Suggestions for OWL

Alan Rector
October 2002

0. Introduction

It is very late to be suggesting significant changes to the OWL standard. However, after learning, using, teaching, (and blundering with) OWL, and the workshops and discussions around EKAW, I am concerned that OWL will not achieve its goals because::

- Reasoners for OWL may scale poorly because make stronger inferences than required or even desirable.
- Its current proposed handling of partial and complete definitions makes the semantics context dependent and requires analogous information to be expressed with radically different constructs for defined and primitive classes This follows usual description logic syntax but is poor human engineering which experience has shown to be confusing to potential authors and users.

There is a danger that constructs may be added to OWL to address these and other concerns which compromise its core semantics. I believe these problems can be addressed without changing the core semantics of OWL while delivering more functionality, at less computational cost, in a more standard way.

The proposals are to make two sorts of changes:

1) To distinguish between 'checks' and 'properties', where checks do not affect inference but raise exceptions. In particular to make it the default that domain and range constraints are 'checks'.
2) To modify the syntax to make the distinction between primitives and defined concepts clearer and to make it easier to add axioms to defined concepts , so that the syntax reflects users' intuitions from systems developed outside the DL community while still retaining the core DL semantics.

Each is discussed in turn.

1. 'checks' and 'properties'

1.1 Motivation

a) Violations of range/domain constraints are likely to be user errors or indicate that information is incomplete.

b) Checks are computationally cheap; the current classification behaviour has a high global overhead. Users are likely to want to specify checks, especially on cardinality, which would be absurdly expensive to check if treated universal constraints to used in testing for satisfiability.

Range checks are currently implemented as axioms. Builders of classifiers have found these difficult to implement efficiently. Users have found both that ontologies containing range constraints classify slowly and that they give unexpected results which are often hard to understand – either causing one (or more frequently, many) concepts to become unsatisfiable or because they cause them to be classified in unexpected ways.

Similarly, cardinality constraints often are required for input validation and other activities but not expected to be used in reasoning. They impose high costs on a reasoner but are easily checked for gross violations.

Most users' requirements would be satisfied if range and domain constraints were implemented as simple checks, performed after classification, which raised exceptions if violated.

Furthermore, violation of checks leads to error messages which are easy to interpret and from which it is easy to locate the fault.

There are two issues

a) How to allow the user to indicate their intention that something be checked rather than implanted by inference.

b) How to provide the checks in a standard way.

In effect, the user needs to indicate that they are willing to accept incomplete reasoning in this situation in return for greater efficiency and transparency. This allows users intents to be clearly indicated in a way that potential implementations can take account of them.

A given implementer could decide whether or not to use the author's meta information to produce a quick but incomplete implementation or ignore them to produce a potentially much slower but complete implementation.

The checks themselves might be specified either as part of the OWL language or, preferably, as part of a separate "OWL Query Language" which performed non-DL reasoning but treated checks in a uniform and well specified way.

1.2 Suggestion

Add two new constructs to OWL:

1.2.1 [check] [range|domain] <remaining syntax parallel to range/domain constraints>

The optional [check] before the [range|domain] clause would indicate that a check should not affect classification. After classification, or in response to a query, an environment would be expected to return a set of pairs of concepts and checks violated by those concepts. The presence of a check clause would *not* effect classification.

Ideally, I would make this the default action, and the use of range|domain on their own either deprecated or for clarity changed to an alternative syntax such as:

[impose]-[range|domain]

EXAMPLE:

variants:

isDiseaseOf check range BodyPart

isDiseaseOf impose range BodyPar

Example:

Hangnail restriction isDiseaseOf someValuesFrom Nail

In the current ontology a Nail is a meta device for fastening. The ontology is incomplete and there are no disjoint axioms separating Nail(of digit) from Nail(fastening device)

Expected behaviour in variant A:

check-violations returns {violation(isDiseaseOf check-domain BodyPart, Nail)…}

Expected behaviour in variant B:

Nail is classified as a BodyPart incorrectly without further comment. The error only comes to light much later in the course of reasoning or when the ontology is further developed and disjoint axioms added.

1.2.2 [check] property <remaining syntax parallel to property>

The check clause would, as above, The check [range|domain] clause would indicate that a check should be performed after classification or in response to a query check-variations which would return a set of pairs of concepts and checks violated by those concepts. The presence of a check clause would *not* effect classification.

Motivation: There are many cases in which authors wish to impose checks on users of the ontology or other authors. Doing so via universal constraints is computationally expensive and

the reason for resulting behaviour is often difficult to pinpoint. Doing so via a post classification check is computationally cheap and pin-points the error precisely.

EXAMPLE

variants:

variant A)

```
class Foo
  restriction hasChildren at-most-15 Person
  ….
```

variant B)

```
class-def Foo
   check restriction hasChildren at-most-15 Person
   …
```

Behaviour in variant A:

If there are many other properties or a complex KB, the computationally effort to determine the satisfiability of Foo or any of its instances grows very rapidly. It is extremely unlikely that the user actually expects to have the reasoner to detect or reason with this information

Behaviour in variant B:

If there are explicitly more than 30 child properties to an instance of Foo, check-violations returns {violation(hasChildren at-most-30 Person, Foo)…} with minimal overheads. It is very likely that the user wishes to have an input check, and may even want to allow the limit to be overridden. It may miss that some set of statements about Foo implies greater than 30 fillers for hasChildren, but this is a rare inference with a high global cost.

2. Make the distinction between defined clearer primitive classes clearer and unify the syntax implied restrictions

2.1 Motivation

- A common error in tools is to omit 'complete' (previously 'defined') in defined concepts. Making inclusion of either 'partial' or 'complete' mandatory would make the structure symmetrical and require a choice. (If tool builders want to provide an option for a default, that should be a matter of the tools rather than the language.)
- In the existing syntax, the form of restrictions which are implied by a concept but not part of its definition – *i.e.* necessary but not sufficient – depends on whether a class is primitive or defined. For primitives, necessary but not sufficient restrictions are expressed as part of the 'partial' description in the class statement. For defined concepts, they must be expressed through separate subclass axioms. This is unintuitive to users and a source of serious confusion when teaching the language.
- In the case of defined classes, the information is not expressed in the 'frame' of the class definitions.. This is a barrier to those experienced in frame systems, and makes the knowledge base difficult to browse.
- In the current syntax, information which is local is expressed as if it were global.
- The difference in expression in the current syntax makes tedious and obscure the natural evolution of ontologies. Concepts are commonly represented as primitive classes when introduced and then transformed in later versions into defined classes as the ontology develops. In our experience this amongst the most common operations in ontology development. Using the current syntax, making such a change requires a major reformulation. Restrictions from the class statement must be transformed into apparently global subclass axioms, Under the proposed modification, the change is simply to partition the restrictions on the original primitive class between a necessary and sufficient definition and a set of additional necessary restrictions, both of which are clearly local. This is likely to correspond much more closely to the author's intent. .

2.2 Suggestion

Take the current "partial"/"Complete" keywords slightly further so that they introduce independent subclauses. I would suggest changing the keywords for clarity from "complete" to

"definition" and from "partial" to "implies" ("implies" is not ideal because the use is narrower than that in the usual Lisp syntax for DLs but does not conflict, insofar as I can tell, with other usages in OWL. Alternative suggestions welcome.)

1. Change the syntax of the so that class can be followed by either or both of two subclauses:

   definedAs <description1>…<description>
   implies <description1>…<description>

   A Class statement without a definedAs clause introduces a "primitive" .
   A Class statement with both a definedAs and a implies clause is treated as if the implies clause were an axiom subclass(classname, <description1>)

A note might be added that restrictions in the implies clause of defined classes should be kept simple.

Example

A) Using current syntax

Defining malePerson as a primitive

```
class MalePerson
    partial (
          Person
          restriction hasSex someValueFrom male
          restriction hasGenitalia  someValueFrom maleGenitalia )
```

Defining malePerson as defined.

```
class MalePerson
    complete (
         Person
          restriction hasSex someValueFrom male)
```

and separately the global statement

```
axiom subclass(MalePerson,  restriction hasGenitalia someValueFrom maleGenitalia)
```

B) Using suggested syntax.

Declaring malePerson as a primitive

```
class MalePerson
    implies (
          Person
          restriction hasSex someValueFrom male
          restriction hasGenitalia  someValueFrom maleGenitalia )
```

Declaring malePerson as defined.

```
class MalePerson
    definition (
          Person
          restriction hasSex someValueFrom male)
    implies(
          restriction hasGenitalia someValueFrom maleGenitalia)
```

Comments

Note that in A), using the existing syntax, there is a major difference between the two constructs resulting in from the evolution of the ontology. In the second, defined, case, the notion of subclass in this context is likely to be is unintuitive to users. In B) The information is held together in a parallel structure. Even worse, consider the following.. Try explaining to users why the isPartOf restrictions have to be handled differently in i) and ii) below:

A – current syntax)

i)

```
class Hand
  partial (Hand
          restriction isPartOf UpperExtremity)
```

ii)

```
class SkinOfHand
  complete (Skin
              restriction isPartOf someValueFrom Hand)

axiom subclass(SkinOfHand, restriction isPartOf someValueFrom
SkinOfUpperExtremity)
```

B – Proposed syntax)

```
class Hand
   implies (Hand
                restriction isPartOf someValueFrom UpperExtremity)


class SkinOfHand
   definition (Skin
                restriction isPartOf Hand)
   implies (
              restriction isPartOf someValueFrom SkinOfUpperExtremity)
```

2.3 Final note

The current syntax is a legacy of the standard Lisp syntax for DLs. Its flaws were less important in early DLs in which it was impossible to add additional necessary restrictions to defined concepts. In modern DLs where both sets of necessary and sufficient conditions and individually necessary conditions can be applied to the same concept, it causes serious confusion and should not be perpetuated. Whether a set of restrictions is necessary and sufficient or just necessary should be uniform and indicated by the syntax of the restrictions, not the syntax declaring the class.