The Namespace of Terms

LANGUAGES ... 2
Membership as Coproduct ... 4
TERMS ... 5
Constructors ... 6
Booleans ... 7
Composite Constructors ... 8
Selectors ... 9
Term Tuples ... *10*
Constructors ... 11
Booleans ... 13
Composite Constructors ... 14
Selectors ... 15
Category-theoretic Operations ... 17

This section offers an axiomatization for terms that is compatible with the IFF Model Theory Ontology. Terms will be merged into the IFF Model Theory Ontology, by replacing substitutions with term tuples. Term tuples subsume the previous notion of substitution. In addition, suitable modifications need to be made to the IFF Model Theory Ontology by replacing the effect of substitutions in language morphisms, language expressions and language colimits, with the effect of term tuples. Also, the traditional functorial semantics for terms needs to be seamlessly merged with the classification-oriented notion of IFF models.

Table 1 lists the terminology for function types and terms.

Table 1: Terminology for terms

	Class	Function
lang	language	variable entity **function** constant reference arity return
		renaming domain codomain
lang .term		**term** arity return
		element substitution-opspan substitutable substitution
		is-element = elementary is-composite is-atom = is-function
		term-substitution-opspan term-substitutable term-substitution
		function tuple resolution
lang .term.tpl		**tuple** index arity
		empty insertible-opspan insertible insertion singleton renamable renaming
		is-empty is-nonempty is-composite is-atom = is-singleton
		tuplable tupling = tuple-insertion
		index-member index-injection index-indication index-projection selection remainder = rest
		composable-opspan composable composition identity

Languages

`lang`

- A 1st-order *type language* $\boldsymbol{L} = \langle \mathit{var}(\boldsymbol{L}), \mathit{ent}(\boldsymbol{L}), \mathit{refer}(\boldsymbol{L}), \mathit{ftn}(\boldsymbol{L}), \mathit{arity}(\boldsymbol{L}), \mathit{rtn}(\boldsymbol{L}) \rangle$ (Figure 1) consists of
  - a set of *variables* $\mathit{var}(\boldsymbol{L})$,
  - a set of *entity types* $\mathit{ent}(\boldsymbol{L})$,
  - a *reference* or sort function
    $*_{\boldsymbol{L}} = \mathit{refer}(\boldsymbol{L}) : \mathit{var}(\boldsymbol{L}) \to \mathit{ent}(\boldsymbol{L})$,
  - a set of *function types* $\mathit{ftn}(\boldsymbol{L})$,
  - an *arity* function
    $\#_{\boldsymbol{L}} = \mathit{arity}(\boldsymbol{L}) : \mathit{ftn}(\boldsymbol{L}) \to \wp\mathit{var}(\boldsymbol{L})$, and
  - a *return* function
    $\text{«}_{\boldsymbol{L}} = \mathit{rtn}(\boldsymbol{L}) : \mathit{ftn}(\boldsymbol{L}) \to \mathit{var}(\boldsymbol{L})$.

$$\begin{tikzcd}
\mathit{var}(\boldsymbol{L}) \arrow[d, "*_{\boldsymbol{L}}"'] & \mathit{ftn}(\boldsymbol{L}) \arrow[l, "\text{«}_{\boldsymbol{L}}"'] \arrow[r, "\#_{\boldsymbol{L}}"] \arrow[dl, "\tau_{\boldsymbol{L}}"] \arrow[dr, "\partial_{\boldsymbol{L}}"'] & \wp\mathit{var}(\boldsymbol{L}) \\
\mathit{ent}(\boldsymbol{L}) & & \mathit{sign}(*_{\boldsymbol{L}}) \arrow[u, "\mathit{refer\text{-}arity}(\boldsymbol{L})"']
\end{tikzcd}$$

Figure 1: Term Language

Just as for relation types, the arity function is equivalent to a *signature* function

$\partial_{\boldsymbol{L}} = \mathit{sign}(\boldsymbol{L}) : \mathit{ftn}(\boldsymbol{L}) \to \mathit{sign}(*_{\boldsymbol{L}})$.

In addition, the return function implies a *type* function

$\tau_{\boldsymbol{L}} = \mathit{return}(\boldsymbol{L}) : \mathit{ftn}(\boldsymbol{L}) \to \mathit{ent}(\boldsymbol{L})$.

The signature and type functions will be ignored at this time. We represent a function type $f \in \mathit{ftn}(\boldsymbol{L})$ having $\mathit{arity}(\boldsymbol{L})(f) = A$ and $\mathit{rtn}(\boldsymbol{L})(f) = a$ with the linear notation

$f(A) : a$.

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

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

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

(4) (SET.FTN$function function)
    (= (SET.FTN$source function) language)
    (= (SET.FTN$target function) set$set)

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

(6) (SET.FTN$function arity)
    (= (SET.FTN$source arity) language)
    (= (SET.FTN$target arity) set.ftn$function)
    (= (SET.FTN$composition [arity set.ftn$source]) function)
    (= (SET.FTN$composition [arity set.ftn$target])
       (SET.FTN$composition [variable set$power]))

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

- A *constant* is a function type whose arity is empty. In particular, the arity fiber at the empty set is called the set of constants of $\boldsymbol{L}$

  $\mathit{const}(\boldsymbol{L}) = \mathit{arity\text{-}fiber}(\boldsymbol{L})(\varnothing)$.

```
(8) (SET.FTN$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 [function set$power]))
    (forall (?l (language ?l))
        (= (arity-fiber ?l)
           (set.ftn$fiber (arity ?l))))

(9) (KIF$function constant)
    (= (SET.FTN$source constant) lang$language)
    (= (SET.FTN$target constant) set$set)
    (forall (?l (language ?l))
        (= (constant ?l)
           ((arity-fiber ?l) set.col$initial)))
```

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

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

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

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

Membership as Coproduct

`set.mbr`

We assume that the following code appears in the IFF Lower Core Ontology in the ‘`set.mbr`’ sub-namespace.

- The extent of the membership relation on subsets of an underlying set A is the coproduct of the power identity function, regarded as a coproduct *arity*

  $\#_G = id_{\wp A} : \wp A \to \wp A.$

```
(1) (SET.FTN$function member-arity)
    (= (SET.FTN$source member-arity) set$set)
    (= (SET.FTN$target member-arity) set.col.art$arity)
    (= (SET.FTN$composition [member-arity set.col.art$index]) set$power)
    (= (SET.FTN$composition [member-arity set.col.art$base]) (SET.FTN$identity set$set))
    (= (SET.FTN$composition [member-arity set.col.art$function])
       (SET.FTN$composition [set$power set.ftn$identity]))
```

- Any set A has a set of membership *cases*

  $mbr(A) = \sum id_{\wp A}$
  $= \sum_{X \in \wp A} id_{\wp A}(X)$
  $= \{(X, x) \mid X \in \wp A, x \in X\},$

  that is the coproduct of its power identity function as arity.

  For any set A and any subset $X \in \wp A$ there is a member *injection* function:

  $inj(A)(X) : X \to mbr(A)$

$$\begin{tikzcd} X \arrow[r, "inj(A)(X)"] \arrow[dr, phantom, "\subseteq"] & mbr(A) \arrow[d, "proj(A)"] \\ & A \end{tikzcd}$$

Diagram 1: Coproduct

  defined by $inj(A)(X)(x) = (X, x)$ for all subsets $X \in \wp A$ and all elements $x \in X$. Obviously, the injections are injective. They commute (Diagram 1) with projection and inclusion.

```
(2) (SET.FTN$function member)
    (= (SET.FTN$source member) set$set)
    (= (SET.FTN$target member) set$set)
    (= member (SET.FTN$composition [member-arity set.col.art$coproduct]))

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

- Any set A defines *indication* and *projection* functions based on its arity (Figure 2):

  $indic(A) : mbr(A) \to \wp A,$

  $proj(A) : mbr(A) \to A.$

  These are defined by

  $indic(A)((X, x)) = X$ and $proj(A)((X, x)) = x$

  for all subsets $X \in \wp A$ and all elements $x \in X$.

$$\begin{tikzcd} & mbr(A) \arrow[dl, "indic(A)"'] \arrow[dr, "proj(A)"] & \\ \wp A \arrow[rr, "\#_A"] & & A \quad \wp A \end{tikzcd}$$

Figure 2: Indication and projection

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

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

Terms

`lang.term`

This section and its term tuple subsection axiomatize the terms for a type language $\boldsymbol{L}$. Terms and term tuples are corecursively defined. This specification replaces the traditional recursive tree-forest set equations

$\boldsymbol{Tree}(\boldsymbol{A}) \cong 1 + \boldsymbol{A} \times \boldsymbol{Forest}(\boldsymbol{A})$,

$\boldsymbol{Forest}(\boldsymbol{A}) \cong \mathit{stack}(\boldsymbol{Tree}(\boldsymbol{A}))$

with the recursive term-tuple set equations

$\boldsymbol{Term}(\boldsymbol{L}) \cong \mathit{ext}(\mathit{mbr}(\mathit{var}(\boldsymbol{L}))) + \mathit{ftn}(\boldsymbol{L}) \otimes \boldsymbol{Tuple}(\boldsymbol{L})$,

$\boldsymbol{Tuple}(\boldsymbol{L}) \cong \mathit{tuple}(\boldsymbol{Term}(\boldsymbol{L}))$,

where '$\otimes$', which denotes the substitution operator, will be generalized in the section on term tuples to composition in a Kleisli-like term category. Categorically, terms and their tuples is the fixpoint solution for an ω-continuous endofunctor on the category $\mathsf{Set}\times\mathsf{Set}$.

○ For any type language $\boldsymbol{L}$, we give a pointwise recursive definition for the set of $\boldsymbol{L}$-terms $\mathit{term}(\boldsymbol{L})$. Like any other recursively defined datatype, beyond the *attribute* functions that describe terms, there will be *basic constructor* functions, *composite constructor* functions, *Boolean tests* for the kinds of terms built by the constructors, and *selector* functions that are generalized inverses to the basic constructors.

A term $\alpha \in \mathit{term}(\boldsymbol{L})$ will have an *arity* $\mathit{arity}(\boldsymbol{L})(\alpha)$ and a *return* $\mathit{rtn}(\boldsymbol{L})(\alpha)$, both denoting entity types via the reference function of the language $\boldsymbol{L}$. Hence, there is

– an *arity* function $\mathit{arity}(\boldsymbol{L}) : \mathit{term}(\boldsymbol{L}) \rightarrow \wp\mathit{var}(\boldsymbol{L})$, and

– an *return* function $\mathit{rtn}(\boldsymbol{L}) : \mathit{term}(\boldsymbol{L}) \rightarrow \mathit{var}(\boldsymbol{L})$.

We picture a term α as in Figure 3, and use the linear notation

$\alpha : A \rhd a$

to denote the term α with $\mathit{arity}(\boldsymbol{L})(\alpha) = A$ and $\mathit{rtn}(\boldsymbol{L})(\alpha) = a$.

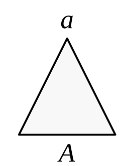

Figure 3: A term α

```
(1) (SET.FTN$function term)
    (= (SET.FTN$source term) lang$language)
    (= (SET.FTN$target term) set$set)

(2) (SET.FTN$function arity)
    (= (SET.FTN$source arity) lang$language)
    (= (SET.FTN$target arity) set.ftn$function)
    (= (SET.FTN$composition [arity set.ftn$source]) term)
    (= (SET.FTN$composition [arity set.ftn$target])
       (SET.FTN$composition [lang$variable set$power]))

(3) (SET.FTN$function return)
    (= (SET.FTN$source return) lang$language)
    (= (SET.FTN$target type) set.ftn$function)
    (= (SET.FTN$composition [return set.ftn$source]) term)
    (= (SET.FTN$composition [return set.ftn$target]) lang$variable)
```

Constructors

- Variables are terms. A pair (A, a) consisting of a variable $a \in \mathit{var}(\boldsymbol{L})$ that is indexed by a subset of variables $A \subseteq \mathit{var}(\boldsymbol{L})$ in which it is a member $a \in A$ (this membership relation extent is the coproduct of the power set identity function regarded as a coproduct arity) defines an *element* or *variable* term

  $\in_{A, a} : A \triangleright a$

  with $\mathit{arity}(\boldsymbol{L})(\in_{A, a}) = A$ and $\mathit{rtn}(\boldsymbol{L})(\in_{A, a}) = a$. A term is *elementary* when it is constructed from an indexed variable.

- Term tuples can be substituted into function symbols. A function $f \in \mathit{ftn}(\boldsymbol{L})$ and a term tuple $\alpha \in \mathit{term}^*(\boldsymbol{L})$ are a *substitutable* pair when the function arity of f is the same as the index of the term tuple α : $\mathit{arity}(\boldsymbol{L})(f) = \mathit{index}(\boldsymbol{L})(\alpha)$. For any substitutable pair

  $f(B) : a$ and $\alpha : B \rightarrow A$

  there is a *substitution* term

  $f\,[\alpha] : A \triangleright a$

  with $\mathit{arity}(\boldsymbol{L})(f\,[\alpha]) = \mathit{arity}(\boldsymbol{L})(\alpha) = A$ and $\mathit{rtn}(\boldsymbol{L})(f\,[\alpha]) = \mathit{rtn}(\boldsymbol{L})(f) = a$. As a special case, by substituting into a function symbol the identity term on the function arity, the function symbol $f(A) : a$ becomes a *function* term $f = f\,[\mathit{id}_A] : A \triangleright a$. A term is *composite* when it is constructed by substitution.

```
(4) (SET.FTN$function element)
    (= (SET.FTN$source element) language)
    (= (SET.FTN$target element) set.ftn$function)
    (= (SET.FTN$composition [element set.ftn$source])
       (SET.FTN$composition [lang$variable set.mbr$member]))
    (= (SET.FTN$composition [element set.ftn$target]) term)
    (forall (?l (lang$language ?l))
        (and (= (set.ftn$composition [(element ?l) (arity ?l)])
                (set.mbr$indication (lang$variable ?l)))
             (= (set.ftn$composition [(element ?l) (return ?l)])
                (set.mbr$projection (lang$variable ?l)))))

(5) (SET.FTN$function substitution-opspan)
    (= (SET.FTN$source substitution-opspan) lang$language)
    (= (SET.FTN$target substitution-opspan) set.lim.pbk$opspan)
    (= (SET.FTN$composition [substitution-opspan set.lim.pbk$set1]) lang$function)
    (= (SET.FTN$composition [substitution-opspan set.lim.pbk$set2]) lang.term.tpl$tuple)
    (= (SET.FTN$composition [substitution-opspan set.lim.pbk$opvertex])
       (SET.FTN$composition [lang$variable set$power]))
    (= (SET.FTN$composition [substitution-opspan set.lim.pbk$opfirst]) lang$arity)
    (= (SET.FTN$composition [substitution-opspan set.lim.pbk$opsecond]) lang.term.tpl$index)

(6) (SET.FTN$function substitutable)
    (= (SET.FTN$source substitutable) lang$language)
    (= (SET.FTN$target substitutable) rel$relation)
    (= substitutable (SET.FTN$composition [substitution-opspan set.lim.pbk$relation]))

(7) (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 [substitutable rel$extent]))
    (= (SET.FTN$composition [substitution set.ftn$target]) term)
    (forall (?l (lang$language ?l))
        (and (= (set.ftn$composition [(substitution ?l) (arity ?l)])
                (set.ftn$composition
                    [(rel$second (substitutable ?l)) (lang.term.tpl$arity ?l)]))
             (= (set.ftn$composition [(substitution ?l) (return ?l)])
                (set.ftn$composition [(rel$first (substitutable ?l)) (lang$return ?l)]))))
```

Booleans

A term is elementary when it is built by the element constructor. A term is composite when it is built by the substitution constructor. A term must be either elementary or composite – the elementary and composite terms comprise a partition for the set of terms. These Boolean tests are used to define composite constructors and the domain of selectors.

```
(8) (SET.FTN$function is-element)
    (SET.FTN$function elementary)
    (= elementary is-element)
    (= (SET.FTN$source is-element) language)
    (= (SET.FTN$target is-element) set$set)
    (forall (?l (lang$language ?l))
        (and (set$subset (is-element ?l) (term ?l))
             (= (is-element ?l) (set.ftn$image (element ?l)))))

(9) (SET.FTN$function is-composite)
    (= (SET.FTN$source is-composite) lang$language)
    (= (SET.FTN$target is-composite) set$set)
    (forall (?l (lang$language ?l))
        (and (set$subset (is-composite ?l) (term ?l))
             (= (is-composite ?l) (set.ftn$image (substitution ?l)))))

(10) (forall (?l (lang$language ?l))
         (and (= (set$binary-intersection [(is-element ?l) (is-composite ?l)]) set.col$initial)
              (= (set$binary-union [(is-element ?l) (is-composite ?l)]) (term ?l))))
```

Composite Constructors

- Term tuples can be substituted into terms. A term $\beta \in$ *term*($\boldsymbol{L}$) and a term tuple $\alpha \in$ *term**($\boldsymbol{L}$) are a *term-substitutable* pair when the arity of β is the same as the index of α : *arity*($\boldsymbol{L}$)(β) = *index*($\boldsymbol{L}$)(α). For any term-substitutable pair

  $\beta : B \triangleright b$ and $\alpha : B \rightharpoonup A$

  there is a *substitution* term

  $\beta[\alpha] : A \triangleright b$

  with *arity*($\boldsymbol{L}$)(β[α]) = *arity*($\boldsymbol{L}$)(α) = A and *rtn*($\boldsymbol{L}$)(β[α]) = *rtn*($\boldsymbol{L}$)(β) = b.

  – If the term β is an elementary term

  $\beta = \in_{B,\, b} : B \triangleright b$ with $b \in B$

  then the term substitution β[α] selects the b^{th} component term

  $\beta[\alpha] = \sigma_b(\alpha) : A \triangleright b$.

  – If the term β is an composite term

  $\beta = g[\gamma] : B \triangleright b$ for some substitutable pair $g(C) : b$ and $\gamma : C \rightharpoonup B$

  then the term substitution resolves as tuple composition $\gamma \circ \alpha$ of the two composable term tuples γ and α followed by substitution of the tuple composite into the function g

  $\beta = g[\gamma \circ \alpha\,] : A \triangleright b$.

```
(11) (SET.FTN$function term-substitution-opspan)
     (= (SET.FTN$source term-substitution-opspan) lang$language)
     (= (SET.FTN$target term-substitution-opspan) set.lim.pbk$opspan)
     (= (SET.FTN$composition [term-substitution-opspan set.lim.pbk$set1]) term)
     (= (SET.FTN$composition [term-substitution-opspan set.lim.pbk$set2]) lang.term.tpl$tuple)
     (= (SET.FTN$composition [term-substitution-opspan set.lim.pbk$opvertex])
        (SET.FTN$composition [lang$variable set$power]))
     (= (SET.FTN$composition [term-substitution-opspan set.lim.pbk$opfirst]) arity)
     (= (SET.FTN$composition [term-substitution-opspan set.lim.pbk$opsecond]) lang.term.tpl$index)

(12) (SET.FTN$function term-substitutable)
     (= (SET.FTN$source term-substitutable) lang$language)
     (= (SET.FTN$target term-substitutable) rel$relation)
     (= term-substitutable (SET.FTN$composition [term-substitution-opspan set.lim.pbk$relation]))

(13) (SET.FTN$function term-substitution)
     (= (SET.FTN$source term-substitution) lang$language)
     (= (SET.FTN$target term-substitution) set.ftn$function)
     (= (SET.FTN$composition [term-substitution set.ftn$source])
        (SET.FTN$composition [term-substitutable rel$extent]))
     (= (SET.FTN$composition [term-substitution set.ftn$target]) term)
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(term-substitution ?l) (arity ?l)])
                 (set.ftn$composition
                     [(rel$second (term-substitutable ?l)) (lang.term.tpl$arity ?l)]))
              (= (set.ftn$composition [(term-substitution ?l) (return ?l)])
                 (set.ftn$composition
                     [(rel$first (term-substitutable ?l)) (return ?l)]))))
     (forall (?l (lang$language ?l))
              ?a (lang.term.tpl$tuple ?a))
         (and (=> (exists (?b (= ?b ((lang.term.tpl$index ?l) ?a)
                              ?x (?b ?x))
                   (= ((term-substitution ?l) [((element ?l) [?b ?x]) ?a])
                      ((lang.term.tpl$selection ?l) [?a ?x]))))
              (=> (exists (?g (lang$function ?g) ?c (lang.tem.tpl$tuple ?c)
                              (substitutable ?g ?c) (lang.term.tpl$composable ?c ?a))
                   (= ((term-substitution ?l) [((substitution ?l) [?g ?c]) ?a])
                      ((substitution ?l) [?g ((lang.term.tpl$composition ?l) [?c ?a])]))))))
```

Selectors

○ Here we define two basic selectors, *function* and *tuple*, and a composite selector called *resolution*. For composite terms the function and tuple selectors return the components used for the substitution constructor. The resolution selector is the pairing of these two.

```
(14) (SET.FTN$function function)
     (= (SET.FTN$source function) lang$language)
     (= (SET.FTN$target function) set.ftn$function)
     (= (SET.FTN$composition [function set.ftn$source]) is-composite)
     (= (SET.FTN$composition [function set.ftn$target]) lang$function)
     (forall (?l (lang$language ?l))
         (= (set.ftn$composition [(substitution ?l) (function ?l)])
            (rel$first (substitutable ?l))))

(15) (SET.FTN$function tuple)
     (= (SET.FTN$source tuple) lang$language)
     (= (SET.FTN$target tuple) set.ftn$function)
     (= (SET.FTN$composition [tuple set.ftn$source]) is-composite)
     (= (SET.FTN$composition [tuple set.ftn$target]) lang.term.tpl$tuple)
     (forall (?l (lang$language ?l))
         (= (set.ftn$composition [(substitution ?l) (tuple ?l)])
            (rel$second (substitutable ?l))))

(16) (SET.FTN$function resolution)
     (= (SET.FTN$source resolution) lang$language)
     (= (SET.FTN$target resolution) set.ftn$function)
     (= (SET.FTN$composition [resolution set.ftn$source]) is-composite)
     (= (SET.FTN$composition [resolution set.ftn$target])
        (SET.FTN$composition [substitutable rel$extent]))
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(resolution ?l) (substitution ?l)])
                 (set.ftn$inclusion [(is-composite ?l) (term ?l)]))
              (forall (?f ((lang$function ?l) ?f) ?t ((lang.term.tpl$tuple ?l) ?t)
                         ((substitutable ?l) [?f ?t]))
                  (= ((resolution ?l) ((substitution ?l) [?f ?t])) [?f ?t]))))
```

Term Tuples

lang.term.tpl

- We next give a pointwise recursive definition for the set of tuples of $\boldsymbol{L}$-terms *term**($\boldsymbol{L}$). A term tuple $\alpha \in$ *term**($\boldsymbol{L}$) will have an index *index*($\boldsymbol{L}$)(α) and an arity *arity*($\boldsymbol{L}$)(α). Hence, there is
  - an *index* function *index*($\boldsymbol{L}$) : *tpl*($\boldsymbol{L}$) $\rightarrow \wp$*var*($\boldsymbol{L}$), and
  - an *arity* function *arity*($\boldsymbol{L}$) : *tpl*($\boldsymbol{L}$) $\rightarrow \wp$*var*($\boldsymbol{L}$).

  We use the notation

  $$\alpha : B \rightharpoondown A$$

  to denote a term with *index*($\boldsymbol{L}$)(α) = B and *arity*($\boldsymbol{L}$)(α) = A.

```
(1) (SET.FTN$function tuple)
    (= (SET.FTN$source tuple) language)
    (= (SET.FTN$target tuple) set$set)
    (forall (?l (lang$language ?l))
        (<=> ((tuple ?l) ?t)
             (and (set.ftn$function ?t)
                  (= (set.ftn$source ?t) ((index ?l) ?t))
                  (= (set.ftn$target ?t) (term ?l))
                  (forall (?x (((index ?l) ?t) ?x))
                      (and (= (lang.term$arity (?t ?x)) ((arity ?l) ?t))
                           (= (lang.term$return (?t ?x)) ?x))))))

(2) (SET.FTN$function index)
    (= (SET.FTN$source index) language)
    (= (SET.FTN$target index) set.ftn$function)
    (= (SET.FTN$composition [index set.ftn$source]) tuple)
    (= (SET.FTN$composition [index set.ftn$target])
       (SET.FTN$composition [variable set$power]))

(3) (SET.FTN$function arity)
    (= (SET.FTN$source arity) language)
    (= (SET.FTN$target arity) set.ftn$function)
    (= (SET.FTN$composition [arity set.ftn$source]) tuple)
    (= (SET.FTN$composition [arity set.ftn$target])
       (SET.FTN$composition [lang$variable set$power]))
```

Constructors

- For any subset of variables $A \subseteq \mathit{var}(\boldsymbol{L})$, regarded as an arity, there is an *empty* term tuple

  $0_A : \varnothing \rightharpoonup A$

  with $\mathit{index}(\boldsymbol{L})(0_A) = \varnothing$ and $\mathit{arity}(\boldsymbol{L})(0_A) = A$. Hence, there is

  – an *empty* function $\mathit{empty}(\boldsymbol{L}) : \wp\mathit{var}(\boldsymbol{L}) \rightarrow \mathit{tpl}(\boldsymbol{L})$.

- Terms can be inserted breadthwise into term tuples. A term $\alpha \in \mathit{term}(\boldsymbol{L})$ and a term tuple $\beta \in \mathit{term}^*(\boldsymbol{L})$ are an *insertible* pair when the arity of the term α is the same as the arity of the term tuple β: $\mathit{arity}(\boldsymbol{L})(\alpha) = \mathit{arity}(\boldsymbol{L})(\beta)$ and the return of α is not contained in the arity. For any insertible pair

  $\alpha : A \triangleright a$ and $\beta : B \rightharpoonup A$ and $a \notin B$

  there is an *insertion* term tuple

  $[\alpha, \beta] : \{a\} \cup B \rightharpoonup A$

  with $\mathit{arity}(\boldsymbol{L})([\alpha, \beta]) = A$ and $\mathit{index}(\boldsymbol{L})([\alpha, \beta]) = \{a\} \cup B$. Hence, there is

  – an *insertion* function $\mathit{insert}(\boldsymbol{L}) : \mathit{ext}(\mathit{insertible}(\boldsymbol{L})) = \mathit{term}(\boldsymbol{L}) \oplus \mathit{tpl}(\boldsymbol{L}) \rightarrow \mathit{tpl}(\boldsymbol{L})$.

  As a special case, by substituting a term into the empty tuple on the term arity,

  $\alpha : A \triangleright a$ and $0_A : \varnothing \rightharpoonup A$

  the term becomes a *singleton* term tuple

  $\alpha : \{a\} \rightharpoonup A$

  with $\mathit{index}(\boldsymbol{L})(\alpha) = \{a\}$ and $\mathit{arity}(\boldsymbol{L})(\alpha) = A$. Hence, there is a

  – an *singleton* function $\{\text{-}\}_L : \mathit{term}(\boldsymbol{L}) \rightarrow \mathit{tpl}(\boldsymbol{L})$.

- Term tuples can be renamed. A *renamable pair* (n, α) consists of a renaming n and a term tuple α where the codomain of the renaming is the index of the term tuple $\mathit{cod}(\boldsymbol{L})(n) = \mathit{index}(\boldsymbol{L})(\alpha)$. For any renamable pair

  $n : C \leftrightarrow B$ and $\alpha : B \rightharpoonup A$

  there is a *renamed* term tuple

  $n \circ \alpha : C \rightharpoonup A$

  with $\mathit{index}(\boldsymbol{L})(n \circ \alpha) = \mathit{dom}(\boldsymbol{L})(n) = C$ and $\mathit{arity}(\boldsymbol{L})(n \circ \alpha) = \mathit{arity}(\boldsymbol{L})(\alpha) = A$. Hence, there is

  – a *renaming* function $\mathit{rename}(\boldsymbol{L}) : \mathit{ext}(\mathit{renamable}(\boldsymbol{L})) \rightarrow \mathit{tpl}(\boldsymbol{L})$.

```
(4) (SET.FTN$function empty)
    (= (SET.FTN$source empty) language)
    (= (SET.FTN$target empty) set.ftn$function)
    (= (SET.FTN$composition [empty set.ftn$source])
       (SET.FTN$composition [lang$variable set$power]))
    (= (SET.FTN$composition [empty set.ftn$target]) tuple)
    (forall (?l (lang$language ?l))
        (and (= (set.ftn$composition [(empty ?l) (index ?l)])
                ((set.ftn$constant
                     [(set$power (lang$variable ?l)) (set$power (lang$variable ?l))])
                     set.col$initial))
             (= (set.ftn$composition [(empty ?l) (arity ?l)])
                (set.ftn$identity (set$power (lang$variable ?l))))
             (forall (?a (set$subset ?a (lang$variable ?l)))
                 (= ((empty ?l) ?a) (set.col$counique (term ?l))))))

(5) (SET.FTN$function insertible)
    (= (SET.FTN$source insertible) lang$language)
    (= (SET.FTN$target insertible) rel$relation)
    (forall (?l (lang$language ?l)
             ?a ((lang.term$term ?l) ?a) ?b ((tuple ?l) ?b))
        (<=> (insertible ?a ?b)
             (and (= ((lang.term$arity ?l) ?a) ((arity ?l) ?b))
                  (not ((index ?l) ?b) ((lang.term$return ?l) ?a))))))
```

```
(6) (SET.FTN$function insertion)
    (= (SET.FTN$source insertion) lang$language)
    (= (SET.FTN$target insertion) set.ftn$function)
    (= (SET.FTN$composition [insertion set.ftn$source])
       (SET.FTN$composition [insertible rel$extent]))
    (= (SET.FTN$composition [insertion set.ftn$target]) tuple)
    (forall (?l (lang$language ?l)
             ?a ((lang.term$term ?l) ?a) ?b ((tuple ?l) ?b)
             (insertible ?a ?b))
        (and (= ((arity ?l) ((insertion ?l) [?a ?b])) ((arity ?l) ?b))
             (= ((index ?l) ((insertion ?l) [?a ?b]))
                (set$binary-union
                    [((set.ftn$singleton (lang$variable ?l)) ((lang.term$return ?l) ?a))
                     ((index ?l) ?b)]))
             (= (((insertion ?l) [?a ?b]) (lang.term$return ?a)) ?a)
             (forall (?x (((index ?l) ?b) ?x))
                 (= (((insertion ?l) [?a ?b]) ?x) (?b ?x)))))

(7) (SET.FTN$function singleton)
    (SET.FTN$function atom)
    (= atom singleton)
    (= (SET.FTN$source singleton) language)
    (= (SET.FTN$target singleton) set.ftn$function)
    (= (SET.FTN$composition [singleton set.ftn$source]) lang.term$term)
    (= (SET.FTN$composition [singleton set.ftn$target]) tuple)
    (forall (?l (lang$language ?l))
        (and (= (set.ftn$composition [(singleton ?l) (index ?l)])
                (set.ftn$composition
                    [lang.term$return (set.ftn$singleton (lang$variable ?l))]))
             (= (set.ftn$composition [(singleton ?l) (arity ?l)]) (lang.term$arity ?l))
             (forall (?t ((lang.term$term ?l) ?t))
                 (= (((singleton ?l) ?t)
                    ((insertion ?l) [?t (empty ((lang.term$arity ?l) ?t))]))))))

(8) (SET.FTN$function renamable-opspan)
     (= (SET.FTN$source renamable-opspan) lang$language)
     (= (SET.FTN$target renamable-opspan) set.lim.pbk$opspan)
     (= (SET.FTN$composition [renamable-opspan set.lim.pbk$set1]) lang$renaming)
     (= (SET.FTN$composition [renamable-opspan set.lim.pbk$set2]) tuple)
     (= (SET.FTN$composition [renamable-opspan set.lim.pbk$opvertex])
        (SET.FTN$composition [lang$variable set$power]))
     (= (SET.FTN$composition [renamable-opspan set.lim.pbk$opfirst]) lang$codomain)
     (= (SET.FTN$composition [renamable-opspan set.lim.pbk$opsecond]) index)

(9) (SET.FTN$function renamable)
    (= (SET.FTN$source renamable) lang$language)
    (= (SET.FTN$target renamable) rel$relation)
    (= renamable (SET.FTN$composition [renamable-opspan set.lim.pbk$relation]))

(10) (SET.FTN$function renaming)
     (= (SET.FTN$source renaming) lang$language)
     (= (SET.FTN$target renaming) set.ftn$function)
     (= (SET.FTN$composition [renaming set.ftn$source])
        (SET.FTN$composition [renamable rel$extent]))
     (= (SET.FTN$composition [renaming set.ftn$target]) tuple)
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(renaming ?l) (arity ?l)])
                 (set.ftn$composition [(rel$second (renamable ?l)) (arity ?l)]))
              (= (set.ftn$composition [(renaming ?l) (index ?l)])
                 (set.ftn$composition [(rel$first (renamable ?l)) (lang$domain ?l)])))
              (forall (?n ((lang$renaming ?l) ?n) ?a ((tuple ?l) ?a) (renamable ?n ?a))
                       ?x (((lang$codomain ?l) ?n) ?x))
                  (= ((renaming ?l) [?n ?a]) ?x) (?a (?n ?x))))))
```

Booleans

A term is empty when it is built by the empty constructor. A term is not empty when it is not so built. A term is an insertion when it is built by the insertion constructor. A term is an atom when it is built by the singleton constructor. An atom is an insertion, which is not empty. These Boolean tests are used to define composite constructors and the domain of selectors. The empty and nonempty term tuples comprise a partition for the set of term tuples. The nonempty term tuples, which are not insertions, are renamings of non-identities.

```
(11) (SET.FTN$function is-empty)
     (= (SET.FTN$source is-empty) language)
     (= (SET.FTN$target is-empty) set$set)
     (forall (?l (lang$language ?l))
         (and (set$subset (is-empty ?l) (tuple ?l))
              (= (is-empty ?l) (set.ftn$image (empty ?l)))))

(12) (SET.FTN$function is-nonempty)
     (= (SET.FTN$source is-nonempty) language)
     (= (SET.FTN$target is-nonempty) set$set)
     (forall (?l (lang$language ?l))
         (and (set$subset (is-nonempty ?l) (tuple ?l))
              (= (is-nonempty ?l) (set$difference [(tuple ?l) (empty ?l)]))))

(13) (SET.FTN$function is-insertion)
     (= (SET.FTN$source is-insertion) language)
     (= (SET.FTN$target is-insertion) set$set)
     (forall (?l (lang$language ?l))
         (and (set$subset (is-insertion ?l) (is-nonempty ?l))
              (= (is-insertion ?l) (set.ftn$image (insertion ?l)))))

(14) (SET.FTN$function is-singleton)
     (SET.FTN$function is-atom)
     (= is-atom is-singleton)
     (= (SET.FTN$source is-singleton) language)
     (= (SET.FTN$target is-singleton) set$set)
     (forall (?l (lang$language ?l))
         (and (set$subset (is-singleton ?l) (is-insertion ?l))
               (= (is-singleton ?l) (set.ftn$image (singleton ?l)))))

(15) (forall (?l (lang$language ?l)
              ?b ((tuple ?l) ?b))
         (=> ((set$difference [(is-nonempty ?l) (is-insertion ?l)]) ?b)
             (exists (?n ((lang$renaming ?l) ?n)
                      ?a ((tuple ?l) ?a) (renamable ?n ?a)
                      (not (= ?n (set.ftn$identity ((index ?l) ?a)))))
                 (= ?b ((renaming ?l) [?n ?a])))))
```

Composite Constructors

○ Term tuples can be tupled. Two term tuples $\alpha_0, \alpha_1 \in$ *term**($\boldsymbol{L}$) are *tuplable* when they share a common arity *arity*($\boldsymbol{L}$)(α_0) = *arity*($\boldsymbol{L}$)(α_1) = A, but have disjoint tuplings *index(*$\boldsymbol{L}$)(α_0) ∩ *index(*$\boldsymbol{L}$)(α_1) = $A_0 \cap A_1 = \emptyset$. For any two tuplable term tuples

$$\alpha_0 : A_0 \rightharpoonup A \text{ and } \alpha_1 : A_1 \rightharpoonup A$$

there is a binary term *tupling*

$$[\alpha_0, \alpha_1] : A_0 \cup A_1 \rightharpoonup A$$

with *index(*$\boldsymbol{L}$)($[\alpha_0, \alpha_1]$) = *index(*$\boldsymbol{L}$)(α_0) ∪ *index(*$\boldsymbol{L}$)(α_1) = $A_0 \cup A_1$ and *arity*($\boldsymbol{L}$)($[\alpha_0, \alpha_1]$) = *arity*($\boldsymbol{L}$)(α_0) = A.

```
(16) (SET.FTN$function tuplable)
     (= (SET.FTN$source tuplable) lang$language)
     (= (SET.FTN$target tuplable) rel$relation)
     (forall (?l (lang$language ?l)
              ?a0 ((tuple ?l) ?a0) ?a1 ((tuple ?l) ?a1))
         (<=> (tuplable ?a0 ?a1)
              (and (= ((arity ?l) ?a0) ((arity ?l) ?a1))
                   (set$disjoint ((index ?l) ?a0) ((index ?l) ?a1)))))

(17) (SET.FTN$function tupling)
     (SET.FTN$function tuple-insertion)
     (= tuple-insertion tupling)
     (= (SET.FTN$source tupling) lang$language)
     (= (SET.FTN$target tupling) set.ftn$function)
     (= (SET.FTN$composition [tupling set.ftn$source])
        (SET.FTN$composition [tuplable rel$extent]))
     (= (SET.FTN$composition [tupling set.ftn$target]) tuple)
     (forall (?l (lang$language ?l)
              ?a0 ((tuple ?l) ?a0) ?a1 ((tuple ?l) ?a1)
              (tuplable ?a0 ?a1))
         (and (= ((arity ?l) ((tupling ?l) [?a0 ?a1])) ((arity ?l) ?a0))
              (= ((index ?l) ((tupling ?l) [?a0 ?a1]))
                 (set$binary-union [((index ?l) ?a0) ((index ?l) ?a1)]))
              (forall (?x0 (((index ?l) ?a0) ?x0))
                  (= (((tupling ?l) [?a0 ?a1]) ?x0) (?a0 ?x0)))
              (forall (?x1 (((index ?l) ?a1) ?x1))
                  (= (((tupling ?l) [?a0 ?a1]) ?x1) (?a1 ?x1)))))
```

Selectors

- The tuple index function

  $\mathit{index}(\boldsymbol{L}) : \mathit{tpl}(\boldsymbol{L}) \rightarrow \wp\mathit{var}(\boldsymbol{L})$

  can serve as a colimit arity.

```
(18) (SET.FTN$function index-arity)
     (= (SET.FTN$source index-arity) set$set)
     (= (SET.FTN$target index-arity) set.col.art$arity)
     (= (SET.FTN$composition [index-arity set.col.art$index]) tuple)
     (= (SET.FTN$composition [index-arity set.col.art$base]) lang$variable)
     (= (SET.FTN$composition [index-arity set.col.art$function]) index)
```

- Any type language $\boldsymbol{L}$ has a set of index members

  $\mathit{index\text{-}mbr}(\boldsymbol{L}) = \sum \mathit{index\text{-}arity}(\boldsymbol{L})$

  $= \sum_{\alpha \in \mathit{tpl}(\boldsymbol{L})} \mathit{index\text{-}arity}(\boldsymbol{L})(\alpha)$

  $= \{(\alpha, x) \mid \alpha \in \mathit{tpl}(\boldsymbol{L}), x \in \mathit{index}(\boldsymbol{L})(\alpha)\}$,

  that is the coproduct of its tuple index function as arity.

  $$\begin{array}{ccc} \mathit{index}(\boldsymbol{L})(\alpha) & \xrightarrow{\mathit{inj}(\boldsymbol{L})(\alpha)} & \mathit{index\text{-}mbr}(\boldsymbol{L}) \\ & \subseteq & \downarrow \mathit{proj}(\boldsymbol{L}) \\ & & \mathit{var}(\boldsymbol{L}) \end{array}$$

  Diagram 2: Coproduct

  For any type language $\boldsymbol{L}$ and any term tuple $\alpha \in \mathit{tpl}(\boldsymbol{L})$ there is a index member *injection* function:

  $\mathit{inj}(\boldsymbol{L})(\alpha) : \mathit{index}(\boldsymbol{L})(\alpha) \rightarrow \mathit{index\text{-}mbr}(\boldsymbol{L})$

  defined by $\mathit{inj}(\boldsymbol{L})(\alpha)(x) = (\alpha, x)$ for all elements $x \in \mathit{index}(\boldsymbol{L})(\alpha)$. Obviously, the injections are injective. They commute (Diagram 2) with projection and inclusion.

```
(19) (SET.FTN$function index-member)
     (= (SET.FTN$source index-member) lang$language)
     (= (SET.FTN$target index-member) set$set)
     (= index-member (SET.FTN$composition [index-arity set.col.art$colimit]))

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

- Any type language $\boldsymbol{L}$ defines *indication* and *projection* functions based on its index arity (Figure 4):

  $\mathit{indic}(\boldsymbol{L}) : \mathit{index\text{-}mbr}(\boldsymbol{L}) \rightarrow \mathit{tpl}(\boldsymbol{L})$,

  $\mathit{proj}(\boldsymbol{L}) : \mathit{index\text{-}mbr}(\boldsymbol{L}) \rightarrow \mathit{var}(\boldsymbol{L})$.

  $$\begin{array}{ccc} & \mathit{index\text{-}mbr}(\boldsymbol{L}) & \\ \swarrow \mathit{indic}(\boldsymbol{L}) & & \searrow \mathit{proj}(\boldsymbol{L}) \\ & & \mathit{var}(\boldsymbol{L}) \\ \mathit{tpl}(\boldsymbol{L}) & \xrightarrow{\mathit{index}(\boldsymbol{L})} & \wp\mathit{var}(\boldsymbol{L}) \end{array}$$

  Figure 4: Indication and projection

  These are defined by

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

  for all term tuples $\alpha \in \mathit{tpl}(\boldsymbol{L})$ and all variables $x \in \mathit{index}(\boldsymbol{L})(\alpha)$.

```
(21) (SET.FTN$function index-indication)
     (= (SET.FTN$source index-indication) lang$language)
     (= (SET.FTN$target index-indication) set.ftn$function)
     (= (SET.FTN$composition [index-indication set.ftn$source]) index-member)
     (= (SET.FTN$composition [index-indication set.ftn$target]) tuple)
     (= index-indication (SET.FTN$composition [index-arity set.col.art$indication]))

(22) (SET.FTN$function index-projection)
     (= (SET.FTN$source index-projection) lang$language)
     (= (SET.FTN$target index-projection) set.ftn$function)
     (= (SET.FTN$composition [index-projection set.ftn$source]) index-member)
     (= (SET.FTN$composition [index-projection set.ftn$target]) lang$variable)
     (= index-projection (SET.FTN$composition [index-arity set.col.art$projection]))
```

- Terms can be selected out of a term tuple, leaving a remainder term tuple. For any pair consisting of a index and a term tuple

  $b \in B$ and $\alpha : B \rightarrow A$

there is a *selection* term

$\sigma_b(\alpha) : A \triangleright b.$

and a *remainder* term tuple

$\rho_b(\alpha) : (B-\{b\}) \rightarrow A.$

The source of selection and remainder is the coproduct of index arity. Hence, there is

– an *selection* function *select*($\boldsymbol{L}$) : *index-mbr*($\boldsymbol{L}$) → *term*($\boldsymbol{L}$), and

– a *remainder* function *rest*($\boldsymbol{L}$) : *index-mbr*($\boldsymbol{L}$) → *term*($\boldsymbol{L}$).

Clearly, an insertion of a term with return b followed by a selection of the b^{th} component term is the identity on terms.

```
(23) (SET.FTN$function selection)
     (= (SET.FTN$source selection) lang$language)
     (= (SET.FTN$target selection) set.ftn$function)
     (= (SET.FTN$composition [selection set.ftn$source]) index-member)
     (= (SET.FTN$composition [selection set.ftn$target]) lang.term$term)
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(selection ?l) (lang.term$arity ?l)])
                 (set.ftn$composition [(index-indication ?l) (arity ?l)]))
              (= (set.ftn$composition [(selection ?l) (lang.term$return ?l)])
                 (index-projection ?l))
              (forall (?a ((tuple ?l) ?a) ?b ((lang$variable ?l) ?b) (((index ?l) ?a) ?b))
                  (= ((selection ?l) [?a ?b]) (?a ?b)))))

(24) (SET.FTN$function remainder)
     (SET.FTN$function rest)
     (= rest remainder)
     (= (SET.FTN$source remainder) lang$language)
     (= (SET.FTN$target remainder) set.ftn$function)
     (= (SET.FTN$composition [remainder set.ftn$source]) index-member)
     (= (SET.FTN$composition [remainder set.ftn$target]) tuple)
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(remainder ?l) (arity ?l)])
                 (set.ftn$composition [(index-indication ?l) (arity ?l)]))
              (forall (?a ((tuple ?l) ?a) ?b ((lang$variable ?l) ?b) (((index ?l) ?a) ?b))
                  (and (= ((index ?l) ((remainder ?l) [?a ?b]))
                          (set$difference
                              [((arity ?l) ?a)
                               ((set.ftn$singleton (lang$variable ?l)) ?b)]))
                       (forall (?x (((index ?l) ?a) ?x) (not (= ?x ?b)))
                           (= (((remainder ?l) [?a ?b]) ?x) (?a ?x)))))))
```

Category-theoretic Operations

- Term tuples can be composed. Two term tuples $\beta, \alpha \in$ *term**(***L***) are *composable* when the arity of the first is the index of the second *arity*(***L***)(β) = *index*(***L***)(α). For any two composable term tuples

  $\beta : C \rightarrow B$ and $\alpha : B \rightarrow A$

  there is a *composition* term tuple

  $\beta \circ \alpha : C \rightarrow A$

  with *index*(***L***)($\beta \circ \alpha$) = *index*(***L***)(β) = C and *arity*(***L***)($\beta \circ \alpha$) = *arity*(***L***)(α) = A. The axiomatic definition proceeds by section then term substitution:

  $(\beta \circ \alpha)(c) \; = \sigma_c(\beta)[\alpha] : A \rhd c.$

  for every variable $c \in C$ = *index*(***L***)(β).

- For any subset of variables $A \subseteq$ *var*(***L***), regarded as an arity, there is an *identity* term tuple

  $id_A : A \rightarrow A$

  with *index*(***L***)(id_A) = A and *arity*(***L***)(id_A) = A. Hence, there is a

  – an *identity* function *id*(***L***) : $\wp$*var*(***L***) $\rightarrow$ *tpl*(***L***).

```
(25) (SET.FTN$function composable-opspan)
     (= (SET.FTN$source composable-opspan) lang$language)
     (= (SET.FTN$target composable-opspan) set.lim.pbk$opspan)
     (forall (?l (lang$language ?l))
         (and (= (set.lim.pbk$set1 (composable-opspan ?l) (tuple ?l))
              (= (set.lim.pbk$set2 (composable-opspan ?l) (tuple ?l))
              (= (set.lim.pbk$opvertex (composable-opspan ?l)
                 (set$power (lang$variable ?l)))
              (= (set.lim.pbk$opfirst (composable-opspan ?l) (arity ?l))
              (= (set.lim.pbk$opsecond (composable-opspan ?l) (index ?l))))

(26) (SET.FTN$function composable)
     (= (SET.FTN$source composable) lang$language)
     (= (SET.FTN$target composable) rel$relation)
     (= composable (SET.FTN$composition [composable-opspan set.lim.pbk$relation]))

(27) (SET.FTN$function composition)
     (= (SET.FTN$source composition) lang$language)
     (= (SET.FTN$target composition) set.ftn$function)
     (= (SET.FTN$composition [composition set.ftn$source])
        (SET.FTN$composition [composable rel$extent]))
     (= (SET.FTN$composition [composition set.ftn$target]) tuple)
     (forall (?l (lang$language ?l)
              ?b ((tuple ?l) ?b) ?a ((tuple ?l) ?a)
              (composable ?b ?a))
         (and (= ((index ?l) ((composition ?l) [?b ?a])) ((index ?l) ?b))
              (= ((arity ?l) ((composition ?l) [?b ?a])) ((arity ?l) ?a))
              (forall (?c (((index ?l) ?b) ?c))
                  (= (((composition ?l) [?a ?b]) ?c)
                     ((lang.term$term-substitution ?l) [((selection ?l) [?b ?c]) ?a])))))

(28) (SET.FTN$function identity)
     (= (SET.FTN$source identity) lang$language)
     (= (SET.FTN$target identity) set.ftn$function)
     (= (SET.FTN$composition [identity set.ftn$source])
        (SET.FTN$composition [lang$variable set$power]))
     (= (SET.FTN$composition [identity set.ftn$target]) tuple)
     (forall (?l (lang$language ?l))
         (and (= (set.ftn$composition [(identity ?l) (index ?l)])
                 (set.ftn$identity (set$power (lang$variable ?l)))
              (= (set.ftn$composition [(identity ?l) (arity ?l)])
                 (set.ftn$identity (set$power (lang$variable ?l)))
              (forall (?a (set$subset ?a (lang$variable ?l)) ?x (?a ?x))
                  (= (((identity ?l) ?a) ?x) (lang.term$element [?a ?x])))))
```