Explaining Inconsistencies in OWL Ontologies

Matthew Horridge, Bijan Parsia, and Ulrike Sattler

The University of Manchester
Oxford Road, Manchester, M13 9PL
{matthew.horridge,bparsia,sattler}@cs.man.ac.uk

Abstract. Justifications play a central role as the basis for explaining entailments in OWL ontologies. While techniques for computing justifications for entailments in consistent ontologies are theoretically and practically well-understood, little is known about the practicalities of computing justifications for inconsistent ontologies. This is despite the fact that justifications are important for repairing inconsistent ontologies, and can be used as a basis for paraconsistent reasoning. This paper presents algorithms, optimisations, and experiments in this area. Surprisingly, it turns out that justifications for inconsistent ontologies are more "difficult" to compute and are often more "numerous" than justifications for entailments in consistent ontologies: whereas it is always possible to compute *some* justifications, it is often not possible to compute *all* justifications for real world inconsistent ontologies.

1 Introduction

An ontology is a machine processable artefact that captures the relationships between concepts and objects in some domain of interest. A logic based ontology language allows ontologies to be specified as logical theories, meaning that is it possible to reason about the relationships between concepts and objects that are expressed in such an ontology. A widely used logic based ontology language is the the Web Ontology Language, OWL [1], the latest version of which is underpinned by a highly expressive Description Logic[1] called $\mathcal{SROIQ}$ [3].

An *inconsistent ontology*, is an ontology that, by virtue of what has been stated in the ontology, cannot have any models, and entails everything. The effect of this is that, using standard classic semantics, no meaningful conclusions can be drawn from an inconsistent ontology. All Description Logic based OWL reasoners merely report "ontology inconsistent" when fed such ontologies.

In practice, inconsistent ontologies can arise due to a number of reasons. For example, an ontology may become inconsistent during the editing process after the addition of some axioms. While the last set of axioms that were added may play a part in making the ontology inconsistent, it may be the case that they were the correct axioms to add from a modelling point of view, but it is the interplay of the newly added axioms with axioms already in the ontology that

[1] Description Logics [2] may be regarded as decidable fragments of First Order Logic.

L. Godo and A. Pugliese (Eds.): SUM 2009, LNAI 5785, pp. 124–137, 2009.

triggers the inconsistency. Inconsistencies can also be the result of *automated* ontology construction techniques. For example, in [4], ontologies are automatically constructed as an output from text mining, and it is possible for the resulting ontologies can be inconsistent.

There are a number of different ways of dealing with an inconsistent ontology. Broadly speaking, either some form of paraconsistent reasoning can be used to draw "meaningful" conclusions from the ontology, as in [5], or the ontology can be *repaired* so that it becomes consistent, thus meaningful conclusions can be drawn from the repaired ontology using standard semantics. The work presented in this paper focuses on this latter "repair strategy", with an assumed scenario that ontologies may become inconsistent during their construction process, but not wanting to publish such ontologies, their authors would repair the ontologies before publication.

Given the assumed "repair and publish" scenario, it is necessary for ontology authors to pin point the reasons for an inconsistent ontology. These reasons may be highly non-obvious, and manually browsing an ontology to determine the causes of inconsistency, if not impossible, can be a wretched, tedious and error prone task. In practice it is therefore necessary for people to have access to some form of automated explanation generation service that, given an inconsistent ontology, can identify *minimal subsets* of the ontology that cause it to be inconsistent. These minimal subsets are known as *justifications* [6,7,8], and a procedure for finding and exposing justifications for inconsistent ontologies is the subject of this paper.

In what follows, an algorithm for computing all justifications for inconsistent ontologies is presented. An investigation on real world, hand-crafted, ontologies is carried out to evaluate the effectiveness of the algorithm for computing justifications. Finally, the problems and challenges of computing justifications for inconsistent ontologies, along with the utility of these justifications, is discussed.

2 Preliminaries: OWL and Justifications

The work presented here focuses OWL 2 ontologies which correspond to $\mathcal{SROIQ}$ knowledge bases. $\mathcal{SROIQ}$ [3] is a highly expressive description logic and can be viewed as a decidable and 2NExpTime-complete [9] fragment of First Order Logic, a close relative of the 2-variable fragment with counting and modal logics [2]. In what follows,

- A and B are used as *class names*–they correspond in FOL to unary predicates,
- R and S are *properties*, i.e., *property names* or *inverse properties* (written R^- for R a property name)–they correspond to binary predicates,
- C and D are (possibly complex) *class expressions* that can be built using Boolean operators on classes ($\sqcap$, $\sqcup$, and $\neg$), value restrictions ($(\exists R.C)$, $(\forall R.C)$), number restrictions ($(\geq nR.C)$, $(\leq nR.C)$ for n a non-negative integer), and enumerations ($\{a\}$)–they correspond to formulae in one free variable,
- a and b are *individual names*–they correspond to constants.

A $\mathcal{SROIQ}$ *ontology* is a finite set of $\mathcal{SROIQ}$ *axioms*, which take the form of subsumptions between (possibly complex) class expressions $C \sqsubseteq D$ or properties $R \sqsubseteq S$, or the form of assertions $C(a)$ or $R(a, b)$.

A definition of the syntax and semantics of $\mathcal{SROIQ}$ can be found in [3]. In what follows a sketch is provided by a translation of the main class, property and axiom forming constructors into FOL. The translation of classes and properties is parametrised by two variables, x and y, and uses counting quantifiers.

$$\begin{aligned}
\pi(\mathcal{O}) &:= \bigwedge_{\alpha \in \mathcal{O}} \pi(\alpha), \\
\pi(C \sqsubseteq D) &:= \forall x.\pi_x(C) \Rightarrow \pi_x(D), \quad \pi(R \sqsubseteq S) := \forall x, y.\pi_{x,y}(R) \Rightarrow \pi_{x,y}(S), \\
\pi(C(a)) &:= \pi_a(C), \quad \pi(R(a, b)) := \pi_{a,b}(R), \\
\pi_{x,y}(R) &:= R(x, y), \quad \pi_{x,y}(R^-) := R(y, x), \\
\pi_x(A) &:= A(x), \quad \pi_x(\top) := \mathsf{true}, \quad \pi_x(\bot) := \mathsf{false}, \quad \pi_x(\{a\}) := (x = a), \\
\pi_x(C \sqcap D) &:= \pi_x(C) \wedge \pi_x(D), \quad \pi_x(C \sqcup D) := \pi_x(C) \vee \pi_x(D), \\
\pi_x(\neg C) &:= \neg \pi_x(C), \\
\pi_x(\exists R.C) &:= \exists y.\pi_{x,y}(R) \wedge \pi_y(C), \quad \pi_x(\forall R.C) := \forall y.\pi_{x,y}(R) \Rightarrow \pi_y(C), \\
\pi_x(\leq nR.C) &:= \exists^{\leq n} y.\pi_{x,y}(R) \wedge \pi_y(C), \quad \pi_x(\geq nR.C) := \exists^{\geq n} y.\pi_{x,y}(R) \wedge \pi_y(C)
\end{aligned}$$

Interpretations, models, and entailments are defined as usual in FOL. In particular, an interpretation $\mathcal{I}$ is said to be a *model of* an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) iff $\mathcal{I} \models \pi(\alpha)$ for every axiom $\alpha \in \mathcal{O}$. A class expression C is *satisfiable* with respect to an ontology $\mathcal{O}$ if there exists a model $\mathcal{I}$ of $\mathcal{O}$ with $\mathcal{I} \models \exists x.\pi_x(C)$. An ontology is *inconsistent* if there does not exist a model of $\mathcal{O}$, i.e., if $\mathcal{O} \models \mathsf{false}(a)$. Given an ontology $\mathcal{O}$ and an axiom η, $\mathcal{O}$ entails η, written $\mathcal{O} \models \eta$ if, for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $\mathcal{I} \models \pi(\eta)$.

Definition 1 (Justification). *Let $\mathcal{O}$ be an ontology such that $\mathcal{O} \models \eta$. $\mathcal{J}$ is a justification for η in $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, and $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subsetneq \mathcal{J}$ $\mathcal{J}' \not\models \eta$.*

Intuitively, a justification for an entailment in an ontology is a minimal subset of the ontology that entails the entailment in question. A justification is a minimal subset in the sense that any proper subset of the justification does not entail the entailment in question. It is important to note that there may be multiple, potentially overlapping, justifications for any given entailment. Note that in the logics considered here, the number of justifications for an entailment in an ontology may be exponential in the size of the ontology [10].

Example: Let $\mathcal{O} = \{A \sqsubseteq B, B \sqsubseteq D, A \sqsubseteq \exists R.C, \exists R.\top \sqsubseteq D, A \sqsubseteq E\}$ so that $\mathcal{O} \models A \sqsubseteq D$. There are two justifications for $\mathcal{O} \models A \sqsubseteq D$, $\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$ and $\mathcal{J}_2 = \{A \sqsubseteq \exists R.C, \exists R.\top \sqsubseteq D\}$.

Definition 2 (Simple Repair). *$\mathcal{S}$ is a simple repair for $\mathcal{O} \models \eta$ if $\mathcal{S} \subseteq \mathcal{O}$ and, for every justification $\mathcal{J}$ for $\mathcal{O} \models \eta$, $\mathcal{S} \cap \mathcal{J} \neq \emptyset$, and for all $\mathcal{S}' \subsetneq \mathcal{S}$ $(\mathcal{O} \setminus \mathcal{S}') \models \eta$*

Hence, in order to construct a repair for an entailment η in an ontology $\mathcal{O}$, it is necessary to choose one axiom from each justification for η in $\mathcal{O}$ and remove this axiom from $\mathcal{O}$. This implies that, in the general sense, all justifications for η in $\mathcal{O}$ are needed in order to construct a simple repair for $\mathcal{O} \models \eta$. The process of finding the justifications for entailments and repairing ontologies is frequently referred to as ontology *debugging*.

3 Related Work and Background

In recent years there has been a significant amount of effort and research into the explanation of entailments that arise in *consistent* ontologies. In a seminal paper in the area of debugging ontologies, Schlobach and Cornet [7] presented minimal unsatisfiable preserving sub-tboxes (MUPS) as a specific kind of justification for unsatisfiable classes in ontologies. Kalyanpur et al. coined the term justification as a form of explanation for any arbitrary entailment that holds in an ontology. In [11] it was shown that it is practical to compute *all* justifications for any given entailment in real, published (consistent) ontologies. The Propositional Logic reasoning community have long used the notion of *minimal conflict sets* for determining the minimal unsatisfiable subsets of a set of clauses for example in [12]. These conflict sets are used to gain insight into why a set of clauses is unsatisfiable. In essence minimal conflict sets are akin to justifications. On a practical level, justifications are used by many ontology development and debugging environments such as Protégé-4 [13], Swoop [14], and RaDON [15], as a popular form of explanation for entailments in ontologies.

3.1 Model Based Diagnosis and Techniques for Computing All Justifications

Techniques for computing all justifications for an entailment, and for computing all minimal conflict sets for a set of unsatisfiable clauses, borrow from techniques used in the field of *Model Based Diagnosis*. Model Based Diagnosis is the overarching name for the process of computing *diagnoses* for a *faulty system*. Briefly, a *diagnosis* is a subset minimal set of *components* from a faulty system, that if replaced would *repair* the system so that it functions as intended. The term *model* refers to the fact that a model is used to obtain the *predicted* behaviour of of a system, which is then compared the *observed* behaviour of the system. Any discrepancy between the predicted and observed behaviours indicates a fault in the system. Model based diagnosis techniques have a wide variety of applications, and for example, are typically used for diagnosing sets of faulty gates and connections in circuits. There is a direct correspondence between the ideas behind model based diagnosis, and the ideas behind justification based ontology debugging. Essentially, a diagnosis is the same as a repair (Definition 2), and *Minimal Conflict Sets*, which are subsets of the faulty system, are equivalent to justifications which are subsets of an ontology. A particularly pertinent

algorithm that is borrowed from the area of Model Based Diagnosis is "Reiter's Hitting Set Tree (HST) Algorithm" [16]. This algorithm essentially constructs a finite tree whose nodes are labelled with minimal conflict sets (justifications), and whose edges are labelled with components (axioms) from the system. In doing this the algorithm finds all minimal hitting sets for the conflict sets in a system. Since a minimal hitting set corresponds to a repair, and a conflict set corresponds to a justification, it is possible to use Reiter's algorithm to find all repairs, and in doing this find all justifications for an entailment in an ontology (proof in [11]).

Continuing with the previous example from Section 2, the hitting set tree algorithm is used to compute all justifications for $\mathcal{O} \models A \sqsubseteq D$ as follows. A subprocedure, computeJustification(η, $\mathcal{O}$), is used, which computes a single justification for the entailment η in an ontology $\mathcal{O}$. Suppose that $\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$ is computed by computeJustification($A \sqsubseteq D$, $\mathcal{O}$). This justification is set as the label of the root of the tree. Next, one of the axioms from $\mathcal{J}_1$, say $A \sqsubseteq B$ is selected and removed from $\mathcal{O}$ to give $\mathcal{O}'$. This essentially repairs $\mathcal{J}_1$ and allows another justification (if one exists) to be computed for the entailment with respect to the modified ontology $\mathcal{O}'$. In this case, computeJustification($A \sqsubseteq D$, $\mathcal{O}'$) computes $\mathcal{J}_2 = \{A \sqsubseteq \exists R.C, \exists R.\top \sqsubseteq D\}$ as a justification. The hitting set tree is extended with an edge from the root, to a node labelled with $\mathcal{J}_2$ (the upper left most edge in Figure 3.1), and the edge is labelled with the axiom that was removed. Next, an axiom from $\mathcal{J}_2$, say $A \sqsubseteq \exists R.C$ is selected and removed from $\mathcal{O}'$ to give $\mathcal{O}''$, and computeJustification($A \sqsubseteq D$, $\mathcal{O}''$) is called to compute further justifications with respect to $\mathcal{O}''$. This time, no further justifications are found since the entailment does not hold in $\mathcal{O}''$, and the hitting set tree is extended with an edge from the node labelled with $\mathcal{J}_2$ to a node with an empty label which marks this branch as complete. The algorithm then backtracks up a level to remove a different axiom from $\mathcal{J}_2$ and the process continues. The algorithm terminates when all branches are marked as complete. It is noticeable, that the axioms labelling the edges in a complete branch form a repair, or a superset of a repair, since removing them from $\mathcal{O}$ results in $\mathcal{O}''$ such that $\mathcal{O}'' \not\models \eta$. A more formal description of this algorithm (with optimisations) is given later in Function-1R in Algorithm 1 (where the entailment that $\mathcal{O}$ is inconsistent is implicit) and is used in the computation of all justifications for inconsistent ontologies.

One of the issues with computing all justifications for an entailment is that the size of the hitting set tree can, in the worst case, be exponential in the size of the set of justifications. However, a suite of optimisations, detailed in [11] and presented in Algorithm 1, can be applied to prune the hitting set tree and minimise its size, so that for practical purposes this is rarely a problem. For example, in [11] it was shown that for real, published ontologies, the number of justifications per entailment is usually small—for the ontologies sampled in [11], the average number of justifications per entailment was two, with one ontology containing a maximum of twenty six justifications for an entailment.

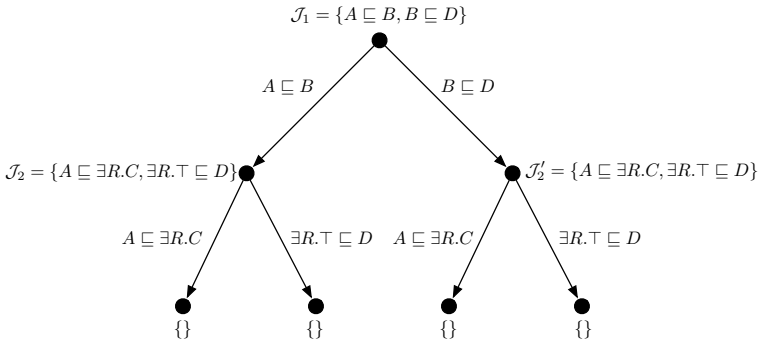

Fig. 1. An example of a hitting set tree (HST) that is constructed by Reiter's HST algorithm

3.2 Computing Single Justifications

The technique described in the previous section for computing *all* justifications for an entailment, uses a sub-procedure $\mathsf{computeJustification}(\eta, \mathcal{O})$ that computes a justification for an entailment η in an ontology $\mathcal{O}$. In this work, a black-box technique, which is not specific to a particular reasoner or proof strategy is used. Black-box techniques typically use an "expand-contract" strategy in combination with calls to an external reasoner. In order to compute a justification for an entailment η in an ontology $\mathcal{O}$, an initial, and small, subset $\mathcal{S}$ of axioms from $\mathcal{O}$ is selected based on the signature of η. The axioms in $\mathcal{S}$ are typically the axioms whose signature has a non-empty intersection with the signature of η, or axioms that "define"[2] terms in the signature of η. A reasoner is then used to check if $\mathcal{S} \models \eta$, if not, $\mathcal{S}$ is expanded by adding a few more axioms from $\mathcal{O}$. This process continues until $\mathcal{S}$ is large enough so that it entails η (this is the expansion phase). When a subset $\mathcal{S}$ of $\mathcal{O}$ that entails η is found, this subset is pruned until it is a minimal set of axioms that entails η i.e. until it is a justification (this is the contraction phase). Through careful selection of axioms in the expansion phase, which minimises the maximum size of $\mathcal{S}$, and various pruning techniques in the contraction phase, it has been found that this algorithm works well in practice for entailments in consistent ontologies [11].

4 A Black-Box Algorithm for Computing Justifications for Inconsistent Ontologies

As described in Section 3.2, the general approach taken by black-box justification finding algorithms is to use a two phase expand-contract procedure. The "seed" for selecting the initial small set axioms for the expansion phase is usually the signature of the entailment in question (the class, property, individual names

[2] For example, the axiom $A \sqsubseteq B$ defines the class name A.

appearing in the entailment). However, an issue when generating justifications for inconsistent ontologies using black-box techniques is "where to start". How should the initial set of axioms be selected? With inconsistent ontologies there is no seed signature that can be used as an input to start the expansion process. This is due to the fact that current OWL reasoners typically do not provide any information or clues as the possible causes of an inconsistent ontology—they just output that the ontology is inconsistent.[3] Therefore, the black-box algorithm that is presented here, consists of a very simple expansion phase, which immediately causes the expansion to contain *all* of the axioms in an ontology, followed by a "divide and conquer" contraction phase, which contracts the set of axioms in the ontology down to a justification. The algorithm for computing all justifications is a hybrid of the algorithm for computing single justifications taken from [17] and combined with the algorithm for computing all justifications (the HST algorithm) taken from [11]. Proof that the Hitting Set Tree algorithm finds all justifications is given in [11]. Algorithm 1 is the algorithm based on Reiter's Hitting Set Tree Algorithm and is used to compute all justifications for an inconsistent ontology. Algorithm 2 is used as a sub-procedure for Algorithm 1 for computing a justification for the inconsistent ontology.

Algorithm 1. ComputeAllJustifications

Function-1: ComputeAllJustifications($\mathcal{O}$)

1: $S, curpath, allpaths \leftarrow \emptyset$
2: ComputeAllJustificationsHST($\mathcal{O}$, S, $curpath$, $allpaths$)
3: **return** S

Function-1R: ComputeAllJustificationsHST($\mathcal{O}$, S, $curpath$, $allpaths$)

1: **for** $path \in allpaths$ **do**
2: **if** $curpath \supseteq path$ **then**
3: **return** //Path termination without consistency check
4: **if** IsConsistent($\mathcal{O}$) **then**
5: $allpaths \leftarrow allpath \cup \{curpath\}$
6: **return**
7: $J \leftarrow \emptyset$
8: **for** $s \in S$ **do**
9: **if** $s \cap path = \emptyset$ **then**
10: $J \leftarrow s$ //Justification reuse (saves recomputing a justification)
11: **if** $J = \emptyset$ **then**
12: $J \leftarrow$ ComputeSingleJustification($\mathcal{O}$)
13: $S \leftarrow S \cup \{J\}$
14: **for** $ax \in J$ **do**
15: $curpath \leftarrow curpaths \cup \{ax\}$
16: ComputeAllJustificationsHST($\mathcal{O} \setminus \{ax\}$, S, $curpath$, $allpaths$)

[3] The exception to this is the Pellet reasoner, which is able to output clash information—while this is a potential optimisation, it is not supported by all reasoners and is therefore not used here.

Algorithm 2. ComputeSingleJustification

Function-2: ComputeSingleJustification($\mathcal{O}$)

1: **return** ComputeSingleJustification($\emptyset$, $\mathcal{O}$)

Function-2R: ComputeSingleJustification(S, F)

1: **if** $|F| = 1$ **then**
2: **return**
3: $S_L, S_R \leftarrow \text{split}(F)$
4: **if** IsInconsistent($S \cup S_L$) **then**
5: **return** ComputeSingleJustification(S, S_L)
6: **if** IsInconsistent($S \cup S_R$) **then**
7: **return** ComputeSingleJustification(S, S_R)
8: $S'_L \leftarrow$ ComputeSingleJustification($S \cup S_R, S_L$)
9: $S'_R \leftarrow$ ComputeSingleJustification($S \cup S'_L, S_R$)
10: **return** $S'_L \cup S'_R$

5 Implementation and Evaluation

Algorithm 1, and its sub-routine Algorithm 2, were implemented in Java using the latest version of the OWL API [18]. The OWL API has excellent support for manipulating ontologies as sets of axioms and provides a common interface to various OWL reasoners. The algorithm was run on the set of ontologies shown in Table 1 using the FaCT++ [19], Pellet [20] and HermiT [21] reasoners. For each ontology, the time to perform a consistency check of the complete ontology was recorded. Then the time to compute a single justification for the inconsistency, and the time to compute as many justifications as possible within a time of 30 minutes was recorded.

5.1 Results Analysis

The ontologies shown in Table 1 are real ontologies in the sense that they were developed by end users using editing tools such as Protégé-4. Since it is rarely

Table 1. Inconsistent Ontologies Used In Experiments

ID	Ontology	Expressivity	Axioms	Cons. HermiT (ms)	Cons. Pellet (ms)	Cons. FaCT++ (ms)
1	Assignment4	$\mathcal{ALCIQ}$	88	-	1301	40
2	Boat	$\mathcal{ALCIF}(\mathcal{D})$	22	34	11	2
3	ComparaGrid	$\mathcal{ALCHIQ}(\mathcal{D})$	409	192	175	-
4	Country	$\mathcal{SROIQ}(\mathcal{D})$	5921	539	1454	333
5	Fish	$\mathcal{ALCH}$	49	36	78	13
6	IedbExport	$\mathcal{SHOIN}(\mathcal{D})$	2417	141	1117	67
7	Micro	$\mathcal{SROIQ}(\mathcal{D})$	1458	-	556	48
8	Spectro	$\mathcal{ALCON}(\mathcal{D})$	26612	1841	1763	378
9	Pizza	$\mathcal{SHOIQ}(\mathcal{D})$	179	25	45	5
10	ClassesAsValues	$\mathcal{ALCO}$	13	3	2	1
11	Travel	$\mathcal{SROIQ}(\mathcal{D})$	6437	443	701	135
12	Units	$\mathcal{ALCOIF}(\mathcal{D})$	2254	217	435	72

the case that inconsistent ontologies are published, all of the ontologies in the sample were obtained either from public mailing lists, where the ontologies had been posted to the mailing list with a request for help in debugging the ontology, or from ontologies that were directly received via email from colleagues and users of Protégé-4 and Swoop.

The rightmost three columns of Table 1 shows the time to load, and check the consistency of the ontologies using HermiT, Pellet and FaCT++. Even for the largest ontology, 8, that contains over twenty six thousand axioms, the consistency check is performed within two seconds using HermiT or Pellet and less that half a second using FaCT++. Note that FaCT++ failed to terminate on the ComparaGrid ontology, where it entered an infinite loop during preprocessing. HermiT incorrectly found Assignment4 consistent, and crashed on the Micro ontology. On the surface, it would appear that even for very large ontologies, the efficiency and speed at which ontologies can be loaded and checked for consistency seems to indicate the feasibility of using a reasoner as an oracle in a black-box based debugging algorithm.

Table 2 shows the performance results for computing justifications for inconsistent ontologies. Showing by ontology and reasoner, the time (ms) to compute one justification, the number of justifications found within 30 minutes, the time (ms) to compute all justifications if less than 30 minutes. A dash indicates that no justifications could be computed. A $\star$ indicates that the algorithm did not terminate within 30 minutes.

It should be noted that for all sampled ontologies, it was possible to compute *at least one* justification for the ontology being inconsistent using *at least one* of the three reasoners. With the exception of the Spectro ontology, it was possible to compute at least one justification *within ten seconds*. This is clearly acceptable performance in the context of generating on demand explanations in an ontology development environment such as Protégé-4. In the case of the Spectro ontology, Pellet was able to compute at least one justification within 11 seconds and HermiT within 31 seconds. Considering that this ontology contains over 26500 axioms, it is arguable that this is acceptable, and even impressive, performance for the purposes of interactive debugging and explanation in an ontology development environment.

It can be seen that for the Assignment4, Fish, Spectro, Travel, and Units ontologies, it was not possible to compute *all* justifications with any of the tested reasoners. However, for these ontologies, with the exception of Assignment4, a sizeable number of justifications could be computed. In all cases, the availability of one or more justifications can provide a vital insight into why an ontology is inconsistent, and allow a person to begin to repair the ontology. Not all justifications could be computed for all ontologies within 30 minutes. Broadly speaking, there are two reasons for this: (1) The reasoner being used found it *much* harder to check if *some subset of an ontology* was consistent, compared to checking if the *whole* ontology was consistent (in all cases, consistency checking performance for whole ontologies was good e.g. less than 2 seconds), and the running algorithm therefore "ground to a halt" during entailment checking. An exam-

Table 2. Times for computing single and all justifications

Ontology	Reasoner	Time for One (ms)	Number found in 30 mins.	Time for all (ms)
Assignment4	HermiT	-	-	⋆
	Pellet	-	-	⋆
	FaCT++	5711	1	⋆
Boat	HermiT	129	1	193
	Pellet	73	1	93
	FaCT++	11	1	41
ComparaGrid	HermiT	2252	9	15901
	Pellet	1515	9	13164
	FaCT++	-	-	⋆
Country	HermiT	4177	4	383963
	Pellet	4564	4	137348
	FaCT++	1726	4	78547
Fish	HermiT	134	162	⋆
	Pellet	-	-	⋆
	FaCT++	115	162	⋆
IedbExport	HermiT	855	2	2415
	Pellet	1257	2	3860
	FaCT++	765	2	2255
Micro	HermiT	4538	1	⋆
	Pellet	2326	1	59090
	FaCT++	809	1	1574
Spectro	HermiT	30930	76	⋆
	Pellet	10768	76	⋆
	FaCT++	-	-	⋆
Pizza	HermiT	114	3	592
	Pellet	7491	3	⋆
	FaCT++	37	3	329
ClassesAsValues	HermiT	21	1	23
	Pellet	4	2	15
	FaCT++	5	2	13
Travel	HermiT	7873	7	⋆
	Pellet	884	492	⋆
	FaCT++	521	163	⋆
Units	HermiT	3023	54	⋆
	Pellet	473	287	⋆
	FaCT++	285	287	⋆
Travel-semi-rep (semi repaired Travel ont.)	HermiT	8309	6	⋆
	Pellet	3975	6	25722
	FaCT++	1331	6	19280

ple of this is the Travel ontology, where using HermiT the procedure was only able to compute 7 justifications, but using Pellet, the procedure was able to compute nearly 500 justifications. (2) The number of justifications for the inconsistent ontology was very large. The runtime performance of the algorithm for computing justifications is in the worst case *exponential* with the number of justifications—computing all justifications is an inherently difficult problem.

6 Discussion

The usefulness of justifications as explanations for inconsistent ontologies. Despite the fact that, in *some cases*, it is not feasible to compute *all* justifications for real inconsistent ontologies, the ability to obtain just one or two justifications can lend a vital insight into why the ontology is inconsistent.

Some ontologies are inconsistent due to highly non-obvious reasons, and it is arguable that without automated explanation support, ontology authors would face a hopeless task of trying to figure out the reasons for an ontology being inconsistent in order to arrive at a manual repair. An example of a justification from the Country ontology is shown in Figure 2. The ontology authors, who made a request for help in trying to track down the reasons for the inconsistency, indicated that it was highly unlikely that they would have discovered the reason, without considerable difficulty, using manual debugging techniques.

Using justifications for inconsistent ontologies for repair. Even though it may not be possible in practice to compute all justifications for a given ontology, with the availability of at least one justification, it is still possible to carry out a manual interactive repair. For example, on closer inspection of the justifications for the travel ontology it was observed that many of them had the form $\{R(a, l), \top \sqsubseteq \forall R.C\}$, where l is a data value not in the value space of C[4]. In fact, structural inspection revealed that there were over 550 of these kinds of justifications for the travel ontology and over 12000 for the Spectro ontology. In order to test the hypothesis that even seeing a handful of justifications is helpful for interactive debugging and can support repair, the travel ontology was modified to produce a new version where all literals in property assertions were typed with the appropriate data type[5]. This new version of the ontology (Travel-semi-rep in Table 2) was still inconsistent, but had just six justifications for it being inconsistent. Thus, even though it may not be possible to compute all justifications is one go, being able to compute some justifications can provide enough insight in order to understand the reasons for an ontology being inconsistent so that is is possible to perform a manual repair.

The difficulty of finding *all* justifications for an inconsistent ontology compared to finding justifications for entailments in consistent ontologies. In other work [6,11] it has been shown that it is practical to compute *all* justifications for entailments in consistent ontologies using similar blackbox algorithms as the one presented here. However, the experiments carried out as part of this research clearly indicate that it is not possible to compute all justifications for some of the sampled inconsistent ontologies. As a means to understanding why this could be the case, consider the following *consistent* ontology $\mathcal{O} = \{B \sqsubseteq A, C \sqsubseteq A, D \sqsubseteq A, \dots\}$ such that B, C and D are entailed to be unsatisfiable. Suppose that there are ten justifications for each unsatisfiable class, meaning that there is a total of *thirty* justifications for the three unsatisfiable classes. In this case, *each entailment can be examined separately*, one after the other, so that the justifications for each entailment are computed separately. In other words, although the total number of justifications for all unsatisfiable classes is thirty, they are *computed in batches of ten*. Now consider $\mathcal{O}' = \mathcal{O} \cup \{B(a), C(b), D(c)\}$. Since B, C and D are unsatisfiable, $\mathcal{O}'$ is entailed to

[4] OWL 2 supports datatypes.

[5] Upon seeing the justifications for this ontology it was deemed that this was a natural repair.

$$\begin{aligned}
&\mathsf{Island}(\mathsf{ireland}) && (1)\\
&\quad \mathsf{Island} \sqsubseteq \mathsf{LandMass} && (2)\\
&\qquad \mathsf{LandMass} \sqsubseteq \mathsf{GeographicalFeature} \sqcap \exists \mathsf{hasCoastline}.\mathsf{Coastline} && (3)\\
&\qquad\quad \mathsf{GeographicalFeature} \sqsubseteq \mathsf{NaturalPhysicalThing} && (4)\\
&\qquad\qquad \mathsf{NaturalPhysicalThing} \sqsubseteq \mathsf{NaturalEntity} && (5)\\
&\qquad\qquad\quad \mathsf{NaturalEntity} \sqsubseteq \mathsf{PhysicalEntity} && (6)\\
&\mathsf{landBoundaryOf}(\mathsf{unitedKingdomIrelandBorder}, \mathsf{ireland}) && (7)\\
&\mathsf{landBoundaryOf} \sqsubseteq \mathsf{hasLandBoundary}^{-} && (8)\\
&\exists \mathsf{hasLandBoundary}.\top \sqsubseteq \mathsf{Country} && (9)\\
&\quad \mathsf{Country} \sqsubseteq \mathsf{AdministrativeDivision} && (10)\\
&\qquad \mathsf{AdministrativeDivision} \sqsubseteq \mathsf{PoliticalEntity} \sqcap \exists \mathsf{directPartOf}.\mathsf{PoliticalDiv} && (11)\\
&\mathsf{PoliticalEntity} \sqsubseteq \neg \mathsf{PhysicalEntity} && (12)
\end{aligned}$$

Fig. 2. An (small) example justification from the (large) country ontology. The individual ireland is entailed to be a PhysicalEntity (by axioms (1)(6)) and is also entailed to be a PoliticalEntity (by axioms (7)-(11)), PhysicalEntity and PoliticalEntity are disjoint with each other (axiom (12)).

be inconsistent, and there are thirty justifications for this entailment. Although the total number of justifications for the entailment of interest in $\mathcal{O}'$ is the same as the total number of justifications for the entailments of interest in $\mathcal{O}$, and the justifications are almost structurally the same, the key difference is that there are now *thirty* justifications for *one* entailment in $\mathcal{O}'$, compared to thirty justifications spread over three entailments in $\mathcal{O}$. Since the algorithm for computing all justifications for an entailment in an ontology is, in the *worst case*, exponential in the number of justifications for the entailment, it is clear to see that it can be *much* more time consuming to compute all justifications for an inconsistent ontology compared to computing all justifications for all entailments of interest in a consistent ontology. In essence, one of the factors that makes it practical to compute all justifications for entailments in consistent ontologies, is that for real ontologies, the average number of justifications per entailment tends to be fairly low. In the case of inconsistent ontologies, it is not possible, using classical reasoning, to take a more fine-grained approach, because it is not possible to recognise entailments other than the single entailment that the ontology is inconsistent.

7 Summary and Future Work

Computing a single justification, as a form of explanation, for real world inconsistent ontologies is possible in practice. However, computing *all* justifications is not possible in practice for some real world ontologies. This is primarily due to the fact that there can be a very large number of justifications for an ontology being inconsistent. Despite this, it is expected that having the ability to compute

just one or two justifications for an inconsistent ontology will be of enormous benefit to people trying to understand why the ontology is inconsistent. As future work, we plan to investigate the possibility of using clash information for further optimisation the current black-box algorithm. We also plan to investigate the use of paraconsistent reasoning as means for exploring justifications for meaningful entailments in inconsistent ontologies with a view to repairing these ontologies.

References

1. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics (2003)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation and Applications (2003)
3. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: KR 2006 (2006)
4. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: AAAI, pp. 299–304 (2007)
5. Ma, Y., Hitzler, P., Lin, Z.: Algorithms for paraconsistent reasoning with OWL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 399–413. Springer, Heidelberg (2007)
6. Kalyanpur, A.: Debugging and Repair of OWL Ontologies. PhD thesis, The Graduate School of the University of Maryland (2006)
7. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: IJCAI (2003)
8. Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. Journal of Automated Reasoning (1995)
9. Kazakov, Y.: $\mathcal{SRIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. Description Logics (2008)
10. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}$. In: KI (2007)
11. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: ISWC (2007)
12. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2004. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)
13. Horridge, M., Tsarkov, D., Redmond, T.: Supporting early adoption of owl 1.1 with protégé-owl and fact++. In: OWL: Experiences and Directions, OWLED (2006)
14. Kalyanpur, A., Parsia, B., Hendler, J.: A tool for working with web ontologies. International Journal on Semantic Web and Information Systems 1 (January - March 2005)
15. Ji, Q., Haase, P., Qu, G., Hitzler, P., Stadtmoeller, S.: RaDON – repair and diagnosis in ontology networks. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simper, L.E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 863–867. Springer, Heidelberg (2009)
16. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32, 57–95 (1987)
17. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: KR-MED (2008)

18. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 touch paper: The OWL API. In: OWLED (2007)
19. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System description. In: IJCAR (2006)
20. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Journal of Web Semantics 5(2) (2007)
21. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hypertableaux. In: CADE 21 (2007)