Standard Upper Ontology Knowledge Interchange Format

6/18/2009

Adam Pease
apease @ articulatesoftware . com

Table of Contents

Introduction

Standard Upper Ontology Knowledge Interchange Format (SUO-KIF) is a language designed for use in the authoring and interchange of knowledge. SUO-KIF has declarative semantics. It is possible to understand the meaning of expressions in the language without appeal to an interpreter for manipulating those expressions. In this way, KIF differs from other languages that are based on specific interpreters, such as Emycin and Prolog. SUO-KIF is also logically comprehensive -- at its most general, it provides for the expression of arbitrary logical sentences. In this way, it differs from relational database languages (like SQL) and logic programming languages (like Prolog). SUO-KIF is intended primarily a first-order language, which is a good compromise between the computational demands of reasoning and richness of representation. In later sections of this document we describe ways in which SUO-KIF is made as expressive as possible without becoming higher-order.

SUO-KIF was derived from KIF (Genesereth, 1992) to support the definition of the Suggested Upper Merged Ontology (Niles & Pease, 2001).

This document contains a guide to writing knowledge in SUO-KIF as well as a more formal reference. Logicians may wish to skip the introductory section “Writing SUO-KIF” and move straight to the details of the language.

Writing SUO-KIF

This section attempts to give a very quick introduction to writing SUO-KIF. A full introduction to logic is however beyond the scope of this document and (Nolt et al, 1998) is recommended. This section will also assume the use of some of the basic ontological content of the SUMO (Niles & Pease, 2001), since otherwise we would have to define relations like **instance** and **subclass** .

SUO-KIF terms can be either individuals such as **BillClinton** and **The82ndAirborne** , or classes, such as **Person** and **MilitaryUnit** . We can define relations such as **agent** and functions such as **GovernmentFn** . Note that relations and functions are instances of the class of all relations and the class of all functions, respectively.

Terms are combined into sentences in order to make statements of fact, for example that "The 82nd Airborne is a military unit", which would be stated in SUO-KIF as

```
(instance The82ndAirborne MilitaryUnit)
```

and "The class of all Person(s) is a subclass of the class of all animals."

```
(subclass Person Animal)
```

SUO-KIF allows us to connect statements with **and** and **or** . "Kofi Annan is a human and he occupies the position of Secretary General at the United Nations."

```
(and
  (instance KofiAnnan Human)
  (occupiesPosition KofiAnnan SecretaryGeneral UnitedNations))
```

Note that the connective **or** differs from common English usage in that **or** means one or the other *or both* . So that the following statement is also true:

```
(or
  (instance KofiAnnan Human)
  (occupiesPosition KofiAnnan SecretaryGeneral UnitedNations))
```

Statements can also be negated with **not** . "Silvio Berlusconi is not the president of Libya."

```
(not
  (occupiesPosition SilvioBerlusconi President Libya))
```

SUO-KIF supports *functions* , which have the same general meaning as the familiar refrain from high-school algebra that "A function is a relation such that every value of the domain has a unique value in the range." Relational expressions are statements with a truth value. For example, the statement that the class of people is a subclass of the class of animals is true. Functional expressions however denote terms. For example **(GovernmentFn Germany)** denotes the individual entity which is the government of

Germany. `(GovernmentFn Mordor)` has no truth value. It simply denotes the government of the fictional country of Mordor.

SUO-KIF uses the characters `=>` to form rules. It is called *implication.* This can be read as *implies* , or as " *if* argument-1 *then* argument-2". Terms can also be combined to form rules such as "If a person is sleeping he or she cannot perform an intentional action".

```
(=>
  (and
    (instance ?P Human)
    (attribute ?SL Asleep))
  (not
    (exists ?ACT
      (and
        (instance ?ACT IntentionalProcess)
        (overlaps ?ACT ?SL)
        (agent ?ACT ?P)))))
```

Less often used is the operator `<=>` , which is called bi-implication. It is a shorthand for a combination of implications. Formally, `(<=> A B)` is equivalent to `(=> A B)` and `(=> B A)` .

The above rule contains a logical operator `exists` , otherwise known as existential quantification. It says that there exists some term, denoted by the variable `?ACT` that possesses the following properties. There may be more than one such entity, but there is at least one. It is a very powerful operator, which lets us talk about something that may not have been named, or whose name is unknown. The entity is described, rather than names, and it is up to an inference engine to apply that description to facts known about the world.

A related operator is `forall` , which states that the following characteristics hold about all entities denoted by a variable. To compare the two operators, consider the following examples

" All farmers like tractors." is stated as

```
(forall (?F ?T)
  (=>
    (and
      (instance ?F Farmer)
      (instance ?T Tractor))
    (likes ?F ?T)))
```

or, literally, "For all F and T if F is a farmer and T is a tractor, F likes T." "Some farmer likes a tractor." is stated as

```
(exists (?F ?T)
  (and
    (instance ?F Farmer)
    (instance ?T Tractor)
    (likes ?F ?T)))
```

or, literally, “There exists F and T such that F is a farmer, T is a tractor, and F likes T.”

Note that universal quantification (forall…) can be omitted. The interpretation of an unquantified variable is that it is quantified at the outermost scope of the formula. For example

```
(exists (?SOMEONE)
  (friend ?SOMEONE ?X))
```

Would be interpreted as “ Everyone has a friend.”

```
(forall (?X)
  (exists (?SOMEONE)
    (friend ?SOMEONE ?X)))
```

If one wanted the alternative interpretation of “ There is someone who is everyone's friend.” the quantification would need to be given explicitly as in

```
(exists (?SOMEONE)
  (forall (?X)
    (friend ?SOMEONE ?X)))
```

Although not a required part of the syntax, by convention, relations are written with an initial lowercase character, and functions, non-relational instances and classes are written with initial capital letters.

Syntax

Introduction

SUO-KIF may be described in three layers. First, there are the basic *characters* of the language. These characters can be combined to form *language elements* . Finally, the lexemes of the language can be combined to form grammatically legal *expressions* . Although this layering is not strictly essential to the specification of SUO-KIF, it simplifies the description of the syntax by dealing with white space at the lexeme level and eliminating that detail from the expression level.

The notation **`nonterminal*`** means zero or more occurrences; **`nonterminal+`** means one or more occurrences; The nonterminals **`space`** , **`tab`** , **`return`** , **`linefeed`** , and **`page`** refer to the characters corresponding to ascii codes 32, 9, 13, 10, and 12, respectively.

Characters

The alphabet of KIF consists of 7 bit blocks of data. In this document, we refer to SUO-KIF data blocks via their usual ASCII encodings as characters.

SUO-KIF characters are classified as upper case letters, lower case letters, digits, alpha characters (non-alphabetic characters that are used in the same way that letters are used), white space, and other characters (every ascii character that is not in one of the other categories). Initial characters which are the first character of a term, must be alphabetic. Constants and variables consist of an initial alphabetic character plus a sequence of alphabetic, numeric or dash characters (prepended with '?' or '@' in the case of variables).

```
upper ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
          N | O | P | Q | R | S | T | U | V | W | X | Y | Z

lower ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
          n | o | p | q | r | s | t | u | v | w | x | y | z

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

special ::= ! | $ | % | & | * | + | - | . | / | < | = | > | ? |
          @ | _ | ~ |

white ::= space | tab | return | linefeed | page

initialchar ::= upper | lower

wordchar ::= upper | lower | digit | - | _

character ::= upper | lower | digit | special | white
```

Use of characters in "special" for word characters is discouraged as they may be given particular meaning in future versions of the standard or its extensions.

Language Elements

A *character string* is a series of characters enclosed in quotation marks. The escape character `\` is used to permit the inclusion of quotation marks and the `\` character itself within such strings.

```
string ::= "character*"
```

A *constant* is a letter or digit followed by any number of other legal word characters.

```
word ::= initialchar wordchar*
```

A *variable* is a word in which the first character is `?` or `@` . A variable with a '@' character is called "row variable" or "sequence variable". It holds a variable number of arguments.

```
variable ::= ?word | @word
```

Semantically, there are four categories of constants in SUO-KIF -- object constants, function constants, relation constants, and logical constants. *Object constants* are used to denote individual objects. *Function constants* denote functions on those objects. *Relation constants* denote relations. *Logical constants* express conditions about the world and are either true or false. SUO-KIF is unusual among logical languages in that there is no syntactic distinction among these four types of constants; any constant can be used where any other constant can be used. The differences between these categories of constants is entirely semantic.

Expressions

The legal expressions of SUO-KIF are formed from lexemes according to the rules presented in this section. Terms are used to denote objects in the world being described; sentences are used to express facts about the world. Sentences can be used as terms, allowing higher-order expressions to be written. A *knowledge base* is a finite set of sentences.

```
number ::= [-] digit+ [. digit+] [exponent]
exponent ::= e [-] digit+
```

There are several types of terms in SUO-KIF -- variables, constants, character strings, and functional terms, as well as sentences themselves.

```
term ::= variable | word | string | funterm | number | sentence
```

A *functional term* consists of a constant and an arbitrary number of *argument* terms surrounded by matching parentheses. Note that there is no syntactic restriction on the number of argument terms; arity restrictions in SUO-KIF are treated semantically.

```
 argument ::= sentence | term

relword ::= initialchar wordchar* | variable
funword ::= initialchar wordchar* | variable
```

No "relword" and "funword" shall have the same character sequence in a particular knowledge base.

```
funterm ::= (funword argument+)
```

The following BNF defines the set of legal sentences in SUO-KIF. There are six types of sentences. We have already mentioned logical constants.

```
sentence ::= word | equation | relsent | logsent | quantsent | ?word
```

An *equation* consists of the `=` operator and two terms.

```
equation ::= (= term term)
```

An *implicit relational sentence* consists of a constant and an arbitrary number of *arguments* . As with functional terms, there is no syntactic restriction on the number of arguments in a relation sentence.

```
relsent ::= (relword argument+)
```

The syntax of *logical sentences* depends on the logical operator involved. A sentence involving the `not` operator is called a *negation* . A sentence involving the `and` operator is called a *conjunction* , and the arguments are called *conjuncts* . A sentence involving the `or` operator is called a *disjunction* , and the arguments are called *disjuncts* . A sentence involving the `=>` operator is called an *implication* ; the first argument is called the *antecedent* ; and the last argument is called the *consequent* . A sentence involving the `<=>` operator is called an *equivalence* .

```
logsent ::= (not sentence) |
            (and sentence+) |
            (or sentence+) |
            (=> sentence sentence) |
            (<=> sentence sentence)
```

There are two types of *quantified sentences* -- a *universally quantified sentence* is signalled by the use of the `forall` operator, and an *existentially quantified sentence* is signalled by the use of the `exists` operator. The first argument in each case is a list of variable specifications. A variable specification is either a variable or a list consisting of a variable and a term denoting a relation that restricts the domain of the specified variable.

```
quantsent ::= (forall (variable+) sentence) |
```

```
(exists (variable+) sentence)
```

Note that, according to these rules, it is permissible to write sentences with *free* variables, i.e. variables that do not occur within the scope of any enclosing quantifiers. The significance of the free variables in a sentence depends on the use of the sentence. When we assert the truth of a sentence with free variables, we are, in effect, saying that the sentence is true for all values of the free variables, i.e. the variables are universally quantified. When we ask whether a sentence with free variables is true, we are, in effect, asking whether there are any values for the free variables for which the sentence is true, i.e. the variables are existentially quantified.

It is important to keep in mind that a knowledge base is a *set* of sentences, not a *sequence* ; and, therefore, the order of forms within a knowledge base is unimportant. Order *may* have heuristic value to deductive programs by suggesting an order in which to use those sentences; however, this implicit approach to knowledge exchange lies outside of the definition of SUO-KIF.

Comments

Comments in SUO-KIF are indicated with a single semi-colon. All characters from the semi-colon to the end of the line can be ignored.

Logic

Logical Sentences

A	(not A)
T	F
F	T

A	B	(or A B)
T	T	T
T	F	T
F	T	T
F	F	F

A	B	(<=> A B)
T	T	T
T	F	F
F	F	T
F	T	F

A	B	(and A B)
T	T	T
T	F	F
F	T	F
F	F	F

A	B	(=> A B)
T	T	T
T	F	F
F	T	T
F	F	T

A negation is true if and only if the negated sentence is false. A conjunction is true if and only if every conjunct is true. A disjunction is true if and only if at least one of the disjuncts is true. If every antecedent in an implication is true, then the implication as a whole is true if and only if the consequent is true. If any of the antecedents is false, then the implication as a whole is true, regardless of the truth value of the consequent. A reverse implication is just an implication with the consequent and antecedents reversed. An equivalence, bi-implication, is equivalent to the conjunction of an implication and a reverse implication.

Quantified Sentences

A simple existentially quantified sentence (one in which the first argument is a list of variables) is true if and only if the embedded sentence is true for *some* value of the variables mentioned in the first argument.

A simple universally quantified sentence (one in which the first argument is a list of variables) is true if and only if the embedded sentence is true for *every* value of the variables mentioned in the first argument.

Note that the significance of free (unquantified) variables depends on context. Free variables in an assertion are assumed to be universally quantified. Free variables in a query are assumed to be existentially quantified. The interpretation of an unquantified variable is that it is quantified at the outermost scope of the formula. The meaning of free variables is determined by the way in which SUO-KIF is used. It cannot be unambiguously defined within SUO-KIF itself. To be certain of the usage in all contexts, use explicit quantifiers.

Functional Terms

The value of a functional term is obtained by applying the function denoted by the function constant in the term to the objects denoted by the arguments.

For example, the value of the term `(+ 2 3)` is obtained by applying the addition function (the function denoted by `+`) to the numbers **`2`** and **`3`** (the objects denoted by the object constants `2` and `3`) to obtain the value **`5`**, which is the value of the object constant `5`.

Relational Sentences

A simple relational sentence is true if and only if the relation denoted by the relation constant in the sentence is true of the objects denoted by the arguments. Equivalently, viewing a relation as a set of tuples, we say that the relational sentence is true if and only if the tuple of objects formed from the values of the arguments is a member of the set of tuples denoted by the relation constant.

Equations and Inequalities

An equation is true if and only if the terms in the equation refer to the same object in the universe of discourse.

An inequality is true if and only if the terms in the equation refer to distinct objects in the universe of discourse.

True and False

The truth value of `true` is true, and the truth value of `false` is false.

Computational Complexity and Expressiveness

Variables in the Predicate Position

SUO-KIF allows variables in the predicate position, such as (?REL ?V1 ?V2). Technically, if the value of ?REL ranges over the universe of all possible relations, then SUO_KIF would be higher-order. However, in a practical reasoning system, ?REL needs only to range over the set of relations already defined in the knowledge base, which is first order. Most reasoning systems however take the broader interpretation, and will disallow variables in the predicate position. One solution for this is to pre-process every statement to add a “dummy” relation, such as (holds ?REL ?V1 ?V2), and then remove the dummy relation in proof output. This is the solution taken in the Sigma knowledge engineering system (Pease, 2003).

Row Variables

While the unbounded implementation the existence of row variables would make SUO-KIF technically an "infinitary logic", with associated issues in efficient implementation, a bounded interpretation does keep SUO-KIF out of first order.

One option is to treat row variables as "macros", which would get expanded automatically so

```
(=>
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 @ROW))
   (holds ?REL2 @ROW))
```

would become

```
(=>
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 ?ARG1))
   (holds ?REL2 ?ARG1))

(=>
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 ?ARG1 ?ARG2))
   (holds ?REL2 ?ARG1 ?ARG2))
```

etc.

Note that this "macro" style expansion has the problem that unlike the true semantics of row variables, that it is not infinite. If the interpretation only expands to five variables, that there is a problem if the knowledge engineer uses a relation with six.

This is the solution taken again the Sigma knowledge engineering system (Pease, 2003).

Quoting

The original version of KIF had an explicit single quote for denoting uninterpreted structures that were essentially terms. This was used to state complex expressions which could be read by humans, without incurring the computational cost of becoming higher-order. For example (believes Mary (likes John Sue)) is a higher-order expression, because the second argument to 'believes' is not a term. (believes Mary '(likes John Sue)) is first order in the original KIF because the single quote character converts the following list into a term. This however is not strictly necessary since a reasoning system can apply a quote automatically when needed. SUO-KIF allows the former expression and leaves it to a reasoning system how it wishes to handle it. If a higher-order interpretation is possible, then that is allowed. If not, then the reasoning system is responsible for quoting any argument to a relation which is not a term. Sigma employs the latter approach.

References

Genesereth, M., (1991). “Knowledge Interchange Format", In Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning, Allen, J., Fikes, R., Sandewall, E. (eds), Morgan Kaufman Publishers, pp 238-249.

Niles, I., & Pease, A., (2001), Toward a Standard Upper Ontology, in Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Chris Welty and Barry Smith, eds. See also www.ontologyportal.org

Nolt, J., Rohatyn, D., and Varzi, A., (1998). Schaum’s Outlines: Logic, second edition, McGraw-Hill.

Pease, A., (2003). The Sigma Ontology Development Environment, in Working Notes of the IJCAI-2003 Workshop on Ontology and Distributed Systems, August 9, Acapulco, Mexico. See also sigmakee.sourceforge.net

Acknowledgements
Many thanks to the US Army for funding the development of the SUMO and SUO-KIF.

Appendix A - BNF Syntax

```
upper ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
          N | O | P | Q | R | S | T | U | V | W | X | Y | Z

lower ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
          n | o | p | q | r | s | t | u | v | w | x | y | z

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

special ::= ! | $ | % | & | * | + | - | . | / | < | = | > | ? |
          @ | _ | ~ |

white ::= space | tab | return | linefeed | page

initialchar ::= upper | lower

wordchar ::= upper | lower | digit | - | _

character ::= upper | lower | digit | special | white

word ::= initialchar wordchar*

string ::= "character*"

variable ::= ?word | @word

number ::= [-] digit+ [. digit+] [exponent]

exponent ::= e [-] digit+

term ::= variable | word | string | funterm | number | sentence

relword ::= initialchar wordchar* | variable

funword ::= initialchar wordchar*

argument ::= sentence | term

funterm ::= (funword argument+)

sentence ::= word | equation | relsent | logsent | quantsent | ?word

equation ::= (= term term)

relsent ::= (relword argument+)

logsent ::= (not sentence) |
            (and sentence+) |
            (or sentence+) |
            (=> sentence sentence) |
            (<=> sentence sentence)

quantsent ::= (forall (variable+) sentence) |
              (exists (variable+) sentence)
```

Appendix B: SUO-KIF as an Internal Format

SUO-KIF is intended as a language for knowledge authoring, unlike the original KIF, which was intended primarily as a language for knowledge interchange. SUO-KIF may also be adapted for use as the internal language for theorem proving. One modification used in the Sigma knowledge engineering system, is the addition of a "quote" operator, which indicates that the formula following the quote is intended not to be interpreted as SUO-KIF, but rather as a list. This allows expression of knowledge that would otherwise be second or higher order, although no second order inference will be performed on that knowledge. To support this modification, the BNF would be changed as follows

```
argument ::= term | ‘sentence
```

Note that the Sigma system quotes higher-order formulas even if they are already enclosed in a higher-order formula, in order to make Vampire's parsing process easier.

Another issue that may be relevant for using SUO-KIF as an internal format is allowing degenerate conjunctions, disjunctions and quantifiers that may be the result of a clause reduction process. This would require the following

```
logsent ::= (not sentence) |
            (and sentence*) |
            (or sentence*) |
            (=> sentence sentence) |
            (<=> sentence sentence)

quantsent ::= (forall (variable*) sentence) |
              (exists (variable*) sentence)
```

The sentence `(and)` would have the value `false` , and the sentence `(or)` would have the value `true`.