup}(\mathit{var}(\boldsymbol{L}_1))$:

- $\mathit{src}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f}))(\mathit{rel}(\boldsymbol{A}))$

  using the classification fiber (inverse image)

  $\mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f})) : \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{L}_2)) \rightarrow \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{L}_1))$

  of the relation (type) function $\mathit{rel}(\alpha) : \mathit{rel}(\boldsymbol{L}_1) \rightarrow \mathit{rel}(\boldsymbol{L}_2)$
- $\mathit{tgt}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{sup}(\mathit{var}(\boldsymbol{L}_1))$
- $\mathit{inst}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \#_{\mathit{inst}(\boldsymbol{A})} \cdot \wp \mathit{var}(\boldsymbol{f})^{-1}$
- $\mathit{typ}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A})) = \#_{\boldsymbol{L}1} = \mathit{arity}(\boldsymbol{L}_1)$

```
(27) (KIF$function model-arity)
     (= (KIF$source model-arity) special)
     (= (KIF$target model-arity) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (model-arity ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (model-arity ?f)) cls.dsgn$designation)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (cls.dsgn$source ((model-arity ?f) ?a))
                          ((set.ftn$type-fiber (relation ?f)) (mod$relation ?a)))
                        (= (cls.dsgn$target ((model-arity ?f) ?a))
                           (set$super (lang$variable (source ?f))))
                        (= (cls.dsgn$instance ((model-arity ?f) ?a))
                           (set.ftn$composition
                               [(hgph$edge-arity (mod$instance ?a))
                                (set.ftn$power (set.ftn$inverse (variable ?f)))]))
                        (= (cls.dsgn$type ((model-arity ?f) ?a))
                           (lang$relation-arity (source ?f)))))))
```

○ There is a *model signature* designation:

$\mathit{mod\text{-}sign}(\boldsymbol{f})(\boldsymbol{A}) = \langle \partial_{\mathit{inst}(\boldsymbol{A})} \cdot \mathit{tuple}(*_{\mathit{inst}(\boldsymbol{f})}), \mathit{sign}(\boldsymbol{L}_1)\rangle : \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f}))(\mathit{rel}(\boldsymbol{A})) \rightrightarrows \mathit{sign}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A}))$:

- $\mathit{src}(\mathit{mod\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f}))(\mathit{rel}(\boldsymbol{A}))$

  using the classification fiber (inverse image)

  $\mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{f})) : \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{L}_2)) \rightarrow \mathit{typ\text{-}fib}(\mathit{rel}(\boldsymbol{L}_1))$

  of the relation (type) function $\mathit{rel}(\boldsymbol{f}) : \mathit{rel}(\boldsymbol{L}_1) \rightarrow \mathit{rel}(\boldsymbol{L}_2)$
- $\mathit{tgt}(\mathit{mod\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \mathit{sign}(\mathit{mod\text{-}arity}(\boldsymbol{f})(\boldsymbol{A}))$
- $\mathit{inst}(\mathit{mod\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \partial_{\mathit{inst}(\boldsymbol{A})} \cdot \mathit{tuple}(\mathit{inst\text{-}ref}(\boldsymbol{f})(\boldsymbol{A}))$
- $\mathit{typ}(\mathit{mod\text{-}sign}(\boldsymbol{f})(\boldsymbol{A})) = \partial_{\boldsymbol{L}1} = \mathit{sign}(\boldsymbol{L}_1)$

```
(28) (KIF$function model-signature)
     (= (KIF$source model-signature) special)
     (= (KIF$target model-signature) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (model-signature ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (model-signature ?f)) cls.dsgn$designation)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (cls.dsgn$source ((model-signature ?f) ?a))
                          ((set.ftn$type-fiber (relation ?f)) (mod$relation ?a)))
                       (= (cls.dsgn$target ((model-signature ?f) ?a))
                          (cls.sdsgn$signature ((model-reference ?f) ?a)))
                       (= (cls.dsgn$instance ((model-signature ?f) ?a))
                          (set.ftn$composition
                              [(hgph$signature (mod$instance ?a))
                               (set.sqtt$tuple ((instance-reference ?f) ?a))]))
                       (= (cls.dsgn$type ((model-signature ?f) ?a))
                          (lang$signature (source ?f)))))))
```

- ○ The *model* (inverse image) function $\mathit{mod}(\boldsymbol{f}) : \mathit{mod}(\boldsymbol{L}_2) \rightarrow \mathit{mod}(\boldsymbol{L}_1)$ for any special type language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ can now be defined. The model operator $\mathit{mod}(\boldsymbol{f})(\boldsymbol{A}) = \boldsymbol{f}^{-1}(\boldsymbol{A})$ is define in terms of the above components.

```
(29) (KIF$function model)
     (= (KIF$source model) special)
     (= (KIF$target model) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (model ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (model ?f)) (lang$model (source ?f)))
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (mod$reference ((model ?f) ?a)) ((model-reference ?f) ?a))
                       (= (mod$arity ((model ?f) ?a)) ((model-arity ?f) ?a))
                       (= (mod$signature ((model ?f) ?a)) ((model-signature ?f) ?a))))))
```

- o The *link* model morphism $\mathit{link}(\boldsymbol{f})(\boldsymbol{A}) : \boldsymbol{f}^{-1}(\boldsymbol{A}) \rightarrow \boldsymbol{A}$ is defined as follows.

```
(30) (KIF$function link)
     (= (KIF$source link) special)
     (= (KIF$target link) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (link ?f)) (lang$model (target ?f)))
              (= (SET.FTN$target (link ?f)) mod.mor$model-morphism)
              (forall (?a ((lang$model (target ?f)) ?a))
                  (and (= (mod.mor$source ((link ?f) ?a)) ?a)
                       (= (mod.mor$target ((link ?f) ?a)) ((model ?f) ?a))
                       (= (mod.mor$type ((link ?f) ?a)) ?f)
                       (= (hgph$reference (mod.mor$instance ((link ?f) ?a)))
                          ((instance-reference ?f) ?a))
                       (= (hgph$edge-arity (mod.mor$instance ((link ?f) ?a)))
                          ((instance-arity ?f) ?a))
                       (= (hgph$signature (mod.mor$instance ((link ?f) ?a)))
                          ((instance-signature ?f) ?a))))))
```

The next question to answer is "How do fibers interact with expressions?" We make this explicit by asking three more specific questions. The answers to these three questions, which each need a proof, will be used to prove the functoriality of the model operator for type language interpretations.

$$\begin{tikzcd}
\boldsymbol{L} \arrow[r, "\eta_{\boldsymbol{L}}"] & \mathit{expr}(\boldsymbol{L}) & \mathit{mod}(\boldsymbol{L}) & \mathit{mod}(\mathit{expr}(\boldsymbol{L})) \arrow[l, "\mathit{mod}(\eta_{\boldsymbol{L}})"'] \\
 & & & \mathit{mod}(\boldsymbol{L}) \arrow[u, "\mathit{mod\text{-}expr}(\boldsymbol{L})"'] \arrow[ul, "\mathit{id}_{\mathit{mod}(\boldsymbol{L})}"]
\end{tikzcd}$$

Diagram 4: Model operator of eta embedding

- ○ How does the model operator interact with the eta embedding?

  The composition of the model expression with the model of the eta (embedding) function is the identity (Diagram 4).

Intuitively, the expression model $\mathit{expr}(\boldsymbol{A})$ of a model $\boldsymbol{A}$, extends $\boldsymbol{A}$ to expressions by inductively defining an extent for each expression starting from the extent of the given relation types. The model of the eta embedding function works in the opposite direction by restricting $\mathit{expr}(\boldsymbol{A})$ to relation types, thus ending up with the original model $\boldsymbol{A}$; that is, $\mathit{mod}(\eta_{\boldsymbol{L}})(\mathit{expr}(\boldsymbol{A})) = \boldsymbol{A}$.

$$\begin{tikzcd}
\mathit{expr}(\mathit{expr}(\boldsymbol{L})) \arrow[r, "\mu_{\boldsymbol{L}}"] & \mathit{expr}(\boldsymbol{L})
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{mod}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) & \mathit{mod}(\mathit{expr}(\boldsymbol{L})) \arrow[l, "\mathit{mod}(\mu_{\boldsymbol{L}})"'] \\
\mathit{mod}(\mathit{expr}(\boldsymbol{L})) \arrow[u, "\mathit{mod\text{-}expr}(\mathit{expr}(\boldsymbol{L}))"] & \\
\mathit{mod}(\boldsymbol{L}) \arrow[u, "\mathit{mod\text{-}expr}(\boldsymbol{L})"] & \mathit{mod}(\boldsymbol{L}) \arrow[l, "id_{\mathit{mod}(\boldsymbol{L})}"] \arrow[uu, "\mathit{mod\text{-}expr}(\boldsymbol{L})"']
\end{tikzcd}$$

Diagram 5: Model operator of mu collapsing

- How does the model operator interact with the mu collapsing function?

  The composition of the model expression with the model of the mu (collapsing) function is the composition of the iterated model expression (Diagram 5).

  Intuitively, the model of the mu collapsing function maps to the expression-of-expression model that attaches to each expression-of-expression the extent of the collapsed expression in $\mathit{expr}(\boldsymbol{A})$, which is built up from the extent of the original relation types in $\boldsymbol{A}$ by induction. This extent construction is equivalent to the iterated construction for expressions-of-expressions, which results from the iterated model expression function $\mathit{expr}(\mathit{expr}(\boldsymbol{A}))$; that is, $\mathit{mod}(\mu_{\boldsymbol{L}})(\mathit{expr}(\boldsymbol{A})) = \mathit{expr}(\mathit{expr}(\boldsymbol{A}))$.

$$\begin{tikzcd}
\mathit{expr}(\boldsymbol{L}_1) \arrow[r, "\mathit{expr}(\boldsymbol{f})"] & \mathit{expr}(\boldsymbol{L}_2) \\
\boldsymbol{L}_1 \arrow[u, "\eta_{L1}"] \arrow[r, "f"'] & \boldsymbol{L}_2 \arrow[u, "\eta_{L2}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{mod}(\mathit{expr}(\boldsymbol{L}_1)) & \mathit{mod}(\mathit{expr}(\boldsymbol{L}_2)) \arrow[l, "\mathit{mod}(\mathit{expr}(\boldsymbol{f}))"'] \\
\mathit{mod}(\boldsymbol{L}_1) \arrow[u, "\mathit{mod\text{-}expr}(\boldsymbol{L}_1)"] & \mathit{mod}(\boldsymbol{L}_2) \arrow[l, "\mathit{mod}(\boldsymbol{f})"] \arrow[u, "\mathit{mod\text{-}expr}(\boldsymbol{L}_2)"']
\end{tikzcd}$$

Diagram 6: Model operator of expression operator

- How does the model operator interact with the expression operator?

  The model of the expression commutes with the model function via the model expressions for source and target languages (Diagram 6).

  Intuitively, let $\boldsymbol{B} \in \mathit{mod}(\boldsymbol{L}_2)$ be any model in the target fiber and $\boldsymbol{A} \in \mathit{mod}(\boldsymbol{L}_1)$ be the model in the target fiber that is its image under the fiber function $\mathit{mod}(\boldsymbol{f})(\boldsymbol{B}) = \boldsymbol{A}$, so that the extent of any relation type $\rho \in \mathit{rel}(\boldsymbol{L}_1)$ in the source language is the same as the extent of the relation type $\mathit{rel}(\boldsymbol{f})(\rho) \in \mathit{rel}(\boldsymbol{L}_2)$ in the target language. By induction this means that the extent of any expression $\varphi \in \mathit{rel}(\mathit{expr}(\boldsymbol{L}_1))$ in the model $\mathit{expr}(\boldsymbol{A})$ is the same as the extent of the expression $\varphi \in \mathit{rel}(\mathit{expr}(\boldsymbol{L}_2))$ in the model $\mathit{expr}(\boldsymbol{B})$; that is, $\mathit{mod}(\mathit{expr}(\boldsymbol{f}))(\mathit{expr}(\boldsymbol{B})) = \mathit{expr}(\boldsymbol{A}) = \mathit{expr}(\mathit{mod}(\boldsymbol{f})(\boldsymbol{B}))$.

- The expression type language function

  $\mathit{expr}(\boldsymbol{f}) : \mathit{expr}(\boldsymbol{L}_1) \rightarrow \mathit{expr}(\boldsymbol{L}_2)$

  for any special type language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ is also special. Its model function

  $\mathit{mod}(\mathit{expr}(\boldsymbol{f})) : \mathit{mod}(\mathit{expr}(\boldsymbol{L}_2)) \rightarrow \mathit{mod}(\mathit{expr}(\boldsymbol{L}_1))$

  factors through recursively defined models (Diagram 7). The *recursive* fiber function

  $\mathit{recur}(\boldsymbol{f}) : \mathit{recur}(\boldsymbol{L}_2) \rightarrow \mathit{recur}(\boldsymbol{L}_1)$

  is the restriction of the above model function to recursively defined models.

$$\begin{tikzcd}
\mathit{mod}(\mathit{expr}(\boldsymbol{L}_1)) & \mathit{mod}(\mathit{expr}(\boldsymbol{L}_2)) \arrow[l, "\mathit{mod}(\mathit{expr}(\boldsymbol{f}))"'] \\
\mathit{recur}(\boldsymbol{L}_1) \arrow[u, "incl"'] & \mathit{recur}(\boldsymbol{L}_2) \arrow[l, "\mathit{recur}(\boldsymbol{f})"'] \arrow[u, "incl"] \\
\mathit{mod}(\boldsymbol{L}_1) \arrow[u, "\cong"'] \arrow[uu, bend left=60, "\mathit{mod\text{-}expr}(\boldsymbol{L}_1)"] & \mathit{mod}(\boldsymbol{L}_2) \arrow[l, "\mathit{mod}(\boldsymbol{f})"'] \arrow[u, "\cong"] \arrow[uu, bend right=60, "\mathit{mod\text{-}expr}(\boldsymbol{L}_2)"']
\end{tikzcd}$$

Diagram 7: Recursive fiber function

```
        (forall (?f (special ?f))
            (and (special (lang.expr.mor$expression ?f))
                 (= (set.ftn$composition
                        [(lang$model-expression (target ?f))
                         (model (lang.expr.mor$expression ?f))])
                    (set.ftn$composition
                        [(model ?f)
                         (lang$model-expression (source ?f))])))))

(31) (KIF$function recursive)
     (= (KIF$source recursive) special)
     (= (KIF$target recursive) SET.FTN$function)
     (forall (?f (special ?f))
         (and (= (SET.FTN$source (recursive ?f)) (lang$recursive (target ?f)))
              (= (SET.FTN$target (recursive ?f)) (lang$recursive (source ?f)))
              (SET.FTN$restriction (recursive ?f) (model (lang.expr.mor$expression ?f)))))

     (forall (?f (special ?f)
              ?a ((lang$model (target ?f)) ?a))
         (= ((recursive ?f) (mod$expression ?a))
            (mod$expression ((model ?f) ?a))))
```

○ Any type language morphism $\boldsymbol{f} = \langle refer(\boldsymbol{f}), sign(\boldsymbol{f})\rangle : \boldsymbol{L} \rightarrow \boldsymbol{K}$ induces an associated *kernel* endorelation $ker(\boldsymbol{f}) = \langle ker(var(\boldsymbol{f})), ker(ent(\boldsymbol{f})), ker(rel(\boldsymbol{f}))\rangle$ on its source language, where the components or the kernel equivalence relations for the variable, entity and relation functions, respectively. The fundamental endorelation constraints follow from the constraints for the reference, arity and signature quartets.

```
(32) (SET.FTN$function kernel)
     (= (SET.FTN$source kernel) language-morphism)
     (= (SET.FTN$target kernel) lang.col.endo$endorelation)
     (= (SET.FTN$composition [kernel lang.col.endo$language]) source)
     (= (SET.FTN$composition [kernel lang.col.endo$variable])
        (SET.FTN$composition [variable set.ftn$kernel]))
     (= (SET.FTN$composition [kernel lang.col.endo$entity])
        (SET.FTN$composition [entity set.ftn$kernel]))
     (= (SET.FTN$composition [kernel lang.col.endo$relation])
        (SET.FTN$composition [relation set.ftn$kernel]))
```

○ Since any type language morphism $\boldsymbol{f} = \langle refer(\boldsymbol{f}), sign(\boldsymbol{f})\rangle : \boldsymbol{L} \rightarrow \boldsymbol{K}$ respects its kernel, it induces an factorization: an *epimorphism* type language morphism

$$epi(\boldsymbol{f}) = canon(ker(\boldsymbol{f})) : \boldsymbol{L} \rightarrow quo(ker(\boldsymbol{f})) = src(\boldsymbol{f})/ker(\boldsymbol{f}),$$

which is the canon of the kernel of $\boldsymbol{f}$, and a *monomorphism* type language morphism

$$mono(\boldsymbol{f}) = comed((\boldsymbol{f}, ker(\boldsymbol{f}))) : \boldsymbol{L} \rightarrow quo(ker(\boldsymbol{f})) = src(\boldsymbol{f})/ker(\boldsymbol{f}),$$

which is the comediator of the morphism-kernel pair. Factorization mean that

$$\boldsymbol{f} = epi(\boldsymbol{f}) \cdot mono(\boldsymbol{f}).$$

```
(33) (SET.FTN$function epimorphism)
     (= (SET.FTN$source epimorphism) language-morphism)
     (= (SET.FTN$target epimorphism) language-morphism)
     (= (SET.FTN$composition [epimorphism source]) source)
     (= (SET.FTN$composition [epimorphism target])
        (SET.FTN$composition [kernel lang.col.endo$quotient]))
     (= epimorphism (SET.FTN$composition [kernel lang.col.endo$canon]))

(34) (SET.FTN$function monomorphism)
     (= (SET.FTN$source monomorphism) language-morphism)
     (= (SET.FTN$target monomorphism) language-morphism)
     (= (SET.FTN$composition [monomorphism source])
        (SET.FTN$composition [kernel lang.col.endo$quotient]))
     (= (SET.FTN$composition [monomorphism target]) target)
     (forall (?f (language-morphism ?f))
         (= (monomorphism ?f) (lang.col.endo$comediator [?f (kernel ?f)])))

     (forall (?f (language-morphism ?f))
         (= (composition [(epimorphism ?f) (monomorphism ?f)]) ?f))
```

Case and Spangraph Morphisms

○ Any type language morphism $\boldsymbol{f} = \langle \mathit{refer}(\boldsymbol{f}), \mathit{sign}(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ defines a coproduct *arity morphism*, whose set quartet is the relation arity quartet

$$\mathit{rel\text{-}arity}(\boldsymbol{f}) = \langle \mathit{rel}(\boldsymbol{f}), \wp\mathit{var}(\boldsymbol{f})\rangle : \mathit{rel\text{-}arity}(\boldsymbol{L}_1) \rightarrow \mathit{rel\text{-}arity}(\boldsymbol{L}_2).$$

```
(32) (SET.FTN$function arity-morphism)
     (= (SET.FTN$source arity-morphism) language-morphism)
     (= (SET.FTN$target arity-morphism) set.col.art.mor$arity-morphism)
     (forall (?f (language-morphism ?f))
          (and (= (set.col.art.mor$source (arity-morphism ?f)) (lang$arity (source ?f)))
               (= (set.col.art.mor$target (arity-morphism ?f)) (lang$arity (target ?f)))
               (= (set.col.art.mor$index (arity-morphism ?f)) (relation ?f))
               (= (set.col.art.mor$base (arity-morphism ?f)) (variable ?f))))
```

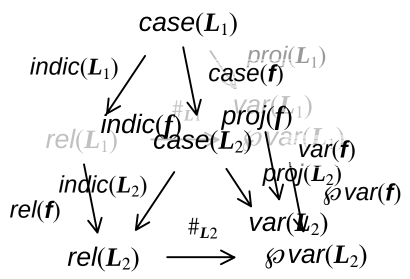


Figure 12: Case function = Coproduct of the Arity of a Type Language Morphism

○ Any type language morphism $\boldsymbol{f} = \langle \mathit{refer}(\boldsymbol{f}), \mathit{sign}(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ defines a *case* function $\mathit{case}(f) = \sum\mathit{arity}(\boldsymbol{f}) : \mathit{case}(\boldsymbol{L}_1) = \sum\mathit{arity}(\boldsymbol{L}_1) \rightarrow \sum\mathit{arity}(\boldsymbol{L}_2) = \mathit{case}(\boldsymbol{L}_2)$. The pointwise definition is:

$$\mathit{case}(\boldsymbol{f})((\rho, x)) = (\mathit{rel}(f)(\rho), \mathit{var}(f)(x))$$

for any relation type $\rho \in \mathit{rel}(\boldsymbol{L}_1)$ and any variable $x \in \mathit{arity}(\boldsymbol{L}_1)(\rho)$. This is well defined since $\boldsymbol{f}$ preserves relation type arity. The abstract definition (Figure 12) is in terms of the coproduct of the arity morphism.

```
(33) (SET.FTN$function case)
     (= (SET.FTN$source case) language-morphism)
     (= (SET.FTN$target case) set.ftn$function)
     (forall (?f (language-morphism ?f))
         (and (= (set.ftn$source (case ?f)) (lang$case (source ?f)))
              (= (set.ftn$target (case ?f)) (lang$case (target ?f)))
              (= (case ?f)
                 (set.col.art.mor$coproduct (arity-morphism ?f)))))
```

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}_1) \arrow[d, "\mathit{rel}(\boldsymbol{f})"'] \arrow[dr, phantom, "\mathit{indic}(\boldsymbol{f})"] & \mathit{case}(\boldsymbol{L}_1) \arrow[l, "\mathit{indic}(\boldsymbol{L}_1)"'] \arrow[r, "\mathit{proj}(\boldsymbol{L}_1)"] \arrow[d, "\mathit{case}(\boldsymbol{f})"] \arrow[dr, phantom, "\mathit{proj}(\boldsymbol{f})"] & \mathit{var}(\boldsymbol{L}_1) \arrow[d, "\mathit{var}(\boldsymbol{f})"] \\
\mathit{rel}(\boldsymbol{L}_2) & \mathit{case}(\boldsymbol{L}_2) \arrow[l, "\mathit{indic}(\boldsymbol{L}_2)"] \arrow[r, "\mathit{proj}(\boldsymbol{L}_2)"'] & \mathit{var}(\boldsymbol{L}_2)
\end{tikzcd}$$

Figure 13: Indication and Projection Quartets

- The case function is the vertical source for two quartets (Figure 13):
  an *indication* quartet $\mathit{indic}(\boldsymbol{f})$ and a *projection* quartet $\mathit{proj}(\boldsymbol{f})$.
  - The commutativity $\mathit{case}(f) \cdot \mathit{indic}(\boldsymbol{L}_2) = \mathit{indic}(\boldsymbol{L}_1) \cdot \mathit{rel}(\boldsymbol{f})$, a property of the coproduct of arities (preservation of indication), is obvious from the pointwise definition of the case function.
  - The commutativity $\mathit{case}(f) \cdot \mathit{proj}(\boldsymbol{L}_2) = \mathit{proj}(\boldsymbol{L}_1) \cdot \mathit{var}(\boldsymbol{f})$, a property of the coproduct of arities (preservation of projection), is obvious from the pointwise definition of the case function.

  By the preservation of relation type arity, the indication quartet is a fibration: for any relation type $\rho_1 \in \mathit{rel}(\boldsymbol{L}_1)$ and any variable $x_2 \in \mathit{arity}(\boldsymbol{L}_2)(\mathit{rel}(f)(\rho_1))$ there is a variable $x_1 \in \mathit{arity}(\boldsymbol{L}_1)(\rho_1)$ such that $\mathit{var}(f)(x_1) = x_2$.

```
(34) (SET.FTN$function indication)
     (= (SET.FTN$source indication) language-morphism)
     (= (SET.FTN$target indication) set.qtt$fibration)
     (forall (?f (language-morphism ?f))
         (and (= (set.qtt$horizontal-source (indication ?f)) (lang$indication (source ?f)))
              (= (set.qtt$horizontal-target (indication ?f)) (lang$indication (target ?f)))
              (= (set.qtt$vertical-source (indication ?f)) (case ?f))
              (= (set.qtt$vertical-target (indication ?f)) (relation ?f))))

(35) (SET.FTN$function projection)
     (= (SET.FTN$source projection) language-morphism)
     (= (SET.FTN$target projection) set.qtt$quartet)
     (forall (?f (language-morphism ?f))
         (and (= (set.qtt$horizontal-source (projection ?f)) (lang$projection (source ?f)))
              (= (set.qtt$horizontal-target (projection ?f)) (lang$projection (target ?f)))
              (= (set.qtt$vertical-source (projection ?f)) (case ?f))
              (= (set.qtt$vertical-target (projection ?f)) (variable ?f))))
```

$$\begin{tikzcd}
\mathit{case}(\boldsymbol{L}_1) \arrow[r, "\tilde{*}_{L1} = \mathit{comed}(\boldsymbol{L}_1)"] \arrow[d, "\mathit{case}(\boldsymbol{f})"'] \arrow[dr, phantom, "\mathit{comed}(\boldsymbol{f})"] & \mathit{ent}(\boldsymbol{L}_1) \arrow[d, "\mathit{ent}(\boldsymbol{f})"] \\
\mathit{case}(\boldsymbol{L}_2) \arrow[r, "\tilde{*}_{L2} = \mathit{comed}(\boldsymbol{L}_2)"'] & \mathit{ent}(\boldsymbol{L}_2)
\end{tikzcd}$$

Figure 14: Comediator Quartet

- Any language morphism $\boldsymbol{f} = \langle \mathit{refer}(\boldsymbol{f}), \mathit{sign}(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ defines a *comediator* quartet $\tilde{*}_f = \mathit{comed}(\boldsymbol{f})$ (Figure 14). The commutativity $\mathit{case}(f) \cdot \mathit{comed}(\boldsymbol{L}_2) = \mathit{comed}(\boldsymbol{L}_1) \cdot \mathit{ent}(\boldsymbol{f})$ holds by a property of the coproduct of arities (preservation of cotupling). Commutativity states that

  $$\partial_{L2}(\mathit{rel}(f)(\rho))(\mathit{var}(f)(x)) = \mathit{ent}(f)(\partial_{L1}(\rho)(x))$$

  for any relation type $\rho \in \mathit{rel}(\boldsymbol{L}_1)$ and any variable $x \in \mathit{var}(\boldsymbol{L}_1)$, which is true by preservation of relation type signature.

```
(36) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) language-morphism)
     (= (SET.FTN$target comediator) set.qtt$quartet)
     (forall (?f (language-morphism ?f))
         (and (= (set.qtt$horizontal-source (comediator ?f)) (lang$comediator (source ?f)))
              (= (set.qtt$horizontal-target (comediator ?f)) (lang$comediator (target ?f)))
              (= (set.qtt$vertical-source (comediator ?f)) (case ?f))
              (= (set.qtt$vertical-target (comediator ?f)) (entity ?f))))
```

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}_1) \arrow[d, "\mathit{rel}(\boldsymbol{f})"'] \arrow[dr, phantom, "\mathit{indic}(\boldsymbol{f})"] & \mathit{case}(\boldsymbol{L}_1) \arrow[l, "\mathit{indic}(\boldsymbol{L}_1)"'] \arrow[r, "\tilde{*}_{L1} = \mathit{comed}(\boldsymbol{L}_1)"] \arrow[d, "\mathit{case}(\boldsymbol{f})"] \arrow[dr, phantom, "\mathit{comed}(\boldsymbol{f})"] & \mathit{ent}(\boldsymbol{L}_1) \arrow[d, "\mathit{ent}(\boldsymbol{f})"] \\
\mathit{rel}(\boldsymbol{L}_2) & \mathit{case}(\boldsymbol{L}_2) \arrow[l, "\mathit{indic}(\boldsymbol{L}_2)"] \arrow[r, "\tilde{*}_{L2} = \mathit{comed}(\boldsymbol{L}_2)"'] & \mathit{ent}(\boldsymbol{L}_2)
\end{tikzcd}$$

Figure 15: Spangraph Morphism of a Type Language Morphism

- Associated with any type language morphism $\boldsymbol{f} = \langle \mathit{refer}(\boldsymbol{f}), \mathit{sign}(\boldsymbol{f})\rangle : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ is a spangraph morphism

  $\mathit{sgph\text{-}mor}(\boldsymbol{f}) = \langle 1^{st}_{\mathit{sgph}(\boldsymbol{f})}, 2^{nd}_{\mathit{sgph}(\boldsymbol{f})}, 3^{rd}_{\mathit{sgph}(\boldsymbol{f})}\rangle : \mathit{sgph}(\boldsymbol{L}_1) \rightarrow \mathit{sgph}(\boldsymbol{L}_2)$ (Figure 15),

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

  $1^{st}_{\mathit{sgph}(\boldsymbol{f})}(f) = \mathit{comed}(\boldsymbol{f})$,
  $2^{nd}_{\mathit{sgph}(\boldsymbol{f})}(f) = \mathit{proj}(\boldsymbol{f})$, and
  $3^{rd}_{\mathit{sgph}(\boldsymbol{f})}(f) = \mathit{indic}(\boldsymbol{f})$.

```
(37) (SET.FTN$function spangraph-morphism)
     (= (SET.FTN$source spangraph-morphism) language-morphism)
     (= (SET.FTN$target spangraph-morphism) sgph.mor$spangraph-morphism)
     (forall (?f (language-morphism ?f))
         (and (= (sgph.mor$source (spangraph-morphism ?f)) (lang$spangraph (source ?f)))
              (= (sgph.mor$target (spangraph-morphism ?f)) (lang$spangraph (target ?f)))
              (= (sgph.mor$first (spangraph-morphism ?f)) (comediator ?f))
              (= (sgph.mor$second (spangraph-morphism ?f)) (projection ?f))
              (= (sgph.mor$third (spangraph-morphism ?f)) (indication ?f))))
```

- For every type language $\boldsymbol{L}$, there is a simple type language morphism

  $\varepsilon_{\boldsymbol{L}} = \langle \mathit{proj}(\boldsymbol{L}), \mathit{id}_{\mathit{ent}(\boldsymbol{L})}, \mathit{id}_{\mathit{rel}(\boldsymbol{L})}\rangle : \mathit{lang}(\mathit{sgph}(\boldsymbol{L})) \rightarrow \boldsymbol{L}$

  to $\boldsymbol{L}$ from the type language of its spangraph, where

  $\mathit{refer}(\varepsilon_{\boldsymbol{L}}) = \mathit{fact}(\boldsymbol{L})$ and $\mathit{rel}(\varepsilon_{\boldsymbol{L}}) = \mathit{id}_{\mathit{rel}(\boldsymbol{L})}$.

  This type language morphism is the $\boldsymbol{L}$th component of a natural transformation

  $\varepsilon : \mathit{sgph} \cdot \mathit{lang} \Rightarrow \mathit{id}_{\text{Language}}$.

```
(38) (SET.FTN$function epsilon)
     (= (SET.FTN$source epsilon) lang$language)
     (= (SET.FTN$target epsilon) language-morphism)
     (forall (?l (lang$language ?l))
         (and (= (source (epsilon ?l)) (sgph$language (lang$spangraph ?l)))
              (= (target (epsilon ?l)) ?l)
              (= (reference (epsilon ?l)) (lang$factorization ?l))
              (= (relation (epsilon ?l)) (set.ftn$identity (lang$relation ?l)))))

     (forall (?l (lang$language ?l))
         (simple (epsilon ?l)))
```

Expressions

`lang.expr`

In this namespace we define terminology for expressions of a type language $\boldsymbol{L}$. Expressions for $\boldsymbol{L}$ form the set of relation types for a type language $expr(\boldsymbol{L})$ that extends $\boldsymbol{L}$.

○ Expressions of any type language $\boldsymbol{L}$ are relation types in a type language $expr(\boldsymbol{L})$ that extends $\boldsymbol{L}$. In order to describe the set of expressions $rel(expr(\boldsymbol{L}))$ over $\boldsymbol{L}$ we use the logical symbols: $\forall$ (universal quantifier), $\exists$ (existential quantifier), $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), and $\Leftrightarrow$ (equivalence). The axiomatic definition of the set of expressions is recursive in nature. An expression is either primitive (a relation type) or constructed from other expressions by binary conjunction, binary disjunction, negation, implication, equivalence, existential or universal quantification, or substitution of variables.

- Any *relation* type $\rho \in rel(\boldsymbol{L})$ is an expression $\rho \in rel(expr(\boldsymbol{L}))$ with the same arity:
  $arity(expr(\boldsymbol{L}))(\rho) = arity(\boldsymbol{L})(\rho)$.
  We identify the set of *relational expressions* with $rel(\boldsymbol{L})$.

- Let $\varphi \in rel(expr(\boldsymbol{L}))$ be any expression.
  The *negation* is an expression $\neg\varphi \in rel(expr(\boldsymbol{L}))$ with the same arity
  $arity(expr(\boldsymbol{L}))(\neg\varphi) = arity(expr(\boldsymbol{L}))(\varphi)$.
  Define the set of *negations* as $neg(\boldsymbol{L}) = \{\neg\}\times rel(expr(\boldsymbol{L})) \cong rel(expr(\boldsymbol{L}))$.

- Let $\varphi \in rel(expr(\boldsymbol{L}))$ and $\psi \in rel(expr(\boldsymbol{L}))$ be any two expressions.

  The *conjunction* is an expression $\varphi\wedge\psi \in rel(expr(\boldsymbol{L}))$ whose arity is the union
  $arity(expr(\boldsymbol{L}))(\varphi\wedge\psi) = arity(expr(\boldsymbol{L}))(\varphi) \cup arity(expr(\boldsymbol{L}))(\psi)$.
  Define the set of *conjunctions* as
  $conj(\boldsymbol{L}) = \{\wedge\}\times rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L})) \cong rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L}))$.

  The *disjunction* is an expression $\varphi\vee\psi \in rel(expr(\boldsymbol{L}))$ whose arity is the union
  $arity(expr(\boldsymbol{L}))(\varphi\vee\psi) = arity(expr(\boldsymbol{L}))(\varphi) \cup arity(expr(\boldsymbol{L}))(\psi)$.
  Define the set of *disjunctions* as
  $disj(\boldsymbol{L}) = \{\vee\}\times rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L})) \cong rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L}))$.

  The *implication* is an expression $\varphi\Rightarrow\psi \in rel(expr(\boldsymbol{L}))$ whose arity is the union
  $arity(expr(\boldsymbol{L}))(\varphi\Rightarrow\psi) = arity(expr(\boldsymbol{L}))(\varphi) \cup arity(expr(\boldsymbol{L}))(\psi)$.
  Define the set of *implications* as
  $impl(\boldsymbol{L}) = \{\Rightarrow\}\times rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L})) \cong rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L}))$.

  The *equivalence* is an expression $\varphi\Leftrightarrow\psi \in rel(expr(\boldsymbol{L}))$ whose arity is the union
  $arity(expr(\boldsymbol{L}))(\varphi\Leftrightarrow\psi) = arity(expr(\boldsymbol{L}))(\varphi) \cup arity(expr(\boldsymbol{L}))(\psi)$.
  Define the set of *equivalences* as
  $equiv(\boldsymbol{L}) = \{\Leftrightarrow\}\times rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L})) \cong rel(expr(\boldsymbol{L}))\times rel(expr(\boldsymbol{L}))$.

- Let $\varphi \in rel(expr(\boldsymbol{L}))$ be any expression and let $x \in arity(expr(\boldsymbol{L}))(\varphi)$ be any variable in its arity.

  The *existential quantification* is an expression $(\exists x)\varphi \in rel(expr(\boldsymbol{L}))$ whose arity is the difference
  $arity(expr(\boldsymbol{L}))((\exists x)\varphi) = arity(expr(\boldsymbol{L}))(\varphi) - \{x\}$.
  Define the set of existential expressions as
  $exist(\boldsymbol{L}) = \{\exists\}\times\sum_{\varphi \in expr(L)} arity(\boldsymbol{L})(\varphi) = \{\exists\}\times case(rel(expr(\boldsymbol{L}))) \cong case(rel(expr(\boldsymbol{L})))$.

  The *universal quantification* is an expression $(\forall x)\varphi \in expr(\boldsymbol{L})$ whose arity is the difference
  $arity(expr(\boldsymbol{L}))((\forall x)\varphi) = arity(expr(\boldsymbol{L}))(\varphi) - \{x\}$.
  We define the set of universal expressions as
  $forall(\boldsymbol{L}) = \{\forall\}\times\sum_{\varphi \in expr(L)} arity(\boldsymbol{L})(\varphi) = \{\forall\}\times case(rel(expr(\boldsymbol{L}))) \cong case(rel(expr(\boldsymbol{L})))$.

- Let $\varphi \in expr(\boldsymbol{L})$ be any expression and let $h : arity(expr(\boldsymbol{L}))(\varphi) \to Y$ be any substitution.
  The *substitution* of h into φ is an expression $\varphi[h] \in rel(expr(\boldsymbol{L}))$ whose arity is Y:
  $arity(expr(\boldsymbol{L}))(\varphi[h]) = Y = cod(h)$.

- The set of expressions is formally defined as the fixpoint of the operator
  $\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
  $= \mathit{rel}(\boldsymbol{L}) + \mathit{neg}(\boldsymbol{L}) + \mathit{conj}(\boldsymbol{L}) + \mathit{disj}(\boldsymbol{L}) + \mathit{impl}(\boldsymbol{L}) + \mathit{equiv}(\boldsymbol{L}) + \mathit{exist}(\boldsymbol{L}) + \mathit{forall}(\boldsymbol{L}) + \mathit{subst}(\boldsymbol{L})$
  $= \mathit{rel}(\boldsymbol{L})$
  $+ \{\neg\}\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
  $+ (\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
  $+ \{\exists, \forall\}\times\mathit{case}(\mathit{expr}(\boldsymbol{L}))$
  $+ \{[\,]\}\times\mathit{subst}(\mathit{expr}(\boldsymbol{L}))$
  where
  $\mathit{case}(\mathit{expr}(\boldsymbol{L})) = \sum_{\varphi \in \mathit{rel}(\mathit{expr}(\boldsymbol{L}))} \mathit{arity}(\mathit{expr}(\boldsymbol{L}))(\varphi)$
  $\mathit{subst}(\mathit{expr}(\boldsymbol{L})) = \{[\,]\}\times\mathit{extent}(\mathit{substitutable}(\mathit{expr}(\boldsymbol{L}))$
  $\cong \mathit{extent}(\mathit{substitutable}(\mathit{expr}(\boldsymbol{L})))$.

$$\begin{tikzcd}
\mathit{rel}(\boldsymbol{L}) \arrow[ddr, "\mathit{inj}(\boldsymbol{L})(\circledR)"] & \\
\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[dr, "\mathit{inj}(\boldsymbol{L})(\neg)"] & \\
\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}\times \mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[r, "\mathit{inj}(\boldsymbol{L})(\wedge,\vee,\Rightarrow,\Leftrightarrow)"] & \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \\
\{\exists, \forall\}\times\mathit{case}(\mathit{expr}(\boldsymbol{L})) \arrow[ur, "\mathit{inj}(\boldsymbol{L})(\exists,\forall)"] & \\
\mathit{subst}(\mathit{expr}(\boldsymbol{L})) \arrow[uur, "\mathit{inj}(\boldsymbol{L})([\,])"] &
\end{tikzcd}$$

®	$\mapsto$	$\mathit{rel}(\boldsymbol{L})$
¬	$\mapsto$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
∧	$\mapsto$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
∨	$\mapsto$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
⇒	$\mapsto$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
⇔	$\mapsto$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$
∃	$\mapsto$	$\mathit{case}(\mathit{expr}(\boldsymbol{L}))$
∀	$\mapsto$	$\mathit{case}(\mathit{expr}(\boldsymbol{L}))$
[	$\mapsto$	$\mathit{subst}(\mathit{expr}(\boldsymbol{L}))$

Figure 16: The set of expressions as a coproduct

- The set of expressions is axiomatized as the coproduct of a tuple of sets. First, create a set of indexing symbols *kind* = {®, ¬, ∧, ∨, ⇒, ⇔, ∃, ∀, []} for *primitive*, *negation*, *conjunctive*, *disjunctive*, *implicative*, *equivalent*, *existential*, *universal* and *substitutive* expressions, respectively. Then define a tuple of sets for expressions indexed by *kind* (Figure 16). This tuple of sets recursively uses the set of expression types and the language of expressions.

```
(1) (set$set kind)
    (kind primitive)
    (kind negative)
    (kind substitutive)

(2) (set$set connective)
    (set$subset connective kind)
    (connective conjunctive) (connective disjunctive)
    (connective implicative) (connective equivalent)

(3) (set$set quantifier)
    (set$subset quantifier kind)
    (quantifier existential) (quantifier universal)

(4) (SET.FTN$function tuple-set)
    (= (SET.FTN$source tuple-set) lang$language)
    (= (SET.FTN$target tuple-set) set.col.coprd$tuple-set)
    (forall (?l (lang$language ?l))
        (and (= (set.col.coprd$index (tuple-set ?l)) kind)
             (= ((tuple-set ?l) primitive) (relation ?l))
             (= ((tuple-set ?l) negative) (set ?l))
             (forall (?k (connective ?k))
                 (= ((tuple-set ?l) ?k)
                    (set.lim.prd2$binary-product [(set ?l) (set ?l)])))
             (forall (?k (quantifier ?k))
                 (= ((tuple-set ?l) ?k) (lang$case (expression ?l))))
             (= ((tuple-set ?l) substitutive)
                (rel$extent (lang$substitutable (expression ?l))))))
```

- Any type language $\boldsymbol{L}$ has a set of *expressions*

  $\mathit{set}(\boldsymbol{L}) = \mathit{rel}(\mathit{expr}(\boldsymbol{L})) = \sum \mathit{tuple\text{-}set}(\boldsymbol{L})$

  $= \sum_{k \in \mathit{kind}} \mathit{tuple\text{-}set}(\boldsymbol{L})(k)$

  $= \{(k, \varphi) \mid k \in \mathit{kind}, \varphi \in \mathit{tuple\text{-}set}(\boldsymbol{L})(k)\}$

  that is the coproduct of its tuple-set (Diagram 8).

$$\begin{tikzcd} \mathit{tuple\text{-}set}(\boldsymbol{L})(k) \arrow[r, "\mathit{inj}(\boldsymbol{L})(k)"] & \mathit{set}(\boldsymbol{L}) = \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \end{tikzcd}$$

Diagram 8: Coproduct

For any type language $\boldsymbol{L}$ and any expression kind $k \in \mathit{kind}$ there is an expression *injection* function (Diagram 8, Table 2 below):

$\mathit{inj}(\boldsymbol{L})(k) : \mathit{tuple\text{-}set}(\boldsymbol{L})(k) \rightarrow \mathit{set}(\boldsymbol{L}) = \mathit{rel}(\mathit{expr}(\boldsymbol{L}))$

defined by $\mathit{inj}(\boldsymbol{L})(k)(\varphi) = (k, \varphi)$ for all expression kinds $k \in \mathit{kind}$ and all expressions of that kind $\varphi \in \mathit{tuple\text{-}set}(\boldsymbol{L})(k)$. Obviously, the injections are injective.

```
(5) (SET.FTN$function set)
    (= (SET.FTN$source set) lang$language)
    (= (SET.FTN$target set) set$set)
    (forall (?l (lang$language ?l))
        (= (set ?l) (set.col.coprd$coproduct (tuple-set ?l))))

(6) (KIF$function injection)
    (= (KIF$source injection) lang$language)
    (= (KIF$target injection) SET.FTN$function)
    (forall (?l (lang$language ?l))
        (and (= (SET.FTN$source (injection ?l)) kind)
             (= (SET.FTN$source (injection ?l)) set.ftn$function)
             (= (SET.FTN$composition [(injection ?l) set.ftn$target]) (set ?l))
             (= (injection ?l) (set.col.coprd$injection (tuple-set ?l)))))
```

- Any type language $\boldsymbol{L}$ defines an expression *indication* function (Figure 17) based on its tuple set:

  $\mathit{indic}(\boldsymbol{L}) : \mathit{case}(\boldsymbol{L}) \rightarrow \mathit{rel}(\boldsymbol{L})$.

  This is defined by $\mathit{indic}(\boldsymbol{L})((k, \varphi)) = k$ for all expression kinds $k \in \mathit{kind}$ and all expressions of that kind $\varphi \in \mathit{tuple\text{-}set}(\boldsymbol{L})(k)$. Obviously, the injections are injective.

$$\begin{tikzcd} & \mathit{set}(\boldsymbol{L}) = \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[dl, "\mathit{indic}(\boldsymbol{L})"'] \\ \mathit{kind} & \end{tikzcd}$$

Figure 17: Indication

```
(7) (SET.FTN$function indication)
    (= (SET.FTN$source indication) lang$language)
    (= (SET.FTN$target indication) set.ftn$function)
    (= (SET.FTN$composition [indication set.ftn$source]) set)
    (= (SET.FTN$composition [indication set.ftn$target])
       ((SET.FTN$constant [lang$language set$set]) kind))
    (forall (?l (lang$language ?l))
        (= (indication ?l) (set.col.coprd$indication (tuple-set ?l))))
```

- The *arity* function

  $\#_{\mathit{expr}(\boldsymbol{L})} = \mathit{rel\text{-}arity}(\mathit{expr}(\boldsymbol{L})) : \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \rightarrow \wp\mathit{var}(\boldsymbol{L})$

  for expression types is defined as the cotupling of an *arity tuple* of functions. The functions in this tuple are defined recursively.

```
(8) (KIF$function arity-tuple)
    (= (KIF$source arity-tuple) lang$language)
    (= (KIF$target arity-tuple) SET.FTN$function)
    (forall (?l (lang$language ?l))
        (and (= (SET.FTN$source (arity-tuple ?l)) kind)
             (= (SET.FTN$target (arity-tuple ?l)) set.ftn$function)
             (= (SET.FTN$composition [(arity-tuple ?l) set.ftn$target])
                ((SET.FTN$constant [kind set$set]) (set$power (lang$variable ?l))))
           (= (set.ftn$source ((arity-tuple ?l) primitive)) (relation ?l))
           (= ((arity-tuple ?l) primitive) (lang$arity ?l))
           (= (set.ftn$source ((arity-tuple ?l) negative)) (set ?l))
           (= ((arity-tuple ?l) negative) (arity (expression ?l)))
           (forall (?k (connective ?k))
               (and (= (set.ftn$source ((arity-tuple ?l) ?k))
                       (set.lim.prd2$binary-product [(set ?l) (set ?l)]))
```

```
                      (= ((arity-tuple ?l) ?k)
                         (set.ftn$composition
                             [(set.lim.prd2$binary-product
                                  [(arity (set ?l)) (arity (set ?l))])
                               set$binary-union]))))
             (forall (?k (quantifier ?k))
                 (and (= (set.ftn$source ((arity-tuple ?l) ?k)) (lang$case (expression ?l)))
                      (= ((arity-tuple ?l) ?k)
                         (set.ftn$composition
                             [(set.lim.prd2.ftn$binary-product
                                  [(set.ftn$composition
                                       [(lang$index (expression ?l)) (arity (expression ?l))])
                                   (set.ftn$composition
                                       [(lang$projection (expression ?l))
                                        (set.ftn$singleton (lang$variable ?l))])])
                              set$difference]))))
             (= (set.ftn$source ((arity-tuple ?l) substitutive))
                (rel$extent (lang$substitutable (expression ?l))))
             (= ((arity-tuple ?l) substitutive)
                (set.ftn$composition
                    [(rel$second (lang$substitutable (expression ?l))) (lang$codomain ?l)]))))
```

```
(9) (SET.FTN$function arity)
    (= (SET.FTN$source arity) lang$language)
    (= (SET.FTN$target arity) set.ftn$function)
    (= (SET.FTN$composition [arity set.ftn$source]) set)
    (= (SET.FTN$composition [arity set.ftn$target])
       (SET.FTN$composition [lang$variable set$power]))
    (forall (?l (lang$language ?l))
         (= (arity ?l)
            ((set.col.coprd$cotupling (tuple-set ?l)) (arity-tuple ?l))))
```

○ For any subset of variables $X \subseteq \mathit{var}(\boldsymbol{L})$ the set

$$\mathit{arity\text{-}fiber}(\boldsymbol{L})(X) = \{\varphi \in \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \mid \mathit{arity}(\mathit{expr}(\boldsymbol{L}))(\varphi) = X\}$$

of all expressions having X as its arity is called the arity fiber of X. This is defined as the fiber at X of the arity function. As X varies we get the *arity fiber* set function

$$\mathit{arity\text{-}fiber}(\boldsymbol{L}) : \wp\mathit{var}(\boldsymbol{L}) \rightarrow \wp\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$$

for language $\boldsymbol{L}$. A *sentence* is an expression whose arity is empty. In particular, the arity fiber at the empty set is called the set of sentences of $\boldsymbol{L}$

$$\mathit{sent}(\boldsymbol{L}) = \mathit{arity\text{-}fiber}(\boldsymbol{L})(\emptyset).$$

```
(10) (KIF$function arity-fiber)
     (= (SET.FTN$source arity-fiber) lang$language)
     (= (SET.FTN$target arity-fiber) set.ftn$function)
     (= (SET.FTN$composition [arity-fiber set.ftn$source])
        (SET.FTN$composition [lang$variable set$power]))
     (= (SET.FTN$composition [arity-fiber set.ftn$target])
        (SET.FTN$composition [set set$power]))
     (forall (?l (language ?l))
         (= (arity-fiber ?l)
            (set.ftn$fiber (arity ?l))))
```

```
(11) (KIF$function sentence)
     (= (SET.FTN$source sentence) lang$language)
     (= (SET.FTN$target sentence) set$set)
     (forall (?l (language ?l))
         (= (sentence ?l)
            ((arity-fiber ?l) set$empty)))
```

○ The *signature* function

$$\partial_{\mathit{expr}(\boldsymbol{L})} = \mathit{sign}(\mathit{expr}(\boldsymbol{L})) : \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \rightarrow \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$$

for the expression language of $\boldsymbol{L}$ is defined in terms of this arity function:

$$\mathit{sign}(\mathit{expr}(\boldsymbol{L})) = \mathit{rel\text{-}arity}(\mathit{expr}(\boldsymbol{L})) \cdot \mathit{refer\text{-}assign}(\boldsymbol{L}),$$

where $\mathit{refer\text{-}assign}(\boldsymbol{L}) = \mathit{sign\text{-}assign}(\tau_{\boldsymbol{L}}) : \wp \mathit{var}(\boldsymbol{L}) \rightarrow \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$ is the signature-assign function for language $\boldsymbol{L}$.

```
(12) (SET.FTN$function signature)
     (= (SET.FTN$source signature) lang$language)
     (= (SET.FTN$target signature) set.ftn$function)
     (= (SET.FTN$composition [signature set.ftn$source]) set)
     (= (SET.FTN$composition [signature set.ftn$target])
        (SET.FTN$composition [reference set.ftn$signature]))
     (forall (?l (language ?l))
         (= (signature ?l)
            (set.ftn$composition [(arity ?l) (lang$reference-assign ?l)])))
```

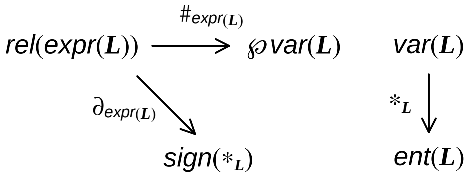
- For any type language

  $\boldsymbol{L} = \langle \mathit{ent}(\boldsymbol{L}), \mathit{rel}(\boldsymbol{L}), \mathit{var}(\boldsymbol{L}), *_{\boldsymbol{L}}, \#_{\boldsymbol{L}}, \partial_{\boldsymbol{L}} \rangle$,

  the associated type language of *expressions* (Figure 18)

  $\mathit{expr}(\boldsymbol{L}) = \langle \mathit{ent}(\boldsymbol{L}), \mathit{rel}(\mathit{expr}(\boldsymbol{L})), \mathit{var}(\boldsymbol{L}), *_{\boldsymbol{L}}, \#_{\mathit{expr}(\boldsymbol{L})}, \partial_{\mathit{expr}(\boldsymbol{L})} \rangle$

  has expressions as its relation types with the same entity types and variables as in $\boldsymbol{L}$. The reference function is unchanged, but the arity function (and also the equivalent signature function) is an extension of the arity function of $\boldsymbol{L}$ from the relation types of $\boldsymbol{L}$ to the expressions of $\boldsymbol{L}$. This has a recursive definition.



$$\begin{tikzcd}
\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[r, "\#_{\mathit{expr}(\boldsymbol{L})}"] \arrow[dr, swap, "\partial_{\mathit{expr}(\boldsymbol{L})}"] & \wp \mathit{var}(\boldsymbol{L}) & \mathit{var}(\boldsymbol{L}) \arrow[d, swap, "*_{\boldsymbol{L}}"] \\
 & \mathit{sign}(*_{\boldsymbol{L}}) & \mathit{ent}(\boldsymbol{L})
\end{tikzcd}$$

Figure 18: Expression Type Language

```
(13) (SET.FTN$function expression)
     (= (SET.FTN$source expression) lang$language)
     (= (SET.FTN$target expression) lang$language)
     (forall (?l (lang$language ?l))
         (and (= (lang$entity (expression ?l))    (lang$entity ?l))
              (= (lang$relation (expression ?l))  (set ?l))
              (= (lang$variable (expression ?l))  (lang$variable ?l))
              (= (lang$reference (expression ?l))      (lang$reference ?l))
              (= (lang$arity (expression ?l))     (arity ?l))
              (= (lang$signature (expression ?l)) (signature ?l))))
```

Table 2: Expression injection functions

$\mathit{inj}(\boldsymbol{L})(®)$	:	$\mathit{rel}(\boldsymbol{L})$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	ρ	$\mapsto$	$(®, \rho)$	=	ρ
$\mathit{inj}(\boldsymbol{L})(\neg)$	:	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	φ	$\mapsto$	$(\neg, \varphi)$	=	$\neg\varphi$
$\mathit{inj}(\boldsymbol{L})(\wedge)$	:	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, ψ)	$\mapsto$	$(\wedge, \varphi, \psi)$	=	$\varphi\wedge\psi$
$\mathit{inj}(\boldsymbol{L})(\vee)$	:	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, ψ)	$\mapsto$	$(\vee, \varphi, \psi)$	=	$\varphi\vee\psi$
$\mathit{inj}(\boldsymbol{L})(\Rightarrow)$	:	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, ψ)	$\mapsto$	$(\Rightarrow, \varphi, \psi)$	=	$\varphi\Rightarrow\psi$
$\mathit{inj}(\boldsymbol{L})(\Leftrightarrow)$	:	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, ψ)	$\mapsto$	$(\Leftrightarrow, \varphi, \psi)$	=	$\varphi\Leftrightarrow\psi$
$\mathit{inj}(\boldsymbol{L})(\exists)$	:	$\mathit{case}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, x)	$\mapsto$	$(\exists, \varphi, x)$	=	$(\exists x)\varphi$
$\mathit{inj}(\boldsymbol{L})(\forall)$	:	$\mathit{case}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, x)	$\mapsto$	$(\forall, \varphi, x)$	=	$(\forall x)\varphi$
$\mathit{inj}(\boldsymbol{L})([)$	:	$\mathit{subst}(\mathit{expr}(\boldsymbol{L}))$	$\rightarrow$	$\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$	:	(φ, h)	$\mapsto$	$([, \varphi, h)$	=	$\varphi[h]$

- For convenience of reference we defined some simplified terminology for the component expression injection functions that map into the set of expressions (Table 2). These terms are useful when building expressions bottom-up.

```
(14) (SET.FTN$function atom)
     (= (SET.FTN$source atom) lang$language)
     (= (SET.FTN$target atom) set.ftn$function)
     (= (SET.FTN$composition [atom set.ftn$source]) lang$relation)
     (= (SET.FTN$composition [atom set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (atom ?l)
            ((injection ?l) primitive)))

(15) (SET.FTN$function negation)
```

```
     (= (SET.FTN$source negation) lang$language)
     (= (SET.FTN$target negation) set.ftn$function)
     (= (SET.FTN$composition [negation set.ftn$source]) set)
     (= (SET.FTN$composition [negation set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (negation ?l)
            ((injection ?l) negative)))

(16) (SET.FTN$function conjunction)
     (= (SET.FTN$source conjunction) lang$language)
     (= (SET.FTN$target conjunction) set.ftn$function)
     (= (SET.FTN$composition [conjunction set.ftn$source])
        (SET.FTN$composition
            [(SET.FTN$composition [set set$diagonal]) set.lim.prd2$binary-product]))
     (= (SET.FTN$composition [conjunction set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (conjunction ?l)
            ((injection ?l) conjunctive)))

(17) (SET.FTN$function disjunction)
     (= (SET.FTN$source disjunction) lang$language)
     (= (SET.FTN$target disjunction) set.ftn$function)
     (= (SET.FTN$composition [disjunction set.ftn$source])
        (SET.FTN$composition
            [(SET.FTN$composition [set set$diagonal]) set.lim.prd2$binary-product]))
     (= (SET.FTN$composition [disjunction set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (disjunction ?l)
            ((injection ?l) disjunctive)))

(18) (SET.FTN$function implication)
     (= (SET.FTN$source implication) lang$language)
     (= (SET.FTN$target implication) set.ftn$function)
     (= (SET.FTN$composition [implication set.ftn$source])
        (SET.FTN$composition
            [(SET.FTN$composition [set set$diagonal]) set.lim.prd2$binary-product]))
     (= (SET.FTN$composition [implication set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (implication ?l)
            ((injection ?l) implicative)))

(19) (SET.FTN$function equivalence)
     (= (SET.FTN$source equivalence) lang$language)
     (= (SET.FTN$target equivalence) set.ftn$function)
     (= (SET.FTN$composition [equivalence set.ftn$source])
        (SET.FTN$composition
            [(SET.FTN$composition [set set$diagonal]) set.lim.prd2$binary-product]))
     (= (SET.FTN$composition [equivalence set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (equivalence ?l)
            ((injection ?l) equivalent)))

(20) (SET.FTN$function existential-quantification)
     (= (SET.FTN$source existential-quantification) lang$language)
     (= (SET.FTN$target existential-quantification) set.ftn$function)
     (= (SET.FTN$composition [existential-quantification set.ftn$source])
        (SET.FTN$composition [expression lang$case]))
     (= (SET.FTN$composition [existential-quantification set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (existential-quantification ?l)
            ((lang.expr$injection ?l) existential)))

(21) (SET.FTN$function universal-quantification)
     (= (SET.FTN$source universal-quantification) lang$language)
     (= (SET.FTN$target universal-quantification) set.ftn$function)
     (= (SET.FTN$composition [universal-quantification set.ftn$source])
        (SET.FTN$composition [expression lang$case]))
     (= (SET.FTN$composition [universal-quantification set.ftn$target]) set)
     (forall (?l (lang$language ?l))
         (= (universal-quantification ?l)
               ((lang.expr$injection ?l) universal)))

    (22) (SET.FTN$function substitution)
         (= (SET.FTN$source substitution) lang$language)
         (= (SET.FTN$target substitution) set.ftn$function)
         (= (SET.FTN$composition [substitution set.ftn$source])
            (SET.FTN$composition
                [(SET.FTN$composition [expression lang$substitutable]) rel$extent]))
         (= (SET.FTN$composition [substitution set.ftn$target]) set)
         (forall (?l (lang$language ?l))
             (= (substitution ?l)
                ((lang.expr$injection ?l) substitutive)))
```

Expression Morphisms

`lang.expr.mor`

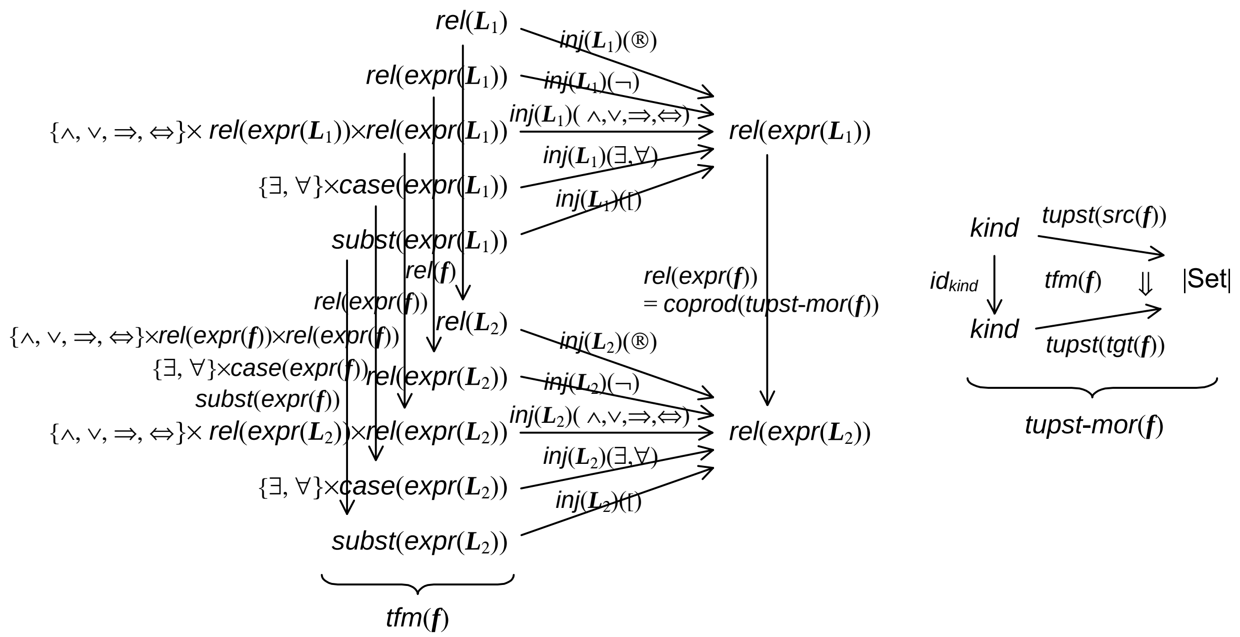


Diagram 9: The expression function as tuple set morphism coproduct

- For any type language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ the expression function

  $rel(expr(\boldsymbol{f})) : rel(expr(\boldsymbol{L}_1)) \rightarrow rel(expr(\boldsymbol{L}_2))$

  is defined as the coproduct $rel(expr(\boldsymbol{f})) = coprd(tupst\text{-}mor(\boldsymbol{f}))$ of the tuple set morphism

  $tupst\text{-}mor(\boldsymbol{f}) = \langle idkind, tfm(\boldsymbol{f})\rangle : tupst(src(\boldsymbol{f})) \rightarrow tupst(tgt(\boldsymbol{f}))$

  from the tuple set of the source language to the tuple set of the target language (Diagram 9). The expression function preserves arity: $expr\text{-}fn(\boldsymbol{f}) \cdot \#_{expr(\boldsymbol{L})} = \#_{\boldsymbol{L}}$.

```
(1) (KIF$function tuple-set-morphism)
     (= (KIF$source tuple-set-morphism) lang.mor$language-morphism)
     (= (KIF$target tuple-set-morphism) set.col.coprd.mor$tuple-set-morphism)
     (forall (?f (lang.mor$language-morphism ?f))
         (and (= (set.col.coprd.mor$source (tuple-set-morphism ?f))
                 (lang.expr$tuple-set (lang.mor$source ?f)))
              (= (set.col.coprd.mor$target (tuple-set-morphism ?f))
                 (lang.expr$tuple-set (lang.mor$target ?f)))
              (= (set.col.coprd.mor$index (tuple-set-morphism ?f))
                 (set.ftn$identity lang.expr$kind))
              (= ((set.col.coprd.mor$transform (tuple-set-morphism ?f)) lang.expr$primitive)
                 (relation ?f))
              (= ((set.col.coprd.mor$transform (tuple-set-morphism ?f)) lang.expr$negative)
                 (function ?f))
              (forall (?k (lang.expr$connective ?k))
                  (= ((set.col.coprd.mor$transform (tuple-set-morphism ?f)) ?k)
                     (set.lim.prd2$binary-product [(function ?f) (function ?f)])))
              (forall (?k (lang.expr$quantifier ?k))
                  (= ((set.col.coprd.mor$transform (tuple-set-morphism ?f)) ?k)
                     (lang.mor$case (expression ?f))))
              (= ((set.col.coprd.mor$transform (tuple-set-morphism ?f)) lang.expr$substitutive)
                 (substitutable (expression ?f)))))

(2) (SET.FTN$function function)
    (= (SET.FTN$source function) lang.mor$language-morphism)
```

```
        (= (SET.FTN$target function) set.ftn$function)
        (= (SET.FTN$composition [function set.ftn$source])
           (SET.FTN$composition [source lang.expr$set]))
        (= (SET.FTN$composition [function set.ftn$target])
           (SET.FTN$composition [target lang$set]))
        (forall (?f (language-morphism ?f))
             (= (function ?f)
                (set.col.coprd.mor$coproduct (tuple-set-morphism ?f))))

    (3) (forall (?f (language-morphism ?f))
              (= (set.ftn$composition [(function ?f) (lang.expr$arity (target ?f))])
                 (lang.expr$arity (source ?f))))
```

$$\begin{tikzcd}
\mathit{var}(\boldsymbol{L}_1) \arrow[r, "\mathit{var}(\boldsymbol{f})"] \arrow[d, "\mathit{refer}(\boldsymbol{L}_1)"'] \arrow[dr, phantom, "\mathit{refer}(\mathit{expr}(\boldsymbol{f}))"] & \mathit{var}(\boldsymbol{L}_2) \arrow[d, "\mathit{refer}(\boldsymbol{L}_2)"] \\
\mathit{ent}(\boldsymbol{L}_1) \arrow[r, "\mathit{ent}(\boldsymbol{f})"'] & \mathit{ent}(\boldsymbol{L}_2)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{rel}(\mathit{expr}(\boldsymbol{L}_1)) \arrow[r, "\mathit{rel}(\mathit{expr}(\boldsymbol{f}))"] \arrow[d, "\mathit{sign}(\mathit{expr}(\boldsymbol{L}_1))"'] \arrow[dr, phantom, "\mathit{sign}(\mathit{expr}(\boldsymbol{f}))"] & \mathit{rel}(\mathit{expr}(\boldsymbol{L}_2)) \arrow[d, "\mathit{sign}(\mathit{expr}(\boldsymbol{L}_2))"] \\
\mathit{sign}(\mathit{type}(\boldsymbol{L}_1)) \arrow[r, "\mathit{sign}(\mathit{type}(\boldsymbol{f}))"'] & \mathit{sign}(\mathit{type}(\boldsymbol{L}_2))
\end{tikzcd}$$

Figure 19: The expression type language morphism

- Any type language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ has an associated *expression* type language morphism $\mathit{expr}(\boldsymbol{f}) : \mathit{expr}(\boldsymbol{L}_1) \rightarrow \mathit{expr}(\boldsymbol{L}_2)$ between expression type languages (Figure 19), which is the identity on entity types and variables. The commutativity of the signature quartet needs an inductive proof.

```
    (4) (SET.FTN$function expression)
        (= (SET.FTN$source expression) lang.mor$language-morphism)
        (= (SET.FTN$target expression) lang.mor$language-morphism)
        (forall (?f (lang.mor$language-morphism ?f))
            (and (= (lang.mor$reference (expression ?f)) (lang.mor$reference ?f))
                 (= (lang.mor$relation (expression ?f)) (function ?f))))
```

- There is a simple type language morphism (Diagram 10)

  $\eta_{\boldsymbol{L}} = \mathit{embed}(\boldsymbol{L}) : \boldsymbol{L} \rightarrow \mathit{expr}(\boldsymbol{L})$

  that *embeds* any type language into its associated expression type language. Both the entity function and the variable function are identities. The relation function $\mathit{rel}(\mathit{embed}(\boldsymbol{L})) : \mathit{rel}(\boldsymbol{L}) \rightarrow \mathit{rel}(\mathit{expr}(\boldsymbol{L}))$ maps any relation type to its corresponding relational expression:

  $(\rho, x_1 : \alpha_1, x_2 : \alpha_2, \ldots x_n : \alpha_n) \mapsto (\rho, x_1 : \alpha_1, x_2 : \alpha_2, \ldots x_n : \alpha_n)$.

  The relation function

  $\mathit{inj}(\boldsymbol{L})(®) : \mathit{rel}(\boldsymbol{L}) \rightarrow \mathit{rel}(\mathit{expr}(\boldsymbol{L}))$

  is the coproduct injection (expression tuple set) on the '`primitive`' expression kind symbol. The commuting Diagram 2 is an exact expression of the definition of the relational expression function. The type language morphism $\eta_{\boldsymbol{L}}$ is the $\boldsymbol{L}$th component of the unit $\eta : \mathit{id}_{\mathsf{Language}} \Rightarrow \mathit{expr}$ for the type language expression monad, a natural transformation from the identity functor on the category of languages to the type language expression endofunctor.

$$\begin{tikzcd}
\boldsymbol{L}_1 \arrow[r, "\eta_{\boldsymbol{L}1}"] \arrow[d, "\boldsymbol{f}"'] & \mathit{expr}(\boldsymbol{L}_1) \arrow[d, "\mathit{expr}(\boldsymbol{f})"] \\
\boldsymbol{L}_2 \arrow[r, "\eta_{\boldsymbol{L}2}"'] & \mathit{expr}(\boldsymbol{L}_2)
\end{tikzcd}$$

Diagram 10: Naturality Square for the Eta Natural Transformation

```
    (5) (SET.FTN$function eta)
        (= (SET.FTN$source eta) lang$language)
        (= (SET.FTN$target eta) lang.mor$simple)
        (forall (?l (lang$language ?l))
            (and (= (lang.mor$source (eta ?l)) ?l)
                 (= (lang.mor$target (eta ?l)) (lang.expr$expression ?l))
                 (= (lang.mor$variable (eta ?l)) (set.ftn$identity (lang$variable ?l)))
```

```
                (= (lang.mor$entity (eta ?l)) (set.ftn$identity (lang$entity ?l)))
                (= (lang.mor$relation (eta ?l))
                   ((lang.expr$arity-tuple ?l) lang.expr$primitive))))
```

○ Any language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ has an associated *free* interpretation $\boldsymbol{f}^{\Delta} = \boldsymbol{f} \cdot \eta_{\boldsymbol{L}2} : \boldsymbol{L}_1 \rightharpoondown \boldsymbol{L}_2$.

```
(6) (SET.FTN$function free)
    (= (SET.FTN$source free) lang.mor$language-morphism)
    (= (SET.FTN$target free) lang.interp$interpretation)
    (forall (?f (lang.mor$language-morphism ?f))
        (and (= (lang.interp$source (free ?f)) (lang.mor$source ?f))
             (= (lang.interp$target (free ?l)) (lang$expression (lang.mor$target ?f)))
             (= (lang.interp$morphism (free ?l))
                (lang.mor$composition [?f (eta (lang.mor$target ?f))]))))
```

○ The multiplication component of the expression monad (see the section on language interpretations) needs careful definition. As one might expect from the developments above, the $\boldsymbol{L}^{\text{th}}$ component of the multiplication natural transformation is a simple type language morphism

$$\mu_{\boldsymbol{L}} = \mathit{collapse}(\boldsymbol{L}) : \mathit{expr}(\mathit{expr}(\boldsymbol{L})) \rightarrow \mathit{expr}(\boldsymbol{L})$$

its entity type and variable functions are identities. So the $\boldsymbol{L}^{\text{th}}$ component reduces to a mystery function $\mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \rightarrow \mathit{rel}(\mathit{expr}(\boldsymbol{L}))$ from the set of relation types of the expressions-of-expressions language $\mathit{expr}(\mathit{expr}(\boldsymbol{L}))$ to the set of relation types of the expression language $\mathit{rel}(\mathit{expr}(\boldsymbol{L}))$. Of course, the latter is just then set of expressions of the type language $\boldsymbol{L}$, but what is the former set? In English, this should be "the expressions of the language whose relation types are the expressions of $\boldsymbol{L}$." Let us

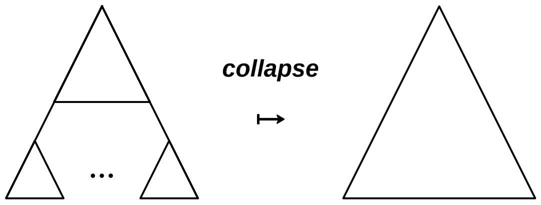


Diagram 11: Expression Collapse

call these "expressions-of-expressions". Thus, the mystery function above takes expressions-of-expressions of $\boldsymbol{L}$ and returns ordinary expressions of $\boldsymbol{L}$. We call this function *collapse*, and its action is graphically represented by Diagram 11, where expressions are visualized as trees with root or outer operator (relation type symbol, connective, quantifier, substitution) at the top and variables at the bottom. The source of the collapse function is the set of relation types of expressions-of-expressions of $\boldsymbol{L}$. This is the set of expressions of the language $\mathit{expr}(\boldsymbol{L})$.

○ We codify this by expressing the typing of the collapse function in logical code.

```
(SET.FTN$function collapse-function)
(= (SET.FTN$source collapse-function) lang$language)
(= (SET.FTN$target collapse-function) set.ftn$function)
(forall (?l (language ?l))
    (and (= (set.ftn$source (collapse-function ?l))
            (lang.expr$set (lang.expr$expression ?l)))
         (= (set.ftn$target (collapse-function ?l))
            (lang.expr$set ?l))))
```

○ Effectively, the collapse function maintains all of the elements of expressions (relation types, connectives, quantifiers, etc.) and all of their positions, but it erases the boundary between the multiple contained expressions and the single containing expression. As usual, we give this a recursive definition, by using a suitable tuple of functions indexed by expression kind, whose component function have as their target the set of ordinary expressions of $\boldsymbol{L}$. Therefore, we need to define these functions, one for each kind of expression. The first step is to figure out their source sets. For any language $\boldsymbol{L}$ these are expressed in boldface in the defining axiom for the term '`lang.expr$tuple-set`'. In our case here, the language $\boldsymbol{L}$ is replaced by the language of $\mathit{expr}(\boldsymbol{L})$ expressions of $\boldsymbol{L}$. There are five cases: primitives, negatives, connectives, quantifiers and substitutives.

primitive:	`(lang$relation (lang.expr$expression ?l))` = **`(lang.expr$set ?l)`**
negative:	**`(lang.expr$set (lang.expr$expression ?l))`**
connective:	**`(set.lim.prd2$binary-product [(lang.expr$set (lang.expr$expression ?l)) (lang.expr$set (lang.expr$expression ?l))])`**
quantifier:	**`(lang$case (lang.expr$expression (lang.expr$expression ?l)))`**
substitutive:	**`(rel$extent (lang$substitutable (lang.expr$expression (lang.expr$expression ?l))))`**

$$\begin{tikzcd}
\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[rrdd, "\mathit{inj}(\mathit{expr}(\boldsymbol{L}))(®)"] \arrow[rrdddddd, "\mathit{id}_{\mathit{rel}(\mathit{expr}(\boldsymbol{L}))}"] & & \\
\mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \arrow[rrd, "\mathit{inj}(\mathit{expr}(\boldsymbol{L}))(\neg)"] \arrow[dddd, "\mathit{collapse}(\boldsymbol{L})"'] & & \\
\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}\times \mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L})))\times\mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \arrow[rr, "\mathit{inj}(\mathit{expr}(\boldsymbol{L}))(\wedge,\vee,\Rightarrow,\Leftrightarrow)"] \arrow[dddd, "\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}\times\mathit{collapse}(\boldsymbol{L})\times\mathit{collapse}(\boldsymbol{L})"'] & & \mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \arrow[dddd, "\mathit{collapse}(\boldsymbol{L})"] \\
\{\exists, \forall\}\times\mathit{case}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \arrow[rru, "\mathit{inj}(\mathit{expr}(\boldsymbol{L}))(\exists,\forall)"] \arrow[dddd, "\{\exists, \forall\}\times\mathit{case}(\mu_{\boldsymbol{L}})"'] & & \\
\mathit{subst}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \arrow[rruu, "\mathit{inj}(\mathit{expr}(\boldsymbol{L}))([)"'] \arrow[dddd, "\mathit{subst}(\mu_{\boldsymbol{L}})"'] & & \\
\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[rrd, "\mathit{inj}(\boldsymbol{L})(\neg)"] & & \\
\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}\times \mathit{rel}(\mathit{expr}(\boldsymbol{L}))\times\mathit{rel}(\mathit{expr}(\boldsymbol{L})) \arrow[rr, "\mathit{inj}(\boldsymbol{L})(\wedge,\vee,\Rightarrow,\Leftrightarrow)"] & & \mathit{rel}(\mathit{expr}(\boldsymbol{L})) \\
\{\exists, \forall\}\times\mathit{case}(\mathit{expr}(\boldsymbol{L})) \arrow[rru, "\mathit{inj}(\boldsymbol{L})(\exists,\forall)"] & & \\
\mathit{subst}(\mathit{expr}(\boldsymbol{L})) \arrow[rruu, "\mathit{inj}(\boldsymbol{L})([)"'] & &
\end{tikzcd}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\mathit{collapse\text{-}tpl}(\boldsymbol{L})}$$

Diagram 12: The collapse function as tuple set coproduct

We next describe the collapsing functions for each kind of expressions (Diagram 12). Think of this as the usual procedure for a recursive call: on the primitive data you take a primitive action and then halt; on the recursive data you dismantle the data, do a recursive call by applying the function to the reduce parts, and then suitably reassemble the result. Thinking of an expression as a tree, taking an operator symbol off the top of an expression is equivalent to using the source for that kind – primitive if it is a relation type symbol, negative for negation symbol, etc. The recursive call applies the collapse function to one or two parts. Replacing the operator symbol back on top of the results is equivalent to the coproduct injection of that kind.

- The primitive function is just the identity on expressions. A primitive expression-of-expressions is just an expression with the single containing expression empty, or better, an empty singleton shell – there being only one contained expression.
- The negative function is the composition of the collapse function followed by the coproduct injection of the resulting negative expression – effectively we take off the negation symbol from the top, we apply the collapse function to the rest getting a single ordinary expression, and then we replace the negation symbol at the top.

- The connective function is the composition of the binary power of the collapse function getting two ordinary expressions, followed by the coproduct injection of the resulting connective expression – effectively we take off the connective symbol from the top, we apply the collapse function to the two remaining expressions-of-expressions, and then we replace the connective symbol at the top of the resulting pair.
- etc.

The collapse function $\mathit{collapse}(\boldsymbol{L}) : \mathit{rel}(\mathit{expr}(\mathit{expr}(\boldsymbol{L}))) \rightarrow \mathit{rel}(\mathit{expr}(\boldsymbol{L}))$ for a type language $\boldsymbol{L}$ is defined as the expression language expression cotupling of the collapse tuple of functions. As evident in Diagram 3, except for the primitive component function, the collapse function is almost definable as a coproduct of a tuple set morphism. The collapse function preserves arity:

$$\mathit{collapse}(\boldsymbol{L}) \cdot \#_{\mathit{expr}(\boldsymbol{L})} = \#_{\mathit{expr}(\mathit{expr}(\boldsymbol{L}))}.$$

```
(7) (KIF$function collapse-tuple)
    (= (KIF$source collapse-tuple) lang$language)
    (= (KIF$target collapse-tuple) SET.FTN$function)
    (forall (?l (lang$language ?l))
        (and (= (SET.FTN$source (collapse-tuple ?l)) lang.expr$kind)
             (= (SET.FTN$target (collapse-tuple ?l)) set.ftn$function)
             (= (SET.FTN$composition [(collapse-tuple ?l) set.ftn$target])
                ((SET.FTN$constant [lang.expr$kind set$set]) (lang.expr$set ?l)))
             (= (set.ftn$source ((collapse-tuple ?l) lang.expr$primitive))
                (lang.expr$set ?l))
             (= ((collapse-tuple ?l) lang.expr$primitive)
                (set.ftn$identity (lang.expr$set ?l)))
             (= (set.ftn$source ((collapse-tuple ?l) lang.expr$negative))
                (lang.expr$set (lang.expr$expression ?l)))
             (= ((collapse-tuple ?l) lang.expr$negative)
                (set.ftn$composition
                    [(collapse ?l)
                     ((set.col.coprd$injection (lang.expr$tuple-set ?l))
                         lang.expr$negative)]))
             (forall (?k (lang.expr$connective ?k))
                 (and (= (set.ftn$source ((collapse-tuple ?l) ?k))
                         (set.lim.prd2$binary-product
                             [(lang.expr$set (lang.expr$expression ?l))
                              (lang.expr$set (lang.expr$expression ?l))]))
                      (= ((collapse-tuple ?l) ?k)
                         (set.ftn$composition
                             [(set.lim.prd2$binary-product
                                  [(collapse ?l) (collapse ?l)])
                              ((set.col.coprd$injection (lang.expr$tuple-set ?l)) ?k)]))))
             (forall (?k (lang.expr$quantifier ?k))
                 (and (= (set.ftn$source ((collapse-tuple ?l) ?k))
                         (lang$case
                             (lang.expr$expression (lang.expr$expression ?l))))
                      (= ((collapse-tuple ?l) ?k)
                         (set.ftn$composition
                             [(lang.mor$case (collapse ?l))
                              ((set.col.coprd$injection (lang.expr$tuple-set ?l)) ?k)]))))
             (= (set.ftn$source ((collapse-tuple ?l) lang.expr$substitutive))
                (rel$extent (lang$substitutable
                    (lang.expr$expression (lang.expr$expression ?l)))))
             (= ((collapse-tuple ?l) lang.expr$substitutive)
                (set.ftn$composition
                    [(rel$extent (lang.mor$substitutable (collapse ?l)))
                     ((set.col.coprd$injection (lang.expr$tuple-set ?l))
                         lang.expr$substitutive)]))))

(8) (SET.FTN$function collapse)
    (= (SET.FTN$source collapse) lang$language)
    (= (SET.FTN$target collapse) set.ftn$function)
    (= (SET.FTN$composition [collapse set.ftn$source])
       (SET.FTN$composition [lang.expr$expression lang.expr$set]))
    (= (SET.FTN$composition [collapse set.ftn$target]) lang.expr$set)
    (forall (?l (lang$language ?l))
        (= (collapse)
            ((set.col.coprd$cotupling (lang.expr$tuple-set (lang.expr$expression ?l)))
                 (collapse-tuple ?l))))

(9) (forall (?l (lang$language ?l))
         (= (set.ftn$composition [(collapse ?l) (lang.expr$arity ?l)])
            (lang.expr$arity (lang.expr$expression ?l))))
```

- There is a simple type language morphism

  $$\mu_{L} = collapse(L) : expr(expr(L)) \to expr(L)$$

  that *collapses* the expression of expression language of any type language into its expression type language. Both the entity function and the variable function are identities. The relation function

  $$rel(embed(L)) : rel(expr(expr(L))) \to rel(expr(L))$$

  is the collapse function. The commuting Diagram 13 needs to be checked. The type language morphism μ_L is the L^{th} component of the multiplication $\mu : expr \bullet expr \Rightarrow expr$ for the type language expression monad, a natural transformation from the square of the type language expression endofunctor on the category of languages to itself.

$$\begin{tikzcd}
expr(expr(L_1)) \arrow[r, "\mu_{L1} = collapse(L_1)"] \arrow[d, "expr(expr(f))"'] & expr(L_1) \arrow[d, "expr(f)"] \\
expr(expr(L_2)) \arrow[r, "\mu_{L2} = collapse(L_2)"'] & expr(L_2)
\end{tikzcd}$$

Diagram 13: Naturality Square of the Mu Natural Transformation

```
(10) (SET.FTN$function mu)
     (= (SET.FTN$source mu) lang$language)
     (= (SET.FTN$target mu) lang.mor$simple)
     (forall (?l (lang$language ?l))
         (and (= (lang.mor$source (mu ?l))
                 (lang.expr$expression (lang.expr$expression ?l)))
              (= (lang.mor$target (mu ?l)) (lang.expr$expression ?l))
              (= (lang.mor$variable (mu ?l)) (set.ftn$identity (lang$variable ?l)))
              (= (lang.mor$entity (mu ?l)) (set.ftn$identity (lang$entity ?l)))
              (= (lang.mor$relation (mu ?l)) (collapse ?l))))
```

Type Language Interpretations

`lang.interp`

We are interested in the Kliesli category – see the monad namespace in the IFF Category Theory Ontology – of the expression monad, and especially interested in the morphisms of this category. These morphisms are called interpretations.

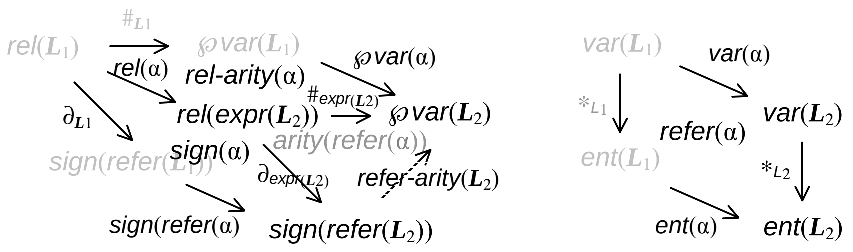


Figure 20: Type Language Interpretation

- A first order type language *interpretation* $\alpha : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_2$ from type language $\boldsymbol{L}_1$ to type language $\boldsymbol{L}_2$ (Figure 20) is a first order type language morphism $mor(\alpha) = \langle var(\alpha), ent(\alpha), rel(\alpha)\rangle : \boldsymbol{L}_1 \rightarrow expr(\boldsymbol{L}_2)$ from type language $\boldsymbol{L}_1$ to the expression type language $expr(\boldsymbol{L}_2)$ of $\boldsymbol{L}_2$. Hence, it is a two dimensional construction consisting of a reference quartet $refer(\alpha)$ and a signature quartet $sign(\alpha)$, where the signature of the reference quartet is the vertical target of the signature quartet

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

  These components resolve into three functions:
  - a function of *variables* $var(\alpha) : var(\boldsymbol{L}_1) \rightarrow var(\boldsymbol{L}_2)$
  - a function of *entity* types $ent(\alpha) : ent(\boldsymbol{L}_1) \rightarrow ent(\boldsymbol{L}_2)$
  - a function of *relation* types $rel(\alpha) : rel(\boldsymbol{L}_1) \rightarrow rel(expr(\boldsymbol{L}_2))$

  which preserve reference, arity and signature. The variable function renames the variables – it is a bijection. The entity function resorts the (entity) types. But clearly, the relation function is of most interest. This maps relation types of $\boldsymbol{L}_1$ to expressions of $\boldsymbol{L}_2$, thus providing a "definition" or "interpretation" for the relation types of $\boldsymbol{L}_1$ in terms of the expressions of $\boldsymbol{L}_2$.

```
(1) (SET$class interpretation)

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) interpretation)
    (= (SET.FTN$target source) lang$language)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) interpretation)
    (= (SET.FTN$target target) lang$language)

(4) (SET.FTN$function morphism)
    (= (SET.FTN$source morphism) interpretation)
    (= (SET.FTN$target morphism) lang.mor$language-morphism)
    (= (SET.FTN$composition [morphism lang.mor$source]) source)
    (= (SET.FTN$composition [morphism lang.mor$target])
       (SET.FTN$composition [target lang.expr$expression]))
```

- For any type language interpretation $\boldsymbol{f} : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_2$ there is an associated *extension* type language morphism $\boldsymbol{f}^{\#} = expr(mor(\boldsymbol{t})) \cdot \mu_{L2} : expr(\boldsymbol{L}_1) \rightarrow expr(\boldsymbol{L}_2)$.

```
(5) (SET.FTN$function extension)
    (= (SET.FTN$source extension) interpretation)
    (= (SET.FTN$target extension) lang.mor$language-morphism)
    (= (SET.FTN$composition [extension lang.mor$source])
       (SET.FTN$composition [source lang.expr$expression]))
    (= (SET.FTN$composition [extension lang.mor$target])
       (SET.FTN$composition [target lang.expr$expression]))
```

```
(forall (?t (interpretation ?t))
    (= (extension ?t)
       (lang.mor$composition
           [(lang.expr.mor$expression (morphism ?t))
            (lang.expr.mor$mu (target ?t))])))
```

○ The extension of the free interpretation of a language morphism $\boldsymbol{f} : \boldsymbol{L}_1 \rightarrow \boldsymbol{L}_2$ is the image of $\boldsymbol{f}$ under the expression function $expr(\boldsymbol{f}) : expr(\boldsymbol{L}_1) \rightarrow expr(\boldsymbol{L}_2)$.

```
(forall (?f (lang.mor$language-morphism ?f))
    (= (extension (lang.expr.mor$free ?f)) (lang.expr.mor$expression ?f)))
```

○ Type language interpretations can be composed. Although the composition is defined in terms of the type language morphism composition, it is different from type language morphism composition. The definition follows the definition of morphisms in the Kliesli category of the expression monad. Two type language interpretations are *composable* when the target of the first is equal to the source of the second. The *composition* $\alpha \circ \beta = mor(\alpha) \cdot mor(\beta)^{\#} : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_3$ of two composable type language interpretations $\alpha : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_2$ and $\beta : \boldsymbol{L}_2 \rightharpoonup \boldsymbol{L}_3$ is defined in terms of the composition of type language morphisms.

```
(6) (SET.LIM.PBK$opspan composable-opspan)
    (= (class1 composable-opspan) interpretation)
    (= (class2 composable-opspan) interpretation)
    (= (opvertex composable-opspan) lang$language)
    (= (first composable-opspan) target)
    (= (second composable-opspan) source)

(7) (REL$relation composable)
    (= (REL$class1 composable) interpretation)
    (= (REL$class2 composable) interpretation)
    (= (REL$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) interpretation)
    (forall (?t1 (interpretation ?t1) ?t2 (interpretation ?t2) (composable ?t1 ?t2))
        (and (= (source (composition [?t1 ?t2])) (source ?t1))
             (= (target (composition [?t1 ?t2])) (target ?t2))
             (= (morphism (composition [?t1 ?t2]))
                (lang.mor$composition [(morphism ?t1) (extension ?t2)]))))
```

o Composition satisfies the usual *associative law*.

```
(forall (?t1 (interpretation ?t1)
         ?t2 (interpretation ?t2)
         ?t3 (interpretation ?t3)
         (composable ?t1 ?t2) (composable ?t2 ?t3))
    (= (composition [?t1 (composition [?t2 ?t3])])
       (composition [(composition [?t1 ?t2]) ?t3])))
```

o For any type language $\boldsymbol{L}$, there is an *identity* type language interpretation, whose underlying type language morphism is the eta (embedding) morphism.

```
(9) (SET.FTN$function identity)
    (= (SET.FTN$source identity) lang$language)
    (= (SET.FTN$target identity) interpretation)
    (forall (?l (lang$language ?l))
        (and (= (source (identity ?l)) ?l)
             (= (target (identity ?l)) ?l)
             (= (morphism (identity ?l)) (lang.expr.mor$eta ?l))))
```

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

```
(forall (?t (interpretation ?t))
    (and (= (composition [(identity (source ?t)) ?t]) ?t)
         (= (composition [?t (identity (target ?t))]) ?t)))
```

○ A *special* type language interpretation is a type language interpretation $\boldsymbol{f} : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_2$ that is bijective on variables; that is, an interpretation whose underlying type language morphism is special. Since the identity interpretations are special and special interpretations are closed under composition, there is a subcategory of special interpretations.

```
(10) (SET$class special)
     (SET$subclass special interpretation)
     (forall (?t (interpretation ?t))
         (<=> (special ?t) (lang.mor$special (morphism ?t))))

    (forall (?l (lang$language ?l))
        (special (identity ?l)))

    (forall (?t1 (special ?t1) ?t2 (special ?t2) (composable ?t1 ?t2))
        (special (composition [?t1 ?t2])))
```

$$\begin{tikzcd} & \mathit{expr}(\boldsymbol{L}_2) \\ \boldsymbol{L}_1 \arrow[ur, "\mathit{mor}(\alpha)"] \arrow[r, "\alpha"'] & \boldsymbol{L}_2 \arrow[u, "\eta_{\boldsymbol{L}2}"'] \end{tikzcd} \qquad \begin{tikzcd} & \mathit{mod}(\mathit{expr}(\boldsymbol{L}_2)) \arrow[dl, "\mathit{mod}(\mathit{mor}(\alpha))"'] \arrow[d, shift right, "\mathit{mod}(\eta_{\boldsymbol{L}2})"'] \\ \mathit{mod}(\boldsymbol{L}_1) & \mathit{mod}(\boldsymbol{L}_2) \arrow[l, "\mathit{mod}(\alpha)"] \arrow[u, shift right, "\mathit{mod\text{-}expr}(\boldsymbol{L}_2)"'] \end{tikzcd}$$

Figure 21: Interpretation and related fiber functions

○ The model fiber (inverse image) operator for special type interpretations is defined in terms of the model fiber operator for special type language morphisms. In particular (Figure 21), the model (inverse image) class function for a special type interpretation $\alpha : \boldsymbol{L}_1 \rightharpoonup \boldsymbol{L}_2$

$$\alpha^{-1} = \mathit{mod}(\alpha) = \mathit{mod\text{-}expr}(\boldsymbol{L}_2) \cdot \mathit{mod}(\mathit{mor}(\alpha)) : \mathit{mod}(\boldsymbol{L}_2) \rightarrow \mathit{mod}(\boldsymbol{L}_1)$$

is defined as the class function composition of the model expression of its target language (maps models to their expression models) with the model of its underlying type language morphism. In particular,

$$\alpha^{-1}(\boldsymbol{A}) = \mathit{mor}(\alpha)^{-1}(\mathit{expr}(\boldsymbol{A}))$$

```
(11) (KIF$function model)
     (= (KIF$source model) special)
     (= (KIF$target model) SET.FTN$function)
     (forall (?t (special ?t))
         (and (= (SET.FTN$source (model ?t)) (lang$model (target ?t)))
              (= (SET.FTN$target (model ?t)) (lang$model (source ?t)))
              (= (model ?t)
                 (SET.FTN$composition
                     [(lang$model-expression (target ?t))
                      (lang.mor$model (morphism ?t))]))))
```

o Define a *linkage* model morphism $\mathit{link}(\alpha)(\boldsymbol{A}) : \alpha^{-1}(\boldsymbol{A}) \rightarrow \mathit{expr}(\boldsymbol{A})$ as follows:

$$\mathit{link}(\alpha)(\boldsymbol{A}) = \mathit{link}(\mathit{mor}(\alpha))(\mathit{expr}(\boldsymbol{A})) : \mathit{mor}(\alpha)^{-1}(\mathit{expr}(\boldsymbol{A})) \rightarrow \mathit{expr}(\boldsymbol{A}).$$

```
(12) (KIF$function link)
     (= (KIF$source link) special)
     (= (KIF$target link) SET.FTN$function)
     (forall (?t (special ?t))
         (and (= (SET.FTN$source (link ?t)) (lang$model (target ?t)))
              (= (SET.FTN$target (link ?t)) mod.mor$model-morphism)
              (= (link ?t)
                 (SET.FTN$composition
                     [(lang$model-inclusion (target ?f))
                      (SET.FTN$composition
                          [mod$expression (lang.mor$link (morphism ?t))])]))))
```

$$\begin{tikzcd}
& expr(\boldsymbol{L}) \\
\boldsymbol{L} \arrow[ur, "\eta_{\boldsymbol{L}}"] \arrow[r, "id_{\boldsymbol{L}}"'] & \boldsymbol{L} \arrow[u, "\eta_{\boldsymbol{L}}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
& & mod(expr(\boldsymbol{L})) \arrow[dll, "mod(\eta_{\boldsymbol{L}})"'] \arrow[d, shift right, "mod(\eta_{\boldsymbol{L}})"'] \\
mod(\boldsymbol{L}) & & mod(\boldsymbol{L}) \arrow[ll, "mod(id_{\boldsymbol{L}})"] \arrow[u, shift right, "mod\text{-}expr(\boldsymbol{L})"']
\end{tikzcd}$$

Diagram 14: Model operator of interpretation identity

- For any type language $\boldsymbol{L}$ the model of the interpretation identity at $\boldsymbol{L}$ is the identity function on the model fiber of $\boldsymbol{L}$ (Diagram 14):

  $id_{\boldsymbol{L}}{}^{-1} = mod(id_{\boldsymbol{L}}) = mod\text{-}expr(\boldsymbol{L}) \cdot mod(\eta_{\boldsymbol{L}}) = id_{fib(\boldsymbol{L})}$.

```
(13) (forall (?l (lang$language ?l))
         (= (SET.FTN$composition
                [(lang$model-expression ?l) (lang.mor$model (lang.mor$eta ?l))])
            (SET.FTN$identity (lang$model ?l))))
```

$$\begin{tikzcd}
& & & & expr(expr(\boldsymbol{L}_3)) \arrow[rr, "\mu_{\boldsymbol{L}3}"] & & expr(\boldsymbol{L}_3) \\
& & expr(\boldsymbol{L}_2) \arrow[urr, "expr(mor(\beta))"] & & expr(\boldsymbol{L}_3) \arrow[u, shift left, "expr(\eta_{\boldsymbol{L}3})"] \arrow[u, shift right, "\eta_{expr(\boldsymbol{L}3)}"'] \arrow[urr, "id_{expr(\boldsymbol{L}3)}"'] & & \\
\boldsymbol{L}_1 \arrow[urr, "mor(\alpha)"] \arrow[rr, "\alpha"'] & & \boldsymbol{L}_2 \arrow[u, "\eta_{\boldsymbol{L}2}"] \arrow[urr, "mor(\beta)"] \arrow[rr, "\beta"'] & & \boldsymbol{L}_3 \arrow[u, "\eta_{\boldsymbol{L}3}"'] & &
\end{tikzcd}$$

$$\begin{tikzcd}
& & & & mod(expr(expr(\boldsymbol{L}_3))) \arrow[dll, "mod(expr(mor(\beta)))"'] \arrow[d, shift right, "mod(\eta_{expr(\boldsymbol{L}3)})"'] & & mod(expr(\boldsymbol{L}_3)) \arrow[ll, "mod(\mu_{\boldsymbol{L}3})"'] \arrow[dll, "id_{mod(expr(\boldsymbol{L}3))}"] \arrow[d, shift right] \\
& & mod(expr(\boldsymbol{L}_2)) \arrow[dll, "mod(mor(\alpha))"'] \arrow[d, shift right, "mod(\eta_{\boldsymbol{L}2})"'] & & mod(expr(\boldsymbol{L}_3)) \arrow[u, shift right, "mod\text{-}expr(expr(\boldsymbol{L}_3))"'] \arrow[dll, "mod(mor(\beta))"'] \arrow[d, shift right, "mod(\eta_{\boldsymbol{L}3})"'] & & mod(\boldsymbol{L}_3) \arrow[u, shift right, "mod\text{-}expr(\boldsymbol{L}_3)"'] \arrow[dll, "id_{mod(\boldsymbol{L}3)}"] \\
mod(\boldsymbol{L}_1) & & mod(\boldsymbol{L}_2) \arrow[ll, "mod(\alpha)"] \arrow[u, shift right, "mod\text{-}expr(\boldsymbol{L}_2)"'] & & mod(\boldsymbol{L}_3) \arrow[ll, "mod(\beta)"] \arrow[u, shift right, "mod\text{-}expr(\boldsymbol{L}_3)"'] & &
\end{tikzcd}$$

Diagram 15: Model operator of interpretation composition

- For any two composable interpretations $\alpha : \boldsymbol{L}_1 \rightharpoondown \boldsymbol{L}_2$ and $\beta : \boldsymbol{L}_2 \rightharpoondown \boldsymbol{L}_3$ the model of the composition of interpretations $\alpha \circ \beta = mor(\alpha) \cdot mor(\beta)^{\#} : \boldsymbol{L}_1 \rightharpoondown \boldsymbol{L}_3$ is the class function composition of the fibers of the component interpretations. This is established with the following identities (Diagram 15).

  $embed(\boldsymbol{L}_3) \cdot mod(\mu_{\boldsymbol{L}3}) \cdot mod(expr(mor(\beta))) \cdot mod(mor(\alpha))$

  $= embed(\boldsymbol{L}_3) \cdot embed(expr(\boldsymbol{L}_3)) \cdot mod(expr(mor(\beta))) \cdot mod(mor(\alpha))$

  $= embed(\boldsymbol{L}_3) \cdot mod(mor(\beta)) \cdot embed(\boldsymbol{L}_2) \cdot mod(mor(\alpha))$

```
(14) (forall (?t1 (interpretation ?t1)
              ?t2 (interpretation ?t2) (composable ?t1 ?t2))
         (= (SET.FTN$composition
                [(SET.FTN$composition
                     [(SET.FTN$composition
                          [(lang$model-expression (target ?t2)
                           (lang.mor$model (lang.mor$mu ?l))])
                      (lang.mor$model (lang.mor$expression (morphism ?t2)))])
                 (lang.mor$model (morphism ?t2))])
            (SET.FTN$composition
                [(SET.FTN$composition
                     [(lang$model-expression (target ?t1)
                      (lang.mor$model (morphism ?t1))])
                 (SET.FTN$composition
                     [(lang$model-expression (target ?t2)
                      (lang.mor$model (morphism ?t2))])]))))
```

- We have shown that the model operator is functorial on interpretations.