A Multi-reasoner, Justification-Based Approach to Reasoner Correctness

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. OWL 2 DL is a complex logic with reasoning problems that have a high worst case complexity. Modern reasoners perform mostly very well on naturally occurring ontologies of varying sizes and complexity. This performance is achieved through a suite of complex optimisations (with complex interactions) and elaborate engineering. While the formal basis of the core reasoner procedures are well understood, many optimisations are less so, and most of the engineering details (and their possible effect on reasoner correctness) are unreviewed by anyone but the reasoner developer. Thus, it is unclear how much confidence should be placed in the correctness of implemented reasoners. To date, there is no principled, correctness unit test-like suite for simple language features and, even if there were, it is unclear that passing such a suite would say much about correctness on naturally occurring ontologies. This problem is not merely theoretical: Divergence in behaviour (thus known bugginess of implementations) has been observed in the OWL Reasoner Evaluation (ORE) contests to the point where a simple, majority voting procedure has been put in place to resolve disagreements.

In this paper, we present a new technique for finding and resolving reasoner disagreement. We use justifications to cross check disagreements. Some cases are resolved automatically, others need to be manually verified. We evaluate the technique on a corpus of naturally occurring ontologies and a set of popular reasoners. We successfully identify several correctness bugs across different reasoners, identify causes for most of these, and generate appropriate bug reports and patches to ontologies to work around the bug.

Keywords: OWL · Reasoning · Debugging · Justifications

1 Introduction

A key advantage of expressive description logic ontologies (such as those encoded into OWL 2 DL) is that automated reasoners *help*. As often stated, reasoners make implicit knowledge explicit and this has benefits both at development time and at run time. One of the most obvious development time uses is for *debugging* ontologies. Reasoners *detect* faulty entailments (e.g., contradictions or unsatisfiable classes) and are a key component in *explaining* them. At runtime,

M. Arenas et al. (Eds.): ISWC 2015, Part II, LNCS 9367, pp. 393–408, 2015.
DOI: 10.1007/978-3-319-25010-6_26

reasoners enable new sorts of functionality such as *post-coordination* [7,11,12] of terminologies as well as the discovery of new knowledge [17] or on the fly data integration [3].

Reasoners are complex pieces of software and their behaviour is opaque even to experts. Modern ontologies are typically too large and complex for any reasonable verification of the reasoners behaviour: Indeed, we rely on reasoners to help us manage those ontologies in the first place. Thus, we need techniques to help verify reasoner correctness.

This is not merely a theoretical issue (as bug lists for various reasoners attest). The complexity of the implementation makes a formal verification of correctness, or even the generation of a non-arbitrary set of automated unit tests, a near impossibility. Incompleteness (i.e., missing some entailments) is particularly challenging for human inspectors to detect both because the number of nonsubsumptions in any ontologies is very large (compared to the number of subsumptions) and because positive information has much higher saliency than missing information.

However, even if we can detect that there is a problem with a reasoner, coping with that problem is also difficult. Ideally, ontology engineers should be able to generate a succinct, informative bug report and a "patch" to their ontology that mitigates the problem (if switching from a buggy reasoner is not possible).

The detection problem can be mitigated by the use of multiple reasoners. If reasoners *disagree* on some entailment we know that there is at least one problem in at least one of the reasoners. Such disagreements naturally emerge in reasoner competitions such as the one conducted as part of the OWL Reasoner Evaluation workshop (ORE). Majority voting (MV) is often used in those competitions to resolve disagreements. When a disagreement occurs in 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 technique is obviously unreliable: the majority might be wrong, in particular because some reasoners share algorithms, optimisations, and even code. Equally obviously, this resolution technique does not help with bug reports or workarounds.

In this paper, we present an extended voting method to determine reasoner correctness and narrow down potential causes of disagreement amongst a set of dissenting reasoners in an efficient manner. It is semi-automated without too heavy a dependence on human expertise.

Similarly to ORE, we first identify disagreements in class hierarchies between reasoners. A disagreement is an entailment that some reasoners infer and others not. For every disagreement, we generate a number of justifications which will be used both to provide an extra check on the reasoners to their commitment of their side of the disagreement and to provide material for debugging. We then apply a series of automated, semi-automated, and manual inspections of the justifications to determine whether they are correct. If the justification is correct, then the disagreement is conclusively resolved in favour of the positive subusmption.

We have evaluated our method using a corpus of BioPortal ontologies. We first identify cases of potential bugs, classify them according to our method, and analyse the result to identify causes of some of the bugs.

2 Preliminaries

We assume a basic familiarity with description logics, OWL, and reasoners. For those unfamiliar with the subject we suggest the Description Logic handbook [2].

Throughout this paper, we use OWL as a short form of OWL 2 DL, $\models$ for the usual entailment relation, and $\mathcal{O}$ for an OWL ontology, i.e., a set of axioms. We use $\widetilde{\mathcal{O}}$ for the signature of $\mathcal{O}$, i.e., the set of class, property and individual names in $\mathcal{O}$. *Classification* is the process of determining, for every $A, B \in \widetilde{\mathcal{O}} \cup \{\bot, \top\}$ whether $\mathcal{O} \models A \sqsubseteq B$. If $A \in \widetilde{\mathcal{O}}$ and $\mathcal{O} \models A \sqsubseteq \bot$, then we call A *unsatisfiable*. We use $\mathcal{R}$ for a description logic reasoner and $\mathcal{E}(\mathcal{O}, \mathcal{R})$ for the set of atomic subsumptions found by $\mathcal{R}$ during classification of $\mathcal{O}$, i.e., $\mathcal{E}(\mathcal{O}, \mathcal{R})$ is the *inferred class hierarchy* of $\mathcal{O}$. $\mathcal{R}$ is *correct on* $\mathcal{O}$ if, for any $A, B \in \widetilde{\mathcal{O}} \cup \{\bot, \top\}$, we have $A \sqsubseteq B \in \mathcal{E}(\mathcal{O}, \mathcal{R})$ if and only if $\mathcal{O} \models A \sqsubseteq B$.

Given $\mathcal{O} \models \eta$, a *justification for* η is a (subset) minimal set of axioms in $\mathcal{O}$ which entails η. That is, $\mathcal{J}$ 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 $\langle \mathcal{O}, \eta, \mathcal{J}, \mathcal{R}_{just}, \mathcal{R}, v_{\mathcal{O}}, v_{\mathcal{J}} \rangle$ a *case*, where $\mathcal{R}_{just}$ is a reasoner that generated the justification $\mathcal{J}$ for the entailment η, $\mathcal{R}$ is the reasoner that tested it and came to the following verdicts:

- $v_{\mathcal{O}} = 1$ if $\eta \in \mathcal{E}(\mathcal{O}, \mathcal{R})$, and 0 otherwise.
- $v_{\mathcal{J}} = 1$ if $\eta \in \mathcal{E}(\mathcal{J}, \mathcal{R})$, and 0 otherwise.

3 A Fine-Grained Justification Based Method for Verifying Reasoner Correctness

We can identify reasoner disagreements in three ways:

- *Ontology level*: reasoners have produced different inferred class hierarchies for the same ontology.
- *Entailment level*: reasoners have different verdicts whether $\mathcal{O} \models \eta$, i.e., $\eta \in \mathcal{E}(\mathcal{J}, \mathcal{R}) \setminus \mathcal{E}(\mathcal{J}, \mathcal{R}')$. An ontology level disagreement requires at least one entailment level disagreement (and vice versa). If we consider more than two reasoners, then there are only two sides to an entailment level disagreement (thus, there will always be two coalitions). This is not true at the ontology level, where a set of n reasoners can produce n distinct class hierarchies.
- *Entailment-justification level*: reasoners have different verdicts whether $\mathcal{J} \models \eta$, for a given justifcation $\mathcal{J}$ for entailment $\eta \in \mathcal{E}(\mathcal{O}, \mathcal{R})$.

At each level, coalitions can occur where varying subsets of the reasoners are in agreement. By looking at more granular decisions (e.g., not just the whole hierarchy but individual entailments; not just entailments from the whole ontology but also from purported justifications), we have more "voting opportunities" which allow us to make more informed decisions about where the errors probably lie as well as justifications which are much easier for humans to verify.

The proposed method has three parts: 1) Discover disagreements and record them in the form of cases, 2) classify cases, and 3) resolve disagreements. Steps 1 and 2 are fully automated, whereas 3 involves some human intervention.

To discover disagreements, given an ontology $\mathcal{O}$ and reasoners $\mathfrak{R} = \mathcal{R}_1, ..., \mathcal{R}_m$ (where $m \geq 2$), we:

1. **Determine ontology agreement:** First, compute $\mathcal{E}(\mathcal{O}, \mathcal{R})$ for each $\mathcal{R} \in \mathfrak{R}$. If there is a pair $\mathcal{R}_j, \mathcal{R}_k \in \mathfrak{R}$ such that $\mathcal{E}(\mathcal{O}, \mathcal{R}_j) \neq \mathcal{E}(\mathcal{O}, \mathcal{R}_k)$, then there is a disagreement with respect to $\mathcal{O}$ and we continue with (2).
2. **Extract entailment disagreements:** compute all entailments H that are found by some but not all reasoners, i.e., $H = \bigcup \mathcal{E}(\mathcal{O}, \mathcal{R}_i) \setminus \bigcap \mathcal{E}(\mathcal{O}, \mathcal{R}_i)$.
3. **Extract justifications:** For all $\eta \in H$ and $\mathcal{R} \in \mathfrak{R}$, we attempt to extract a justification[1] in $\mathcal{O}$ for η using $\mathcal{R}$ (we try all reasoners for this, including the ones for which $\eta \notin \mathcal{E}(\mathcal{O}, \mathcal{R})$). If successful, we record the pair $\langle \mathcal{J}, \mathcal{R} \rangle$. Please note that $\mathcal{J}$ may be the same or different for the same entailment across different reasoners.
4. **Justification testing:** For each justification $\mathcal{J}$ for η and each reasoner $\mathcal{R} \in \mathfrak{R}$ we check whether $\eta \in \mathcal{E}(\mathcal{J}, \mathcal{R})$ and record the resulting case (see above) in $\mathcal{C}$.

Next, we classify each case $c = \langle \mathcal{O}, \eta, \mathcal{J}, \mathcal{R}_{just}, \mathcal{R}, v_{\mathcal{O}}, v_{\mathcal{J}} \rangle \in \mathcal{C}$ in one of four categories:

1. **Consistent Yes**, where $v_{\mathcal{O}}$ and $v_{\mathcal{J}}$ are both 1. This means that $\mathcal{R}$ found η by classification and verified the justification.
2. **Consistent No**, where $v_{\mathcal{O}}$ and $v_{\mathcal{J}}$ are both 0. This means that $\mathcal{R}$ did not find η by classification and rejected the justification.
3. **Definite Bug**, where $v)\mathcal{O}$ is 0 and $v_{\mathcal{J}}$ is 1. Here, $\mathcal{R}$ did not find η by classification but accepted the justification. This is quite problematic since montonicity dictates that if η follows from a subset of $\mathcal{O}$ it follows from $\mathcal{O}$, so *this reasoner* has an error.
4. **Possible Bug** where $v_{\mathcal{O}}$ is 1 and $v_{\mathcal{J}}$ is 0. Here, $\mathcal{R}$ found η by classification but rejected the justification. This is either due to an error in $\mathcal{R}$ or in $\mathcal{R}_{just}$ which caused it to generate a spurious $\mathcal{J}$. Thus this case indicates a possible bug with this reasoner.

[1] Note that we attempt to extract only 1 justification (per reasoner) per entailment. While there might well be multiple putative justifications that could be extracted using a given reasoner, and the more justifications, the more cases, there is a high computational cost to extracting all justifications and the cost to human verifiers is potentially even higher. As we will see, attempting 1 has been highly successful.

Finally, we resolve disagreements: having computed and categorised cases, we still do not know which side of a disagreement is correct. However, we can do more granular comparisons. Consider the following pair of (abstract) cases:

$$c_1 = \langle \mathcal{O}, \eta, \mathcal{J}, \mathcal{R}, \mathcal{R}_1, 1, 1 \rangle$$
$$c_2 = \langle \mathcal{O}, \eta, \mathcal{J}, \mathcal{R}, \mathcal{R}_2, 0, 1 \rangle$$

$\mathcal{R}_1$ found $\mathcal{O} \models \eta$ during classification and verified that $\mathcal{J} \models \eta$ for a justification $\mathcal{J} \subseteq \mathcal{O}$ extracted by $\mathcal{R}$. Contrariwise, $\mathcal{R}_2$ did not find $\mathcal{O} \models \eta$ during classification, but verified that $\mathcal{J} \models \eta$. We know, from our case classification, that $\mathcal{R}_2$ has a bug. Given that $\mathcal{R}_1$ answers consistently and that justifications are less likely to cause errors (being smaller and simpler than their parent ontologies), it is reasonable to bet on $\mathcal{R}_1$ in this case. Unless we want to seek conclusive human verification, this may be the best we can do.

To properly resolve disagreements, we need to determine whether a given justification is correct or not. If a justification is, indeed, a justification (that is, it is conclusively determined that $\mathcal{J} \models \eta$) then any reasoner which finds that $\mathcal{O} \models \eta$ is correct with respect to η. Unfortunately, the mere fact that some proposed $\mathcal{J}$ is conclusively determined *not* to be a justification, does not show that η is a nonsubsumption. It merely shows that $\mathcal{J}$ is not a justification and that the reasoner that extracted it has a bug. Even if we extracted "all" putative justifications for each reasoner and conclusively rejected them all, it would still be possible that all the reasoners were similarly generating spurious justifications. Of course, it seems rather unlikely that this would happen, but this is just a yet more detailed voting scheme. Of course, analysis of the spurious justifications might lead to a hypothesis about the reasoner's buggy behaviour which then leads to a resolution. If reasoners would present putative witness countermodels, then we could get conclusive evidence for the nonsubsumption. However, non-entailment explanation is currently a wide open problem.

There are some cases where verifying the justification is automatable. For example, if the justification is a self-justification, that is, $\mathcal{J} = \eta$ then we can just check whether $\eta \in \mathcal{O}$. Of course, this would be a very odd case for a reasoner to miss, but as we will see below, it does happen.

It is well know that sets of justifications exhibit various structures and commonalities that mean that principled comprehension of the entire set can be achieved more efficiently than by individual examination of each justification in turn. As the infrastructure for these techniques is not readily available, for the purposes of this study we experiment with some ad hoc versions. It is part of future work to provide proper support.

4 Experimental Design

Note that datasets and supporting materials (including a description of an earlier version of the experiment) are available at http://bit.ly/1Gzi7PB.

4.1 Reasoners, and Machines, and Corpus

For the experiment the OWL API (v. 3.5.0) implementations [9] of four state of the art reasoners were used: FaCT++ 1.6.4 [16], JFact 1.2.3, Pellet 2.3.1 [13] and HermiT 1.3.8 [6].

We ran the experiment on 6 Amazon Web Service r3.large instances, 15.25 GB RAM, memory-optimised, Intel Xeon E5-2670 v2 CPU @ 2.5 GHz with a timeout of 5 hours for the overall process (including 80 minutes for each individual classification). 22 ontologies were not successfully processed: 15 failed due to timeout, 4 ran out of memory, 2 had a stack overflow (Java) and 1 caused a segmentation fault.

Corpus From a corpus of 339 BioPortal ontologies, we filtered out those that were not valid OWL DL (53), had TBoxes smaller then 50 axioms (+26), or were merely RDFS (+47) or AL (+1). This left us with 212 ontologies. Out of the 212, 190 were successfully processed according to our method in Section 3. Every process was run in a fresh Java 8 Virtual Machine, and involved generating, for 1 ontology, inferred hierarchies by four reasoners (Pellet 2.3.1, HermiT 1.3.8, JFact 1.2.3, FaCT++ 1.6.4 (snapshot)) and generating and verifying their justifications for all not agreed-upon entailments. We explicitly allowed individual reasoners to fail generating their hierarchies as long as at least two reasoners succeeded.

Identifying and Classifying Problem Cases. Of the 190 successfully processed ontologies, 181 had complete agreeement at Class Hierarchy level. For the remaining 9 ontologies, we generate and verify justifications using the OWL Explanation Framework [8] to generate explanations.

Each explanation is stored in .owl format allowing us to reload them for the purposes of checking them against the reasoner and so that if needed, we can evaluate the file directly. We also generate a human readable version of the justification in DL Syntax, split into the ABox, TBox and RBox.

5 Results

The ontology level agreements are summarised in Table 1. The first thing to note is that the reasoners exhibit a great deal of agreement: they concur on 181 of the ontologies while disagreeing on only 9. While significant, it is not evidence that the current suite of reasoners are *wildly* buggy. While nearly half of the ontologies occur in polynomial profiles, all of the bug witnesses are in OWL DL. 8 out of 9 also contain datatypes. This conforms to the expectation that the more complex (i.e., OWL DL handling) or under tested (i.e., datatype handling) parts of reasoners are more likely to be buggy. For the rest of the rows the key thing to notice is that for most of them, the bug witnesses are smaller than either the successful and agreed upon cases or the timeouts (the timeouts are significantly larger in general).

Table 2 shows key statistics about the 9 problem ontologies. The first interesting point is that FaCT++ and JFact do disagree in spite of sharing a common code lineage — JFact was produced by a translation of FaCT++ from C++ to Java and the implementation follows FaCT++ to some degree.

Table 1. Filtered corpus statistics.

	Agreement	Disagreement	Processing failed
Ontologies	181	9	22
In a Polynomial Profile	89	0	4
Contain datatypes	53	8	5
TBox (mean)	8,833	1,287	266,674
TBox (max)	415,494	3,547	2,249,883
ABox (mean)	680	1,331	10,082
ABox (max)	89,292	11,575	220,948
Nr. Classes (mean)	3,643	651	92,369
Nr. Classes (max)	110,717	1,851	517,023
Nr. Object Prop. (mean)	36	64	134
Nr. Object Prop. (max)	463	189	950
Nr. Data Prop. (mean)	5	13	2
Nr. Data Prop. (max)	117	37	11
Nr. Individuals (mean)	237	204	35
Nr. Individuals (max)	40,069	1,605	634

Table 2. Description of the problematic cases. CLS, OP, DP, and IN stand for number of classes, object properties, data properties and individuals, respectively. Coal. stands for ontology level agreement coalitions. Dis. stands for the number of entailment level disagreements. The last 4 columns list the number of axioms in the generated class hierarchy by a specific reasoner, either **F**aCT++, **H**ermiT, **J**FaCT, or **P**ellet. The blank entires indicate that the specific reason either timed out for this case, or rejected it due to not understanding a datatype (only FaCT++).

$\mathcal{O}$	TBox	ABox	Expressivity	CLS	OP	DP	IN	Coal.	Dis.	F	H	J	P
bco	599	25	SROIF(D)	136	189	16	24	2	8	571	571	571	563
cao	442	0	SHIQ(D)	204	35	2	0	3	160	1249	1379	1249	1355
dikb	648	12	ALCHOIN(D)	125	70	37	9	2	5	282	282	279	282
gro	955	7	ALCHIQ(D)	507	24	6	4	2	3	3269	3270	3269	3270
heio	295	11575	ALCHIF(D)	124	11	37	1605	2	14	172	172	193	172
nemo	2686	182	SHIQ(D)	1851	89	4	120	2	1574	14665	14665	16305	14665
obcs	1126	33	SROIQ(D)	629	36	6	22	3	75		4122	4055	4116
obi_bcgo	3586	57	SROIN(D)	1673	71	1	38	2	22		16840	16818	
stato	1678	85	SROIQ(D)	609	48	4	13	2	44		4102	4062	

However for gro, JFact and FaCT++ form a coalition against HermiT and Pellet. In the ORE majority voting scheme, this would go to a coin toss, although the verdict should be weighted toward HermiT and Pellet, as they are completely independent implementations of different underlying calculi. It is also worth noting that the entailment level differences are quite small, esp. on a percentage basis. These are not cases of large, easily detectable problems.

Table 3 shows the breakdown of cases into our four categories for each reasoner. Even this high level view is informative. Consider cao where there are three coalitions (FaCT++ and JFact vs. HermiT vs. Pellet), so majority vote picks FaCT++ and JFact. However, FaCT++, JFact, and Pellet all have significant numbers of Definite Bugs (270, 270, and 72 resp.) while HermiT has

none. Clearly, just because FaCT++ and JFact have (or seem to have) the same underlying bug is not a good reason to let them win! Similarly, for gro, we have two coalitions (FaCT++ and JFact vs. HermiT and Pellet). Thus, majority voting would flip a coin, but FaCT++ and JFact are known buggy here, while the other coalition is not. We clearly should prefer the HermiT and Pellet coalition. JFact has 12 Definite Bugs for dikb while the rest of the reasoners have none. This suggests that our efforts are best spent understanding why JFact fails to find those entailments during classification. Care must be taken with the last three rows as FaCT++ did not provide a class hierarchy for them and Pellet did not for the last two. Thus, all of their cases for those ontologies will have 0 for finding that $\mathcal{O} \models \eta$. Hence their verifying the corresponding justification might not indicate a bug, but that they would have reasoned correctly on that case if they had not failed to complete the classification. This suggests that either our procedure should be slightly modified or (better) that our case categories be. In any case, with only a bit more information, we are able to make more informed judgements about how to resolve disagreements as well as steer the manual investigation.

Table 3. Classification of all cases where the reasoner extracting the justification is different from the reasoner testing the justification, grouped by reasoner. DB = Definite bug, CY= Consistent Yes, PB = Possible Bug, and CN = Consistent No. Note that HermiT has no cases in DB so that column was omitted.

	FaCT++				HermiT			JFact				Pellet			
$\mathcal{O}$	DB	CY	PB	CN	CY	PB	CN	DB	CY	PB	CN	DB	CY	PB	CN
bco	0	16	0	0	16	0	0	0	16	0	0	0	0	0	24
cao	278	72	0	0	220	0	0	278	72	0	0	72	148	0	0
dikb	0	10	2	1	10	2	1	12	0	0	0	0	10	2	1
gro	6	1	0	0	5	1	2	6	1	0	0	0	5	1	2
heio	0	0	0	14	0	0	14	0	14	0	0	0	0	0	14
nemo	0	38	0	1273	38	0	1273	0	0	15	39	0	38	0	1273
obcs	167	0	0	5	111	4	1	150	12	4	1	16	99	0	0
obi_bcgo	52	0	0	0	38	0	0	52	0	0	0	38	0	0	0
stato	109	0	0	4	84	2	2	101	0	0	0	84	0	0	4
SUM	612	137	2	1297	522	9	1293	599	101	19	40	210	300	3	1318

5.1 Justification Analysis

A total of 1,622 distinct purported justifications were extracted from these 9 ontologies, which is a daunting number, but not inherently infeasible to survey. Furthermore, Table 2 showed that 7 of the problem ontologies had less than 100 suspect entailments (5 less than 50 with several having very few). Thus, resolving the disagreement w.r.t. most ontologies is a much easier task.

In addition to the total number, the verification effort is determined by their difficulty which is often proportionate to their size (Table 4). 99% of all justifications have less than 9 axioms in them, which is typically quite manageable. The min being 1 is not inherently surprising, but the fact that 13% (204) of justifications are of size 1 is a bit surprising. (We do further analysis of those

cases below.) These results suggest that crowdsourcing justifications verification (or, at least, sufficient elbow grease) is quite feasible (though perhaps not for real time applications like live competitions). Due to time constraints we do not attempt to verify all the justifications, but we did attempt some in order to see how feasible it is.

Table 4. Justification size distribution

Min.	1st Qu.	Median	Mean	3rd Qu.	99%	Max.
1.00	3.00	3.00	3.33	4.00	9	100.00

A-Self Justifications. In principle, such justifications can require at least a bit of reasoning (e.g., $A \sqsubseteq (B \sqcap C)$), but all 204 such justifications in this experiment are *self* justifications, that is where $\mathcal{J} = \{\eta\}$. This is quite surprising since this means that the missing η is asserted in $\mathcal{O}$ and if a reasoner should get any entailments right it should find all the asserted axioms. Indeed, we can (and do!) verify self justifications simply by checking whether $\eta \in \mathcal{O}$. Thus, all 204 are verified automatically.

For 378 *cases* in Definite Bug, the justification in question was a self justification which resolves the issue of *where* the reasoner error is (i.e., in the classification or in testing the justification). These cases occur for FaCT++ and JFact. We reported such problems discovered in an earlier round of experimentation (see http://bit.ly/1Gzi7PB) to the developer of FaCT++, who accepted the reports and attempted a fix. The current experiment used this version, but still found FaCT++ Definite Bug cases with self-justification. In personal communication with the developer, we found that the problem seems to be in the interaction between the OWL API and FaCT++ (notably, the command line version of FaCT++ does not exhibit these problems).

Note that fixing these missing entailments has a cascade effect as other justifications which depended on those lost axioms now are verified by the buggy reasoners (and, indeed, all those dependent entailments are found during classification).

This case is interesting because 1) testing for the correctness of checking the entailment of asserted axioms is not an obvious testing move 2) it highlights that problems with reasoners may stem from outside the reasoner proper, and 3) it highlights the need for analysis of the justificatory structure.

B-Manual Inspection: Pellet and BCO. We selected the 24 Consistent No cases for Pellet against the bcp ontology because Pellet was in conflict with all the other reasoners. There were 8 distinct justifications in the 24 cases, thus each non-Pellet reasoner extracted and verified each of the 8. All 8 were rather similar structurally. For example, one purported justification for $A \sqsubseteq B$ was:

$A \sqsubseteq \exists S.D$
$D \sqsubseteq \exists U.(B \sqcap \exists R.C)$
$U \sqsubseteq P$
$S \circ P \sqsubseteq Q$
$\exists Q.\top \sqsubseteq B$

All 8 of the justifications were verified independently by all the authors as correct explanations for their entailment. Thus, we verified that Pellet was incorrect.

C-Manual Inspection: FaCT++, JFact and Unsatisfiable Classes. We found two justifications produced by FaCT and JFact that asserted a particular class in the gro ontology was unsatisfiable. Since debugging unsatisfiable classes are a classic explanation target, we decided to verify them. Examination of the justifications directly showed that the class Decrease was being 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 opposed to the specified range of the polarity datatype, which were all xs:string. The actual possible value was correct, but the types differed. If the types are, in fact, different, the purported justifications are correct. However, according to the W3C specifications on plain literals, such a substitution was allowed: rdf plain literals should be interpreted as xs:strings [1]. Consequently, "negative" cast as plainLiteral should be accepted as a xs:string. "negative"^^xsd:string denotes the same as the plain literal "negative". This showed that JFact and FaCT++ had problems with particular data types.

To test this, we created a minimal type casting test to run the reasoners on. We created a simple ABox and RBox:

a:R value "x"
a:R value "x"^^string
Functional(R)

Since no individual can have more than one R successor in this ontology, it is consistent just in case the plain literal "x" can be cast to a string. This was inconsistent for FaCT++ and JFact, but not Pellet or HermiT. This test case clearly shows that FaCT++ and JFact are incorrect. This example suggested other similar examples, such as:

a:R value "x"^^rdfs:Literal
a:R value "x"^^string
Functional(R)

This was found to be inconsistent by Pellet while it crashed FaCT++.

This provides specific evidence to diagnose a data type error within FaCT and JFact (and incidentally, Pellet).

D-Manual Inspection: Heio and JFact. To accumulate evidence for or against a contested assumption, Consistent Yes and Consistent No cases tend to cancel each other out. If we have only a single case (in either category) we don't have any reason to contradict the case. However, if we have a balanced set of Consistent Yeses and Nos (i.e., half of each), then we have no information to choose between them. Contrariwise, an unbalanced set of Consistent Yeses and Nos tends to verify the cases in the majority. In the case of heio, there were 14 Consistent Yeses for JFact, and 14 Nos for each of the other reasoners, and JFact was the only reasoner to produce a justification. In this case, the smart money is against JFact. Given that there were only 14 justifications to verify and they were all of size 2, we decided to completely verify this set.

The justifications were all structurally similar:

$$\begin{aligned} \mathrm{A} &\equiv \mathrm{C} \sqcap \exists \mathrm{P}.\mathrm{integer}[>3] \\ \mathrm{B} &\equiv \mathrm{C} \sqcap \exists \mathrm{P}.\mathrm{integer}[\leq 3] \end{aligned}$$

In each case, two classes (e.g., A and B) are modelled as non-equivalent by having incompatible ranges as the value of some datatype property. Clearly, these are not real justifications for $A \sqsubseteq B$. Moreover, JFact will infer from such justifications (and thus from heio) that $A \equiv B$. Clearly, JFact is not appropriately coping with the data range facets.

A Modest Generalisation. While we exhausted all the self justifications (A), the remaining case studies are suggestive: B involves role chains, while C and D involve datatypes. Both these features are comparatively new (in their current form) as of OWL 2 and thus comparatively little used or tested. Of all our justifications, 1363 involve a datatype, of those 1347 use facets, while only 24 use role chains. (Those 24 were fully verified.) As FaCT++ and Pellet both rejected ontologies on the basis of datatypes, we have further confirmation that datatypes need special attention. Modellers using elaborate datatype modelling would be well advised to test their ontologies against a set of reasoners, not just one, and compare the results.

6 Discussion

One important consequence of our justification based method is its implications for crude methods such as majority voting. Recall that majority voting considers if there is total, partial or no consensus in the inferred class hierarchies. In the case of partial consensus, any given reasoner is correct if it agrees with the majority. In the case of no consensus, some tie-breaking method is applied (this can be as simple as flipping a coin).

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.

With respect to the reliability of MV, we have produced evidence to suggest that it's reliability is questionable. Given that we can find a clear instance of a tie-breaking method picking from a set of reasoners that are suspected to exhibit problems, this undermines MV's effectiveness. We believe that a more granular voting system paralleling some of our analyses above would produce a more satisfactory mechanisms. If a contestant wanted to dispute the voting results, they would have the comparatively easy task of verifying some justifications.

It is clear that the system is informative with respect to how the errors are produced by the reasoners. Although we were unable to diagnose the source of each error in every evaluation, we were fairly successful for comparatively minor effort. In particular, we were able to generate minimal test cases with detailed explanation of the erroneous behaviour very suitable for bug reports. Our initial bug reports have been well received. Providing this information and a strict methodology to narrow down these minimal patches should provide basis for future work. It can also be seen that for Ontology Engineers, we are either able to provide effective minimal patches to the Ontology, to allow one to work around the problem with respect to reasoning, or provide an indication that no such work around is available such as in the case of FaCT++ and JFact handling of some self-justifications.

A general shortcoming of this system is that it does not catch all errors generated by the reasoners. We will not catch errors where the reasoners are all in agreement. Either they all infer an entailment that is not correct or they all fail to infer an entailment that should follow from the ontology. Against this, we argue that as this test can be performed with multiple reasoners, it is unlikely that this will occur. The more reasoners that are in accord with each other, the more confidence we may have that such inferred entailments are correct.

Another potential source of problems might be our restriction to a single justification per reasoner entailment pair. There are only two cases where this could be a problem: (1) The reasoner generates some justifications and some non-justifications for a given entailment and (2) a reasoner is able to verify some, but not all justifications. Without formally verifying this conjecture, we have not encountered any cases like this in some preliminary experiments. It might be necessary to rule this problem out in the future with more experiments. However, the restriction does not seem to hurt us with any of the examples we have pursued.

7 Related Work

Our initial motivation was the problem of reasoner disagreement in the context of ORE. We found Majority Voting to be a very unsatisfying disagreement resolution especially when it degrades to random choice. In prior ORE competitions known approximate reasoners, such as TrOWL, have been deployed over logic levels for which their incompleteness is a known fact (so for instance TrOWL approximates OWL DL by converting statements into a level of expressivity lower then OWL DL). The balance between approximation and speed is

an understandable one and it is unlikely that incomplete reasoners will ever form a majority for MV, as most reasoners within these competitions are complete. However, a method to adjudicate against this possibility is very useful. Moreover, within the competition, it would be useful to know the degree to which an automated reasoner is incomplete with respect to the majority (of presumably complete reasoners), because this can allow us to discern the degree of penalisation to those incomplete reasoners. Finally, a key goal of ORE is to improve reasoners. While it has proved helpful in pushing the performance bound, correctness is similarly important.

Justifications have been used in benchmarking, where there analysability was held up as a virtue. In order to verify that the justifications being used as benchmarks were reliable [4, p.38], every justification generated was checked against every reasoner. Additionally for completeness purposes, the authors use a number of reasoners to generate all entailments and their justifications. They note that this "...is very time consuming and infeasible for some ontologies". This problem with generating justifications means it is necessary to have some form of hard limit on generation in order to ensure the process is not overly time consuming.

Similar work for automated reasoners has been performed with respect to queries [14]. 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. Our work is complementary to this. [14] is evaluating an expected feature of the reasoners, in so much as the implementations of them are incomplete in order to have a satisfactory level of efficiency. By comparison, our method is concerned with error, with unexpected levels of incompleteness. It is also important to note that our method fulfils the invariant condition that [14] stipulate. This method can be used with any corpus of ontologies.

In general in Automated Reasoning there are a variety of techniques deployed for debugging. We note however, that a good deal of the techniques used as detailed in the literature refer to debugging of SAT or SMT solvers, rather then any reasoner that underlies semantic web technology. For instance, fuzzing is used by Brummayer to debug SAT and SMT solvers [5]. Briefly stated fuzzing is the use of deliberate random input for bug generation and is typically used in software engineering. Moreover, there exist libraries of test cases such as "The Thousands of Problems for Theorem Provers" (TTFP) for First Order Logic and typed higher order formed logic [15]. TTFP helpfully contains solutions to these problems, allowing reasoner developers to verify the output of their solvers. The library of problems forms a benchmark for solver developers as well as a nexus for the community as a whole.

8 Conclusions and Future Work

In this paper, we have presented a justification based method to identify bugs in 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 or spurious entailments.

A limitation of the work is that it is blind in terms of exhaustiveness of error. That is, we have produced evidence to show that there is something wrong with particular reasoners on particular ontologies, axioms and entailments. However, we do not know the extent to which this is exhaustive of erroneous behaviour. While this is not within the scope of the work, it means that we must apply a conservative mindset to the results of the method. It is not that the method has shown that reasoning software is in fact sound and complete. It has only shown that it has certain errors and certain agreements.

Its known that there is a good deal of diversity between reasoner implementation, in terms of code base, calculus and use of modularity and that this does not even take into account differences in degrees of completeness. We may naively assume that such a diversity influences the effectiveness of our method, in the sense that multiple pieces of complex software that agree should grant a certain confidence in that result. However, we note that a certain degree of care needs to be made when making claims regarding the diversity of software acting as a guarantee of reasoner soundness and being a principle factor in the effectiveness of the method. This is largely because of prior research on N Version Software Development. N Version Software Development, is the concept that a way of developing reliable software is to have multiple parallel developments attempt to produce the same piece of software. Certain communication restrictions are placed on the programmers developing the software in parallel, with the intention of enforcing a diversity of methods in its creation and hence a reliability in the overall product, when they are evaluated against each other.

There are parallels with our method - we are using multiple reasoners against each other, each from separate developers. This means that certain criticisms that were made of N Version Software Development need to be kept in mind. These principally come from [10]. They test the idea that the failures that occur within strictly separated programmers (in their case, students) can be considered as independent events, that is that failure does not occur in correlation (note that they do not specify how the failure occurs). The authors conclude that such a thing is statistically unlikely to be independent. They are keen to stress that this conclusion only applies to the application they used in their experiment. While there is no straightforward application of their results in this scenario, what can be learnt is a measure of caution about how diverse we may assume the reasoners are and whether diversity of implementation itself is playing a key role in our method. This is a possible avenue for further research. Our situation is even less restricted then the one that took place in their experiment, given the vitality of the semantic web community as a whole.

Our process involves a great deal of duplication of human evaluation work, in situations where justifications are duplicated, either for the same entailment or in the cases involving unsatisfiable classes, offered elsewhere as justifications for an aspect of the class hierarchy. Compounding this problem are situations where malign justifications are generated for correct entailments. Moreover, it can be seen that with axiom swallowing, justifications for one entailment can be subsets for another justification for a different entailment (if a reasoner misses an explanation for entailments, it may miss it for situations where the entailment is used as a step elsewhere). Hence, knowing how justifications interact either as subsets or in duplication and being able to collapse these cases down into a single human readable case would make the process of analysis far more efficient.

This provides an intention for future work. We could use tools for diffing, isomorphism detection, detecting root and derived justifications and pattern detection. Deployment of anyone of these tools could greatly speed up the analysis: focusing on root justifications, for example, reduces both the number of justifications one needs to inspect and their average size. Isomorphism could easily show that a similarity of pattern has occurred across justifications. At the moment these tools are not integrated together into a suite of tools. An ideal end goal would be to produce such a suite alongside an implementation of our method to allow end users to test reasoners and ontologies for bugs and then analyse such results.

In the future, we would like to provide a web service supporting our methods. 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). Ontology engineers would be able to verify that their ontologies are treated consistently by all reasoners. In both cases, we would attempt to facilitate crowdsourcing of the justification verification task.

Furthermore, standard testing methodologies such as test case minimisation and mutation seem worth exploring in this context. While independently useful, they might also complement the other comprehension tools.

Perhaps the most important extension is the generation of a potentially conclusive witness for non-entailment. The classic technique for non-entailment explanation is the generation and display of counter models. Current implementations, being mostly of a model construction flavour, seem well suited for this. However, the structures that reasoners use internally are not directly a model, nor, in complex logics, are they particular close to sharable structures like aBoxes (which can completely capture certain sorts of model and then be tested by other reasoners). Furthermore, the models (and internal structures) can be quite large and complex potentially defying human inspection.

However, our reasoner correctness task provides a good test scenario for non-entailment explanation. One of the great barriers to such research is the difficulty of finding "interesting" non-entailments to experiment with. Our technique generates interesting non-entailments with no need for domain or ontology familiarity and with a strong motivation for understanding them.

References

1. W3C Recommendations RDF Semantics (2004). [Online; accessed April 15, 2015]
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook, 2nd edn. Cambridge University Press (2007)
3. Bagosi, T., Calvanese, D., Hardi, J., Komla-Ebri, S., Lanti, D., Rezk, M., Rodríguez-Muro, M., Slusnys, M., Xiao, G.: The ontop framework for ontology based data access. In: Zhao, D., Du, J., Wang, H., Wang, P., Ji, D., Pan, J.Z. (eds.) CSWS 2014. CCIS, vol. 480, pp. 67–77. Springer, Heidelberg (2014)
4. Bail, S., Parsia, B., Sattler, U.: JustBench: a framework for OWL benchmarking. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 32–47. Springer, Heidelberg (2010)
5. Brummayer, R., Biere, A.: Fuzzing and delta-debugging SMT solvers. In: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories, SMT 2009, New York, NY, USA, pp. 1–5. ACM (2009)
6. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: An OWL 2 Reasoner. J. Autom. Reasoning **53**(3), 245–269 (2014)
7. Hedeler, C., Parsia, B., Brandt, S.: Estimating and analysing coordination in medical terminologies. In: 2014 IEEE 27th International Symposium on Computer-Based Medical Systems, New York, NY, USA, May 27–29, 2014, pp. 357–362 (2014)
8. Horridge, M.: Justification Based Explaination in Ontologies. PhD thesis, University of Manchester (2011)
9. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011)
10. Knight, J.C., Leveson, N.G.: An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. IEEE Trans. Software Eng. **12**(1), 96–109 (1986)
11. Rector, A.L., Iannone, L.: Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in SNOMED CT. Journal of Biomedical Informatics **45**(2), 199–209 (2012)
12. Rogers, J.E.: Development of a methodology and an ontological schema for medical terminology. PhD thesis, University of Manchester (2004)
13. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. **5**(2), 51–53 (2007)
14. Stoilos, G., Grau, B.C., Horrocks, I.: How incomplete is your semantic web reasoner? In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010 (2010)
15. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning **43**(4), 337–362 (2009)
16. Tsarkov, Dmitry, Horrocks, Ian: FaCT++ description logic reasoner: system description. In: Furbach, Ulrich, Shankar, Natarajan (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)
17. Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Turi, D., Stevens, R.: A little semantic web goes a long way in biology. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 786–800. Springer, Heidelberg (2005)