INCOMPLETE DRAFT /DO NOT COPY OR CIRCULATE/INCOMPLETE DRAFT

The Sorted Extension for KIF. [Note 0: numbered notes enclosed in square brackets are points for discussion, not part of the text.]

1. Sorts.

1.1 Introduction

The general idea is that the universe of discourse is divided into particular named non-empty subsets, and arguments and values of relations and functions are restricted to be in some of these subsets. Not all subsets of the universe qualify, and the ones that do are labelled with entities called sorts. The exact restrictions for each symbol, and the subset relations between sorts, are described by expressions called sorting declarations.

An ontology is called sorted if it contains a [Note 1: do we want to allow for several declarations? See //] sorting declaration for every individual, relation and function name used in any expression of the ontology. The set of all declarations in a sorted ontology is called the declaration set of the ontology. The declaration set is intended to provide enough information to enable a parser to efficiently decide what the intended sort of every subexpression in the language is, and to reject expressions which do not conform to the declared sort structure. It is recommended practice (though not obligatory) that the first occurrence of any name should be in its declaration.

Exactly what a sort is, in the metaphysical sense, is not specified; only that the set of all sorts is itself one of the sorted subsets, so that being a sort is a logically recognizable property. (Sorts can be thought of as syntactic labels rather than entities in any substantial ontological sense, but we include them in the universe as an expressive convenience, in order to be able to quantify over them.) We will also require that the set of sorts is closed under certain sort-forming operations, described below.

Three sorts are logically defined: the sort of the set of sorts is Sort, the sort of the truthvalues is Bool, and the sort of the universe is Top. Other sorts can be defined by the user. The logic does not require any particular relation to hold between elements of Sort and other entities, but it is recommended practice that Sort be declared to be separate from other sorts.

1.2 Sort predicates

Since each sort corresponds to a subset of the universe, there is a natural correspondence between sorts and properties. We will call the property which is true of all and only the things in a set labelled with a sort, the sorting property of that sort. It is natural to identify sorts with their sorting properties, and since predicates are first-class entities in the KIF2 core, we can do so without limiting the expressiveness of the language. We will therefore adopt a convention whereby any sort name can be used as a unary predicate symbol, called the sort predicate, where it denotes the sorting property of the sort. We will call this the sort predicate (SP) convention.
(Note that a sort property may or may not be true of the sort itself. In particular, all of
(Top Sort)
(Sort Top)
 (Top Top)
(Top Bool)
(Sort Bool)
(Sort Sort)
are logically true (using the SP convention), but
(Bool Bool)
(Bool Top)
(Bool Sort)
are logically false, and the analogous statements for other sorts will usually be false; for example the sort Integer is not itself an integer.)

1.3 Relations between sorts

Relationships between things and sorts are described by three basic relations, whose names are considered to be logical symbols in the sorted extension. (There are several other reserved names for describing the sort heirarchy, described later.)

The general relation between a member of a sorted subset and the sort which labels that subset is described by a binary relation SortOf.

(SortOf ?x ?y) is true if ?x is in a set labelled with sort ?y, false otherwise.

Note that since sorts are associated with sets, a given entity may have several sorts.
The basic relationships between sorts are described by the binary relations SubSort and DisjointSorts .

(SubSort ?x ?y) is true only if the subset labelled by ?x is wholly included in the subset labelled by ?y.

The subset relationship might hold even if SubSort is false: we allow such 'accidental' relationships between sort predicates. They represent truths which are not handled by the sorting machinery, ie are considered to be matters of 'fact' rather than matters of 'meaning'.

(DisjointSorts ?x ?y) is true if the subsets labelled by ?x and ?y have an empty intersection.

[Note 1a. Do we need any other kinds of relationship between sorts? DAML+OIL has quite a few more. See Note 3]

The following are logical truths. The first one states the SP convention, and the others use it:
(forall (?x ?y)(iff (SortOf ?x ?y)(and (Sort ?y)(?y ?x))))
(forall (?y)(iff (exists (?x)(SortOf ?x ?y))(Sort ?y)))
(forall (?x ?y ?z)(implies (and (SubSort ?x ?y)(?x ?z))(?y ?z)))
(forall (?x)(implies (Sort ?x)(SubSort ?x Top)))
(forall (?x ?y ?z)(implies (and (DisjointSorts ?x ?y)(?x ?z))(not (?y ?z))))

1.4 Individual sort declarations

Every nonlogical symbol in the sorted extension has a unique sort which is specified by a sort declaration. A declaration set must contain one [Note 2: exactly one? See ///] sorting declaration for every nonlogical symbol used in the ontology, including every sort name other than Top and Sort. (Although it is not recommended practice, Top can be used as a default assignment of a 'blank' sort to expressions which are not declared: see section ///.)

We will consider sort declarations for function and relation symbols later, but the basic syntax is the same for all sort declarations. A sort declaration has the form:

(SortOf <name> <SortExp>)

where <SortExp> is an expression describing the sort assigned to <name>. The simplest sort expression is simply the name of a sort. For example

(SortOf Number Sort)

declares the name 'Number' to have the sort Sort, ie to be a sort name. This is how names of new sorts are declared. Once declared, they can be used in further declarations.

Notice that the above can be abbreviated, using the SP convention, as:

(Sort Number)

This is always legal since 'Sort' is a logical symbol of sort Sort. The relations SubSort and DisjointSorts are used to specify relationships between sorts. For example, to specify that Integer and Real are disjoint sorts which are subsorts of Number, write the declarations:

(Sort Integer)
(Sort Real)
(SubSort Integer Number)
(SubSort Real Number)
(DisjointSorts Real Integer)

These might seem to have the same intended meanings as sentences which can be written in unsorted KIF (using the SP convention) , eg the last two above could be rendered:

(implies (Real ?x)(Number ?x))
(not (and (Real ?x)(Integer ?x)))

However, these are not precisely equivalent in meaning to the declarations, since they convey no information to the parser that the symbols 'Real' and 'Integer' are sorts rather than ordinary predicate symbols. In general, a sorting declaration has both logical content and syntactic weight. The former can be expressed in unsorted KIF, but the latter has an import within sorted KIF which is unobtainable in the unsorted language. In particular, an attempt to assert a sentence which violates the sort heirarchy would give rise to a syntax error in a reasoning engine conforming to sorted KIF; whereas its translation into unsorted KIF would be parsable, but might give rise to a logical contradiction. Much of the utility of a sorted logic lies in the fact that a large class of errors can be detected by the parser and thereby prevented from infecting a knowledge-base.

If the syntactic role of sorts are ignored, then the bare semantic content of sorted KIF can be translated into unsorted KIF using the SP convention, where every sort name is translated into a unary predicate symbol. The same information can also be rendered into or from many 'object-oriented' description logics, such as CLASSIC (ref///), by thinking of a sort as a class. (We give translations of KIF declaration syntax into CLASSIC and DAML+OIL in //// below.)

The relation symbols SortOf , SubSort and DisjointSorts can be used anywhere in a KIF expression, with their intended meanings, but such usage does not in general constitute a declaration. A declaration must be either a ground form with one of these as its first subexpression, or an abbreviation of this using the SP convention.

[NOTE 3: how much expressive power do we want to have available to describe the sort heirarchy? For example, do we want cardinality constraints, partitioning, property restrictions. etc.? (all available in DAML+OIL!) Should we take an existing description logic as our guide? I would guess that partitioning a sort might be handy, and it might also be useful to have classes of sorts.

Closely related question: how complex can the defining criteria for sorts be allowed to get? eg suppose we want to declare a sort 'transitive' for binary relations: we can declare it, but how do we say what it means?? Of course we can make assertions about everything in the sort, but can we expect the syntactic parser to be able to draw arbitrary conclusions in order to check sort compliance? (No!)
It may be that we will simply have to say that communities of users can reserve certain sort names for special usage for particular reasoners, and give transitivity and unification as one way that this might be used. Then the role of the sorting machinery is not to check meaning logically, but only transmit intended meaning through the syntax, as it were. But this means that there is no way to convey the intended content that can be so transmitted in KIF itself!]

Boolean sort expressions

In order to increase the expressive power of the language, it is useful to be able to attach sort restrictions which are combinations of other sorts. For example, we might want to say that something is both human and female, or is either a bird or an airplane, or that it is not a number. These could be stated by defining new sorts for each combination, but it is more convenient to allow Boolean combinations of sorts to occur as sort expressions, with the following syntax:

<literalSortExp> ::= <sortName> | (not <sortName>)
<booleanSortExp> ::= <literalSortExp> | (and <booleanSortExp>+) |(or <booleanSortExp>+)

These have the obvious meanings, for example:

(SortOf foo (not Integer))

declares foo to have a sort which specifies only that it is anything outside the sort Integer. Notice that this implies that the set of sorting properties must be closed under complementation.

(SortOf foo (or (and cat female) (and dog male)))

declares foo to have a sort which would ensure no breeding could take place. Note that this is well-formed only if cat, dog, male and female have already been declared as sorts.

Boolean sort expressions can only be used inside declarations. Boolean sort expressions use the same syntax as connective names, but with a different meaning. For example, conjunction ('and' applied to propositional expressions) means a function on truth-values, but 'and' used in a sort expression means set-theoretic intersection.
Note that the SP convention only applies to simple names, not to more complex sort expressions.

[Note 4: do we want this? We could allow it more genearlly but the expressions seem hard to read, and the expressive advantage seems small.]

Several equivalences hold between Boolean sort expressions:

(= (and ?x ?y)(and ?y ?x))
(= (or ?x ?y)(or ?y ?x))
(= (and ?x (or ?y ?z)) (or (and ?x ?y)(and ?x ?z)))
(= (or ?x (and ?y ?z)) (and (or ?x ?y)(or ?x ?z)))
(= (and ?x Top) ?x)
(= (or ?x Top) Top)

Relation and function sorts

Relation and function symbols can be assigned sorts just like other names, but they can also be given more complex sorts which are functions of the sorts of their arguments.

To say that a relation's arguments are restricted to a certain sort class, enclose the sort names corresponding to each argument inside parentheses. This is a relational sort expression. For example:

(SortOf Married (Human Human))

If one of the arguments can be something of any sort, one can use 'Top' as the sort name for that argument. In this context, as in a Boolean combination, Top acts a 'blank' sort, in that it imposes no sorting constraint. In particular, to say that a relation is of a certain fixed arity without specifying the sorts of its arguments, use a relational sort expression consisting entirely of Top, eg

(SortOf R (Top Top Top))

restricts R to have precisely three arguments.
Function symbols can be considered to be a special case of relation symbols and are sorted in the same way. The sort of the value of the function is the last sort listed in the relation expression. For example, the binary function divide on integers could be assigned a sort by

(SortOf divide (integer integer rational))

[Note. This means that a sort declaration does not distinguish between function and relation symbols. If we want this

classification to be part of the sort declaration then we will need to extend the syntax in some characteristic way. However I dont think there is any need to do so.]

Note that it is syntactically illegal to quantify into a fixed-arity relational expression using a row quantifier. Relation symbols which can be applied to any number of arguments can be assigned sorts by using a sequence sort expression, with the simple syntax

<seqSortExpression> ::= (@ <sortExp>).

For example, the sort expression

(@ (or Real Integer))

indicates that any finite sequence of reals or integers can occur as arguments in the argument position indicated by the expression. Sequence sort expressions using '@' can be used only inside a relation or function sort expression.

Relational and sequence expressions can be combined using boolean operations. For example, the relation married can be restricted to exclude same-sex marriages:

(SortOf married
(or ((and male human)(and female human))(and human female)(and human male))))

Complex sort expressions can often be abbreviated by equivalent forms. For example the sorting of married can be written in the equivalent form:

(SortOf married (and (human human)(or (male female)(female male))))

The complete system of identities for combining relational, boolean and sequence sort expressions is given in /// below.
