Computing Explanations for Entailments in Description Logic Based Ontologies

Matthew Horridge* Bijan Parsia* Ulrike Sattler*
*The University of Manchester
Oxford Road Manchester, M13 9PL
{matthew.horridge|bparsia|sattler}@cs.man.ac.uk

1 Introduction

Trying to track down and understand the causes of entailments in Description Logic based ontologies[1] can be a wretched and error prone task. Without some kind of tools support many users of ontology editing tools, such as Protégé-4, find that it is impossible to determine the reasons for unsatisfiable classes or other undesirable entailments that can arise during the process of constructing an ontology. Indeed, users of such tools have been seen to switch tool purely for the benefits of explanation (Kalyanpur et al. (2007)). In recent years, there has been a significant amount of interest in generating explanations for entailments in ontologies. Generally speaking the focus has moved from finding and generating explanations that correspond closely with a particular proof technique, such as natural deduction, to finding and generating explanations whose sub-structure is at the level of asserted axioms. This has given rise to explanations that, in their most general form are known as *Justifications* (Kalyanpur et al. (2007)). A justification for an entailment in an ontology is a minimal subset of the ontology that is sufficient for the entailment to hold. More precisely, given an ontology $\mathcal{O}$ such that $\mathcal{O} \models \eta$, $\mathcal{J}$ is justification for η with respect to $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subsetneq \mathcal{J}$, $\mathcal{J}' \not\models \eta$. Note that there may be multiple, potentially overlapping, justifications for a given entailment.

2 Computing Justifications

Methods of computing justifications are broadly categorised into *glass-box* methods and *black-box* methods. While both types of method depend upon reasoning, a glass-box implementation is specific to a given reasoner and therefore reasoning technique, while a black-box method does not depend on a specific reasoner or reasoning technique.

Glass-box methods usually require thorough and non-trivial modifications to reasoner internals. The tableaux based reasoner Pellet was augmented with tableaux *tracing* whereby, as the tableaux is expanded, Pellet tracks the axioms that are used in the expansion. Computing *all* justifications for an entailment using this technique would require saturation of the completion graph, and would require many optimisations to be rolled back. Therefore a hybrid approach, combining glass-box methods with black-box methods and model diagnosis techniques is used to compute all justifications.

Black-box methods are much easier to implement than glass-box methods as they just require a reasoner[2] that can perform entailment testing, and some (goal directed) procedure to examine subsets of an ontology in order to compute all justifications. Black-box implementations typically use some optimised "expand/shrink" strategy. For example, the signature of an entailment is used as an input to a selection function that is repeatedly used to select larger and larger subsets of the ontology until the entailment in question holds in the subset, at which point axioms in the subset that are not relevant for the entailment are gradually pruned away.

The work presented in Kalyanpur et al. (2007) provides descriptions of a Pellet based glass-box implementation and black-box implementation that uses a simple expand/shrink strategy. Empirical investigation showed that it is practical to compute justifications for a range of ontologies varying in size and expressivity. More recently, we sampled entailments from over twenty published ontologies ranging from $\mathcal{ALC}$ to $\mathcal{SHOIQ}$, and from tens of axioms to tens of thousands of axioms. With the necessary optimisations to the black-box implementation, it was found that black-box justification finding can perform equally well, if not better, than glass-box justification finding—the tracing technique used in the glass box justification finding does impose a slight overhead on satisfiability testing. Given that modifying an existing reasoner to support glass-box justification finding is highly non-trivial, and binds the implementation to a specific reasoner, we recommend the use and continued optimisation of black-box methods.

[1] Description Logics being decidable fragments of First Order Logic
[2] A reasoner being an implementation of a decision procedure for satisfiability testing etc.

3 Fine-grained Justifications: Laconic and Precise Justifications

Given a, potentially very large, ontology, justifications pick out the handfuls of axioms that are responsible for the entailment. This is hugely useful since it allows a user to focus on what could be a very small part of the ontology. Thus, when trying to understand the reasons for an entailment, the user can devote their attention to examining just a few axioms compared to examining the whole ontology. However, it is frequently the case that not all *parts* of axioms are required for an entailment to hold. This has given rise to the notion of *fine-grained* justifications, which are justifications whose axioms do not contain any superfluous parts.

Numerous groups of researchers have identified fine-grained justifications as being important. However, the notion of fine-grained justifications was only recently formalised in Horridge et al. (2008), which split fined-grained justifications into laconic justifications and precise justifications. Using the well known structural transformation given in Plaisted and Greenbaum (1986) to identify the "parts" of axioms, laconic justifications are justifications all of whose axioms do not contain any superfluous parts and, more over, all of whose parts of axioms are as weak as possible. Precise justifications are laconic justifications whose parts of axioms have been transferred into separate axioms. Laconic justifications are geared towards user understanding, while precise justifications are geared towards repair.

The definition of laconic and precise justifications given in Horridge et al. (2008) is given with respect to the deductive closure of an ontology. In practice, laconic and precise justifications are computed with respect to a filter on the deductive closure of an ontology. The filter is used to select justifications that contain axioms that have a strong syntactic resemblance to the axioms in the original ontology. In essence this is essential for usability. The filters used in practice tend to generate from the ontology an expanded set of axioms that contains controlled, stepwise weakenings of the asserted axioms. While this generally results in many more justifications for an entailment, various optimisations can be used to ensure good algorithmic performance on real ontologies. Indeed, results presented in Horridge et al. (2008) show that it is practical to compute laconic justifications using black-box methods for a range of published ontologies, with all laconic justifications for an entailment typically being computed in tens of seconds on a standard laptop computer.

4 Lemmatising Justifications

We have observed that some justifications can be very difficult for people to understand. In a user study we found that approximately 20% of justifications that were generated from over 100 entailments taken from published ontologies could not be understood by a wide range of people. Justifications essentially gather the premises of a proof together and present them to a user. The user is left to fill in the gaps and work out how the interplay between axioms in order to figure out how they result in the conclusion, i.e. the entailment of interest. One of the areas that we are currently exploring is the lemmatisation of justifications so as to make helpful intermediate inference steps explicit. A justification can be lemmatised by replacing one or more axioms with summarising/bridging axiom. The result is a simpler and easier to understand justification. The replacement axiom is essentially a lemma, with the axioms that were replaced being a justification for this lemma. There can be multiple lemmatisations per justification, and the justification for lemmas can themselves be lemmatised. This results in a proof DAG that can be used as an input to a presentation service for use in end user tools such as Protégé-4. It should be noted that the notion of how easy or difficult a justification is to understand is governed by a complexity model, which is used to predict whether or not a justification ought to be lemmatised. We developed a complexity model based on the results from the aforementioned user study and used this to compute lemmatised justifications for entailments from published ontologies. It was found that it was feasible to lemmatise justifications, with many lemmatisations being computed in the order of tens of seconds.

References

Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In *ISWC 08 The International Semantic Web Conference 2008, Karlsruhe, Germany*, 2008.

Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. In *ISWC 07 The International Semantic Web Conference 2007, Busan, Korea*, 2007.

David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 1986.