

The IFF SCL (meta) Ontology (IFF-SCL)

THE OBJECT ASPECT OF SCL 1

SCL Syntax..... 1

 Terms 3

 Sentences 7

SCL Semantics 17

 Vocabularies 17

 Interpretations 21

 Satisfaction 25

SCL to FOL Translation 27

APPENDIX 28

Sequences..... 28

 Basics 28

 Relations and Functions 35

The Object Aspect of SCL

SCL Syntax

scl.obj

This document axiomatizes the Simple Common Logic (meta) Ontology (IFF-SCL). Its two main sections deal with the syntax and semantics of Simple Common Logic (SCL). Table 1 lists the 69 terms in the syntactic kernel of the IFF-SCL – there are 18 terms in the term part and 50 terms in the sentence part. The 16 terms in bold correspond to non-terminals or reserve-words in the SCL kernel grammar. There are 18 terms dealing with the vocabulary of an SCL text – 4 in the term part and 14 in the sentence part.

Table 1: Terminology for the Objective Syntactic Aspect of the SCL (meta) Ontology

	Other	Object	Morphism
scl .obj		name-set	
scl .obj .trm		term term-sequence elementary composite	element select-name application select-term select-term-sequence resolution
	application-diagram	application-pair	projection1 projection2
			term-objects term-sequence-objects term-functions term-sequence-functions
scl .obj .snt	relater-diagram role-pair-diagram role-description-diagram	atom equation relater role-pair role-description	term1 term2 term term-sequence role-name role-value property role-set
	quantification-pair-diagram	sentence text atomic-sentence conjunctive-sentence negative-sentence quantification-pair quantified-sentence	holds select-atom conjunction select-sentence-subset negation select-sentence name-projection sentence-projection quantification select-quantification-pair variable body
			role-pair-objects role-pair-functions role-set-objects role-set-functions atom-objects sentence-objects text-objects atom-functions sentence-functions text-functions atom-relations sentence-relations text-relations vocabulary

The IFF Simple Common Logic (meta) Ontology

- There are five binary Cartesian products used to axiomatize the SCL. Each comes equipped with a pair of product projections. Term application requires the binary product $trm \times trm\text{-seq}$ between the set of terms and the set of term sequences. The relater component of atoms is the binary product $rel = trm \times trm\text{-seq}$ between the set of terms and the set of term sequences. The role-description component of atoms requires the binary product $rol\text{-pr}$ between the name-set and the set of terms, and is the binary product $rol\text{-desc} = trm \times (rol\text{-pr} \circ \wp)$ between the set of terms and the set of role-pair subsets. The quantification component of sentences requires the binary product $name \times sent$ between the name-set and the set of sentences.
- There are three sums (disjoint unions) used to axiomatize the SCL. Each comes equipped with a collection of sum injections. The set of terms is the binary sum $trm = trm + trm\text{-seq}$ of terms and term-sequences. The sum injections are the element and application constructor functions. The set of atoms is the ternary sum $atm = eqn + rel + rol\text{-desc}$ of equations relaters and role-descriptions. The sum injections are named. The set of sentences is the quaternary sum $sent = atm + sent + name \times sent + sent \circ \wp$ of atoms, sentences, name-sentence pairs and sentence-subsets. The sum injections are the holds, negation, conjunction and quantification constructor functions. Selector partial functions and Boolean tests (in the form of subsets) complete the abstract syntax for terms and sentences.

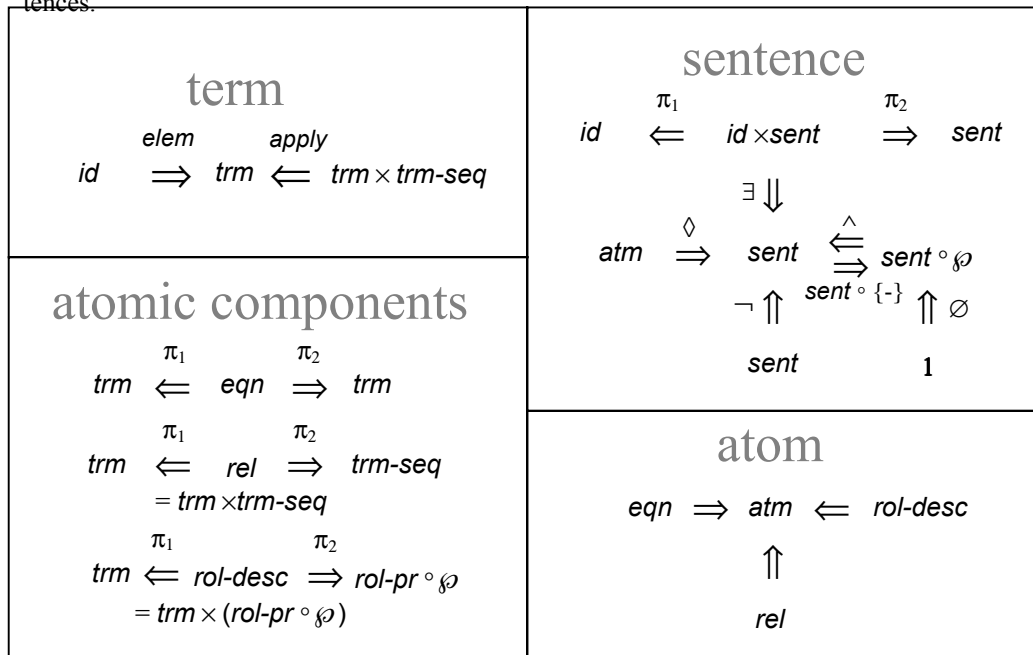


Figure 1: SCL Architecture

SCL text is based upon sets of names. For this there is a class. We identify name sets with abstract sets.

```
(1) (SET$class name-set)
    (= name-set set.obj$set)
```

Terms

scl.obj.trm

SCL syntax is obtained from conventional FO syntax by removing signature-related restrictions, so that both SCL atomic sentences and terms can be described uniformly as a term followed by a sequence of argument terms. Both terms and atomic sentences use the notion of a sequence of terms representing a vector of arguments to a function or relation. Names count as terms, and a complex (application) term consists of a term denoting a function with a vector of arguments.

For any name-set N , the term-set $trm(N)$ is the fixpoint solution of the set operator $F(X) = N + (X \times seq(X))$ consisting of the sum of N and the binary Cartesian product between the set X and the set of X -sequences. As the fixpoint solution, the term-set $trm(N)$ comes equipped with two injections into it: an element map from the name-set N and an application map from the binary Cartesian product between the set $trm(N)$ and the set of $trm(N)$ -sequences. The image of the element map is called the set of elementary N -terms. These are the N -terms that are just names. The image of the application map is called the set of composite N -terms. These are N -terms of the form (t, seq) for N -term t and N -term sequence seq .

- (1) (SET.FTN\$function term)
 - (= (SET.FTN\$source term) name-set)
 - (= (SET.FTN\$target term) set.obj\$set)
- (2) (SET.FTN\$function term-sequence)
 - (= (SET.FTN\$source term-sequence) name-set)
 - (= (SET.FTN\$target term-sequence) set.obj\$set)
 - (= term-sequence (SET.FTN\$composition [term set.obj.seq\$sequence]))

Term Constructors

A term is either elementary (just a name) or composite (a tree of names) (Figure 2). A term is *elementary* when it is constructed from a name. Names are terms. Hence, there is an *elem* function

$$elem(N) : N \rightarrow trm(N).$$

A term is *composite* when it is constructed by application. Term sequences can be substituted into functions. For any pair consisting of a term $t \in trm(L)$ and term sequence $\varphi \in trm-seq(L)$, there is a *application* term $t[\varphi]$. In contrast to traditional first order logic, there is no requirement that the arity of a function symbol is equal to the length of the term sequence. Here there is no arity and there are no functions (so far – later we will talk about terms in function position). Hence, there is a *application* function

$$apply(N) : trm(N) \times trm-seq(N) \rightarrow trm(N).$$

- (3) (SET.FTN\$function element)
 - (= (SET.FTN\$source element) name-set)
 - (= (SET.FTN\$target element) set.mor\$function)
 - (= (SET.FTN\$composition [element set.mor\$source]) (SET.FTN\$identity name-set))
 - (= (SET.FTN\$composition [element set.mor\$target]) term)
- (4) (SET.FTN\$function application-diagram)
 - (= (SET.FTN\$source application-diagram) name-set)
 - (= (SET.FTN\$target application-diagram) set.lim.prd2\$diagram)
 - (= (SET.FTN\$composition [application-diagram set.lim.prd2\$set1]) term)
 - (= (SET.FTN\$composition [application-diagram set.lim.prd2\$set2]) term-sequence)
- (5) (SET.FTN\$function application-pair)
 - (= (SET.FTN\$source application-pair) name-set)
 - (= (SET.FTN\$target application-pair) set.obj\$set)
 - (= application-pair (SET.FTN\$composition [application-diagram set.lim.prd2\$binary-product]))

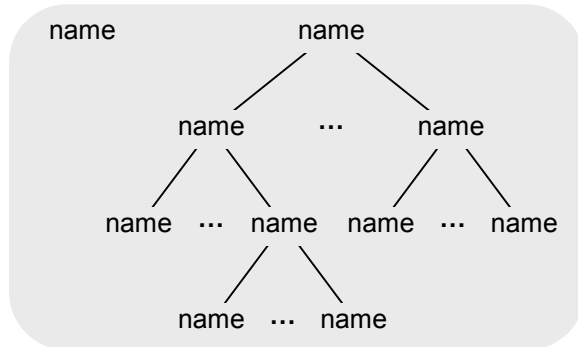


Figure 2: Terms

```
(6) (SET.FTN$function projection1)
    (= (SET.FTN$source projection1) name-set)
    (= (SET.FTN$target projection1) set.mor$function)
    (= (SET.FTN$composition [projection1 set.mor$source]) application-pair)
    (= (SET.FTN$composition [projection1 set.mor$target]) term)
    (= projection1
       (SET.FTN$composition [application-diagram set.lim.prd2$projection1]))

(7) (SET.FTN$function projection2)
    (= (SET.FTN$source projection2) name-set)
    (= (SET.FTN$target projection2) set.mor$function)
    (= (SET.FTN$composition [projection2 set.mor$source]) application-pair)
    (= (SET.FTN$composition [projection2 set.mor$target]) term-sequence)
    (= projection2
       (SET.FTN$composition [application-diagram set.lim.prod2projection2]))

(8) (SET.FTN$function application)
    (= (SET.FTN$source application) name-set)
    (= (SET.FTN$target application) set.mor$function)
    (= (SET.FTN$composition [application set.mor$source]) application-pair)
    (= (SET.FTN$composition [application set.mor$target]) term)
```

Objects and Functions

Any term contains a set of objects (subterms). An elementary term contains only its name as an object. A composite term contains itself plus the objects in its term and term-sequence components. Any term sequence contains the union of the objects in its coordinates. A term can be a function within another term. Any term contains a set of function (terms in function position). An elementary term contains no function. A composite term contains its term plus the functions in its term and term-sequence components. Any term sequence contains the union of the functions in its coordinates. A term can be a relation within an atom. However, a term does not contain any relation.

```
(9) (SET.FTN$function term-objects)
    (= (SET.FTN$source term-objects) name-set)
    (= (SET.FTN$target term-objects) set.mor$function)
    (= (SET.FTN$composition [term-objects set.mor$source]) term)
    (= (SET.FTN$composition [term-objects set.mor$target])
       (SET.FTN$composition [term set.obj$power]))
    (= (set.mor$composite [element term-objects])
       (set.mor$composite [element (SET.FTN$composition [term set.mor$singleton])]))
    (= (set.mor$composite [application term-objects])
       (set.mor$composite
        [(set.lim.prd3$tripling-parametric
          [(set.mor$composite [application (SET.FTN$composition [term set.mor$singleton])])
            (set.mor$composite [projection1 term-objects])
            (set.mor$composite [projection2 term-sequence-objects])])])
        (SET.FTN$composition [term set.obj$ternary-union])]))

(10) (SET.FTN$function term-sequence-objects)
    (= (SET.FTN$source term-sequence-objects) name-set)
    (= (SET.FTN$target term-sequence-objects) set.mor$function)
    (= (SET.FTN$composition [term-sequence-objects set.mor$source]) term-sequence)
    (= (SET.FTN$composition [term-sequence-objects set.mor$target])
       (SET.FTN$composition [term set.obj$power]))
    (= (set.mor$composite
       [(SET.FTN$composition [term set.obj.seq$concatenation]) term-sequence-objects])
       (set.mor$composite
        [(set.lim.prd2$pairing-parametric
          [(set.mor$composite
            [(SET.FTN$composition [term set.obj.seq$sequence-sequence-projection1])
              term-sequence-objects])
            (set.mor$composite
              [(SET.FTN$composition [term sequence-sequence-projection2])
                term-sequence-objects])])])
          (SET.FTN$composition [term set.obj$binary-union])]))
    (forall (?N (name-set ?N))
      (and (= (set.mor$composition [(set.obj.seq$empty (term ?N)) (term-sequence-objects ?N)])
              ((set.mor$element (set.obj$power (term ?N))) set.obj$empty)))
```

```
(11) (SET.FTN$function term-functions)
    (= (SET.FTN$source term-functions) name-set)
    (= (SET.FTN$target term-functions) set.mor$function)
    (= (SET.FTN$composition [term-functions set.mor$source]) term)
    (= (SET.FTN$composition [term-functions set.mor$target])
        (SET.FTN$composition [term set.obj$power]))
    (= (set.mor$composition [(application ?N) (term-functions ?N)])
        (set.mor$composite
            [(set.lim.prd3$tripling-parametric
                [(set.mor$composite [projection1 (SET.FTN$composition [term set.mor$singleton])])
                    (set.mor$composite [projection1 term-functions])
                    (set.mor$composite [projection2 term-sequence-functions])])
                (SET.FTN$composition [term set.obj$ternary-union])]))
    (= (set.mor$composite [element term-functions])
        ((set.mor$constant-parametric
            [term (SET.FTN$composition [term set.obj$power])]) set.obj$empty))

(12) (SET.FTN$function term-sequence-functions)
    (= (SET.FTN$source term-sequence-functions) name-set)
    (= (SET.FTN$target term-sequence-functions) set.mor$function)
    (= (SET.FTN$composition [term-sequence-functions set.mor$source]) term-sequence)
    (= (SET.FTN$composition [term-sequence-functions set.mor$target])
        (SET.FTN$composition [term set.obj$power]))
    (= (set.mor$composite
        [(SET.FTN$composition [term set.obj.seq$concatenation]) term-sequence-functions])
        (set.mor$composite
            [(set.lim.prd2$pairing-parametric
                [(set.mor$composite
                    [(SET.FTN$composition [term set.obj.seq$sequence-sequence-projection1])
                        term-sequence-functions])
                    (set.mor$composite
                        [(SET.FTN$composition [term set.obj.seq$sequence-sequence-projection2])
                            term-sequence-functions])])
                (SET.FTN$composition [term set.obj$binary-union])]))
    (forall (?N (name-set ?N))
        (= (set.mor$composition
            [(set.obj.seq$empty (term ?N)) (term-sequence-functions ?N)])
            ((set.mor$element (set.obj$power (term ?N)) set.obj$empty))


```

Booleans

A term is elementary when it is built by the element constructor. Let $elem(N) \subseteq trm(N)$ denote the set of elementary terms. A term is composite when it is built by the application constructor. Let $comp(N) \subseteq trm(N)$ denote the set of composite terms. A term must be either elementary or composite – the elementary and composite terms comprise a partition for the set of terms: $trm(N) = elem(N) + comp(N)$. These Boolean tests are used to define the domains of selectors.

```
(13) (SET.FTN$function elementary)
    (= (SET.FTN$source elementary) name-set)
    (= (SET.FTN$target elementary) set.obj$set)
    (= elementary (SET.FTN$composition [element set.mor$image])
        (set.obj$subset-parametric elementary term)

(14) (SET.FTN$function composite)
    (= (SET.FTN$source composite) name-set)
    (= (SET.FTN$target composite) set.obj$set)
    (= composite (SET.FTN$composition [application set.mor$image])
        (set$subset-parametric composite term)

(15) (= (set$binary-union-parametric [elementary composite]) term)
    (= (set$binary-intersection-parametric [elementary composite])
        (set.obj$empty-parametric name-set))


```

Selectors

Here we define one elementary selector, *name*, two composite selectors, *term* and *term-sequence*, and a combined selector called *resolution*. For elementary terms, the name selector returns the name used for the element constructor. For composite terms, the term and term-sequence selectors return the components used for the application constructor. The resolution selector is the pairing of the two. Hence, there are *name*, *term* and *term-sequence* partial functions

The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 6

May 9, 2004

$name(N) : trm(N) \rightarrow N$ with domain $elem(N)$,
 $trm(N) : trm(N) \rightarrow trm(N)$ with domain $comp(N)$,
 $trm-seq(L) : trm(L) \rightarrow trm-seq(L)$ with domain $comp(N)$,

and a *resolution* function

$res(N) = (trm(N), trm-seq(N)) : trm(N) \rightarrow trm(N) \times trm-seq(N)$.

- ```
(16) (SET.FTN$function select-name)
 (= (SET.FTN$source select-name) name-set)
 (= (SET.FTN$target select-name) set.pfn$partial-function)
 (= (SET.FTN$composition [select-name set.pfn$source]) term)
 (= (SET.FTN$composition [select-name set.pfn$domain] elementary)
 (SET.FTN$composition [select-name set.pfn$target] (SET.FTN$identity name-set)))

(17) (SET.FTN$function select-term)
 (= (SET.FTN$source select-term) name-set)
 (= (SET.FTN$target select-term) set.pfn$partial-function)
 (= (SET.FTN$composition [select-term set.pfn$source]) term)
 (= (SET.FTN$composition [select-term set.pfn$domain]) composite)
 (= (SET.FTN$composition [select-term set.pfn$target]) term)
 (= (set.pfn$composite [application select-term]) projection1)

(18) (SET.FTN$function select-term-sequence)
 (= (SET.FTN$source select-term-sequence) name-set)
 (= (SET.FTN$target select-term-sequence) set.pfn$partial-function)
 (= (SET.FTN$composition [select-term-sequence set.pfn$source]) term)
 (= (SET.FTN$composition [select-term-sequence set.pfn$domain]) composite)
 (= (SET.FTN$composition [tuple set.pfn$target]) term-sequence)
 (= (set.pfn$composite [application select-term-sequence]) projection2)

(19) (SET.FTN$function resolution)
 (= (SET.FTN$source resolution) name-set)
 (= (SET.FTN$target resolution) set.mor$function)
 (= (SET.FTN$composition [resolution set.mor$source]) composite)
 (= (SET.FTN$composition [resolution set.mor$target]) application-pair)
 (= (set.mor$composite [resolution application])
 (set.mor$inclusion-parametric [composite term]))
 (= (set.mor$composite [application resolution])
 (set.mor$identity-parametric application-pair))
```

## Sentences

### scl.snt

SCL syntax is obtained from conventional FO syntax by removing signature-related restrictions, so that both SCL atomic sentences and terms can be described uniformly as a term followed by a sequence of argument terms. Atoms or atomic sentences are the simplest kind of sentence. For any name set  $N$ , they come in three varieties: equations, relaters and role descriptions.

$$atm(N) = eqn(N) + rel(N) + rol(N).$$

Equations are atomic. Relaters are similar in structure to term applications. Role descriptions consist of sets of (role-name, term) pairs.

- ```
(1) (SET.FTN$function atom)
    (= (SET.FTN$source atom) name-set)
    (= (SET.FTN$target atom) set.obj$set)
    (= atom (set.obj$ternary-union-parametric [equation relater role-description]))
```

Equations are distinguished as a special category because of their special semantic role and special handling by many applications. Note that the equality sign is not a term. For any name set N , there are *term* projection (selector) functions

$$trm1_N : eqn(N) \rightarrow trm(N)$$

$$trm2_N : eqn(N) \rightarrow trm(N)$$

from equations to terms.

- ```
(2) (SET.FTN$function equation)
 (= (SET.FTN$source equation) name-set)
 (= (SET.FTN$target equation) set.obj$set)

(3) (SET.FTN$function term1)
 (= (SET.FTN$source term1) name-set)
 (= (SET.FTN$target term1) set.mor$function)
 (= (SET.FTN$composition [term1 set.mor$source]) equation)
 (= (SET.FTN$composition [term1 set.mor$target]) term) term)

(4) (SET.FTN$function term2)
 (= (SET.FTN$source term2) name-set)
 (= (SET.FTN$target term2) set.mor$function)
 (= (SET.FTN$composition [term2 set.mor$source]) equation)
 (= (SET.FTN$composition [term2 set.mor$target]) term) term)
```

A *relater*  $(R, \alpha)$  is a substituted relation  $R[\alpha]$ . It consists of two components, a term  $R \in trm(N)$  that denotes a relation and a term sequence  $\alpha \in trm-seq(N)$ . There is no constraint that the arity of the relation is equal to the length of the term sequence, since there is no arity and there is no relation. The set of relaters is the binary product of the set of terms and the set of term sequences  $rel(N) = trm(N) \times trm-seq(N)$ . For any name set  $N$ , there are *term* and *term sequence* selector functions

$$trm_N : rel(N) \rightarrow trm(N)$$

$$trm-seq_N : rel(N) \rightarrow trm-seq(N)$$

from relaters to terms and term sequences, respectively.

- ```
(5) (SET.FTN$function relater-diagram)
    (= (SET.FTN$source relater-diagram) scl.obj$name-set)
    (= (SET.FTN$target relater-diagram) set.lim.prd2$diagram)
    (= (SET.FTN$composition [relater-diagram set.lim.prd2$set1]) scl.obj.term$term)
    (= (SET.FTN$composition [relater-diagram set.lim.prd2$set2]) scl.obj.term$term-sequence)

(6) (SET.FTN$function relater)
    (= (SET.FTN$source relater) scl.obj$name-set)
    (= (SET.FTN$target relater) set.obj$set)
    (= relater (SET.FTN$composition [relater-diagram set.lim.prd2$binary-product]))

(7) (SET.FTN$function term)
    (= (SET.FTN$source term) scl.obj$name-set)
    (= (SET.FTN$target term) set.mor$function)
    (= (SET.FTN$composition [term set.mor$source])) relater)
```

The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 8

May 9, 2004

```
(= (SET.FTN$composition [term set.mor$target])) scl.obj.trm$term)
(= term (SET.FTN$composition [relater-diagram set.lim.prd2$projection1]))
```

```
(8) (SET.FTN$function term-sequence)
(= (SET.FTN$source term-sequence) scl.obj$name-set)
(= (SET.FTN$target term-sequence) set.mor$function)
(= (SET.FTN$composition [term-sequence set.mor$source])) relater)
(= (SET.FTN$composition [term-sequence set.mor$target])) scl.obj.trm$term-sequence)
(= term-sequence (SET.FTN$composition [relater-diagram set.lim.prd2$projection2]))
```

A *role pair* is a role-name and a term called the role-value. A *role-set* is a set of role pairs. An atomic sentence can be represented as a *role description* consisting of a unary *property* of a state of affairs, along with associated roles and fillers; that is, as a (term, role-set) pair. The set of role descriptions is the binary product of the set of terms and the power set of role pairs $rol-desc(N) = trm(N) \times \wp rol-pr(N)$.

For any name set N , there are *role-name* and *role-value* selector (projection) functions

$$name_N : rol-pr(N) \rightarrow N$$
$$val_N : rol-pr(N) \rightarrow trm(N)$$

from role pairs to names and terms, respectively.

For any name set N , there are *property* and *role set* selector (projection) functions

$$prop_N : rol-desc(N) \rightarrow trm(N)$$
$$rol-set_N : rol-desc(N) \rightarrow \wp rol-pr(N)$$

from role descriptions to terms and role sets, respectively.

```
(9) (SET.FTN$function role-pair-diagram)
(= (SET.FTN$source role-pair-diagram) scl.obj$name-set)
(= (SET.FTN$target role-pair-diagram) set.lim.prd2$diagram)
(= (SET.FTN$composition [role-pair-diagram set.lim.prd2$set1])
    (SET.FTN$identity scl.obj$name-set))
(= (SET.FTN$composition [role-pair-diagram set.lim.prd2$set2]) scl.obj.trm$term)

(10) (SET.FTN$function role-pair)
(= (SET.FTN$source role-pair) scl.obj$name-set)
(= (SET.FTN$target role-pair) set.obj$set)
(= role-pair (SET.FTN$composition [role-pair-diagram set.lim.prd2$binary-product]))

(11) (SET.FTN$function role-name)
(= (SET.FTN$source role-name) scl.obj$name-set)
(= (SET.FTN$target role-name) set.mor$function)
(= (SET.FTN$composition [role-name set.mor$source])) role-pair)
(= (SET.FTN$composition [role-name set.mor$target])) (SET.FTN$identity scl.obj$name-set))
(= role-name (SET.FTN$composition [role-pair-diagram set.lim.prd2$projection1]))

(12) (SET.FTN$function role-value)
(= (SET.FTN$source role-value) scl.obj$name-set)
(= (SET.FTN$target role-value) set.mor$function)
(= (SET.FTN$composition [role-value set.mor$source])) role-pair)
(= (SET.FTN$composition [role-value set.mor$target])) term)
(= role-value (SET.FTN$composition [role-pair-diagram set.lim.prd2$projection2]))

(13) (SET.FTN$function role-description-diagram)
(= (SET.FTN$source role-description-diagram) scl.obj$name-set)
(= (SET.FTN$target role-description-diagram) set.lim.prd2$diagram)
(= (SET.FTN$composition [role-description-diagram set.lim.prd2$set1]) scl.obj.trm$term)
(= (SET.FTN$composition [role-description-diagram set.lim.prd2$set2])
    (SET.FTN$composition [role-pair set.obj$power]))

(14) (SET.FTN$function role-description)
(= (SET.FTN$source role-description) scl.obj$name-set)
(= (SET.FTN$target role-description) set.obj$set)
(= role-description
    (SET.FTN$composition [role-description-diagram set.lim.prd2$binary-product]))

(15) (SET.FTN$function property)
(= (SET.FTN$source property) scl.obj$name-set)
(= (SET.FTN$target property) set.mor$function)
```

The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 9

May 9, 2004

```
(= (SET.FTN$composition [property set.mor$source])) role-description)
(= (SET.FTN$composition [property set.mor$target])) scl.obj.trm$term)
(= property (SET.FTN$composition [role-description-diagram set.lim.prd2$projection1]))

(16) (SET.FTN$function role-set)
(= (SET.FTN$source role-set) scl.obj$name-set)
(= (SET.FTN$target role-set) set.mor$function)
(= (SET.FTN$composition [role-set set.mor$source])) role-description)
(= (SET.FTN$composition [role-set set.mor$target]))
(SET.FTN$composition [role-pair set.obj$power]))
(= role-set (SET.FTN$composition [role-description-diagram set.lim.prd2$projection2]))
```

For any name-set N , the set of SCL sentences $\mathit{sent}(N)$ is defined inductively with the set of atoms as a base. The set of sentences is inductively defined as follows.

- Any atom $\alpha \in \mathit{atm}(N)$ is an *atomic sentence* $\diamond_N(\alpha) = \mathit{holds}(N)(\alpha) \in \mathit{sent}(N)$;
- The *negation* of any sentence $\varphi \in \mathit{sent}(N)$ is a *negative sentence* $\neg_N\varphi = \mathit{neg}(N)(\varphi) \in \mathit{sent}(N)$;
- The *conjunction* of any subset of sentences $\Phi \in \wp \mathit{sent}(N)$ is a *conjunctive sentence* $\wedge_N\Phi = \mathit{conj}(N)(\Phi) \in \mathit{sent}(N)$; and
- The (*existential*) *quantification* of any name-sentence pair $(x, \varphi) \in N \times \mathit{sent}(N)$ is a *quantified sentence* $\exists_N(x, \varphi) = \mathit{exists}(N)((x, \varphi)) \in \mathit{sent}(N)$; and
- Nothing else is a *sentence* (this requirement is enforced by the axioms for constructors and selectors).

The set of atoms is the source of the *holds* (constructor) function

$$\diamond_N = \mathit{holds}(N) : \mathit{atm}(N) \rightarrow \mathit{sent}(N),$$

which embeds atoms as sentences. Its range is the set of atomic sentences. The *atom* (selector) partial function

$$\diamond_N^{-1} = \mathit{atm}(N) : \mathit{sent}(N) \rightarrow \mathit{atm}(N)$$

returns the atom that makes up an atomic sentence; it is inverse to the atom constructor:

$$\mathit{holds}(N) \cdot \mathit{atm}(N) = \mathit{id}_{\mathit{atm}(N)},$$

$$\mathit{atm}(N) \cdot \mathit{holds}(N) = \mathit{id}_{\mathit{atm}\text{-}\mathit{sent}(N)},$$

implying that the holds function is injective.

```
(17) (SET.FTN$function holds)
(= (SET.FTN$source holds) scl.obj$name-set)
(= (SET.FTN$target holds) set.mor$injection)
(= (SET.FTN$composition [holds set.mor$source]) atom)
(= (SET.FTN$composition [holds set.mor$target]) sentence)

(18) (SET.FTN$function atomic-sentence)
(= (SET.FTN$source atomic-sentence) scl.obj$name-set)
(= (SET.FTN$target atomic-sentence) set.obj$set)
(= atomic-sentence (SET.FTN$composition [holds set.mor$range]))

(19) (SET.FTN$function select-atom)
(= (SET.FTN$source select-atom) scl.obj$name-set)
(= (SET.FTN$target select-atom) set.pfn$partial-function)
(= (SET.FTN$composition [select-atom set.pfn$source]) sentence)
(= (SET.FTN$composition [select-atom set.pfn$domain]) atomic-sentence)
(= (SET.FTN$composition [select-atom set.pfn$target]) atom)

(20) (= (set.pfn$composite [holds select-atom])
(SET.FTN$composition [atom set.mor$identity]))
(= (set.pfn$composite [select-atom holds])
(SET.FTN$composition [atomic-sentence set.mor$identity]))
```

The *conjunction* (constructor) function

$$\wedge_N = \mathit{conj}(N) : \wp \mathit{sent}(N) \rightarrow \mathit{sent}(N)$$

embeds subsets of sentences as sentences. Its range is the set of *conjunctive sentences*. The *sentence subset* (selector) partial function

$$\wedge_N^{-1} = \mathit{subset}(N) : \mathit{sent}(N) \rightarrow \wp \mathit{sent}(N)$$

returns the sentence subset underling a conjunctive sentence; it is inverse to the conjunction constructor:

$$\begin{aligned} \text{conj}(N) \cdot \text{subset}(N) &= id_{\emptyset \text{sent}(N)}, \\ \text{subset}(N) \cdot \text{conj}(N) &= id_{\text{conj-sent}(N)}, \end{aligned}$$

implying that the conjunction function is injective.

- ```
(21) (SET.FTN$function conjunction)
 (= (SET.FTN$source conjunction) scl.obj$name-set)
 (= (SET.FTN$target conjunction) set.mor$injection)
 (= (SET.FTN$composition [conjunction set.mor$source]
 (SET.FTN$composition [sentence set.obj$power])))
 (= (SET.FTN$composition [conjunction set.mor$target]) sentence)

(22) (SET.FTN$function conjunctive-sentence)
 (= (SET.FTN$source conjunctive-sentence) scl.obj$name-set)
 (= (SET.FTN$target conjunctive-sentence) set.obj$set)
 (= conjunctive-sentence (SET.FTN$composition [conjunction set.mor$range]))

(23) (SET.FTN$function select-sentence-subset)
 (= (SET.FTN$source select-sentence-subset) scl.obj$name-set)
 (= (SET.FTN$target select-sentence-subset) set.pfn$partial-function)
 (= (SET.FTN$composition [select-sentence-subset set.pfn$source] sentence)
 (SET.FTN$composition [select-sentence-subset set.pfn$domain] conjunctive-sentence)
 (SET.FTN$composition [select-sentence-subset set.pfn$target]
 (SET.FTN$composition [sentence set.obj$power])))

(24) (= (set.pfn$composite [conjunction select-sentence-subset])
 (SET.FTN$composition [(SET.FTN$composition
 [sentence set.obj$power]) set.mor$identity]))
 (= (set.pfn$composite [select-sentence-subset conjunction])
 (SET.FTN$composition [conjunctive-sentence set.mor$identity]))
```

The *negation* (constructor) function

$$\neg_N = \text{neg}(N) : \text{sent}(N) \rightarrow \text{sent}(N)$$

embeds sentences as sentences. Its range is the set of *negative sentences*. The *sentence* (selector) partial function

$$\neg^{-1}_N = \text{sent}(N) : \text{sent}(N) \rightarrow \text{sent}(N)$$

returns the *sentence* that underlies a *negative sentence*; it is inverse to the negation constructor:

$$\begin{aligned} \text{neg}(N) \cdot \text{sent}(N) &= id_{\text{sent}(N)}, \\ \text{sent}(N) \cdot \text{neg}(N) &= id_{\text{neg-sent}(N)}, \end{aligned}$$

implying that the negation function is injective.

- ```
(25) (SET.FTN$function negation)
      (= (SET.FTN$source negation) scl.obj$name-set)
      (= (SET.FTN$target negation) set.mor$injection)
      (= (SET.FTN$composition [negation set.mor$source] sentence)
          (SET.FTN$composition [negation set.mor$target] sentence))

(26) (SET.FTN$function negative-sentence)
      (= (SET.FTN$source negative-sentence) language)
      (= (SET.FTN$target negative-sentence) set.obj$set)
      (= negative-sentence (SET.FTN$composition [negation set.mor$range]))

(27) (SET.FTN$function select-sentence)
      (= (SET.FTN$source select-sentence) language)
      (= (SET.FTN$target select-sentence) set.pfn$partial-function)
      (= (SET.FTN$composition [select-sentence set.pfn$source] sentence)
          (SET.FTN$composition [select-sentence set.pfn$domain] negative-sentence)
          (SET.FTN$composition [select-sentence set.pfn$target] sentence))

(28) (= (set.pfn$composite [negation select-sentence])
        (SET.FTN$composition [sentence set.mor$identity]))
      (= (set.pfn$composite [select-sentence negation])
          (SET.FTN$composition [negative-sentence set.mor$identity]))
```

The (*existential*) *quantification* (constructor) function

The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 11

May 9, 2004

$$\exists_N = \text{exists}(N) : N \times \text{sent}(N) \rightarrow \text{sent}(N),$$

embeds quantification (variable, sentence) pairs as sentences. Its range is the set of *quantified sentences*. The *pair* (selector) partial function

$$\exists^{-1}_N = \text{pr}(N) : \text{sent}(N) \rightarrow N \times \text{sent}(N)$$

returns the quantification pair that underlies a quantified sentence; it is inverse to the quantification constructor:

$$\text{exists}(N) \cdot \text{pr}(N) = \text{id}_{N \times \text{sent}(N)},$$

$$\text{pr}(N) \cdot \text{exists}(N) = \text{id}_{\text{quant-sent}(N)},$$

implying that the quantification function is injective. It is convenient to also define selectors for the variable and body of a quantified sentence. There is a variable and body (selector) partial functions

$$\text{var}(N) : \text{sent}(N) \rightarrow N,$$

$$\text{body}(N) : \text{sent}(N) \rightarrow \text{sent}(N).$$

The pair selector is the pairing of the variable and body selectors.

- ```
(29) (SET.FTN$function quantification-pair-diagram)
 (= (SET.FTN$source quantification-pair-diagram) name-set)
 (= (SET.FTN$target quantification-pair-diagram) set.lim.prd2$diagram)
 (= (SET.FTN$composition [quantification-pair-diagram set.lim.prd2$set1])
 (SET.FTN$identity scl.obj$name-set))
 (= (SET.FTN$composition [quantification-pair-diagram set.lim.prd2$set2]) sentence)

(30) (SET.FTN$function quantification-pair)
 (= (SET.FTN$source quantification-pair) name-set)
 (= (SET.FTN$target quantification-pair) set.obj$set)
 (= quantification-pair
 (SET.FTN$composition [quantification-pair-diagram set.lim.prd2$binary-product]))

(31) (SET.FTN$function name-projection)
 (= (SET.FTN$source name-projection) name-set)
 (= (SET.FTN$target name-projection) set.mor$function)
 (= (SET.FTN$composition [name-projection set.mor$source]) quantification-pair)
 (= (SET.FTN$composition [name-projection set.mor$target])
 (SET.FTN$identity scl.obj$name-set))
 (= name-projection
 (SET.FTN$composition [quantification-pair-diagram set.lim.prd2$projection1]))

(32) (SET.FTN$function sentence-projection)
 (= (SET.FTN$source sentence-projection) name-set)
 (= (SET.FTN$target sentence-projection) set.mor$function)
 (= (SET.FTN$composition [sentence-projection set.mor$source]) quantification-pair)
 (= (SET.FTN$composition [sentence-projection set.mor$target]) sentence)
 (= sentence-projection
 (SET.FTN$composition [quantification-pair-diagram set.lim.prd2$projection2]))

(33) (SET.FTN$function quantification)
 (= (SET.FTN$source quantification) scl.obj$name-set)
 (= (SET.FTN$target quantification) set.mor$injection)
 (= (SET.FTN$composition [quantification set.mor$source]) quantification-pair)
 (= (SET.FTN$composition [quantification set.mor$target]) sentence)

(34) (SET.FTN$function quantified-sentence)
 (= (SET.FTN$source quantified-sentence) scl.obj$name-set)
 (= (SET.FTN$target quantified-sentence) set.obj$set)
 (= quantified-sentence (SET.FTN$composition [quantification set.mor$range]))

(35) (SET.FTN$function select-quantification-pair)
 (= (SET.FTN$source select-quantification-pair) scl.obj$name-set)
 (= (SET.FTN$target select-quantification-pair) set.pfn$partial-function)
 (= (SET.FTN$composition [select-quantification-pair set.pfn$source]) sentence)
 (= (SET.FTN$composition [select-quantification-pair set.pfn$domain]) quantified-sentence)
 (= (SET.FTN$composition [select-quantification-pair set.pfn$target]) quantification-pair)
 (= select-quantification-pair
 (set.lim.prd2$pairing-parametric [variable body]))
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 12

May 9, 2004

```
(36) (= (set.pfn$composite [quantification select-quantification-pair])
 (SET.FTN$composition [quantification-pair set.mor$identity]))
 (= (set.pfn$composite [select-quantification-pair quantification])
 (SET.FTN$composition [quantified-sentence set.mor$identity]))

(37) (SET.FTN$function variable)
 (= (SET.FTN$source variable) scl.obj$name-set)
 (= (SET.FTN$target variable) set.pfn$partial-function)
 (= (SET.FTN$composition [variable set.pfn$source]) sentence)
 (= (SET.FTN$composition [variable set.pfn$domain]) quantified-sentence)
 (= (SET.FTN$composition [variable set.pfn$target])
 (SET.FTN$identity scl.obj$name-set))

(38) (SET.FTN$function body)
 (= (SET.FTN$source body) scl.obj$name-set)
 (= (SET.FTN$target body) set.pfn$partial-function)
 (= (SET.FTN$composition [body set.pfn$source]) sentence)
 (= (SET.FTN$composition [body set.pfn$domain]) quantified-sentence)
 (= (SET.FTN$composition [body set.pfn$target]) sentence)
```

A *sentence* is an atomic sentence, a negative sentence, a conjunctive sentence, or a quantified sentence; but nothing else. Atomic sentences, negative sentences, conjunctive sentences and quantified sentences partition the set of sentences. The holds, negation, conjunction and quantification synthetic/constructor functions map the set of atoms, the set of sentences, the set of sentence subsets, and the set of quantification pairs into the set of sentences.

```
(39) (SET.FTN$function sentence)
 (= (SET.FTN$source sentence) scl.obj$name-set)
 (= (SET.FTN$target sentence) set.obj$set)

(40) (forall (?N (scl.obj$name-set ?N))
 (set.obj$quaternary-partition
 (sentence ?1) [(atomic-sentence ?1)
 (negative-sentence ?1)
 (conjunctive-sentence ?1)
 (quantified-sentence ?1)]))
```

A *text* is a subset of sentences. Texts are used for the contextual definition of signatures (vocabularies).

```
(41) (SET.FTN$function text)
 (= (SET.FTN$source text) scl.obj$name-set)
 (= (SET.FTN$target text) set.obj$set)
 (= text (SET.FTN$composition [sentence set.obj$power]))
```

## Vocabulary: Objects, Functions and Relations

A role set contains the objects of its role-value components. A role set contains its role-name as a single function and contains the functions of its role-value components. A role set contains no relations. There is a *role pair objects* function and a *role pair functions* function

$$objs(N) : role-pr(N) \rightarrow \wp trm(N),$$

$$ftns(N) : role-pr(N) \rightarrow \wp trm(N),$$

and a *role set objects* function and a *role set functions* function:

$$objs(N) : role-set(N) \rightarrow \wp trm(N),$$

$$ftns(N) : role-set(N) \rightarrow \wp trm(N).$$

```
(42) (SET.FTN$function role-pair-objects)
 (= (SET.FTN$source role-pair-objects) name-set)
 (= (SET.FTN$target role-pair-objects) set.mor$function)
 (= (SET.FTN$composition [role-pair-objects set.mor$source]) role-pair)
 (= (SET.FTN$composition [role-pair-objects set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= role-pair-objects (set.mor$composite [role-value scl.obj.trm$term-objects]))

(43) (SET.FTN$function role-pair-functions)
 (= (SET.FTN$source role-pair-functions) name-set)
 (= (SET.FTN$target role-pair-functions) set.mor$function)
 (= (SET.FTN$composition [role-pair-functions set.mor$source]) role-pair)
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 13

May 9, 2004

```
(= (SET.FTN$composition [role-pair-functions set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= role-pair-functions
 (set.mor$composite
 [(set.lim.prd2$pairing
 [(set.mor$composite
 [(set.mor$composite [role-name scl.obj.trm$element])
 (SET.FTN$composition [scl.obj.trm$term set.mor$singleton])])
 (set.mor$composite [role-value scl.obj.trm$term-functions])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))))

(44) (SET.FTN$function role-set-objects)
(= (SET.FTN$source role-set-objects) name-set)
(= (SET.FTN$target role-set-objects) set.mor$function)
(= (SET.FTN$composition [role-set-objects set.mor$source]) role-set)
(= (SET.FTN$composition [role-set-objects set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= role-set-objects
 (set.mor$composite
 [(SET.FTN$composition [role-pair-objects set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))

(45) (SET.FTN$function role-set-functions)
(= (SET.FTN$source role-set-functions) name-set)
(= (SET.FTN$target role-set-functions) set.mor$function)
(= (SET.FTN$composition [role-set-functions set.mor$source]) role-set)
(= (SET.FTN$composition [role-set-functions set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= (role-set-functions ?N)
 (set.mor$composition
 [(SET.FTN$composition [role-pair-functions set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))
```

Any atom contains a set of objects (subterms). An equation contains the objects in its two component terms. A relater contains the objects in its term and term-sequence components. A role description contains the objects of its property component and the objects of each role-pair in its role-set component. A similar argument holds for functions (terms in function position). Any atom contains a set of functions. An equation contains the functions in its two component terms. A relater contains the functions in its term and term-sequence components. A role description contains the functions of its property component and the functions of each role-pair in its role-set component. Any atom may contain a single relation (terms in relation position). An equation contains no relations. A relater contains its term as its single relation. A role description contains no relations. There is an *atom objects* function, an *atom functions* function and an *atom relations* function:

$$objs(N) : atm(N) \rightarrow \wp trm(N),$$

$$fns(N) : atm(N) \rightarrow \wp trm(N),$$

$$rels(N) : atm(N) \rightarrow \wp trm(N).$$

```
(46) (SET.FTN$function atom-objects)
(= (SET.FTN$source atom-objects) name-set)
(= (SET.FTN$target atom-objects) set.mor$function)
(= (SET.FTN$composition [atom-objects set.mor$source]) atom)
(= (SET.FTN$composition [atom-objects set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [equation atom]) atom-objects])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [term1 scl.obj.trm$term-objects])
 (set.mor$composite [term2 scl.obj.trm$term-objects])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [relater atom]) atom-objects])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [term scl.obj.trm$term-objects])
 (set.mor$composite [term-sequence scl.obj.trm$term-sequence-objects])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))))
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 14

May 9, 2004

```
(= (set.mor$composite
 [(set.mor$inclusion-parametric [role-description atom]) atom-objects])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [property scl.obj.trm$term-objects])
 (set.mor$composite [role-set role-set-objects])])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))

(47) (SET.FTN$function atom-functions)
(= (SET.FTN$source atom-functions) name-set)
(= (SET.FTN$target atom-functions) set.mor$function)
(= (SET.FTN$composition [atom-functions set.mor$source]) atom)
(= (SET.FTN$composition [atom-functions set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [equation atom]) atom-functions])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [term1 scl.obj.trm$term-functions])
 (set.mor$composite [term2 scl.obj.trm$term-functions])])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [relater atom]) atom-functions])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [term scl.obj.trm$term-functions])
 (set.mor$composite [term-sequence scl.obj.trm$term-sequence-functions])])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [role-description atom]) atom-functions])
 (set.mor$composite
 [(set.lim.prd2$pairing-parametric
 [(set.mor$composite [property scl.obj.trm$term-functions])
 (set.mor$composite [role-set role-set-functions])])])
 (SET.FTN$composition [scl.obj.trm$term set.obj$binary-union])]))

(48) (SET.FTN$function atom-relations)
(= (SET.FTN$source atom-relations) name-set)
(= (SET.FTN$target atom-relations) set.mor$function)
(= (SET.FTN$composition [atom-relations set.mor$source]) atom)
(= (SET.FTN$composition [atom-relations set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [equation atom]) atom-relations])
 ((set.mor$constant-parametric
 [equation (SET.FTN$composition [term set.obj$power])]) set.obj$empty))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [relater atom]) atom-relations])
 (set.mor$composite [term (SET.FTN$composition [scl.obj.trm$term set.mor$singleton])]))
(= (set.mor$composite
 [(set.mor$inclusion-parametric [role-description atom]) atom-relations])
 ((set.mor$constant-parametric
 [role-description (SET.FTN$composition [term set.obj$power])]) set.obj$empty))
```

Any sentence contains a set of objects (subterms). The objects of an atom are defined above. A negation contains the objects in its sentence component. A conjunction contains the objects of its sentence subset component. A quantification contains the objects of its quantification pair. Similar arguments hold for the functions (terms in function position) and relations (terms in relation position) for any sentence.

```
(49) (SET.FTN$function sentence-objects)
(= (SET.FTN$source sentence-objects) name-set)
(= (SET.FTN$target sentence-objects) set.mor$function)
(= (SET.FTN$composition [sentence-objects set.mor$source]) sentence)
(= (SET.FTN$composition [sentence-objects set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
(= (set.mor$composite [holds sentence-objects]) atom-objects)
(= (set.mor$composite [negation sentence-objects]) sentence-objects)
(= (set.mor$composite [conjunction sentence-objects])
 (set.mor$composite
 [(SET.FTN$composition [sentence-objects set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 15

May 9, 2004

```
(= (set.mor$composite [quantification sentence-objects])
 (set.mor$composite [sentence-projection sentence-objects]))

(50) (SET.FTN$function sentence-functions)
 (= (SET.FTN$source sentence-functions) name-set)
 (= (SET.FTN$target sentence-functions) set.mor$function)
 (= (SET.FTN$composition [sentence-functions set.mor$source]) sentence)
 (= (SET.FTN$composition [sentence-functions set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= (set.mor$composite [holds sentence-functions]) atom-functions)
 (= (set.mor$composite [negation sentence-functions]) sentence-functions)
 (= (set.mor$composite [conjunction sentence-functions])
 (set.mor$composite
 [(SET.FTN$composition [sentence-functions set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))
 (= (set.mor$composite [quantification sentence-functions])
 (set.mor$composite [sentence-projection sentence-functions]))

(51) (SET.FTN$function sentence-relations)
 (= (SET.FTN$source sentence-relations) name-set)
 (= (SET.FTN$target sentence-relations) set.mor$function)
 (= (SET.FTN$composition [sentence-relations set.mor$source]) sentence)
 (= (SET.FTN$composition [sentence-relations set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= (set.mor$composite [holds sentence-relations]) atom-relations)
 (= (set.mor$composite [negation sentence-relations]) sentence-relations)
 (= (set.mor$composite [conjunction sentence-relations])
 (set.mor$composite
 [(SET.FTN$composition [sentence-relations set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))
 (= (set.mor$composite [quantification sentence-relations])
 (set.mor$composite [sentence-projection sentence-relations]))
```

Any text contains a set of objects (subterms), a set of functions (terms in function position), and a set of relations (terms in relational position). The objects of a text are the objects of the sentences in the text. A similar argument holds for functions and relations.

```
(52) (SET.FTN$function text-objects)
 (= (SET.FTN$source text-objects) name-set)
 (= (SET.FTN$target text-objects) set.mor$function)
 (= (SET.FTN$composition [text-objects set.mor$source]) text)
 (= (SET.FTN$composition [text-objects set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= text-objects
 (set.mor$composite
 [(SET.FTN$composition [sentence-objects set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))

(53) (SET.FTN$function text-functions)
 (= (SET.FTN$source text-functions) name-set)
 (= (SET.FTN$target text-functions) set.mor$function)
 (= (SET.FTN$composition [text-functions set.mor$source]) text)
 (= (SET.FTN$composition [text-functions set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= text-functions
 (set.mor$composite
 [(SET.FTN$composition [sentence-functions set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))

(54) (SET.FTN$function text-relations)
 (= (SET.FTN$source text-relations) name-set)
 (= (SET.FTN$target text-relations) set.mor$function)
 (= (SET.FTN$composition [text-relations set.mor$source]) text)
 (= (SET.FTN$composition [text-relations set.mor$target])
 (SET.FTN$composition [scl.obj.trm$term set.obj$power]))
 (= text-relations
 (set.mor$composite
 [(SET.FTN$composition [sentence-relations set.mor$power])
 (SET.FTN$composition [scl.obj.trm$term set.obj$union])]))
```

For any text  $T$ , the (concrete) *vocabulary*  $voc(T) = \langle obj(T), ftn(T), rel(T) \rangle$  of  $T$  consists of

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 16

May 9, 2004

- $obj(T) \subseteq name(T)$ , the set of names in *object-position* (the set of terms of  $T$ ).
- $ftn(T) \subseteq name(T)$ , the set of names in *function-position*.
- $rel(T) \subseteq name(T)$ , is the set of names in *relation-position*.

```
(55) (SET.FTN$function vocabulary)
 (= (SET.FTN$source vocabulary) name-set)
 (= (SET.FTN$target vocabulary) set.mor$function)
 (= (SET.FTN$composition [vocabulary set.mor$source]) text)
 (= (SET.FTN$composition [vocabulary set.mor$target]) scl.obj.voc$vocabulary)
 (= (set.mor$composite [vocabulary scl.obj.voc$object]) text-objects)
 (= (set.mor$composite [vocabulary scl.obj.voc$function]) text-functions)
 (= (set.mor$composite [vocabulary scl.obj.voc$relation]) text-relations)
```

## SCL Semantics

The semantics axiomatize in this section is called the ‘moderately’ folded semantics. The concept of a vocabulary base is associated with the ‘minimally’ folded semantics. The moderately folded semantics suggests using a special class to denote the universe in translating from traditional FOL to SCL. This may lead to recognition of an adjunction, rather than an inverse relationship, between traditional FOL and the SCL.

### Vocabularies

`scl.obj.voc`

Table 2 lists the 29 terms for the semantics of the IFF-SCL – 13 in the vocabulary namespace and 16 in the interpretation namespace.

**Table 2: Terminology for the Objective Semantic Aspect of the SCL (meta) Ontology**

|                                          | Other                          | Object                                                                 | Morphism                                                 |
|------------------------------------------|--------------------------------|------------------------------------------------------------------------|----------------------------------------------------------|
| <b>scl</b><br><b>.obj</b><br><b>.voc</b> |                                | vocabulary                                                             |                                                          |
|                                          |                                | object object-base<br>function function-base<br>relation relation-base | choice resolution<br>function-selector sequence-selector |
|                                          |                                | elementary composite                                                   |                                                          |
|                                          |                                |                                                                        |                                                          |
| <b>scl</b><br><b>.obj</b><br><b>.int</b> |                                | interpretation                                                         |                                                          |
|                                          |                                | vocabulary<br>universe                                                 | object function relation<br>function-fold relation-fold  |
|                                          |                                |                                                                        | indication projection application                        |
|                                          | name-value-diagram             | name-value                                                             | name-map                                                 |
|                                          | legal-sentence<br>satisfaction |                                                                        |                                                          |

For any name-set  $N$ , an (abstract) *vocabulary*  $V = \langle obj_N(V), ftn_N(V), rel_N(V) \rangle \in voc(N)$  consists of

- $obj_N(V) \subseteq trm(N)$ , a set of terms,
  - $ftn_N(V) \subseteq obj(V)$ , a set of terms (in function-position), and
  - $rel_N(V) \subseteq obj(V)$ , a the set of terms (in relation-position).
- (1) (SET.FTN\$function vocabulary)  
 (= (SET.FTN\$source vocabulary) scl.obj\$name-set)  
 (= (SET.FTN\$target vocabulary) SET\$class)
  - (2) (KIF\$function object)  
 (= (KIF\$source object) scl.obj\$name-set)  
 (= (KIF\$target object) SET.FTN\$function)  
 (forall (?N (scl.obj\$name-set ?N))  
 (and (= (SET.FTN\$source (object ?N)) (vocabulary ?N))  
 (= (SET.FTN\$target (object ?N)) set.obj\$set)  
 (forall (?V ((vocabulary ?N) ?V))  
 (set.obj\$subset ((object ?N) ?V) (scl.obj.trm\$term ?N))))))
  - (3) (KIF\$function function)  
 (= (KIF\$source function) scl.obj\$name-set)  
 (= (KIF\$target function) SET.FTN\$function)  
 (forall (?N (scl.obj\$name-set ?N))  
 (and (= (SET.FTN\$source (function ?N)) (vocabulary ?N))  
 (= (SET.FTN\$target (function ?N)) set.obj\$set)  
 (forall (?V ((vocabulary ?N) ?V))  
 (set.obj\$subset ((function ?N) ?V) ((object ?N) ?V))))))
  - (4) (KIF\$function relation)  
 (= (KIF\$source relation) scl.obj\$name-set)  
 (= (KIF\$target relation) SET.FTN\$function)  
 (forall (?N (scl.obj\$name-set ?N))  
 (and (= (SET.FTN\$source (relation ?N)) (vocabulary ?N))  
 (= (SET.FTN\$target (relation ?N)) set.obj\$set)  
 (forall (?V ((vocabulary ?N) ?V))  
 (set.obj\$subset ((relation ?N) ?V) ((object ?N) ?V))))))

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 18

May 9, 2004

An object term is elementary when it is an elementary term. Let  $elem_N(V) \subseteq obj_N(V)$  denote the set of elementary objects of a vocabulary. An object term is composite when it is a composite term. Let  $comp_M(V) \subseteq obj_M(V)$  denote the set of composite objects. An object must be either elementary or composite – the elementary and composite objects comprise a partition for the set of objects:  $comp_M(V) = obj_M(V) - elem_N(V)$ ;  $obj_M(V) = elem_N(V) + comp_M(V)$ .

- ```
(5) (KIF$function elementary)
    (= (KIF$source elementary) scl.obj$name-set)
    (= (KIF$target elementary) SET.FTN$function)
    (forall (?N (scl.obj$name-set ?N))
      (and (= (SET.FTN$source (elementary ?N)) (vocabulary ?N))
            (= (SET.FTN$target (elementary ?N)) set.obj$set)
            (forall (?V ((vocabulary ?N) ?V))
              (= ((elementary ?N) ?V)
                  (set.obj$binary-intersection [((object ?N) ?V) (scl.obj.trm$elementary ?N)]))))))

(6) (KIF$function composite)
    (= (KIF$source composite) scl.obj$name-set)
    (= (KIF$target composite) SET.FTN$function)
    (forall (?N (scl.obj$name-set ?N))
      (and (= (SET.FTN$source (composite ?N)) (vocabulary ?N))
            (= (SET.FTN$target (composite ?N)) set.obj$set)
            (forall (?V ((vocabulary ?N) ?V))
              (= ((composite ?N) ?V)
                  (set.obj$binary-intersection
                    [((object ?N) ?V) (scl.obj.trm$composite ?N)]))))))

(7) (forall (?N (scl.obj$name-set ?N))
      ?V ((vocabulary ?N) ?V))
    (and (= (set$binary-union [((elementary ?N) ?V) ((composite ?N) ?V)])
           ((object ?N) ?V))
          (= (set$binary-intersection [((elementary ?N) ?V) ((composite ?N) ?V)]) set$empty)))
```

For any name-set N , the *vocabulary base* $B = \langle obj\text{-base}_N(V), ftn\text{-base}_N(V), rel\text{-base}_N(V) \rangle$ for a vocabulary $V \in voc(N)$ is a triple of sets, where the function and relation bases $ftn\text{-base}(V)$ and $rel\text{-base}(V)$ are the function and relation elementary terms (names), and the object base are the elementary object terms that appear as arguments (the coordinates of the sequence selections of composite object terms).

- $obj\text{-base}_N(V) \subseteq obj_N(V) \cap N$, a set of names (in object-position),
- $ftn\text{-base}_N(V) = ftn_N(V) \cap N$, a set of names (in function-position), and
- $rel\text{-base}_N(V) = rel_N(V) \cap N$, a the set of names (in relation-position).

Any vocabulary can be recursively generated with its base. Any two vocabularies with the same base are equal. A vocabulary base, not a vocabulary, was the original notion of vocabulary in the SCL standards document.

- ```
(8) (KIF$function object-base)
 (= (KIF$source object-base) scl.obj$name-set)
 (= (KIF$target object-base) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (object-base ?N)) (vocabulary ?N))
 (= (SET.FTN$target (object-base ?N)) set.obj$set)
 (forall (?V ((vocabulary ?N) ?V))
 (set.obj$subset
 ((object-base ?N) ?V)
 (set.obj$binary-intersection [((object ?N) ?V) ?N])))))

(9) (KIF$function function-base)
 (= (KIF$source function-base) scl.obj$name-set)
 (= (KIF$target function-base) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (function-base ?N)) (vocabulary ?N))
 (= (SET.FTN$target (function-base ?N)) set.obj$set)
 (forall (?V ((vocabulary ?N) ?V))
 (= ((function-base ?N) ?V)
 (set.obj$binary-intersection [((function ?N) ?V) ?N])))))

(10) (KIF$function relation-base)
```

```

(= (KIF$source relation-base) scl.obj$name-set)
(= (KIF$target relation-base) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (relation-base ?N)) (vocabulary ?N))
 (= (SET.FTN$target (relation-base ?N)) set.obj$set)
 (forall (?V ((vocabulary ?N) ?V))
 (= ((relation-base ?N) ?V)
 (set.obj$binary-intersection [((relation ?N) ?V) ?N])))
))
(11) (forall (?N (scl.obj$name-set ?N)
 ?V1 ((vocabulary ?N) ?V1)
 ?V2 ((vocabulary ?N) ?V2))
 (=> (and (= ((object-base ?N) ?V1) ((object-base ?N) ?V2))
 (= ((function-base ?N) ?V1) ((function-base ?N) ?V2))
 (= ((relation-base ?N) ?V1) ((relation-base ?N) ?V2)))
 (= ?V1 ?V2)))

```

The set of composite terms  $comp_N(V)$  are the terms of the form  $t[\tau] = (t, \tau)$  for  $N$ -term  $t$  and  $N$ -term sequence  $\tau$ . We make the two requirements:  $t \in ftn_N(V)$  and  $\tau \in seq(obj_N(V))$ , that  $t$  is a function term and  $\tau$  is a sequence of object terms. That is, for any composite term  $t[\tau] \in comp_N(V)$ , the term component is in function position  $t \in ftn_N(V)$  and the coordinates of the term sequence  $\tau = (t_0, t_1, \dots, t_{n-1})$  are in object position  $t_k \in obj_N(V)$  for all  $0 \leq k < n$ . We regard the term  $t$  as a symbol that denotes a function on object terms, mapping the  $obj_N(V)$ -sequence  $\tau$  to the object term  $t[\tau] = (t, \tau)$ . Depending on the vocabulary  $V$ , there may be many such functions. We choose one and record this with the *choice* function

$$choice_N(V) : ftn_N(V) \rightarrow ftn(obj_N(V))$$

from the function terms of the vocabulary  $V$  to the actual functions on sequences of object terms. Closely associated with this is a *resolution* injection

$$res_N(V) : comp_N(V) \rightarrow dom\text{-}coprod(obj_N(V))$$

from the composite terms of the vocabulary  $V$  to the coproduct of the domain tuple of functions on sequences of object terms. Since the application of the resolution of a composite term is itself, the resolution function is almost left inverse to the application function for the domain coproduct:

$$res_N(V) \cdot apply(obj_N(V)) = incl_{comp(V), obj(V)}$$

That is, the inclusion map of  $comp_N(V)$  into the set of object terms  $obj_N(V)$  factors through the domain coproduct  $dom\text{-}coprod(obj_N(V))$  for the set of object terms  $obj_N(V)$ . The requirements that composite object terms resolve into function terms and object term sequences means that the composition of the resolution map with the indication and projection functions for the coproduct factor through  $ftn_N(V)$  and  $seq(obj_N(V))$ , respectively. This means that from  $comp_N(V)$  there are two maps – a *function* selector to  $ftn_N(V)$  and a *sequence* selector to  $seq(obj_N(V))$ , respectively

$$ftn_N(V) : comp_N(V) \rightarrow ftn_N(V),$$

$$seq_N(V) : comp_N(V) \rightarrow seq(obj_N(V)).$$

satisfying the constraints:

$$ftn_N(V) \cdot choice_N(V) = res_N(V) \cdot indic(obj_N(V)) \text{ and } seq_N(V) = res_N(V) \cdot proj(obj_N(V)).$$

The function and sequence selectors return the components of composite object terms. We further require that the function selector be surjective. This means that any term in  $ftn_N(V)$  occurs in function position in some term in  $obj_N(V)$ . We do not require that the sequence selector be surjective; that is, there may be sequences of terms in  $obj_N(V)$  that do not appear in object position sequence. The resolution map (and the function and sequence selectors) is important for defining interpretations.

```

(12) (KIF$function choice)
 (= (KIF$source choice) scl.obj$name-set)

```

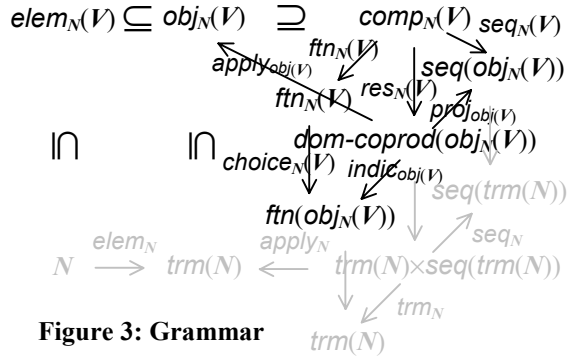


Figure 3: Grammar

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 20

May 9, 2004

```
(= (KIF$target choice) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (choice ?N)) (vocabulary ?N))
 (= (SET.FTN$target (choice ?N)) set.mor$function)))
(= (SET.FTN$composition [(choice ?N) set.mor$source]) (function ?N))
(= (SET.FTN$composition [(choice ?N) set.mor$target])
 (SET.FTN$composition [(object ?N) set.obj.seq$function])))

(13) (KIF$function resolution)
(= (KIF$source resolution) scl.obj$name-set)
(= (KIF$target resolution) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (resolution ?N)) (vocabulary ?N))
 (= (SET.FTN$target (resolution ?N)) set.mor$function)))
(= (SET.FTN$composition [(resolution ?N) set.mor$source]) (composite ?N))
(= (SET.FTN$composition [(resolution ?N) set.mor$target])
 (SET.FTN$composition [(object ?N) set.obj.seq$domain-coproduct])))

(14) (KIF$function function-selector)
(= (KIF$source function-selector) scl.obj$name-set)
(= (KIF$target function-selector) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (function-selector ?N)) (vocabulary ?N))
 (= (SET.FTN$target (function-selector ?N)) set.mor$function)))
(= (SET.FTN$composition [(function-selector ?N) set.mor$source]) (composite ?N))
(= (SET.FTN$composition [(function-selector ?N) set.mor$target]) (function ?N)))

(15) (KIF$function sequence-selector)
(= (KIF$source sequence-selector) scl.obj$name-set)
(= (KIF$target sequence-selector) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (sequence-selector ?N)) (vocabulary ?N))
 (= (SET.FTN$target (sequence-selector ?N)) set.mor$function)))
(= (SET.FTN$composition [(sequence-selector ?N) set.mor$source]) (composite ?N))
(= (SET.FTN$composition [(sequence-selector ?N) set.mor$target])
 (SET.FTN$composition [(object ?N) set.obj.seq$sequence])))

(16) (forall (?N (scl.obj$name-set ?N)
 ?V ((vocabulary ?N) ?V))
 (and (= (set.mor$composition
 [(resolution ?N) ?V] (set.obj.seq$indication ((object ?N) ?V))))
 (set.mor$composition
 [(function-selector ?N) ?V] ((choice ?N) ?V))))
 (= (set.mor$composition
 [(resolution ?N) ?V] (set.obj.seq$projection ((object ?N) ?V))))
 ((sequence-selector ?N) ?V))))
```

Note that the only constraint on  $rel_N(V)$  is that it be a subset of the set of objects terms  $obj_N(V)$ .

## Interpretations

scl.obj.int

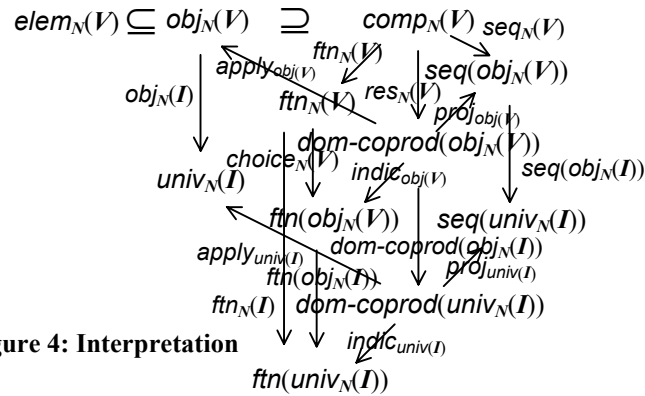


Figure 4: Interpretation

An interpretation  $I \in int_N$  for a vocabulary  $V = \langle obj_N(V), ftn_N(V), rel_N(V) \rangle \in voc_N$  has a nonempty set  $univ_N(I)$  called the universe, three functions

$$obj_N(I) : obj_N(V) \rightarrow univ_N(I), ftn_N(I) : ftn_N(V) \rightarrow ftn(univ_N(I)), \text{ and } rel_N(I) : rel_N(V) \rightarrow rel(univ_N(I)),$$

and two partial functions (folds)

$$function_N(I) : univ_N(I) \rightarrow ftn(univ_N(I)) \text{ and } relation_N(I) : univ_N(I) \rightarrow rel(univ_N(I)),$$

where

- o the value of  $obj_N(I)$  on any composite term  $t[\tau] \in comp_N(V)$  is the value of  $ftn_N(I)(t)$  applied to the sequential extension  $seq(obj_N(I))(\tau)$ ,
- o the restriction of  $rel_N(I)$  to  $ftn_N(V) \cap rel_N(V)$  is equal to the restriction of  $ftn_N(I)$  to  $ftn_N(V) \cap rel_N(V)$  composed with the function to relation embedding  $ftn2rel(univ_N(I))$ ,
- o the image of  $ftn_N(V)$  under  $obj_N(I)$  is a subset of the domain of  $function_N(I)$

$$\wp obj_N(I)(ftn_N(V)) \subseteq dom(function_N(I))$$

and  $function_N(I)(obj_N(I)(t)) = ftn_N(I)(t)$  for any  $t \in ftn_N(V)$

$$obj_N(I) \cdot ftn(function_N(I)) = ftn_N(I),$$

- o the image of  $rel_N(V)$  under  $obj_N(I)$  is a subset of the domain of  $relation_N(I)$

$$\wp obj_N(I)(rel_N(V)) \subseteq dom(relation_N(I))$$

and  $relation_N(I)(obj_N(I)(t)) = rel_N(I)(t)$  for any  $t \in rel_N(V)$

$$obj_N(I) \cdot ftn(relation_N(I)) = rel_N(I).$$

- (1) (SET.FTN\$function interpretation)  
 (= (SET.FTN\$source interpretation) scl.obj\$name-set)  
 (= (SET.FTN\$target interpretation) SET\$class)
- (2) (KIF\$function vocabulary)  
 (= (KIF\$source vocabulary) scl.obj\$name-set)  
 (= (KIF\$target vocabulary) SET.FTN\$function)  
 (forall (?N (scl.obj\$name-set ?N))  
 (and (= (SET.FTN\$source (vocabulary ?N)) (interpretation ?N))  
 (= (SET.FTN\$target (vocabulary ?N)) (scl.obj.voc\$vocabulary ?N))))
- (3) (KIF\$function universe)  
 (= (KIF\$source universe) scl.obj\$name-set)  
 (= (KIF\$target universe) SET.FTN\$function)  
 (forall (?N (scl.obj\$name-set ?N))  
 (and (= (SET.FTN\$source (universe ?N)) (interpretation ?N))  
 (= (SET.FTN\$target (universe ?N)) set.obj\$set)))
- (4) (KIF\$function object)  
 (= (KIF\$source object) scl.obj\$name-set)

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 22

May 9, 2004

```
(= (KIF$target object) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (object ?N)) (interpretation ?N))
 (= (SET.FTN$target (object ?N)) set.mor$function)))
 (= (SET.FTN$composition [(object ?N) set.mor$source])
 (SET.FTN$composition [(vocabulary ?N) (scl.obj.voc$object ?N)]))
 (= (SET.FTN$composition [(object ?N) set.mor$target]) (universe ?N))))

(5) (KIF$function function)
(= (KIF$source function) scl.obj$name-set)
(= (KIF$target function) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (function ?N)) (interpretation ?N))
 (= (SET.FTN$target (function ?N)) set.mor$function)))
 (= (SET.FTN$composition [(function ?N) set.mor$source])
 (SET.FTN$composition [(vocabulary ?N) (scl.obj.voc$function ?N)]))
 (= (SET.FTN$composition [(function ?N) set.mor$target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$function]))))

(6) (KIF$function relation)
(= (KIF$source relation) scl.obj$name-set)
(= (KIF$target relation) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (relation ?N)) (interpretation ?N))
 (= (SET.FTN$target (relation ?N)) set.mor$function)))
 (= (SET.FTN$composition [(relation ?N) set.mor$source])
 (SET.FTN$composition [(vocabulary ?N) (scl.obj.voc$relation ?N)]))
 (= (SET.FTN$composition [(relation ?N) set.mor$target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$relation]))))

(7) (forall (?N (scl.obj$name-set ?N))
 ?I ((interpretation ?N) ?I))
(= ((set.mor$restrict ((relation ?N) ?I))
 [(set.obj$binary-intersection
 [(scl.obj.voc$function ?N) (vocabulary ?I))
 ((scl.obj.voc$relation ?N) (vocabulary ?I))])
 (set.mor$target ((relation ?N) ?I))])
(set.mor$composition
 [((set.mor$restrict ((function ?N) ?I))
 [(set.obj$binary-intersection
 [(scl.obj.voc$function ?N) (vocabulary ?I))
 ((scl.obj.voc$relation ?N) (vocabulary ?I))])
 (set.mor$target ((function ?N) ?I))])
 (set.obj.seq$function2relation ((universe ?N) ?I))]))

(8) (KIF$function function-fold)
(= (KIF$source function-fold) scl.obj$name-set)
(= (KIF$target function-fold) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (function-fold ?N)) (interpretation ?N))
 (= (SET.FTN$target (function-fold ?N)) set.pfn$partial-function)))
 (= (SET.FTN$composition [(function-fold ?N) set.pfn$source]) (universe ?N))
 (= (SET.FTN$composition [(function-fold ?N) set.pfn$target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$function]))
 (forall (?I ((interpretation ?N) ?I))
 (set.obj$subset
 ((set.mor$power ((function ?N) ?I))
 ((scl.obj.voc$function ?N) ((vocabulary ?N) ?I)))
 (set.pfn$domain ((function-fold ?N) ?I)))))

(9) (KIF$function relation-fold)
(= (KIF$source relation-fold) scl.obj$name-set)
(= (KIF$target relation-fold) SET.FTN$function)
(forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (relation-fold ?N)) (interpretation ?N))
 (= (SET.FTN$target (relation-fold ?N)) set.pfn$partial-function)))
 (= (SET.FTN$composition [(relation-fold ?N) set.pfn$source]) (universe ?N))
 (= (SET.FTN$composition [(relation-fold ?N) set.pfn$target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$function]))
 (forall (?I ((interpretation ?N) ?I))
 (set.obj$subset
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 23

May 9, 2004

```
((set.mor$power ((relation ?N) ?I))
 ((scl.obj.voc$relation ?N) ((vocabulary ?N) ?I)))
(set.pfn$domain ((relation-fold ?N) ?I))))
```

The interpretation  $I$  contains the function  $obj_M(I) : obj_M(V) \rightarrow univ_N(I)$  that defines several other functions:

- a *coproduct* function  $dom-coprod(obj_N(I)) : dom-coprod(obj_N(V)) \rightarrow dom-coprod(univ_N(I))$ ;
- a *function* function  $ftn(obj_M(I)) : ftn(obj_M(V)) \rightarrow ftn(univ_N(I))$ ; and
- a *sequence* function  $seq(obj_N(I)) : seq(obj_N(V)) \rightarrow seq(univ_N(I))$ .

We refer to these functions only indirectly. We require the interpretation to honor the choice function of the vocabulary, as represented by the identity:

$$ftn_N(I) = choice_M(V) \cdot ftn(obj_M(I)).$$

```
(10) (forall (?N (scl.obj$name-set ?N)
 ?I ((interpretation ?N) ?I))
 (= ((function ?N) ?I)
 (set.mor$composition
 [(scl.obj.voc$choice ?N) ((vocabulary ?N) ?I)]
 (set.seq.mor$function ((object ?N) ?I)))))
```

The set sequence indication functions for the sets  $obj_N(V)$  and  $univ_N(I)$  are the horizontal source and target for a set quartet with the two functions  $ftn(obj_M(I))$  and  $dom-coprod(obj_M(I))$ : the quartet constraint is  $dom-coprod(obj_N(I)) \cdot indic(univ_N(I)) = indic(obj_M(V)) \cdot ftn(obj_M(I))$ . This results in the identities:  $res_N(V) \cdot dom-coprod(int_N(I)) \cdot indic(univ_N(I)) = res_N(V) \cdot indic(obj_M(V)) \cdot ftn(obj_M(I)) = ftn_M(V) \cdot actual(obj_N(V)) \cdot ftn(obj_M(I))$  mapping composite object terms to functions of the interpretation universe.

The set sequence projection functions for the set  $obj_M(V)$  and  $univ_N(I)$  are the horizontal source and target for a set quartet with the two functions  $seq(obj_N(I))$  and  $dom-coprod(obj_M(I))$ : the quartet constraint is  $dom-coprod(obj_N(I)) \cdot proj(univ_N(I)) = proj(obj_M(V)) \cdot seq(obj_N(I))$ . This results in the identities:  $res_N(V) \cdot dom-coprod(int_N(I)) \cdot proj(univ_N(I)) = res_N(V) \cdot proj(obj_M(V)) \cdot seq(obj_N(I)) = seq_M(V) \cdot seq(obj_N(I))$  mapping composite object terms to sequences of the interpretation universe.

The set sequence application functions for the sets  $obj(V)$  and  $univ(I)$  are the horizontal source and target for a set quartet with the two functions  $obj_N(I)$  and  $dom-coprod(obj_M(I))$ : the quartet constraint is  $dom-coprod(obj_N(I)) \cdot apply(univ_N(I)) = apply(obj_M(V)) \cdot obj_N(I)$ . This results in the identities:  $res_N(V) \cdot dom-coprod(int_N(I)) \cdot apply(univ_N(I)) = res_N(V) \cdot apply(obj_M(V)) \cdot obj_N(I) = incl_{comp(V), obj(V)} \cdot obj_N(I)$  mapping composite object terms to elements of the interpretation universe.

```
(11) (KIF$function indication)
 (= (KIF$source indication) scl.obj$name-set)
 (= (KIF$target indication) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (indication ?N)) (interpretation ?N))
 (= (SET.FTN$target (indication ?N)) set.qtt$quartet)))
 (= (SET.FTN$composition [(indication ?N) set.qtt$horizontal-source])
 (SET.FTN$composition [(SET.FTN$composition
 [(vocabulary ?N) (scl.obj.voc$object ?N)] set.obj.seq$indication)]))
 (= (SET.FTN$composition [(indication ?N) set.qtt$horizontal-target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$indication]))
 (= (SET.FTN$composition [(indication ?N) set.qtt$vertical-source])
 (SET.FTN$composition [(object ?N) set.seq.mor$domain-coproduct]))
 (= (SET.FTN$composition [(indication ?N) set.qtt$vertical-target])
 (SET.FTN$composition [(object ?N) set.seq.mor$function]))
```

```
(12) (KIF$function projection)
 (= (KIF$source projection) scl.obj$name-set)
 (= (KIF$target projection) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (projection ?N)) (interpretation ?N))
 (= (SET.FTN$target (projection ?N)) set.qtt$quartet)))
 (= (SET.FTN$composition [(projection ?N) set.qtt$horizontal-source])
 (SET.FTN$composition [(SET.FTN$composition
 [(vocabulary ?N) (scl.obj.voc$object ?N)] set.obj.seq$projection)]))
 (= (SET.FTN$composition [(projection ?N) set.qtt$horizontal-target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$projection]))
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 24

May 9, 2004

```
(= (SET.FTN$composition [(projection ?N) set.qtt$vertical-source])
 (SET.FTN$composition [(object ?N) set.seq.mor$domain-coproduct]))
(= (SET.FTN$composition [(projection ?N) set.qtt$vertical-target])
 (SET.FTN$composition [(object ?N) set.seq.mor$sequence]))

(13) (KIF$function application)
 (= (KIF$source application) scl.obj$name-set)
 (= (KIF$target application) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (application ?N)) (interpretation ?N))
 (= (SET.FTN$target (application ?N)) set.qtt$quartet)))
 (= (SET.FTN$composition [(application ?N) set.qtt$horizontal-source])
 (SET.FTN$composition [(SET.FTN$composition
 [(vocabulary ?N) (scl.obj$.voc$object ?N)]
 set.obj.seq$application]))
 (= (SET.FTN$composition [(application ?N) set.qtt$horizontal-target])
 (SET.FTN$composition [(universe ?N) set.obj.seq$application]))
 (= (SET.FTN$composition [(application ?N) set.qtt$vertical-source])
 (SET.FTN$composition [(object ?N) set.seq.mor$domain-coproduct]))
 (= (SET.FTN$composition [(application ?N) set.qtt$vertical-target])
 (object ?N))))
```

We could define the composite term evaluation function

$$eval_M(V) = res_M(V) \cdot dom\text{-coprod}(obj_M(I)) \cdot apply(univ_M(I)) : comp_M(V) \rightarrow univ_M(I).$$

This implies the identity:  $eval_M(V) = incl_{comp(V), obj(V)} \cdot obj_M(I)$ .

For any name-set  $N$  and any interpretation  $I \in int(N)$ , in general a name map is a tuple for the set pair  $\langle N, univ_N(I) \rangle$ ; that is, a name map is a function  $n : N \rightarrow univ(I)$  on some subset of names  $N \subseteq N$ . However, here we need only the special case where  $N$  is a singleton set  $N = \{x\}$ , where  $x$  is the quantified variable for some quantified sentence. Hence, we define a name-map to be a pair  $(x, a)$ , where  $x$  is a variable  $x \in N$  and  $a$  is a universe element  $a \in univ_N(I)$ . That is, the set of name-maps is the binary Cartesian product  $N \times univ_N(I)$ . The name-map function maps a pair  $(x, a) \in N \times univ_N(I)$  to the augmented interpretation  $map_N(I)((x, a))$ , which is recursively defined on the base component of  $voc_N(I)$ : the function and relation base functions are the same as  $I$ , and the object base function is  $a$  on the name  $x$ , and  $I$  otherwise. There is an interpretation class function  $map_N(I) : N \times univ_N(I) \rightarrow int(N)$ .

```
(14) (KIF$function name-value-diagram)
 (= (KIF$source name-value-diagram) scl.obj$name-set)
 (= (KIF$target name-value-diagram) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (name-value-diagram ?N)) (interpretation ?N))
 (= (SET.FTN$target (name-value-diagram ?N)) set.lim.prd2$diagram)
 (= (SET.FTN$composition [(name-value-diagram ?N) set.lim.prd2$set1]) ?N)
 (= (SET.FTN$composition [(name-value-diagram ?N) set.lim.prd2$set2])
 ((universe ?N) ?I))))
```

```
(15) (KIF$function name-value)
 (= (KIF$source name-value) scl.obj$name-set)
 (= (KIF$target name-value) SET.FTN$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (SET.FTN$source (name-value ?N)) (interpretation ?N))
 (= (SET.FTN$target (name-value ?N)) set.obj$set)
 (= (name-value ?N)
 (SET.FTN$composition
 [(name-value-diagram ?N) set.lim.prd2$binary-product])))
```

```
(16) (KIF$function name-map)
 (= (KIF$source name-map) scl.obj$name-set)
 (= (KIF$target name-map) KIF$function)
 (forall (?N (scl.obj$name-set ?N))
 (and (= (KIF$source (name-map ?N)) (interpretation ?N))
 (= (KIF$target (name-map ?N)) SET.FTN$function)
 (forall (?I ((interpretation ?N) ?I))
 (and (= (SET.FTN$source ((name-map ?N) ?I)) ((name-value ?N) ?I))
 (= (SET.FTN$target ((name-map ?N) ?I)) (interpretation ?N))
 (forall (?n (?N ?n) ?a (((universe ?N) ?I) ?a))
 (and (= ((function ?N) ((name-map ?N) ?I) [?n ?a]))
 ((function ?N) ?I))))
```

```
(= ((relation ?N) ((name-map ?N) ?I) [?n ?a]))
 ((relation ?N) ?I))
(forall (?m ((object-base ?N) ((vocabulary ?N) ?I)) ?m))
 (and (=> (= ?m ?n)
 (= ((object ?N) ((name-map ?N) ?I) [?n ?a])) ?m) ?a)
 (=> (not (= ?m ?n))
 (= ((object ?N) ((name-map ?N) ?I) [?n ?a])) ?m)
 ((object ?N) ?I) ?m)))))))))
```

## Satisfaction

When the (concrete) vocabulary of a sentence  $s$  (or a text  $T$ ) is bounded by the vocabulary of an interpretation  $I$ ,  $I$  is said to be a *legal interpretation* of  $s$  (or  $T$ ). We use the notation  $I \models s$  to denote that sentence  $s$  is true in interpretation  $I$ . We also say that  $I$  *satisfies*  $s$ . Satisfaction assumes legality. In the following, we assume that  $I$  is a legal interpretation of any text to be evaluated. We temporarily ignore sequence variables.<sub>012</sub>

### Atoms:

$I \models (t_1 = t_2)$  iff  $obj_N(I)(t_1) = obj_N(I)(t_2)$ .

$I \models r(\sigma)$  iff  $rel_N(I)(r)$  contains  $obj_N^*(I)(\sigma)$ .

### Boolean Sentence:

$I \models \neg s$  iff not  $I \models s$ .

$I \models \wedge ()$ .

$I \models \wedge (s)$  iff  $I \models s$ .

$I \models \wedge (s_1 s_2 \dots s_n)$  iff  $I \models s_1$  and  $I \models \wedge (s_2 \dots s_n)$ .

$I \models \vee (s_1 \dots s_n)$  iff  $I \models \neg(\wedge(\neg s_1 \dots \neg s_n))$ .

$I \models (s_1 \Rightarrow s_2)$  iff  $I \models \vee(\neg s_1 s_2)$ .

$I \models (s_1 \Leftrightarrow s_2)$  iff  $I \models (s_1 \Rightarrow s_2)$  and  $I \models (s_2 \Rightarrow s_1)$ .

### Quantified Sentence:

$I \models (\forall(x) \beta)$  iff  $map_N(I)((x, a)) \models \beta$  for all  $a \in univ_N(I)$ .

$I \models (\exists(x) \beta)$  iff  $I \models \neg(\forall(x) \neg\beta)$ .

### Text:

$I \models \{\}$ .

$I \models \{s\}$  iff  $I \models s$ .

$I \models \{s_1, s_2, \dots s_n\}$  iff  $I \models s_1$  and  $I \models \{s_2, \dots s_n\}$ .

```
(17) (KIF$function legal-sentence)
 (= (KIF$source legal-sentence) scl.obj$name-set)
 (= (KIF$target legal-sentence) REL$relation)
 (forall (?N (name-set ?N))
 (and (= (SET.FTN$class1 (legal-sentence ?N)) (interpretation ?N))
 (= (SET.FTN$class2 (legal-sentence ?N)) (scl.obj.snt$sentence ?N))))
 (forall (?I ((interpretation ?N) ?I)
 ?s ((scl.obj.snt$sentence ?N) ?s))
 (<=> (legal-interpretation ?N) ?I ?s)
 (scl.obj.voc$subvocabulary
 ((scl.obj.snt$vocabulary ?N) ?s)
 ((vocabulary ?N) ?I))))))
```

```
(18) (KIF$function satisfaction)
 (= (KIF$source satisfaction) scl.obj$name-set)
 (= (KIF$target satisfaction) REL$relation)
 (forall (?N (scl.obj$name-set ?N))
```

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 26

May 9, 2004

```
(and (= (SET.FTN$class1 (satisfaction ?N)) (interpretation ?N))
 (= (SET.FTN$class2 (satisfaction ?N)) (scl.obj.snt$sentence ?N)))
(REL$subrelation (satisfaction ?N) (legal-interpretation ?N)))

(19) (forall (?N (name-set ?N))
 (forall (?I ((interpretation ?N) ?I)
 ?s ((scl.obj.snt$sentence ?N) ?s)
 ((legal-interpretation ?N) ?I ?s))
 (and (=> ((scl.obj.snt$equation ?N) ?s)
 (<=> ((satisfaction ?N) ?I ?s)
 (= ((object ?N) ?I) ((scl.obj.snt$term1 ?N) ?s))
 ((object ?N) ?I) ((scl.obj.snt$term2 ?N) ?s))))
 (=> ((scl.obj.snt$relater ?N) ?s)
 (<=> ((satisfaction ?N) ?I ?s)
 (((relation ?N) ?I) ((scl.obj.snt$term ?N) ?s))
 ((set.seq.mor$sequence ((object ?N) ?I))
 ((scl.obj.snt$term-sequence ?N) ?s))))
 (=> ((scl.obj.snt$conjunctive-sentence ?N) ?s)
 (<=> ((satisfaction ?N) ?I ?s)
 (forall (?s1 ((scl.obj.snt$select-sentence-subset ?N) ?s) ?s1))
 ((satisfaction ?N) ?I ?s1))))
 (=> ((scl.obj.snt$negative-sentence ?N) ?s)
 (<=> ((satisfaction ?N) ?I ?s)
 (not ((satisfaction ?N) ?I ((scl.obj.snt$select-sentence ?N) ?s))))
 (=> ((scl.obj.snt$quantified-sentence ?N) ?s)
 (<=> ((satisfaction ?N) ?I ?s)
 (forall (?a ((universe ?N) ?I) ?a)
 ((satisfaction ?N)
 ((name-map ?N) ?I) [(scl.obj.snt$variable ?s) ?a]
 (scl.obj.snt$body ?s)))))))))
```

## SCL to FOL Translation

`scl.obj.fol`

We can translate any SCL kernel text (with no sequence variables) into FOL text with essentially the same meaning. This uses various functions to translate SCL terms into FOL terms, SCL term sequences into FOL term tuples, SCL atoms into FOL atoms, and SCL sentences into FOL sentences. For any SCL name-set  $N$  and any SCL vocabulary  $V \in \text{voc}(N)$ , let  $\text{scl2fol}(V)$  denote the SCL to FOL translation of sentences and text compatible with that vocabulary. Intuitively, the FOL language (or signature) of  $\text{scl2fol}(V)$  contains all the object terms  $\text{obj}(V)$ , classified as constants (individual names), plus the countable sets  $\{\text{holds}_0, \text{holds}_1, \text{holds}_2, \dots\}$  of relation symbols with arities respectively 1, 2, 3, ... and  $\{\text{app}_0, \text{app}_1, \text{app}_2, \dots\}$  of function symbols with arities respectively 1, 2, 3, ... .

In more detail, the language (signature) of  $L = \text{scl2fol}(V)$  has a set of variables  $\text{var}(L) = \{\text{var}(0), \text{var}(1), \text{var}(2), \dots\}$  in one-one correspondence with the natural numbers  $\aleph = \{0, 1, 2, \dots\}$ , has the disjoint union  $\text{obj}(V) + \text{app}(L) = \text{obj}(V) + \{\text{app}(0), \text{app}(1), \text{app}(2), \dots\}$  as its set of function symbols with valence define by  $\text{val}(t) = 0$  for each object term  $t \in \text{obj}(V)$  and  $\text{val}(\text{app}(n)) = n+1$  for each natural number  $n \in \aleph$  (the constants are  $\text{obj}(V)$ ), and has a set of relation symbols  $\text{rel}(L) = \{\text{hlds}(0), \text{hlds}(1), \text{hlds}(2), \dots\}$  in one-one correspondence with the natural numbers  $\aleph$  with valence define by  $\text{val}(\text{holds}(n)) = n+1$  for each natural number  $n \in \aleph$ . We can define special generating injections  $\text{app} : \aleph \rightarrow \text{ftn}(L)$  and  $\text{holds} : \aleph \rightarrow \text{rel}(L)$ . The translation  $\text{scl2fol}(V)$  maps the terms and atoms of sentences and texts into those of  $\text{scl2fol}(V)$  by application of the following recursively defined functions.

The *term* function

$$\text{trm} : \text{obj}(V) \rightarrow \text{trm}(\text{scl2fol}(V)),$$

where  $\text{trm}(n) = n$ , for all elementary terms (names)  $n \in \text{elem}(V) \subseteq N$ , using constant embedding to terms, and  $\text{trm}((t, \tau)) = \text{apply}(\text{app}(|\tau|), \text{insert}(\text{trm}(t), \text{tuple}(\tau)))$  for all composite terms  $t[\tau] = (t, \tau) \in \text{comp}(V)$ .

The *tuple* function

$$\text{tuple} : \text{seq}(\text{obj}(V)) \rightarrow \text{tuple}(\text{scl2fol}(V)),$$

where  $\text{tuple}(\epsilon) = \epsilon$  for the empty sequence of terms,  $\text{tuple}((t)_{\text{obj}(V)}) = \{\text{var}(0), t\}$  for a singleton term tuple  $t \in \text{obj}(V)$ , and  $\text{tuple}(\tau_1 \oplus \tau_2) = \text{tuple}(\tau_1) \oplus \text{tuple}(\tau_2)$  for composite sequence of terms.

The *atom* function

$$\text{atom} : \text{atm}(V) \rightarrow \text{atm}(\text{scl2fol}(V)),$$

where  $\text{atom}((= t_1 t_2)) = \text{eqn}(\text{trm}(t_1), \text{trm}(t_2))$  for equation  $(= t_1 t_2) \in \text{eqn}(V)$ , and  $\text{atom}((t, \tau)) = \text{holds}(\text{hlds}(|\tau|), \text{insert}(\text{trm}(t), \text{tuple}(\tau)))$  for all relaters  $t \in \text{rel}(V)$  and  $\tau \in \text{seq}(\text{obj}(V))$ .

The *sentence* function

$$\text{sent} : \text{sent}(V) \rightarrow \text{sent}(\text{scl2fol}(V)),$$

where  $\text{sent}(s) = \text{atom}(s)$  for atom  $s \in \text{atm}(V)$ ,  $\text{sent}((\text{not } s)) = \text{neg}(\text{sent}(s))$  for sentence  $s \in \text{sent}(V)$ ,  $\text{sent}((\text{and } S)) = \text{conj}(\wp \text{sent}(S))$  for sentence subset  $S \in \wp \text{sent}(V)$ , and  $\text{sent}((\text{forall } (x) s)) = \text{quan}(x, s)$  for variable  $x \in N$  and sentence  $s \in \text{sent}(V)$ .

## Appendix

### Sequences

set.obj.seq

Table 1 lists the 39 terms in the set sequence namespace of the lower core (meta) ontology (IFF-LCO). The 8 terms in bold are terms in popular use (e.g., in the SCL kernel grammar).

**Table 1: Terminology for Set Sequences**

|                                          | Other                                                                                         | Object                                                    | Morphism                                                                                                                                                                                       |
|------------------------------------------|-----------------------------------------------------------------------------------------------|-----------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>set</b><br><b>.obj</b><br><b>.seq</b> | power                                                                                         | <b>sequence</b><br>empty-sequence<br>nonempty-sequence    | <b>injection length singleton</b><br><b>empty concatenation</b>                                                                                                                                |
|                                          |                                                                                               |                                                           | left-insertion left-most<br>left-deletion left-resolution<br>right-insertion right-most<br>right-deletion right-resolution                                                                     |
|                                          | sequence-sequence-<br>diagram<br>element-sequence-<br>diagram<br>sequence-element-<br>diagram | sequence-sequence<br>element-sequence<br>sequence-element | sequence-sequence-projection1<br>sequence-sequence-projection1<br>element-sequence-projection1<br>element-sequence-projection2<br>sequence-element-projection1<br>sequence-element-projection2 |
|                                          |                                                                                               | <b>function relation</b>                                  | function2relation<br>function-inclusion                                                                                                                                                        |
|                                          | domain-arity                                                                                  | domain-coproduct                                          | injection indication projection<br><b>application</b>                                                                                                                                          |

### Basics

The set of sequences of  $A$  elements  $seq(A)$  is the least fixpoint solution of the set equation

$$X \cong 1 + A \times X.$$

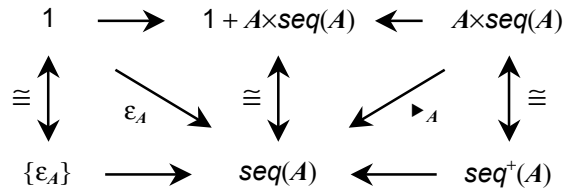
Categorically, the sequence set  $seq(A)$  is the fixpoint solution (Figure 1)

$$seq(A) \cong fixpt(A)(seq(A)),$$

for the  $\omega$ -continuous endofunctor  $fixpt(A) : Set \rightarrow Set$  on the category  $Set$  defined by

$$fixpt(A)(X) = 1 + A \times X.$$

The abstract syntax of sequences is parametric: there is a name for the parameter set ' $A$ ' and also a name for  $A$ -sequences ' $seq(A)$ '. There is a collection of synthetic/constructor operators, a collection of analytic-selector partial operators and axioms that relate the two collections.



**Figure 1: Sequence Fixpoint Solution**

There is a collection of two synthetic/constructor injective operators on sequences:

$$\epsilon_A = empty(A) : 1 \rightarrow seq(A)$$

$$\blacktriangleright_A = insert(A) : A \times seq(A) \rightarrow seq(A),$$

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 29

May 9, 2004

where the empty operator ' $\epsilon_A$ ' is the empty  $A$ -sequence, and the left insertion operator ' $\blacktriangleright_A$ ' inserts  $A$ -elements into  $A$ -sequences. The coproduct copairing of these two functions is the fixpoint isomorphism

$$\cong_A : 1 + A \times \text{seq}(A) \rightarrow \text{seq}(A).$$

There is a collection of analytic-selector operators on sequences with the two subgroup clusters in one-one correspondence with the two synthetic/constructor operators (their inverse):

$$\text{is-empty}_A : 1 \rightarrow \text{bool}$$

$$\Delta_A : \text{seq}(A) \rightarrow 1,$$

where the boolean operator ' $\text{is-empty}_A$ ' tests for the empty sequence, and the partial operator ' $\Delta_A$ ' maps the empty sequence to the element of the unit set.

$$\text{is-nonempty}_A : 1 \rightarrow \text{bool}$$

$$\text{left}_A : \text{seq}(A) \rightarrow A,$$

$$\text{left-del}_A : \text{seq}(A) \rightarrow \text{seq}(A),$$

where the boolean operator ' $\text{is-nonempty}_A$ ' tests for a nonempty sequence, the leftmost partial operator ' $\text{left}_A$ ' selects the leftmost element of a nonempty sequence, and the left deletion partial operator ' $\text{left-del}_A$ ' selects the remaining elements to the right of the leftmost element of a nonempty sequence.

These operators satisfy the defining FOL term abstract data type semantics: there are axioms for the elementary and composite operators.

## Elementary

$$\forall (* : 1)$$

$$(\text{is-empty}_A(\epsilon_A(*)) \\ \& \Delta_A(\epsilon_A(*)) = *)$$

which states the trivial fact that the empty sequence is an empty sequence

$$\forall (a : \text{seq}(A))$$

$$(\text{is-empty}_A(a) \Rightarrow (\epsilon_A(\Delta_A(a)) = a))$$

which states that the empty sequence is constructible using the empty operator.

## Composite

$$\forall (a : A, \underline{a} : \text{seq}(A))$$

$$(\text{is-nonempty}_A(\blacktriangleright_A(a, \underline{a})) \\ \& \text{left}_A(\blacktriangleright_A(a, \underline{a})) = a \\ \& \text{left-del}_A(\blacktriangleright_A(a, \underline{a})) = \underline{a})$$

which states that any sequence constructed by the left insertion operator is nonempty and is decomposable into the original  $A$ -element and  $A$ -sequence.

$$\forall (a : \text{seq}(A))$$

$$(\text{is-nonempty}_A(a) \Rightarrow (\blacktriangleright_A(\text{left}_A(a), \text{left-del}_A(a)) = a))$$

which states that every nonempty sequence is constructible using the insertion operator from the pair of its components.

---

For any set  $A$ , a finite *sequence*  $a = (a_0, a_1, a_2, \dots, a_{n-1})$  of length  $n \in \aleph$  is any list of  $n$  elements of  $A$ ; that is,  $a_k \in A$  for all  $0 \leq k < n$ . Let  $\text{seq}(A)$  denote the set of all finite sequences over  $A$ . Another way to describe this is that  $a : \downarrow n \rightarrow A$  is a map from  $\downarrow n = \{0, 1, 2, \dots, n-1\}$ , the interval of natural numbers below  $n$ , to  $A$ . Clearly, there is a *length* function

$$|-|_A = \text{len}(A) : \text{seq}(A) \rightarrow \aleph.$$

The set of sequences is the coproduct of the  $n^{\text{th}}$  power tuple of sets

$$\text{seq}(A) = \sum_{n \in \aleph} A^n.$$

For each natural number  $n \in \aleph$ , there is a coproduct injection

$$\iota_A : A^n \rightarrow \text{seq}(A).$$

- (1) (KIF\$function power)
  - (= (KIF\$source power) set.obj\$set)
  - (= (KIF\$target power) set.dgm.tpl\$tuple)
  - (forall (?a (set.obj\$set ?a))
    - (and (= (set.dgm.tpl\$index (power ?a)) natno\$natural-number))
      - (forall (?n (natno\$natural-number ?n))
        - (= ((set.dgm.tpl\$set (power ?a)) ?n)
 ((set.lim.prd.fin\$power ?a) ?n))))
- (2) (SET.FTN\$function sequence)
  - (= (SET.FTN\$source sequence) set.obj\$set)
  - (= (SET.FTN\$target sequence) set.obj\$set)
  - (= (sequence (SET.FTN\$composition [power set.col.coprds\$coproduct]))
- (3) (KIF\$function injection)
  - (= (KIF\$source injection) set.obj\$set)
  - (= (KIF\$target injection) SET.FTN\$function)
  - (forall (?a (set.obj\$set ?a))
    - (and (= (SET.FTN\$source (injection ?n)) natno\$natural-number)
      - (= (SET.FTN\$target (injection ?n)) set.mor\$function)
      - (= (SET.FTN\$composition [(injection ?n) set.mor\$source])
 ((set.dgm.tpl\$set (power ?a)) ?n))
      - (= (SET.FTN\$composition [(injection ?n) set.mor\$target])
 (sequence ?a))
      - (forall (?n (natno\$natural-number ?n))
        - (= ((injection ?a) ?n)
 ((set.col.coprds\$injection (power ?a)) ?n))))
  - (4) (SET.FTN\$function length)
    - (= (SET.FTN\$source length) set.obj\$set)
    - (= (SET.FTN\$target length) set.mor\$function)
    - (= (SET.FTN\$composition [length set.mor\$source]) sequence)
    - (= (SET.FTN\$composition [length set.mor\$target])
 ((SET.FTN\$constant [set.obj\$set set.obj\$set]) natno\$natural-number))
    - (forall (?a (set.obj\$set ?a)
 ?n (natno\$natural-number ?n))
      - (= (set.mor\$composition [(injection ?a) ?n] (length ?a))
 ((set.mor\$constant [(set.dgm.tpl\$set (power ?a)) ?n] natno\$natural-number) ?n)))

Any element  $a \in A$  can be embedded as a singleton sequence  $(a)_A \in \mathbf{seq}(A)$  of length 1. Hence, there is a *singleton* function

$$(-)_A = \mathbf{sln}(A) : A \rightarrow \mathbf{seq}(A),$$

which satisfies the constraint:  $(-)_A \circ |-_A = \Delta_A(1)$ , the function with constant value 1. This asserts “the length of a singleton sequence is 1”.

- (5) (SET.FTN\$function singleton)
  - (= (SET.FTN\$source singleton) set.obj\$set)
  - (= (SET.FTN\$target singleton) set.mor\$function)
  - (= (SET.FTN\$composition [singleton set.mor\$source]) (SET.FTN\$identity set.obj\$set))
  - (= (SET.FTN\$composition [singleton set.mor\$target]) sequence)
  - (forall (?a (set.obj\$set ?a))
    - (= (set.mor\$composition [(singleton ?a) (length ?a)])
 (set.mor\$constant [?a natno\$natural-number] natno\$1)))

Any two sequences  $a = (a_0, a_1, \dots, a_{n-1})$  and  $b = (b_0, b_1, \dots, a_{m-1})$  can be concatenated, resulting in the sequence  $a \oplus_A b = (a_0, a_1, \dots, a_{n-1}, b_0, b_1, \dots, a_{m-1}) \in \mathbf{seq}(A)$  of length  $n + m$ . Hence, there is a *concatenation* function

$$\oplus_A = \mathbf{conc}(A) : \mathbf{seq}(A) \times \mathbf{seq}(A) \rightarrow \mathbf{seq}(A),$$

which satisfies the constraint:  $\oplus_A \circ |-_A = (|-_A \times |-_A) \circ +$ . This asserts, “the length of a concatenated sequence is the sum of the lengths of the two component sequences”. Concatenation also satisfies the associative law:  $(a \oplus_A b) \oplus_A c = a \oplus_A (b \oplus_A c)$ .

- (6) (SET.FTN\$function sequence-sequence-diagram)
  - (= (SET.FTN\$source sequence-sequence-diagram) set.obj\$set)
  - (= (SET.FTN\$target sequence-sequence-diagram) set.lim.prd2\$diagram)
  - (= (SET.FTN\$composition [sequence-sequence-diagram set.lim.prd2\$set1]) sequence)
  - (= (SET.FTN\$composition [sequence-sequence-diagram set.lim.prd2\$set2]) sequence)

- (7) (SET.FTN\$function sequence-sequence)  
 (= (SET.FTN\$source sequence-sequence) set.obj\$set)  
 (= (SET.FTN\$target sequence-sequence) set.obj\$set)  
 (= sequence-sequence  
 (SET.FTN\$composition [sequence-sequence-diagram set.lim.prd2\$binary-product]))
- (8) (SET.FTN\$function sequence-sequence-projection1)  
 (= (SET.FTN\$source sequence-sequence-projection1) set.obj\$set)  
 (= (SET.FTN\$target sequence-sequence-projection1) set.mor\$function)  
 (= (SET.FTN\$composition [sequence-sequence-projection1 set.mor\$source]) sequence-sequence)  
 (= (SET.FTN\$composition [sequence-sequence-projection1 set.mor\$target]) sequence)  
 (= sequence-sequence-projection1  
 (SET.FTN\$composition [sequence-sequence-diagram set.lim.prd2\$projection1]))
- (9) (SET.FTN\$function sequence-sequence-projection2)  
 (= (SET.FTN\$source sequence-sequence-projection2) set.obj\$set)  
 (= (SET.FTN\$target sequence-sequence-projection2) set.mor\$function)  
 (= (SET.FTN\$composition [sequence-sequence-projection2 set.mor\$source]) sequence-sequence)  
 (= (SET.FTN\$composition [sequence-sequence-projection2 set.mor\$target]) sequence)  
 (= sequence-sequence-projection2  
 (SET.FTN\$composition [sequence-sequence-diagram set.lim.prd2\$projection2]))
- (10) (SET.FTN\$function concatenation)  
 (= (SET.FTN\$source concatenation) set.obj\$set)  
 (= (SET.FTN\$target concatenation) set.mor\$function)  
 (= (SET.FTN\$composition [concatenation set.mor\$source]) sequence-sequence)  
 (= (SET.FTN\$composition [concatenation set.mor\$target]) sequence)
- (11) (forall (?a (set.obj\$set ?a))  
 (= (set.mor\$composition [(concatenation ?a) (length ?a)])  
 (set.mor\$composition [(set.mor\$binary-product [(length ?a) (length ?a)] natno\$sum])))

There is a distinguished empty sequence  $\varepsilon_A = ()_A \in \mathbf{seq}(A)$  of length 0. Hence, there is an *empty* function

$$\varepsilon_A = \mathbf{empty}(A) : \mathbf{1} \rightarrow \mathbf{seq}(A),$$

which satisfies the constraint:  $\varepsilon_A \circ |-|_A = 0$ . This asserts, “the length of the empty sequence is 0”. Empty also satisfies the unit laws:  $\varepsilon_A \oplus_A a = a = a \oplus_A \varepsilon_A$ . A sequence is *empty* when it is equal to the empty sequence. Let  $\mathbf{seq}^+(A) \subseteq \mathbf{seq}(A)$  denote the set of all finite nonempty sequences over  $A$ . The singleton function factors through the set of nonempty sequences.

- (12) (SET.FTN\$function empty)  
 (= (SET.FTN\$source empty) set.obj\$set)  
 (= (SET.FTN\$target empty) set.mor\$function)  
 (= (SET.FTN\$composition [empty set.mor\$source])  
 ((set.mor\$constant [set\$set set.obj\$set]) set.obj\$unit))  
 (= (SET.FTN\$composition [empty set.mor\$target]) sequence)  
 (forall (?a (set.obj\$set ?a))  
 (= (set.mor\$composition [(empty ?a) (length ?a)] natno\$zero)))
- (13) (SET.FTN\$function empty-sequence)  
 (= (SET.FTN\$source empty-sequence) set.obj\$set)  
 (= (SET.FTN\$target empty-sequence) set.obj\$set)  
 (forall (?a (set.obj\$set ?a))  
 (= (empty-sequence) (set.mor\$image (empty ?a))))
- (14) (SET.FTN\$function nonempty-sequence)  
 (= (SET.FTN\$source nonempty-sequence) set.obj\$set)  
 (= (SET.FTN\$target nonempty-sequence) set.obj\$set)  
 (forall (?a (set.obj\$set ?a))  
 (= (nonempty-sequence)  
 (set.obj\$difference [(sequence ?a) (empty-sequence ?a)])))

Any element  $a \in A$  of  $A$  can be inserted onto the left side of a sequence  $(a_0, a_1, a_2, \dots, a_{n-1}) \in \mathbf{seq}(A)$  of length  $n$ , getting another sequence  $(a, a_0, a_1, a_2, \dots, a_{n-1}) \in \mathbf{seq}(A)$  of length  $n + 1$ . Hence, there is a *left insertion* function

$$\blacktriangleright_A = \mathbf{left-ins}(A) : A \times \mathbf{seq}(A) \rightarrow \mathbf{seq}(A),$$

which satisfies the constraint:  $\blacktriangleright_A \circ |\cdot|_A = \pi_2 \circ |\cdot|_A \circ (-)+1$ , which asserts that “the length of an left inserted sequence is the one plus the length of the component sequence”. Concatenation and left insertion are related by the constraint:  $\blacktriangleright_A = ((-)_A \times id_{seq(A)}) \circ \oplus_A$ , which asserts that left insertion can be defined as the concatenation of the singleton crossed with the identity”.

- ```
(15) (SET.FTN$function element-sequence-diagram)
    (= (SET.FTN$source element-sequence-diagram) set.obj$set)
    (= (SET.FTN$target element-sequence-diagram) set.lim.prd2$diagram)
    (= (SET.FTN$composition [element-sequence-diagram set.lim.prd2$set1])
        (SET.FTN$identity set.obj$set))
    (= (SET.FTN$composition [element-sequence-diagram set.lim.prd2$set2]) sequence)

(16) (SET.FTN$function element-sequence)
    (= (SET.FTN$source element-sequence) set.obj$set)
    (= (SET.FTN$target element-sequence) set.obj$set)
    (= element-sequence
        (SET.FTN$composition [element-sequence-diagram set.lim.prd2$binary-product]))

(17) (SET.FTN$function element-sequence-projection1)
    (= (SET.FTN$source element-sequence-projection1) set.obj$set)
    (= (SET.FTN$target element-sequence-projection1) set.mor$function)
    (= (SET.FTN$composition [element-sequence-projection1 set.mor$source]) element-sequence)
    (= (SET.FTN$composition [element-sequence-projection1 set.mor$target])
        (SET.FTN$identity set.obj$set))
    (= element-sequence-projection1
        (SET.FTN$composition [element-sequence-diagram set.lim.prd2$projection1]))

(18) (SET.FTN$function element-sequence-projection2)
    (= (SET.FTN$source element-sequence-projection2) set.obj$set)
    (= (SET.FTN$target element-sequence-projection2) set.mor$function)
    (= (SET.FTN$composition [element-sequence-projection2 set.mor$source]) element-sequence)
    (= (SET.FTN$composition [element-sequence-projection2 set.mor$target]) sequence)
    (= element-sequence-projection2
        (SET.FTN$composition [element-sequence-diagram set.lim.prd2$projection2]))

(19) (SET.FTN$function left-insertion)
    (= (SET.FTN$source left-insertion) set.obj$set)
    (= (SET.FTN$target left-insertion) set.mor$function)
    (= (SET.FTN$composition [left-insertion set.mor$source]) element-sequence)
    (= (SET.FTN$composition [left-insertion set.mor$target]) sequence)
    (forall (?a (set.obj$set ?a))
        (= (left-insertion ?a)
            (set.mor$composition
                [(set.mor.prd2$binary-product [(singleton ?a) (set.mor$identity (sequence ?a))]
                    (concatenation ?a)])))

(20) (forall (?a (set.obj$set ?a))
    (= (set.mor$composition [(left-insertion ?a) (length ?a)])
        (set.mor$composition [(set.mor$composition [
            (element-sequence-projection2 ?a) (length ?a)] natno$successor]]))
```

Any nonempty sequence $a \in seq^+(A)$ resolves into a *leftmost* element $left_A(a) \in A$ and a *left-deletion* sequence $left-del_A(a) \in seq(A)$. Hence, there is a *leftmost* partial function and a *left deletion* partial function whose product pairing called *left resolution* is a bijection between $seq^+(A)$ and the binary product $A \times seq(A)$

$$left_A : seq(A) \rightarrow A,$$

$$left-del_A : seq(A) \rightarrow seq(A),$$

$$left-res_A = (left_A, left-del_A) : seq^+(A) \rightarrow A \times seq(A),$$

both with domain $seq^+(A)$, which satisfies the abstract syntax axioms discussed below.

- ```
(21) (SET.FTN$function leftmost)
 (= (SET.FTN$source leftmost) set.obj$set)
 (= (SET.FTN$target leftmost) set.pfn$partial-function)
 (= (SET.FTN$composition [leftmost set.pfn$source]) sequence)
 (= (SET.FTN$composition [leftmost set.pfn$domain]) nonempty-sequence)
 (= (SET.FTN$composition [leftmost set.pfn$target]) (SET.FTN$identity set.obj$set))
```

- (22) (SET.FTN\$function left-deletion)  
 (= (SET.FTN\$source left-deletion) set.obj\$set)  
 (= (SET.FTN\$target left-deletion) set.pfn\$partial-function)  
 (= (SET.FTN\$composition [left-deletion set.pfn\$source]) sequence)  
 (= (SET.FTN\$composition [left-deletion set.pfn\$domain]) nonempty-sequence)  
 (= (SET.FTN\$composition [left-deletion set.pfn\$target]) sequence)
- (23) (SET.FTN\$function left-resolution)  
 (= (SET.FTN\$source left-resolution) set.obj\$set)  
 (= (SET.FTN\$target left-resolution) set.mor\$bijection)  
 (= (SET.FTN\$composition [left-resolution set.pfn\$source]) nonempty-sequence)  
 (= (SET.FTN\$composition [left-resolution set.pfn\$target]) element-sequence)  
 (forall (?a (set.obj\$set ?a))  
 (= (left-resolution ?a)  
 (set.lim.prd2\$pairing  
 [(set.pfn\$function (leftmost ?a))  
 (set.pfn\$function (left-deletion ?a))]))))
- (24) (forall (?a (set.obj\$set ?a))  
 (= (set.mor\$composition [(left-resolution ?a) (left-insertion ?A)])  
 (set.mor\$inclusion [(nonempty-sequence ?a) (sequence ?a)])))
- (25) (forall (?a (set.obj\$set ?a))  
 (and (= (set.mor\$composition [(left-insertion ?a) (leftmost ?a)])  
 (element-sequence-projection1 ?a))  
 (= (set.mor\$composition [(left-insertion ?a) (left-deletion ?a)])  
 (element-sequence-projection2 ?a))))
- Any element  $a \in A$  of  $\mathcal{A}$  can be inserted onto the right side of a sequence  $(a_0, a_1, a_2, \dots, a_{n-1}) \in \mathbf{seq}(\mathcal{A})$  of length  $n$ , getting another sequence  $(a_0, a_1, a_2, \dots, a_{n-1}, a) \in \mathbf{seq}(\mathcal{A})$  of length  $n + 1$ . Hence, there is a *right insertion* function
- $$\blacktriangleleft_{\mathcal{A}} = \mathit{right-ins}(\mathcal{A}) : \mathbf{seq}(\mathcal{A}) \times \mathcal{A} \rightarrow \mathbf{seq}(\mathcal{A}),$$
- which satisfies the constraint:  $\blacktriangleleft_{\mathcal{A}} \circ |-_{\mathcal{A}} = \pi_1 \circ |-_{\mathcal{A}} \circ (-)+1$ , which asserts that “the length of an right inserted sequence is the one plus the length of the component sequence”. Concatenation and right insertion are related by the constraint:  $\blacktriangleleft_{\mathcal{A}} = (id_{\mathbf{seq}(\mathcal{A})} \times (-)_{\mathcal{A}}) \circ \oplus_{\mathcal{A}}$ , which asserts that right insertion can be defined as the concatenation of the identity crossed with the singleton”.
- (26) (SET.FTN\$function sequence-element-diagram)  
 (= (SET.FTN\$source sequence-element-diagram) set.obj\$set)  
 (= (SET.FTN\$target sequence-element-diagram) set.lim.prd2\$diagram)  
 (= (SET.FTN\$composition [sequence-element-diagram set.lim.prd2\$set1]) sequence)  
 (= (SET.FTN\$composition [sequence-element-diagram set.lim.prd2\$set2])  
 (SET.FTN\$identity set.obj\$set))
- (27) (SET.FTN\$function sequence-element)  
 (= (SET.FTN\$source sequence-element) set.obj\$set)  
 (= (SET.FTN\$target sequence-element) set.obj\$set)  
 (= sequence-element  
 (SET.FTN\$composition [sequence-element-diagram set.lim.prd2\$binary-product]))
- (28) (SET.FTN\$function sequence-element-projection1)  
 (= (SET.FTN\$source sequence-element-projection1) set.obj\$set)  
 (= (SET.FTN\$target sequence-element-projection1) set.mor\$function)  
 (= (SET.FTN\$composition [sequence-element-projection1 set.mor\$source]) sequence-element)  
 (= (SET.FTN\$composition [sequence-element-projection1 set.mor\$target]) sequence)  
 (= sequence-element-projection1  
 (SET.FTN\$composition [sequence-element-diagram set.lim.prd2\$projection1]))
- (29) (SET.FTN\$function sequence-element-projection2)  
 (= (SET.FTN\$source sequence-element-projection2) set.obj\$set)  
 (= (SET.FTN\$target sequence-element-projection2) set.mor\$function)  
 (= (SET.FTN\$composition [sequence-element-projection2 set.mor\$source]) sequence-element)  
 (= (SET.FTN\$composition [sequence-element-projection2 set.mor\$target])  
 (SET.FTN\$identity set.obj\$set))  
 (= sequence-element-projection2  
 (SET.FTN\$composition [sequence-element-diagram set.lim.prd2\$projection2]))
- (30) (SET.FTN\$function right-insertion)

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 34

May 9, 2004

```
(= (SET.FTN$source right-insertion) set.obj$set)
(= (SET.FTN$target right-insertion) set.mor$function)
(= (SET.FTN$composition [right-insertion set.mor$source]) sequence-element)
(= (SET.FTN$composition [right-insertion set.mor$target]) sequence)
(forall (?a (set.obj$set ?a))
 (= (right-insertion ?a)
 (set.mor$composition
 [(set.mor.prd2$binary-product [(set.mor$identity (sequence ?a)) (singleton ?a)]
 (concatenation ?a)])))
(31) (forall (?a (set.obj$set ?a))
 (= (set.mor$composition [(right-insertion ?a) (length ?a)]
 (set.mor$composition [(set.mor$composition [
 (sequence-element-projection1 ?a) (length ?a)] natno$successor]])))
```

Any nonempty sequence  $a \in \text{seq}^+(A)$  resolves into a *rightmost* element  $\text{right}_A(a) \in A$  and a *right-deletion* sequence  $\text{right-del}_A(a) \in \text{seq}(A)$ . Hence, there is a *rightmost* partial function and a *right deletion* partial function whose product pairing called *right resolution* is a bijection between  $\text{seq}^+(A)$  and the binary product  $\text{seq}(A) \times A$

$$\text{right}_A : \text{seq}(A) \rightarrow A,$$

$$\text{right-del}_A : \text{seq}(A) \rightarrow \text{seq}(A),$$

$$\text{right-res}_A = (\text{right-del}_A, \text{right}_A) : \text{seq}^+(A) \rightarrow \text{seq}(A) \times A,$$

both with domain  $\text{seq}^+(A)$ , which satisfies the abstract syntax axioms discussed below.

```
(32) (SET.FTN$function rightmost)
(= (SET.FTN$source rightmost) set.obj$set)
(= (SET.FTN$target rightmost) set.pfn$partial-function)
(= (SET.FTN$composition [rightmost set.pfn$source]) sequence)
(= (SET.FTN$composition [rightmost set.pfn$domain]) nonempty-sequence)
(= (SET.FTN$composition [rightmost set.pfn$target]) (SET.FTN$identity set.obj$set))
(33) (SET.FTN$function right-deletion)
(= (SET.FTN$source right-deletion) set.obj$set)
(= (SET.FTN$target right-deletion) set.pfn$partial-function)
(= (SET.FTN$composition [right-deletion set.pfn$source]) sequence)
(= (SET.FTN$composition [right-deletion set.pfn$domain]) nonempty-sequence)
(= (SET.FTN$composition [right-deletion set.pfn$target]) sequence)
(34) (SET.FTN$function right-resolution)
(= (SET.FTN$source right-resolution) set.obj$set)
(= (SET.FTN$target right-resolution) set.mor$bijection)
(= (SET.FTN$composition [right-resolution set.pfn$source]) nonempty-sequence)
(= (SET.FTN$composition [right-resolution set.pfn$target]) sequence-element)
(forall (?a (set.obj$set ?a))
 (= (right-resolution ?a)
 (set.lim.prd2$pairing
 [(set.pfn$function (right-deletion ?a))
 (set.pfn$function (rightmost ?a)])))
(35) (forall (?a (set.obj$set ?a))
 (= (set.mor$composition [(right-resolution ?a) (right-insertion ?a)]
 (set.mor$inclusion [(nonempty-sequence ?a) (sequence ?a)])))
(36) (forall (?a (set.obj$set ?a))
 (and (= (set.mor$composition [(right-insertion ?a) (rightmost ?a)]
 (sequence-element-projection2 ?a))
 (= (set.mor$composition [(right-insertion ?a) (right-deletion ?a)]
 (sequence-element-projection1 ?a))))
```

## Relations and Functions

For any set  $A$ , a *relation* (*relational extension*)  $P \in \mathit{rel}(A)$  on  $A$  is a subset  $P \subseteq \mathit{seq}(A)$  of finite *sequences* of  $A$ . The set of all relations is the powerset of  $\mathit{rel}(A) = \wp \mathit{seq}(A)$ . A *function* (*functional extension*)  $\varphi \in \mathit{ftn}(A)$  on  $A$  is a partial function  $\varphi : \mathit{seq}(A) \rightarrow A$  which is total on each arity. The latter assertion means that if  $x \in \mathit{dom}(\varphi)$  then  $\varphi$  is defined on all of  $A^{|x|}$ . There is an *ftn2rel* injection

$$\mathit{ftn2rel}(A) : \mathit{ftn}(A) \rightarrow \mathit{rel}(A)$$

that embeds functions as relations. The image of  $\mathit{ftn2rel}(A)$  is the subset of functional relations.

```
(37) (SET.FTN$function function)
 (= (SET.FTN$source function) set.obj$set)
 (= (SET.FTN$target function) set.obj$set)
 (forall (?a (set.obj$set ?a))
 (and (set.obj$subset (function ?a) ((SET$hom partial-function) [(sequence ?a) ?a]))
 (forall (?n (natno$natural-number ?n))
 (or (not (set.obj$intersects ((set.dgm.tpl$set (power ?a)) ?n) (function ?a)))
 (set.obj$subset ((set.dgm.tpl$set (power ?a)) ?n) (function ?a))))))

(38) (SET.FTN$function relation)
 (= (SET.FTN$source sequence) set.obj$set)
 (= (SET.FTN$target sequence) set.obj$set)
 (= relation (SET.FTN$composition [sequence set.obj$power]))

(39) (SET.FTN$function function2relation)
 (= (SET.FTN$source function2relation) set.obj$set)
 (= (SET.FTN$target function2relation) set.mor$injection)
 (= (SET.FTN$composition [function2relation set.mor$source]) function)
 (= (SET.FTN$composition [function2relation set.mor$target]) relation)
 (forall (?A (set.obj$set ?A)) ?f ((function ?A) ?f)
 ?x ((sequence ?A) ?x) ?a (?A ?a)
 (<=> (((function2relation ?a) ?f) ((right-insert ?A) [?x ?a]))
 (and ((set.pfn$domain ?f) ?x)
 (= (?f ?x) ?a))))
```

Quite a bit of the function concepts and axiomatization below come from the fact that  $(\mathit{seq}(A), A)$  form an interesting set pair. The function inclusion function maps the set of functions of a set  $A$  into the class of functions.

```
(40) (KIF$function function-inclusion)
 (= (KIF$source function-inclusion) set.obj$set)
 (= (KIF$target function-inclusion) SET.FTN$function)
 (forall (?A (set.obj$set ?A))
 (and (= (SET.FTN$source (function-inclusion ?a)) (function ?A))
 (= (SET.FTN$target (function-inclusion ?a)) set.ftn$function)
 (= (function-inclusion ?A)
 (SET.FTN$inclusion [(function ?A) set.mor$function]))))
```

The arity set of  $\varphi$  is the subset  $\#_A(\varphi) = \mathit{arity}_A(\varphi) = \{n \in \mathbb{N} \mid \varphi \text{ defined on } A^n\}$ . The domain of  $\varphi$  is the subset of sequences  $\mathit{dom}_A(\varphi) = \cup_{n \in \#(\varphi)} A^n$ . The tuple domain function

$$\mathit{dom}(A) : \mathit{ftn}(A) \rightarrow \wp \mathit{seq}(A)$$

can serve as a colimit arity.

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

Any set  $A$  has a set of domain cases

$$\begin{aligned} \mathit{dom-coprod}(A) &= \sum \mathit{dom-arity}(L) \\ &= \sum_{\varphi \in \mathit{ftn}(A)} \mathit{dom}(A)(\varphi) \\ &= \{(\varphi, a) \mid \varphi \in \mathit{ftn}(L), a \in \mathit{dom}(A)(\varphi)\}, \end{aligned}$$

that is the coproduct of its index arity.

# The IFF Simple Common Logic (meta) Ontology

Robert E. Kent

Page 36

May 9, 2004

For any set  $A$  and any function  $\varphi \in \mathit{ftn}(A)$  there is a case *injection* function:

$$\mathit{inj}(A)(\varphi) : \mathit{dom}(A)(\varphi) \rightarrow \mathit{dom-coprod}(A)$$

defined by  $\mathit{inj}(A)(\varphi)(a) = (\varphi, a)$  for all  $A$ -sequence  $a \in \mathit{dom}(A)(\varphi)$ . Obviously, the injections are injective. They commute with projection and inclusion.

```
(42) (SET.FTN$function domain-coproduct)
 (= (SET.FTN$source domain-coproduct) set.obj$set)
 (= (SET.FTN$target domain-coproduct) set.obj$set)
 (= domain-coproduct (SET.FTN$composition [domain-arity set.col.art$colimit]))
```

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

Any set  $A$  defines *indication* and *projection* functions based on its domain arity:

$$\mathit{indic}(A) : \mathit{dom-coprod}(A) \rightarrow \mathit{ftn}(A),$$

$$\mathit{proj}(A) : \mathit{dom-coprod}(A) \rightarrow \mathit{seq}(A).$$

These are defined by

$$\mathit{indic}(A)((\varphi, a)) = \varphi \text{ and } \mathit{proj}(A)((\varphi, a)) = a$$

for all function  $\varphi \in \mathit{ftn}(A)$  and all  $A$ -sequences  $a \in \mathit{dom}(A)(\varphi)$ .

```
(44) (SET.FTN$function indication)
 (= (SET.FTN$source indication) set.obj$set)
 (= (SET.FTN$target indication) set.mor$function)
 (= (SET.FTN$composition [indication set.mor$source]) domain-coproduct)
 (= (SET.FTN$composition [indication set.mor$target]) function)
 (= indication (SET.FTN$composition [domain-arity set.col.art$indication]))
```

```
(45) (SET.FTN$function projection)
 (= (SET.FTN$source projection) set.obj$set)
 (= (SET.FTN$target projection) set.mor$function)
 (= (SET.FTN$composition [projection set.mor$source]) domain-coproduct)
 (= (SET.FTN$composition [projection set.mor$target]) sequence)
 (= projection (SET.FTN$composition [domain-arity set.col.art$projection]))
```

Any set  $A$  defines an *application* function:

$$*_A = \mathit{apply}(A) : \mathit{dom-coprod}(A) \rightarrow A.$$

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

$$\mathit{apply}(A)((\varphi, a)) = \varphi(a)$$

for all functions  $\varphi \in \mathit{ftn}(A)$  and all sequences  $a \in \mathit{dom}(A)(\varphi)$ . Abstractly, it is the comediator of the coproduct cotuple (cocone) consisting of the collection of functions regarded as functions,

$$\{\varphi : \mathit{dom}(A)(\varphi) \rightarrow A \mid \varphi \in \mathit{ftn}(A)\}.$$

The application function commutes with the injection function and the function itself. If the indexed sequences in  $\mathit{dom-coprod}(A)$  are viewed as roles (prehensions), the application is a reference function from roles to objects (actualities).

```
(46) (SET.FTN$function application)
 (= (SET.FTN$source application) set.obj$set)
 (= (SET.FTN$target application) set.mor$function)
 (= (SET.FTN$composition [application set.mor$source]) domain-coproduct)
 (= (SET.FTN$composition [application set.mor$target])
 (SET.FTN$identity set.obj$set))
 (forall (?A (set.obj$set ?A))
 (= (application ?A)
 ((set.col.art$cotupling (domain-arity ?A)) (function-inclusion ?A))))
```