Verifying Reasoner Correctness - A Justification Based Method

Michael Lee, Nico Matentzoglu, Bijan Parsia, and Uli Sattler

The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
{michael.lee-5,nicolas.matentzoglu,bijan.parsia,uli.sattler}@manchester.ac.uk

Abstract. DL reasoners are complex pieces of software that work on even more complex input which makes manual verification difficult. A single ontology can have hundreds or thousands of classes and thus its classification involve an unsurveyable number of subsumption tests. We propose a new method for debugging classification across multiple reasoners which employs justifications generated from the set of entailments that reasoners disagree upon to determine the cause of the disagreement.

Keywords: OWL, reasoning, debugging, justifications

1 Introduction

One of the major advantages of description logic reasoning are the sound and complete decision procedures for the known fragments that underpin OWL 2 and its various profiles. However, implementing these procedures can be very difficult due to inter-related optimisations and language specific limitations. It is very likely that no implementation is actually correct for all legal inputs. We have found that, in practice, reasoners do disagree on particular entailments in the inferred class hierarchy. The complexity of the implementation makes a formal verification of correctness, or even a generation of a non-arbitrary set of automated unit tests, a near impossibility. Thus, it is difficult to resolve disputes.

Majority voting (MV) is a resolution method used in reasoner competitions [4]. When a disagreement occurs over the inferred class hierarchy, the verdict of the majority of reasoners is taken as truth. In case of a tie, the correct reasoner is selected at random. This method has the key problem that it is not inherently sensitive to the truth of the matter.

We present a novel semi-automated method to determine reasoner correctness and narrow down potential causes of disagreement amongst a set of dissenting reasoners in an efficient manner. We have evaluated our method using a corpus of BioPortal ontologies.

2 The Method

Throughout, we denote $\mathcal{R}$ a description logic reasoner and $\mathcal{E}(\mathcal{O},\mathcal{R})$ the set of atomic subsumptions found by $\mathcal{R}$ (the *class hierarchy*). $\mathcal{J}$ is a justification for

$\mathcal{O} \models \eta$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$ and there is no $\mathcal{J}' \subset \mathcal{J}$ such that $\mathcal{J}' \models \eta$. We call a tuple $< \mathcal{O}, \eta, \mathcal{J}, \mathcal{R}_{just}, \mathcal{R} >$ a *case*, where $\mathcal{R}_{just}$ is the reasoner that generated the justification $\mathcal{J}$ for the entailment η and $\mathcal{R}$ is the reasoner that verified it. Aside from these notions we assume a basic familiarity with description logics, OWL and reasoners (see [1]).

The method: First, for a given ontology $\mathcal{O}$, we ask reasoners $\mathcal{R}_1, ..., \mathcal{R}_m$, denoted $\mathfrak{R}$, to compute the $\mathcal{E}(\mathcal{O}, \mathcal{R}_i)$ for all $\mathcal{R}_i \in \mathfrak{R}$. From these, we compute the set of disagreed upon entailments, denoted $\mathcal{D}$, by taking the union of all inferred class hierarchies and removing the intersection of them, $\bigcup \mathcal{E}(\mathcal{O}, \mathcal{R}_i) \setminus \bigcap \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$.

For each given entailment $\eta \in \mathcal{D}$ and every $\mathcal{R}_i \in \mathfrak{R}$, we generate a sequence of justifications $\mathcal{J}_1, \mathcal{J}_2, ...\mathcal{J}_n$. Note that we attempt to generate justifications with all reasoners, including the ones for which $\eta \notin \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$.

For each justification $\mathcal{J}_i$ for η and each reasoner $\mathcal{R}_i \in \mathfrak{R}$[1] we check whether η follows directly from the given justification (*justification verification*). We then classify all *cases* according to one of four categories. (1) If $\mathcal{R}_i$ infers $\mathcal{J}_i \models \eta$ and also initially considered $\eta \in \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$, then we classify this behaviour as *inconspicuous.* (2) If $\mathcal{R}_i$ infers $\mathcal{J}_i \models \eta$ and did not compute η as part of the generated class hierarchy, then we consider $\mathcal{R}_i$ to have a simple *bug.* This could happen because of a bug in the classification traversal algorithm or because of faulty caching. (3) If $\mathcal{R}_i$ infers $\mathcal{J}_i \not\models \eta$ and $\eta \notin \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$, then we assume that either $\mathcal{R}_i$ or the reasoner that generated the justification contains a bug. These cannot be automatically decided and we pass the justification to an expert. If the expert determines $\mathcal{J}_i$ to be a justification for η, than $\mathcal{R}_i$ is considered buggy, else the justification generating reasoner. (4) If $\mathcal{R}_i$ infers $\mathcal{J}_i \not\models \eta$ and initially returned $\eta \in \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$, then we consider this a serious error. Such cases also require expert review. In our experiment, this case did not occur.

3 Experimental Design

Corpus, Reasoners and Machines: For the experiment the OWL API (v. 3.5.0) implementations [6] of four state of the art reasoners were used: FaCT++ 1.6.3 [9], JFact 1.2.3, Pellet 2.3.1 [7] and HermiT 1.3.8 [3]. Aside from the fact that these are popular, FaCT++ and JFact were also picked because of their similarity, in an attempt to see if that similarity would produce any change to the MV and the results from the method. Moreover, the number of reasoners were set to 4 to provide enough variation between reasoners, but also to include the possibility of deadlock between them. Our corpus is a snapshot of 339 BioPortal ontologies (January 2015). All experiments are performed on a desktop computer running 4 i5-3479 CPU at 3.20 GHz with a 64 bit Ubuntu 14.04.

Identifying potential ontologies: From our corpus, we filtered out those ontologies that were merely RDFS or AL and the ones that do not fall under OWL 2 DL. For the remaining 240, we computed the inferred class hierarchy (1 hour timeout) for all four reasoners. We excluded a further 80 ontologies, for which there was a timeout. The inferred hierarchies were *normalised* using

[1] For reliability, we let reasoners verify their own justifications.

default techniques for dealing with owl:Nothing, owl:Thing, atomic subsumptions and equivalent classes. For the 160 ontologies that remained, the reasoners agreed in 155 cases and disagreed in 5.

Identifying and classifying problem cases: For the remaining 5 ontologies, we generate and verify justifications according to the method described above. We used the OWL Explanation Framework by Horridge [5][2] to generate explanations for each $\eta \in \mathcal{D}$. For performance reasons, we generate only one justification per reasoner per entailment.

Each explanation is stored in .owl format allowing us to reload them to check them against the reasoners and if needed, perform direct evaluation. For direct evaluation cases, we generate a human readable version of the justification.

Testing hypotheses and generating patches: We test four hypotheses that emerged from initial observations we will describe in more depth later: (1) The reasoner *swallows* asserted axioms, (2) the reasoner does not correctly implement the data type map (3) interdependent justifications can be reduced to one or more root justifications of an entailment that can be fixed by asserting the entailment and (4) we can use the intersection of explanations to generate patches that fix a large number of bugs.

4 Results

The 5 ontologies in our in-depth experiment were the Biological Collections Ontology (BCO), the Gene Regulation Ontology (GRO), the Sysmo Jerm Ontology of Systems Biology for Micro Organism (JERM), the Clusters of Orthologous Groups Cog Analysis Ontology (CAO) and the Cell Culture Ontology (CCONT).

Across these 5 ontologies there was one instance of total disagreement and 4 partial majorities. With respect to the total disagreement, it was found that 3 out of the 4 reasoners verified a justification for an entailment missing from their class hierarchy. HermiT, at least for our small set, did not exhibit any buggy behaviour. Table 1 shows for each ontology and reasoner $\mathcal{R}$ and ontology $\mathcal{O}$:

- Was the reasoner part of the majority (MAJ) according to the MV method?
- Was there total disagreement (DIS) or total agreement (AGG) with respect to this ontology between all reasoners?
- If there was a case that was unclear involving $\mathcal{R}$ (TIE), was the reasoner generating the justification (TIE ($\mathcal{J}$)) or verifying it (TIE ($\mathcal{R}$))?
- Was there one or more case for $\mathcal{R}$ which implicated a bug (BUG)?
- Does the verdict differ from simple majority voting (dff2MV)?

As we can see, only BCO produced at least one unclear case. According to MV, FaCT and HermiT's inferred hierarchy would be in the majority. However Pellet produced TIE cases, which meant that we needed further analysis to determine which entailment was correct. Expert review showed that Pellet in fact failed to entail something correctly, as detailed in Section 4.1. In the case of the CAO ontology (total disagreement), we could identify clear BUG cases for all

[2] https://github.com/matthewhorridge/owlexplanation

Table 1. Reasoner verdicts

$\mathcal{O}$	$\mathcal{R}$	MAJ	DIS	AGG	TIE ($\mathcal{J}$)	TIE ($\mathcal{R}$)	BUG	diff2MV
BCO	jfact	0	0	0	1	0	0	diff
BCO	fact	1	0	0	1	0	0	same
BCO	pellet	0	0	0	0	1	0	diff
BCO	hermit	1	0	0	1	0	0	same
GRO	jfact	0	0	0	1	0	1	same
GRO	fact	0	0	0	1	0	1	same
GRO	pellet	1	0	0	0	1	0	same
GRO	hermit	1	0	0	0	1	0	same
JERM	jfact	1	0	0	0	0	0	same
JERM	fact	0	0	0	0	0	1	same
JERM	pellet	1	0	0	0	0	0	same
JERM	hermit	1	0	0	0	0	0	same
CAO	jfact	0	1	0	0	0	1	same
CAO	fact	0	1	0	0	0	1	same
CAO	pellet	0	1	0	0	0	1	same
CAO	hermit	0	1	0	0	0	0	diff
CCONT	jfact	1	0	0	0	0	0	same
CCONT	fact	0	0	0	0	0	1	same
CCONT	pellet	1	0	0	0	0	0	same
CCONT	hermit	1	0	0	0	0	0	same

reasoners except HermiT. A MV based technique would chose the victor randomly, which disadvantages HermiT. Reasoning over the other three ontologies each produced a majority that we could successfully provide further confidence for. For each, any dissenting reasoner was found to have produced a BUG case and hence missed an entailment in its overall class hierarchy.

4.1 Analysis of Errors and Method for Patching

Of the 5336 cases generated by our method, 3718 were classed as inconspicuous, 1590 were classed as BUG and 28 as TIE. A further diagnostic investigation was carried out on the 1618 error cases. This diagnosis found four different types of problem for the reasoners. These are, (1) removal of asserted axioms from the ontology by FaCT++ and JFact, (2) a difficulty for FaCT++ and JFact to resolve type information, (3) Pellet being unable to determine a particular subsumption relation and (4) a case of Pellet "missing" a set of strong structurally similar justifications. For each case we used a different method to provide a diagnosis of the problem or a minimal patch to the ontology that the developer could then use to test against. This was a prudent measure, given the large number of cases. These techniques are naturally dependent on the nature of the errors produced. Ideally, we would like to provide developers a tuple of ontology, entailment, justification and patch that illustrate the faulty behaviour.

Axiom swallowing: FaCT++, JFact In analysis of those cases classed as BUG, it was discovered that 116 of the justifications were of cardinality 1. Flagging for entailments being contained in the ontology showed that the subsumption axioms were directly asserted in the ontologies. All such cases were instances of FaCT++ or JFact verifying a justification for an entailment missing in it's inferred class hierarchy. Specifically 100 of these cases were produced by FaCT, 16 by JFact. All JFact cases were "replicated" by FaCT++. These 116 cases are 28 particular axioms being missed by FaCT++ and JFact, with JFact missing 7.

This suggests a pre-processing bug on the part of FaCT++ and JFact. Of the bug cases, 1440 cases (approximately 90 %) rely upon these 28 axioms. This suggests that the vast majority of our BUG cases are down to this error.

Further support for this hypothesis comes from analysis of the overall cases. 400 out of the total 5336 cases had justifications of size 1. 4640 require these 400 cases to hold in the sample overall. This suggests that most of work was generated by FaCT++ and JFact failing to entail the result.

Data Type errors: FaCT++, JFact From the TIE cases, we found two justifications produced by FaCT and JFact that asserted a particular class was unsatisfiable. Direct examination showed that the class Decrease was classified as unsatisfiable because of the specifications of the data type.

Decrease had for its data property *polarity* a specified datatype of rdf:PlainLiteral. This was distinct from the specified range of the polarity datatype, which were all xs:string. The actual possible value was correct, but the types differed. According to W3C specifications on plain literals, such a substitution is allowed: rdf plain literals should be interpreted as xs:strings [10]. Consequently, "negative" cast as plainLiteral would be accepted as a string. This suggested that JFact and FaCT++ had problems with particular data types. We verified this fact with a pair of minimal test cases.

Wrongly Missing Entailment: Pellet The 24 additional cases classed as TIE were all generated by Pellet asserting that a particular entailment did not follow from a given justification. These cases could be collapsed into 8 unique justifications (in-between reasoner redundancy). Interestingly, each of these justifications took on a similar structural form.

All 8 justifications were verified as correct for their entailment (expert review). Because of their structural similarities and because upon examination we discovered them to have ontology classes on both the left and right hand side of the entailment, we attempted to patch the problem by taking the most general left hand side class name and the most specific right hand side class names in terms of subsumption and inserting that into the ontology. Starting with the most general on the left hand side and most specific on the right means that we are assured to affect every entailment. In this case inserting $IAO\_0000109 \sqsubseteq BFO\_0000040$ into the ontology allowed Pellet to infer the correct class hierarchy and eliminate all problematic cases from the ontology.

Classification bugs: Pellet All BUG cases generated by Pellet were found to have justifications of size 5 and to have entailments of the form $A \sqsubseteq CAO\_0000323$

(the RHS is a named class in the CAO). These similarities made us suspect that the justifications might be structurally similar and possibly share axioms.

Checking the intersection of all the justifications produced three shared axioms, $Inverse(CAO\_0000052, CAO\_0000059)$, $CAO\_0000055 \sqsubseteq \exists CAO\_0000059.CAO\_0000323$, $CAO\_0000323 \equiv CAO\_0000048 \sqcap \forall CAO\_0000052.CAO\_0000171$.

We suspected that one of these axioms caused Pellet the difficulty. We singled out the problematic axiom with the following procedure: for a given set of justifications $\mathcal{J}_1, ...\mathcal{J}_n$ take $\mathcal{S} = \bigcap_i \mathcal{J}_i$. For each $\alpha \in \mathcal{S}$ and for each reasoner $\mathcal{R}_j$ infer the class hierarchy from $\mathcal{O} \setminus \alpha$. If no bugs occur then stop. We call the removal of α a patch for the problem case.

The problematic axiom was singled out as $CAO\_0000055 \sqsubseteq \exists CAO\_0000059.CAO\_0000323$. This provided a minimal patch that a developer can use to understand the problematic reasoner behaviour.

5 Discussion

One important consequence of our justification based method is its implications for majority voting. In two of the cases we found reasons to suspect that the majority (or lack of) might lead to wrong reasoners being classified as “correct”. Moreover, we could produce information to justify the choices picked through our method and those cases where our method agreed with MV. The evidence shows a clear difference between the method stipulated and MV.

A general shortcoming of this system is that it does not catch all errors generated by the reasoners in particular those where the reasoners are all in agreement (which is the majority of the time). A more artificial problem is our restriction to a single justification per reasoner entailment pair. This problem is avoidable with greater computational resources.

To our knowledge, there is little similar work. The JustBench benchmarking methodology [2], verifies the justifications being used as benchmarks by cross checking them against all reasoners (this also emerges naturally when benchmarking). Similar work for automated reasoners has been performed with respect to queries [8]. The authors create test units (A-Boxes representations of the query) to form a test base. For certain benchmark ontologies that are used to assess reasoners, this provides a metric that evaluates the completeness of the reasoners. Importantly they also stress the need for such methods to be invariant or independent of the ontology being tested against.

6 Conclusions and Future Work

In this paper, we have presented a justification based method to reliably identify bugs in the classification algorithm of OWL reasoners. Furthermore, our method allows us to narrow down possible sources of bugs, providing a starting point for reasoner debugging. Ideally, we would like to ensure that reasoners are as correct as possible for key reasoning services such as classification to avoid confusions caused by wrongly missing entailments.

In the future, we would like to provide the method as a web service. Developers would be able to test their reasoners against a set of standard reasoners and then obtain cases that pinpoint potential bugs consisting of justifications, missing entailments and ontology patches (that make the problem disappear).

References

1. Edited by Baader F., Calavenese D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook 2nd Edition Cambridge University Press (2007)
2. Bail, S., Parsia, B. Sattler,U.: JustBench: A Framework for OWL Benchmarking. In: The Semantic Web ISWC 2010, pp. 32–47. Springer (2010)
3. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: An OWL 2 Reasoner,. In: J. Autom. Reasoning, Volume 53, pp.245–269, (2014). http://dx.doi.org/10.1007/s10817-014-9305-1.
4. Gonsalves, R.S., Bail, S., Jimenez-Ruiz, E., Matentzoglu, N., Parsia, B., Glimm, B., Kazakov, Y.: OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report. CEUR Workshop Proceedings, ORE (2013)
5. Horridge, M.: Justification Based Explaination in Ontologies. Thesis, University of Manchester (2011)
6. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. In: Semantic Web, Volume 2, Number 1, pp.11–21 (2011). http://dx.doi.org/10.3233/SW-2011-0025
7. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A. Katz, Y.: Pellet: A practical OWL-DL reasoner. In: J. Web. Sem., Volume 5, Number 2, pp.51–53, (2007) http://dx.doi.org/10.1016/j.websem.2007.03.004,
8. Stoilos, G., Grau, B.C., Horrocks, I.: How Incomplete is Your Semantic Reasoner? Systematic Analysis of the Completeness of Query Answering System. In: AAAI (2010)
9. Tsarkov, D. and Horrocks, I.: FaCT++ Description Logic Reasoner. In: IJCAR pp.292-297 (2006) http://dx.doi.org/10.1007/11814771_26
10. W3C Recommendations RDF Semantics, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#DtypeRules