Explanation of OWL Entailments in Protégé 4

Matthew Horridge
The University of Manchester
horridge@cs.man.ac.uk

Bijan Parsia
The University of Manchester
parsia@cs.man.ac.uk

Ulrike Sattler
The University of Manchester
sattler@cs.man.ac.uk

ABSTRACT

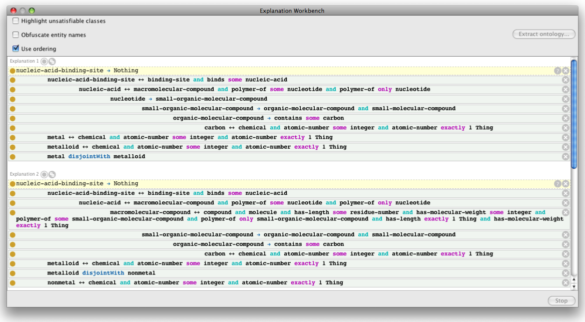
This poster and demo presents new OWL ontology explanation tools and facilities that are available in Protégé 4. These explanations take the form of *justifications*. A justification is a minimal set of axioms that is sufficient for a given entailment to hold. Justification finding services for Protégé 4 are presented, including what have become defacto explanation services such as root/derived pinpointing, and justification presentation. In addition to this, an implementation of recent theoretical work that computes so-called *precise justifications* is presented. Finally, preliminary work and new ideas of how justifications might be made easier to understand is a topic for discussion. All feedback and discussion is welcomed. Protégé 4 is open source, freely available software.

Keywords

OWL, Reasoning, Explanation

1. INTRODUCTION

Explanation of entailments is now recognised as a highly desirable, if not critical, feature that is required by end users of OWL ontology browsing and editing environments. Originally exposed to the masses via the ontology editor Swoop, explanation facilities have trickled through into other browsing and editing tools such as Protégé 4, OWL Sight and Top Braid Composer. Such facilities typically provide explanations in the form of *justifications* [5, 2], which are minimal sets of axioms that are sufficient for a particular entailment to hold. This poster showcases the explanation tools that have recently been developed for Protégé 4 (Figure 1)



```
[ ] Highlight unsatisfiable classes
[ ] Obfuscate entity names
[✓] Use ordering

Explanation 1
nucleic-acid-binding-site → Nothing
    nucleic-acid-binding-site ↔ binding-site and binds some nucleic-acid
        nucleic-acid ↔ macromolecular-compound and polymer-of some nucleotide and polymer-of only nucleotide
            nucleotide → small-organic-molecular-compound
                small-organic-molecular-compound → organic-molecular-compound and small-molecular-compound
                    organic-molecular-compound → contains some carbon
                        carbon ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
    metal ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
    metalloid ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
    metal disjointWith metalloid

Explanation 2
nucleic-acid-binding-site → Nothing
    nucleic-acid-binding-site ↔ binding-site and binds some nucleic-acid
        nucleic-acid ↔ macromolecular-compound and polymer-of some nucleotide and polymer-of only nucleotide
            macromolecular-compound ↔ compound and molecule and has-length some residue-number and has-molecular-weight some integer and
polymer-of some small-organic-molecular-compound and polymer-of only small-organic-molecular-compound and has-length exactly 1 Thing and has-molecular-weight
exactly 1 Thing
                small-organic-molecular-compound → organic-molecular-compound and small-molecular-compound
                    organic-molecular-compound → contains some carbon
                        carbon ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
    metalloid ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
    metalloid disjointWith nonmetal
    nonmetal ↔ chemical and atomic-number some integer and atomic-number exactly 1 Thing
```

Figure 1: A screenshot of the explanation work bench in Protégé 4

2. BASIC EXPLANATION SERVICES

Previous versions of Protégé include basic debugging support. For example, the colouring of unsatisfiable class names in red to permit manual tracing in order to attempt to find a root unsatisfiable class. The OWLDebugger plugin [6] also attempts to guide users in tracking down the causes of unsatisfiable classes. While these approaches are obviously improvements on no debugging support, they are only useful for trying to understand the reasons as to why a class is unsatisfiable. They do not handle demands for the explanation of arbitrary entailments. Moreover, they are heuristic based approaches and suffer from incompleteness. The latest tools that are available in Protégé 4 remedy this situation by providing the kinds of explanation facilities that users have come to expect. Specifically, the means to automatically pinpoint root unsatisfiable classes [2] and the ability to generate all justifications for any arbitrary entailment [2]. The latest incarnation of the tools bring state of the art explanation facilities to an ontology editing environment that is widely used, and it is expected that users will have the opportunity to provide feedback and suggestions on the user interfaces that present these explanations.

3. PRECISE JUSTIFICATIONS

Due to the typical construction of rich ontologies, and the way in which ontology development environments display and make it easy to edit axioms, it is frequently the case that axioms can be rather long. Hence, justifications can contain "long" axioms, where only *part* of the axioms are required for the entailment in question to hold. In many cases, these parts can obfuscate the true reasons as to why an entailment holds. Justifications that contain long axioms could also result in information being unnecessarily lost when repairing an ontology, because it isn't clear which parts of the axioms contribute to the entailment explained by the justification. Further more, in certain situations, justifications can masked other justifications [1].

Justifications that contain axioms that do not contain any redundant parts tend to be known as *precise justifications*. Recent theoretical work by the authors [1][1] has provided a, previously lacking, formal definition of precise justifications. Prior to this, it was not clear as to what constituted a precise justification, which resulted in a situation where different implementers used different ad-hoc approaches to computing justifications such that the axioms in these justifications do not contain "redundant parts". Examples include the repair tool developed as part of Lam's PhD thesis [4], or the heuristic based strikeout feature used in Swoop [3]. With a formal definition precise justifications at hand, an optimised implementation that computes precise justifications was implemented using the OWL API. This implementation has been integrated into Protégé 4 along with a user interface for browsing and working with precise justifications.

This work on precise justifications resulted in some surprising results when experiments were performed on several publicly available ontologies. For example for some ontologies it was found that for a given entailment the number of precise justifications were fewer in number that the number of regular justifications. Example were also found where regular justifications *masked* further precise justifications. Full details are available in [1].

4. LEMMAS

While justifications have proved to be incredibly useful for end users when debugging ontologies, preliminary experimental evidence suggests that, in many cases, even with justifications in hand, users can still find it difficult to understand the causes of entailments. The exact reasons for this are unknown. However, in the course of observing users who are tying to understand justifications, it has been noted that there are certain justifications that seem difficult for most users to understand. The authors hypothesise that these justifications contain non-obvious ("hidden") entailments, and in order to understand the whole justification, a user must spot these non-obvious entailments.

An example of such a case is shown in Figure 2 which is taken from an ontology about movies that was posted to the Protégé mailing list [2]. In this example, the justification for $Person \sqsubseteq Movie$ also entails that the class `Movie` is equivalent to *Thing*, and hence every class is a subclass of *Movie*. However, this isn't explicit in the justification, and for most people this is far from obvious, yet it is critical to realise that this entailment holds in order to understand the explanation.

One possible solution is to augment justifications with automatically generated lemmas. These lemmas can help to bridge the gap in understanding, highlighting the non-obvious entailments that are required in order to understand a justification. What lemmas should be used in what context is the subject of further research.

[1]Accepted as a paper in the research track at the main conference.

[2]`http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/22321/focus=22370`

$$ParentalAdvSuggested \equiv \forall hasViolenceLevel.Medium$$
$$hasViolenceLevel\ domain\ Movie$$
$$ParentalAdvSuggested \sqsubseteq CertificationCatMovie$$
$$CertificationCatMovie \sqsubseteq Movie$$

Figure 2: A justification for $Person \sqsubseteq Movie$. This is an example of a justification that seems to be difficult for most users to understand.

5. UTILISING PRECISE JUSTIFICATIONS AND LEMMAS

The use of lemmas in order to aid understanding, in conjunction with precise justifications raises issues of how they should be presented an incorporated into a user's workflow. In particular,it may not necessarily be clear how lemmas and precise justifications relate back to asserted axioms or how axioms should be modified in order to generate a repair. The challenge remains, how can lemma generation services be effectively used so as to make the services useful to end users? This is open for discussion during the poster session.

6. SUMMARY

It is hoped that this work will go some way to making it easier for users to develop OWL ontologies, giving them confidence in the knowledge that if they make a change to an ontology that results in some undesirable entailment, then they will have a better chance of understanding and removing the entailment than exists at present. In turn, this will encourage and foster the development of rich ontologies that will contribute to the enhancement of the semantic web.

7. REFERENCES

[1] M. Horridge, B. Parsia, and U. Sattler. Computing precise justifications for entailments in owl. In *ISWC 08 The International Semantic Web Conference 2008, Karlsruhe, Germany*, 2008.

[2] A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, The Graduate School of the University of Maryland, 2006.

[3] A. Kalyanpur, B. Parsia, and B. C. Grau. Beyond asserted axioms: Fine-grain justifications for owl-dl entailments. In *DL 2006, Lake District, U.K.*, 2006.

[4] S. C. J. Lam. *Methods for Resolving Inconsistencies In Ontologies*. PhD thesis, Department of Computer Science, Aberdeen, 2007.

[5] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *IJCAI International Joint Conference on Artificial Intelligence*, 2003.

[6] H. H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg. Debugging owl-dl ontologies: A heuristic approach. In *ISWC 05 The International Semantic Web Conference 2005, Galway, Ireland*, 2005.