Appendix: The KIF Encoding for a Classification Ontology

This paper provides a formal semantics for classifications. Following the thesis that classifications are a preliminary representation or prototype for ontological models, by extension it provides a formal semantics for ontologies. This appendix discusses the KIF encoding for an ontology of classifications. Again, since classifications are a prototype for ontological models, this KIF ontology could be viewed of as a theory of ontologies, or in other words an "ontology of ontologies."

A key observation for accomplishing this formalism goes as follows. For each example of a KIF ontology previously given, there has been only one universal classification with incidence relation symbolized by the absolute KIF instantiation-predication notation

`(`*`type instance`*`).`

In a building blocks approach to ontology development there is a need for multiple classifications. One straightforward way for enabling this relative specification is (1) to use a generic incidence predicate '`|=`' and (2) to use a parameter that indicates the specific classification. The following KIF-like instantiation-predication notation symbolizes the incidence relation of these relative classifications

`(|= `*`classification instance type`*`).`

Basic Formalisms

We assume the classes `UnaryRelation`, `BinaryRelation`, `TernaryRelation`, `UnaryFunction` and `BinaryFunction` have already been defined. We use the term `Instance` instead of `Entity` and the term `Type` instead of `Class`. The following core axiom states that "in the role of instantiation-predication only instances are prehending and only types are prehended." We make no assumptions about either disjointness or subtyping between `Instance` and `Type`.

```
(UnaryRelation Instance)
(UnaryRelation Type)
(forall (?i ?t) (=> (?t ?i) (and (Instance ?i) (Type ?t))))
```

The following is a KIF representation for the elements of a classification. Such elements are useful for the declaration and population of a classification. Classifications are specified by declaration and population. The unary KIF predicate '`Classification`' represents the object aspect of the category Classification – it allows one to *declare* classifications. The binary KIF predicates '`inst`' and '`typ`' and the ternary KIF predicate '`|=`' resolve classifications into their parts, thus allowing one to *populate* classifications. Relative instantiation-predication represented by '`|=`' is compatible with the absolute KIF instantiation-predication.

```
(UnaryRelation Classification)
(BinaryRelation inst)
(forall (?c ?i) (=> (inst ?c ?i) (and (Classification ?c) (Instance ?i))))
(BinaryRelation typ)
(forall (?c ?t) (=> (typ ?c ?t) (and (Classification ?c) (Type ?t))))
(TernaryRelation |=)
(forall (?c ?i ?t) (=> (|= ?c ?i ?t) (and (Classification ?c) (Instance ?i) (Type ?t))))
(forall (?c ?i ?t)
    (and (=> (|= ?c ?i ?t) (?t ?i))
         (=> (not (|= ?c ?i ?t)) (not (?t ?i)))))
```

Classifications are related through infomorphisms. The following is a KIF representation for the elements of an infomorphism. Such elements are useful for the definition of an infomorphism. Infomorphisms are also specified by declaration and population. The unary KIF predicate '`Infomorphism`', along with the two unary KIF functions '`src`' and '`tgt`', represent the morphism (arrow) aspect of the category Classification – the predicate '`Infomorphism`' allows one to *declare* infomorphisms themselves, and the functions '`src`' and '`tgt`' allow one to *declare* their associated domain and codomain classifications, respectively. The binary KIF functions '`instFn`' and '`typFn`' resolve infomorphisms into their parts, thus allowing one to *populate* infomorphisms.

```
(UnaryRelation Infomorphism)
(UnaryFunction src)
(forall (?f ?c) (=> (= (src ?f) ?c) (and (Infomorphism ?f) (Classification ?c))))
(UnaryFunction tgt)
```

```
(forall (?f ?c) (=> (= (tgt ?f) ?c) (and (Infomorphism ?f) (Classification ?c))))
(BinaryFunction instFn)
(forall (?f ?i2 ?i1)
    (=> (= (instFn ?f ?i2) ?i1)
        (and (Infomorphism ?f) (inst (tgt ?f) ?i2) (inst (src ?f) ?i1))))
(BinaryFunction typFn)
(forall (?f ?t1 ?t2)
    (=> (= (typFn ?f ?t1) ?t2)
        (and (Infomorphism ?f) (typ (src ?f) ?t1) (typ (tgt ?f) ?t2))))
```

Here is the KIF for the fundamental property of an infomorphism

```
(forall (?f ?i2 ?t1)
    (=> (and (Infomorphism ?f) (inst (tgt ?f) ?i2) (typ (src ?f) ?t1))
        (<=> (|= (src ?f) (instFn ?f ?i2) ?t1)
             (|= (tgt ?f) ?i2 (typFn ?f ?t1)))))
```

The unary KIF function ‘`ident`’ associates a well-defined (identity) infomorphism with any classification, whose instance function is the identity on instances and whose type function is the identity on types.

```
(UnaryFunction ident)
(forall (?c ?f) (=> (= (ident ?c) ?f) (and (Classification ?c) (Infomorphism ?f))))
(forall (?c) (=> (Classification ?c)
             (and (= (src (ident ?c)) ?c) (= (tgt (ident ?c)) ?c)))
(forall (?c ?i ?t)
    (and (=> (inst ?c ?i) (= (instFn (ident ?c) ?i) ?i))
         (=> (typ ?c ?t) (= (typFn (ident ?c) ?t) ?t))))
```

The binary KIF function ‘`comp`’ operates on any two infomorphisms that are compatible in the sense that the target classification of the first is equal to the source classification of the second. It returns a well-defined (composition) infomorphism, whose instance function is the composition of the instance functions of the components and whose type function is the composition of the type functions of the components.

```
(BinaryFunction comp)
(forall (?f1 ?f2 ?f)
    (=> (= (comp ?f1 ?f2) ?f)
        (and (Infomorphism ?f1) (Infomorphism ?f2) (Infomorphism ?f))))
(forall (?f1 ?f2)
    (=> (and (Infomorphism ?f1) (Infomorphism ?f2) (exists (?f) (= ?f (comp ?f1 ?f2))))
        (= (tgt ?f1) (src ?f2))))
(forall (?f1 ?f2)
    (=> (and (Infomorphism ?f1) (Infomorphism ?f2) (exists (?f) (= ?f (comp ?f1 ?f2))))
        (and (= (src ?f) (src ?f1)) (= (tgt ?f) (tgt ?f2)))))
(forall (?f1 ?f2)
    (=> (and (Infomorphism ?f1) (Infomorphism ?f2) (exists (?f) (= ?f (comp ?f1 ?f2))))
        (and (forall (?i3)
                 (=> (inst (tgt ?f2) ?i3)
                     (= (instFn f ?i3) (instFn ?f1 (instFn ?f2 ?i3)))))
             (forall (?t1)
                 (=> (typ (src ?f1) ?t1)
                     (= (typFn f ?t1) (typFn ?f2 (typFn ?f1 ?t1))))))))
```

This KIF formalism satisfies the axioms for a category (see the text *Categories for the Working Mathematician* by Saunders Mac Lane): the KIF predicate ‘`Classification`’ specifies the objects of the category; the KIF predicate ‘`Infomorphism`’ specifies the morphisms (arrows) of the category; the KIF predicates ‘`src`’ and ‘`tgt`’ provide for domain and codomain operations that assign classifications to infomorphisms; the KIF function ‘`ident`’ provides identities; and the KIF function ‘`comp`’ provides for an associative composition of infomorphisms.

```
(forall (?a ?f)
    (=> (and (Classification ?a) (Infomorphism ?f) (= ?a (src ?f)))
        (= (comp (ident ?a) ?f) ?f)))
(forall (?f ?b)
    (=> (and (Infomorphism ?f) (Classification ?b) (= (tgt ?f) ?b))
        (= (comp ?f (ident ?b)) ?f)))
(forall (?f1 ?f2 ?f3)
    (=> (and (Infomorphism ?f1) (Infomorphism ?f2) (Infomorphism ?f3))
        (=> (and (= (tgt ?f1) (src ?f2)) (= (tgt ?f2) (src ?f3)))
            (= (comp (comp ?f1 ?f2) ?f3) (comp ?f1 (comp ?f2 ?f3))))))
```

Observation: Classification, *the category of classifications and infomorphisms, has a KIF formalism.*

Furthermore, letting Set denote the category of sets and functions, the KIF predicate ‘`inst`’ and the KIF function ‘`instFn`’ represent a KIF formalism for the functor *inst* from Classification to Set^{op} (the opposite of the category Set), and the KIF predicate ‘`typ`’ and the KIF function ‘`typFn`’ represent a KIF formalism for the functor *typ* from Classification to Set.

Other Formalisms

The ternary KIF predicates ‘`coext`’ and ‘`indist`’ represent type coextension and instance indistinguishability in KIF, respectively. The unary KIF predicates ‘`ext`’ and ‘`sep`’ represent classification extensionality and separateness in KIF, respectively.

```
(TernaryPredicate coext)
(forall (?c ?t1 ?t2)
    (=> (coext ?c ?t1 ?t2)
        (and (Classification ?c) (typ ?c ?t1) (typ ?c ?t2))))
(forall (?c ?t1 ?t2)
    (=> (coext ?c ?t1 ?t2)
        (forall (?i) (=> (inst ?c ?i) (<=> (|= ?c ?i ?t1) (|= ?c ?i ?t2))))))
(TernaryPredicate indist)
(forall (?c ?i1 ?i2)
    (=> (indist ?c ?i1 ?i2)
        (and (Classification ?c) (inst ?c ?i1) (inst ?c ?i2))))
(forall (?c ?i1 ?i2)
    (=> (indist ?c ?i1 ?i2)
        (forall (?t) (=> (typ ?c ?t) (<=> (|= ?c ?i1 ?t) (|= ?c ?i2 ?t))))))
(UnaryPredicate ext)
(forall (?c) (=> (ext ?c) (Classification ?c)))
(forall (?c)
    (=> (ext ?c)
        (forall (?t1 ?t2) (=> (and (typ ?c ?t1) (typ ?c ?t2))
                              (=> (coext ?c ?t1 ?t2) (= ?t1 ?t2))))))
(UnaryPredicate sep)
(forall (?c) (=> (sep ?c) (Classification ?c)))
(forall (?c)
    (=> (sep ?c)
        (forall (?i1 ?i2) (=> (and (inst ?c ?i1) (inst ?c ?i2))
                              (=> (indist ?c ?i1 ?i2) (= ?i1 ?i2))))))
```

Examples

Here are examples of classifications and infomorphisms taken from the text *Information Flow: The Logic of Distributed Systems* by Barwise and Seligman.

Webster	Noun	Int-Vb	Tr-Vb	Adj
bet	1	1	1	0
eat	0	1	1	0
fit	1	1	1	1
friend	1	0	1	0
square	1	0	1	1
…		…		

The following KIF represents the *Webster* classification on page 70 of Barwise and Seligman. This classification, which is (a small part of) the classification of English words according to parts of speech as given in Webster’s dictionary, is diagrammed above.

```
(Classification Webster)
(typ Webster Noun)
(typ Webster Intransitive-Verb)
(typ Webster Transitive-Verb)
(typ Webster Adjective)
(inst Webster bet)
(inst Webster eat)
```

```
(Classification Webster)
(typ Webster Noun)
(typ Webster Intransitive-Verb)
(typ Webster Transitive-Verb)
(typ Webster Adjective)
(inst Webster bet)
(inst Webster eat)
(inst Webster fit)
(inst Webster friend)
(inst Webster square)
...
(|= Webster bet Noun)
(|= Webster bet Intransitive-Verb)
(|= Webster bet Transitive-Verb)
(not (|= Webster bet Adjective))
(not (|= Webster eat Noun))
(|= Webster eat Intransitive-Verb)
(|= Webster eat Transitive-Verb)
(not (|= Webster fit Adjective))
(|= Webster fit Noun)
(|= Webster fit Intransitive-Verb)
(|= Webster fit Transitive-Verb)
(|= Webster fit Adjective)
(|= Webster friend Noun)
(not (|= Webster friend Intransitive-Verb))
(|= Webster friend Transitive-Verb)
(not (|= Webster friend Adjective))
(|= Webster square Noun)
(not (|= Webster square Intransitive-Verb))
(|= Webster square Transitive-Verb)
(|= Webster square Adjective)
...
```

The following KIF represents the infomorphism defined on page 73 of Barwise and Seligman. This represents the way that punctuation at the end of a sentence carries information about the type of the sentence. The infomorphism is from a *Punctuation* classification to a *Sentence* classification. The instances of *Punctuation* are the inscriptions of the punctuation marks of English. These marks are classified by the terms ‘`Period`, ‘`Exclamation-Mark`’, ‘`Question-Mark`’, ‘`Comma`’, etc. The instances of *Sentence* are inscriptions of grammatical sentences of English. There are but three types of *Sentence*: ‘`Declarative`’, ‘`Question`’, and ‘`Other`’. The instance function of the infomorphism assigns to each sentence its own terminating punctuation mark. The type function of the infomorphism assigns ‘`Declarative`’ to ‘`Period`’ and ‘`Exclamation-Mark`’, ‘`Question`’ to ‘`Question-Mark`’, and ‘`Other`’ to other types of *Punctuation*. The fundamental property of this infomorphism is the requirement that a sentence be of the type indicated by its punctuation.

Let ‘`yakity-yak`’ denote the command “Take out the papers and the trash!” with its punctuation symbol ‘`yy-punc`’ being the exclamation symbol at the end of the sentence. Let ‘`gettysburg1`’ denote the statement that “Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal.” with its punctuation symbol ‘`g1-punc`’ being the period at the end of the sentence. Let ‘`angels`’ denote the question “How many angels can fit on the head of a pin?” with its punctuation symbol ‘`ag-punc`’ being the question mark at the end of the sentence.

```
(Classification Punctuation)
(typ Punctuation Period)
(typ Punctuation Exclamation-Mark)
(typ Punctuation Question-Mark)
(typ Punctuation Comma)
(inst Punctuation yy-punc)
(inst Punctuation g1-punc)
(inst Punctuation ag-punc)
...
(not (|= Punctuation yy-punc Period))
(|= Punctuation yy-punc Exclamation-Mark)
(not (|= Punctuation yy-punc Question-Mark))
...

(Classification Sentence)
```

```
(typ Sentence Declarative)
(typ Sentence Question)
(typ Sentence Other)
(inst Sentence yakity-yak)
(inst Sentence gettysburg1)
(inst Sentence angels)
...
(|= Sentence yakity-yak Declarative)
(not (|= Sentence yakity-yak Question))
(not (|= Sentence yakity-yak Other))
...

(Infomorphism punct-type)
(= (src punct-type) Punctuation)
(= (tgt punct-type) Sentence)

(= (instFn punct-type yakity-yak) yy-punc)
(= (instFn punct-type gettysburg1) g1-punc)
...
(= (typFn punct-type Period) Declarative)
(= (typFn punct-type Exclamation-Mark) Declarative)
(= (typFn punct-type Question-Mark) Question)
(= (typFn punct-type Comma) Other)
...
```