Macro expansion for OWL

Chris Mungall, Alan Ruttenberg, David Osumi-Sutherland, ...

April 9, 2010

Abstract

Accurate representation of complex domains such as biology demand powerful and expressive ontology languages such as OWL. However, the complex nested class expressions required for modeling some domains can be a hindrance to ontology authoring and adoption. These expressions can appear opaque to domain experts, and even users proficient in OWL can benefit from some kind of syntactic sugar, especially when authoring large ontologies.

One solution is to have domain experts fill in simple templates and translate the results into more complex axioms, but this has the disadvantage of being disconnected from full ontology authoring and reasoning environment.

We present here a method of specifying shortcut properties directly in OWL. These shortcut properties can be used in similar ways as object properties within the OWL environment, with the resulting simple axioms translated automatically to more complex axioms via macro expansion.

One of the main implications of this work is a way to simplify the translation between OBO format and OWL, and the use of RDF triple-stores.

1 Introduction

Accurate representation of complex domains such the biosciences demand powerful and expressive ontology languages such as OWL. However, the very expressive power of OWL can be a hindrance to widespread adoption and effective use - OWL axioms and class expression may not always mirror the internal cognitive models of domain experts.

In addition, the development of large ontologies often requires the repeated application of the same templates for different combinations of classes. Table 1 shows some examples.

Template	**Example**
hasPart some ('plasma membrane' and hasPart ?Y)	Class: 'alpha-beta T cell' EquivalentTo: 'T cell' and hasPart some ('plasma membrane' and hasPart 'alpha-beta TCR complex')
bearerOf some (realizedBy only ?Y)	Class: 'immune cell' EquivalentTo: 'cell' and bearerOf some (realizedBy only 'immune system process')
(?X partOf some ?Y) (?Y hasPart some ?X)	Class: finger SubClassOf: partOf some hand Class: hand SubClassOf: hasPart some finger
hasPart exactly 0 ?Y	Class: 'enucleate cell' EquivalentTo: cell and hasPart exactly 0 nucleus

Table 1: Example templates.

To the seasoned ontologist accustomed to writing complex nested expressions this may seem a trivial problem. A common response to this issue is to develop some concise "syntactic sugar" or intermediate representation, and have this translated behind the scenes to OWL. For example, on particular approach is to have domain experts fill in an Excel table, with columns corresponding to variables in the above, and to write a script to generate the OWL.

However, this approach can be unsatisfying for a number of reasons. A scripting approach is ad-hoc and difficult to integrate with existing OWL toolchains. In addition, the original intermediate representation is lost on translation into OWL. Ideally the intermediate representation would still be visible within generic OWL viewers and development tools such as Protege.

Here we describe a macro expansion language that allows the representation in a high-level intermediate form directly in OWL. A generic tool can be used to translate back and forth between the intermediate form and the expanded form. This allows users to view and edit at either level, as appropriate.

We also discuss the usage of a macro expansion language versus using and extending existing OWL2 constructs such as property chains.

Domain	Annotation Property	Use
Annotation Property	expandAssertionTo	Annotation assertion axioms connecting two classes
Object Property	expandExpressionTo	Class expressions

Table 2: Two annotation properties for describing shortcut relations.

Annotation Property	Matches	Expansion
expandAssertionTo	PropertyAssertion(?R,?X,?Y)	substitute(?R,?X,?Y)
expandExpressionTo	?R some ?Y	substitute(?R,?Y)
expandExpressionTo	?R value ?Y	substitute(?R,?Y)

Table 3: Macro expansion of shortcut relations.

2 Macro Expansion Language

We propose a macro expansion language that can be represented directly within OWL-DL. The macro expansions are represented as OWL Manchester Syntax[2] with the addition of template variables. Either annotation properties or object properties ("shortcut" relations) are annotated with macros, and these can be used to expand individual axioms, either during the authoring process or prior to reasoning. Table 2 shows the two different annotation properties that can be used to define shortcut relations, and table 3 specifies how expansion occurs.

Table **??** shows some examples. We now describe these in more detail.

2.1 Macro Expansion of Annotation Properties

We propose an annotation property expandAssertionTo that specifies how to expand a single OWL PropertyAssertion (i.e. rdf triple) into a more complex axiom by substituting two variables $?X$ and $?Y$ with the subject and target of the assertion respectively.

The rule can be specified as:

```
PropertyAssertion(?R,?X,?Y) ==> substitute(?R,?X,?Y)
```

For example:

```
AnnotationProperty: spatially_disconnected_from
```

```
Annotations: expandAssertionTo
  "(part_of some ?X and part_of some ?Y) SubClassOf owl:Nothing"
```

The following triple:

```
Class: brain
Annotations: spatially_disconnected_from 'spinal cord'
```

would expand to a general class inclusion axiom:

```
(part_of some brain and part_of some 'spinal cord') SubClassOf owl:Nothing"
```

TODO: HOW DO WE WRITE GCIs IN MANCHESTER?

However, the approach of expanding individual property assertions does not work well when we want to use our high-level properties in equivalence axioms.

Consider the following expansion rule:

```
ObjectProperty: has_plasma_membrane_part
Annotations: macro "Class ?X: SubClassOf has_part some
   ('plasma membrane' and has_part some ?Y)"
```

This works fine for:

```
Class: 'alpha-beta T cell'
Annotations: has_plasma_membrane_part some ('alpha-beta TCR complex')
```

Which would expand to:

```
Class: 'alpha-beta T cell'
SubClassOf: has_part some
   ('plasma membrane' and
    has_part some 'alpha-beta TCR complex')
```

However, because this expansion is only defined for property assertions, it cannot be used in equivalence axioms. If the ontology author wishes to write:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and
   has_plasma_membrane_part some ('alpha-beta TCR complex')
```

Then this requires a more complex means of treating expandAssertionTo assertions. In addition, the above axiom is problematic as it is using an annotation property where an object property should be used [PUNNING?]

2.2 Macro Expansion of Object Properties

As an alternative that works for both SubClass and EquivalentTo axioms, we propose a different expansion rule for a new annotation property expandExpressionTo. Here, the macro describes how to expand a class expression, not a property assertion, and only uses a single variable.

The macro object property should be used in existential restrictions, and the translation rule is:

```
?R some ?Y ==> substitute(?R,?Y)
```

For example:

```
ObjectProperty: has_plasma_membrane_part
Annotations: expandExpressionTo "has_part some ('plasma membrane' and has_part som
```

And the following intermediate-form axiom:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and has_plasma_membrane_part some ('alpha-beta TCR complex'
```

Would expand to:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and has_part some ('plasma membrane' and has_part some 'alp
```

However, this is problematic if used in certain circumstances. Consider:

```
ObjectProperty: lacks_part
Annotations: expandExpressionTo "has_part exactly 0 ?Y"
```

The author would write in the intermediate form:

```
Class: 'enucleate cell'
EquivalentTo: cell and lacks_part some nucleus
```

Which would expand to:

```
Class: 'enucleate cell'
EquivalentTo: cell and has_part exactly 0 nucleus
```

Note that the intermediate form would produce different inferences from the expanded form. This is a major point against this scenario.

We can fix this by introducing an expansion rule for hasValue restrictions:

```
?R value ?Y ==> substitute(?R,?Y)
```

Here we are treating the intermediate property as a relation between an individual and a class.

Now we can write:

```
Class: 'enucleate cell'
EquivalentTo: cell and lacks_part value nucleus
```

This expands to the correct form. Whilst the intermediate form does not produce all the same inferences as the intermediate form, the intermediate form does not produce any incorrect inferences.

We can also safely introduce an expansion rule for individuals

```
PropertyValue(?R,?X,?Y) ==> ?X Types: substitute(?R,?Y)
```

We can write property assertions:

```
Individual: imaged-cell00001234
Properties: lacks_part nucleus

Individual: imaged-cell00001234
Types: has_part exactly 0 nucleus
```

3 Extending Property Chains

TODO - merge this with discussion.

OWL2 introduces additional more expressive constructs that reduces the need for syntactic sugar or intermediate representations. One such construct is property chains - we will first explore the potential for using property chains to abstract away from complex repetitive axioms.

A biological ontology representing types of cells would likely include classifications based on proteins expressed on the surface of the cell (the *plasma membrane*). The general template can be specified as:

```
Class: ?X
EquivalentTo: 'T cell' and
  has_part some ('plasma membrane' and has_part ?Y)
```

For example, an $\alpha\beta$ T cell is a defined to be a *T cell* that has an $\alpha\beta$ *T cell receptor complex* expressed on the plasma membrane:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and
  has_part some ('plasma membrane' and has_part 'alpha-beta TCR complex')
```

With OWL2 we can use property chains and self-restrictions to declare a new property:

```
ObjectProperty: has_plasma_membrane_part
SubPropertyChain: has_part o is_plasma_membrane o has_part

ObjectProperty: is_plasma_membrane

Class: 'plasma membrane'
SubClassOf: is_plasma_membrane Self
```

This is somewhat complex for many domain experts who are not computer scientists, but if there are a limited number of such properties then the labor can be divided such that an OWL expert defines the properties and the domain expert defines classes using these properties.

Now we can use this new property to define cells in terms of their plasma membrane parts:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and has_plasma_membrane_part some ('alpha-beta TCR complex'
```

Unfortunately, the semantics of this axiom is not exactly equivalent to the previous fully expanded form. For example, if we ask for subclasses of *hasPart some plasma membrane* or *hasPart some TCR receptor complex*[1], then only the first definition works. In fact, if we ask for subclasses of *hasPart some (plasma membrane and hasPart some abTCR complex)* then again only the first fully-expanded definition works.

[1] we assume transitivity of hasPart

In order to have the same semantics, the implication of the property chain would have to go in both directions, but this is not permissible in OWL2.

It is important to point out that this may not be immediately obvious to many OWL users. Many users may be using properties as if the implication works in both directions.

One option here is to annotate the SubPropertyChain axiom:

```
SubPropertyChainOf:
  Annotations: isReversible "true"
  has_part o is_plasma_membrane o has_part
```

This in itself does not change the semantics. However, some kind of macro expansion tool could use this annotation to guide the expansion of axioms that use the contracted form to the expanded form.

The advantage here is that a subset of users (both ontology creators and downstream users of ontologies) can view the ontology through a simpler prism. The disadvantage is that it introduces complications - if for some reasoner the axioms are not expanded it will change the results of reasoning. Although in the above case it will not introduce incorrect inferences, it will fail to produce some correct inferences.

The other main disadvantage here is that the intermediate representation is not sufficiently expressive to abstract over other useful common templates, such as:

```
bearer_of some (realized_by only ?Y)
```

This kind of combination of an existential and universal restriction over an intermediate anonymous class is common in many BFO-compliant ontologies. For example:

```
Class: 'immune cell'
EquivalentTo: 'cell' and bearer_of some
     (realized_by only 'immune system process')
```

Immune cells are cells that bear some role or function that is realized by an immune system process. We could attempt to approximate this using a property chain:

```
ObjectProperty: bearer_of_realized_by
SubPropertyChain: bearer_of o realized_by
```

This could be used to write more compact axioms:

```
Class: 'immune cell'
EquivalentTo: 'cell' and
   bearer_of_realized_by some 'immune system process'
```

However, this does not have the same semantics as the original expanded form, which combines an existential and universal quantification. This means the ontology developer is forced to explicitly write out the expanded form.

4 Implementation

We have implemented a tool called Annotation Property Expander (APEX) that will apply these macros.

OK WE HAVENT YET BUT I CAN DO THIS QUICKLY

5 Challenges

Recursive expansion

6 Uses

7 Discussion

7.1 Do we need shortcut relations?

Some ontology authors may have no objection to explicitly writing the fully expanded forms of axioms. But this is not really practical for the development of large ontologies by domain experts with intermediate-level OWL skills. In these cases it is tremendously beneficial to be able to use shortcut relations in axioms, especially where these shortcut relations are used repeatedly. Ideally these shortcut relations would be both usable within the standard OWL-centric ontology authoring environment.

This also works well in terms of division of labor - a smaller number of OWL experts can define a limited set of shortcut relations for a particular domain, and a larger number of domain experts can apply these.

7.2 The hidden dangers of property chains

Property chains are a powerful feature. However, the ontology author must take care and not assume that a property restriction is *defining* the relation.

For example, a standard axiom in mereology[**?**] defines an overlaps relation in terms of parthood:

```
overlaps(x,y) <-> exists z: hasPart(x,z), partOf(z,y)
```

The overlaps relation is useful in bio-ontologies[**?**], so it can be tempting to declare an object property for it, and provide a property chain:

```
ObjectProperty: overlaps
SubPropertyChain: has_part o part_of
```

But this axiom is weaker, and there is a danger if this property is used in assertions thinking it has the same semantics. If an ontology asserts that a overlaps some b, then the class hasPart some partOf b will *not* subsume a.

It may be useful to have ontology "spell-checker" detect when properties with property chains are used in assertions, and ask the author if they really intended to write out the fully expanded form. As a matter of naming convention, it may be wise to call the object property above something different, like "infers-overlap"???

Self-restrictions add further power to property chains, by allowing class membership to be 'checked' at points along the chain. But this comes with the same health warnings as property chains, and an additional disadvantage of adding opaque axioms to OWL classes, where regular users will be looking, rather than burying the workings in property annotations, where they probably won't. Presumably in future there will be some syntactic sugar to hide this.

Overall, the temptation to define shortcut relations using property chains should be avoided. Property chains should only be used when the unidirectional implication is truly intended (and in these situations they are of course very useful).

7.3 Macro-expansion strategies

Given that property chains are insufficient in themselves to support the requirements for shortcut relations we have proposed some strategies involving an explicit macro-expansion step.

We first explored the possibility of annotating property chain axioms to indicate that the implication is intended to be bi-directional. However, this is likely to cause confusion. We decided it would be better to segregate shortcut relations completely from other object properties.

We have explored two different types of shortcut relations. This first type is modeled as an annotation property between two classes. This is then expanded to a more complex axiom or collection of axioms - possible a GCI axiom. These shortcut relations can not be used in class expressions e.g. to define classes.

The second type can formally be seen as a relation between an individual and a class. It expands to a more complex axiom involving two classes. It can be used within class expressions, and can be used to define classes. When used as a class expression, it can be used as an existential or value restriction.

7.4 Practical usage and implementation: ontology views

The scheme we have proposed is relatively simple and can be implemented easily.

However, to be maximally useful, the translator could be integrated into ontology editing environments such as Protege4[**?**]. Expanded axioms could be marked using axiom annotations. Users would have the option of viewing the ontology at the intermediate level, the expanded level, or both. Changes at one level would automatically expand/contract to the other level, and feed in to incremental reasoning.

However, even without this tight integration we believe that a translation tool implementing the macro language specified here would be useful.

There is a possible concern about reasoning over the intermediate view, prior to expansion. However, our strategy here insulates us from making incorrect inferences at the intermediate level. Users will still have to be aware of the differences between the two levels to know why they get more inferences in the expanded view.

7.5 Applications for OBO to OWL translation

OBO-Edit is an ontology editing environment popular amongst biologists[1]. The native model for this tool is the OBO file format, which can be translated to a subset of OWL[**?**][3]. Whilst it is possible to write nested class expressions in OBO format by using b-node style anonymous classes, this is not well-supported and best avoided. In addition, there is no standard way

to write other OWL constructs such as negation and universal restriction, which are becoming increasingly necessary for bio-ontologies.

Many OBO-Edit authored ontologies have started informally adopting shortcut relations with a view to translating these to expanded OWL axioms further down the line. One approach is to do this as part of the obo to owl translation, but we believe there are advantages to doing the expansion in a separate layer. For one thing, it simplifies the already complicated translation, which has some problems in the existing translation. It also has the advantage of maintaining the shortcut relations in the OWL environment.

Currently a piece of obo-format may look like this:

```
[Typedef]
id: ro:has_part
is_transitive: true
inverse_of: ro:part_of

[Typedef]
id: lacks_part
expandExpressionTo: "ro:has_part 0 ?Y" []

[Term]
id: cl:enucleate_cell
intersection_of: cl:cell
intersection_of: lacks_part go:nucleus
```

Using a simple extension of the Horrocks proposal, this would translate to:

```
ObjectProperty: lacks_part
Annotations:
  expandExpressionTo "ro:has_part exactly 0 ?Y"

Class: cl:enucleate_cell
EquivalentTo: cl:cell and lacks_part value go:nucleus
```

(PUNNING?)

Which would not lead to any incorrect inferences.

Applying the expansions rules we would get:

```
ObjectProperty: ro:has_part
Characteristics: Transitive
```

```
ObjectProperty: lacks_part
Annotations:
  expandExpressionTo "ro:has_part exactly 0 ?Y"

Class: cl:enucleate_cell
EquivalentTo: cl:cell and ro:has_part exactly 0 go:nucleus
```

Which has the intended semantics.

7.6 Applications for RDF and semantic web applications

If an OWL ontology contains complex axioms involving nested class expressions this can result in RDF triples that are difficult to work with at the RDF level. This difficulty is ameliorated by languages such as SPARQL-DL, but there is still problem for many triple stores and RDF libraries that are not OWL-aware.

The shortcut relation strategy provided here could provide a means of providing a "triple-friendly" view over complex OWL ontologies.

7.7 N-ary relations

7.8 Automatically extracting shortcut relations

8 Conclusions

9 References

References

[1] John Day-Richter, Midori A Harris, Melissa Haendel, Gene Ontology OBO-Edit Working Group, and Suzanna Lewis. OBO-Edit–an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, Aug 2007.

[2] M.Horridge, N.Drummond, J.Goodwin, A.Rector, R.Stevens, and H.Wan. The Manchester OWL Syntax. *OWL: Experience and Directions 2006*, 2006.

[3] S.H. Tirmizi, S. Aitken, D.A. Moreira, C. Mungall, J. Sequeda, N.H. Shah, and D.P. Miranker. OBO & OWL: Roundtrip Ontology Transformations. In *Semantic Web Applications and Tools for Life Sciences*, 2009.