Root Justifications for Ontology Repair

Kodylan Moodley[1,2], Thomas Meyer[1,2], and Ivan José Varzinczak[1,2]

[1] CSIR Meraka Institute, Pretoria, South Africa
{kmoodley, tmeyer, ivarzinczak}@csir.co.za
[2] School of Computer Science, University of KwaZulu-Natal, South Africa

Abstract. In recent years, there has been significant progress in developing tools for debugging and repairing Description Logic (DL)-based ontologies with erroneous consequences. However, these tools place more emphasis on explaining *why* the consequences follow from the ontology rather than on eliminating them. Another shortcoming in existing tools is that there is no common approach for eliminating or handling different types of erroneous consequences in the ontology. By extending existing principles in ontology repair, our goal is to define one such a common approach. We have implemented a Protégé plugin to demonstrate our approach and we evaluate it against the traditional ones.

1 Introduction

An *ontology* (also referred to as a *terminology*, *knowledge base*) is an entity used to represent some domain (field of knowledge). To be more specific, an ontology precisely depicts some representation of the domain. Usually the building blocks of an ontology include categories (concepts), relations (roles) and objects (individuals).

If we were to consider the domain of a university, we could use concept names such as Lecturer, Student and Module, and role names such as teaches and enrolledFor, and combine these constructs to form sentences which describe the domain, like "A Lecturer is someone who teaches a Module" or "A Student is someone who enrolledFor a Module". An ontology can be viewed as a set of such sentences or as a taxonomy in which the concepts are classified as more general (*super-concepts*) or more specific (*sub-concepts*) in relation to one another. For example the sentence "Every Student is a Person" means that the Person concept is more general than the Student concept.

There are many different formalisms for representing ontologies. Description Logics (DLs) are a family of such formalisms. They are widely accepted as an appropriate class of knowledge representation languages to formalize and reason about ontologies [1].

DL reasoners, which are tools for performing standard reasoning tasks with ontologies such as satisfiability and consequence checking, have grown increasingly powerful and sophisticated in the last decade [15, 12]. Quite often during the development of ontologies, ontology developers make modelling errors. These errors can introduce *unwanted consequences* in the ontology. The process

of identifying, explaining and eliminating these unwanted consequences is known as *ontology debugging and repair* or simply *ontology repair*.

The *identification* of unwanted consequences in DL ontologies is taken care of by the algorithms implemented in DL reasoners. The remaining steps, of explaining *why* the unwanted consequences arise and devising ways to *eliminate* them, can be divided into two main approaches namely *Glass-box* [11, 6, 7] and *Black-box* [5, 3, 16]. Glass-box approaches usually require non-trivial modification of the reasoner while Black-box approaches, as the name suggests, treat the reasoner as a "black-box" which basically answers 'yes' or 'no' to the question: does consequence α follow from the ontology?

In this paper, we focus on the Black-box approach and extend the principles introduced in a previous work [8]. We present an improved implementation of the *Black-box* ontology repair method discussed there and we also provide some experimental results comparing the performance of our approach to that of the standard *Black-box* approach to ontology repair. The standard approach has been applied mostly to the specific ontology errors called *unsatisfiable concepts* but have not been extended for other errors. Furthermore, this approach deals with the unsatisfiable concepts by finding an advantageous sequence in which to eliminate them. Therefore, it does not eliminate all the unsatisfiabilities simultaneously.

The outline for the rest of this paper is as follows. After some logical preliminaries (Section 2), we briefly summarize existing work. In Section 4, we present our method for eliminating a set of errors from the ontology, based on the notion of *root justifications* [8, 9]. In Section 5, we discuss the implementation of a Protégé[3] plugin which demonstrates our approach to ontology repair. In this section we also discuss some experimental results comparing the performance of the plugin to the standard approach for debugging ontology errors. Finally, we conclude with a summary of our contributions and open questions for future investigation.

2 Description Logics

In this paper we work in the Description Logic $\mathcal{ALC}$ [1], however, all we shall say in the sequel can in principle be stated for any description logic.

The language of $\mathcal{ALC}$ is built upon a (finite) set of atomic *concept names* $\mathsf{N}_{\mathscr{C}}$ and a (finite) set of *role names* $\mathsf{N}_{\mathscr{R}}$, where $\mathsf{N}_{\mathscr{C}} \cap \mathsf{N}_{\mathscr{R}} = \emptyset$, together with the constructors $\sqcap$, $\neg$, and $\exists$ and the distinguished concept $\top$. An atomic concept is denoted by A, and a role name by r. Complex concepts are denoted by $C, D, \ldots$ and constructed as follows:

$$C ::= A \mid \top \mid C \sqcap C \mid \neg C \mid \exists r.C$$

Concepts built with the constructors $\sqcup$, and $\forall$ as well as the concept $\bot$ are defined in terms of the others in the usual way. We let $\mathcal{L}$ denote the set of all $\mathcal{ALC}$ concepts.

[3] `http://protege.stanford.edu`

The semantics is the standard set theoretic Tarskian one. An *interpretation* is a structure $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain*, and $\cdot^{\mathcal{I}}$ is an *interpretation function* mapping concept names A to subsets of $\Delta^{\mathcal{I}}$, and mapping role names r to binary relations on $\Delta^{\mathcal{I}}$:

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}},\ \top^{\mathcal{I}} = \Delta^{\mathcal{I}},\ r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

We extend $\cdot^{\mathcal{I}}$ to interpret complex concepts as follows:

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\ \ (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}},$$
$$(\exists r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$$

Given $C, D \in \mathcal{L}$, $C \sqsubseteq D$ is a *subsumption statement* which can be read "C is subsumed by D". $C \equiv D$ is an abbreviation for both $C \sqsubseteq D$ and $D \sqsubseteq C$. A *TBox* $\mathcal{T}$ is a finite set of subsumption statements. We use the Greek symbols α, β etc. to refer to subsumption statements and, following in the tradition of the DL community, we call them *axioms*.

We say $\mathcal{I}$ *satisfies* $C \sqsubseteq D$ (denoted $\mathcal{I} \Vdash C \sqsubseteq D$) if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I} \Vdash C \equiv D$ if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$. $C \sqsubseteq D$ is *entailed* by a TBox $\mathcal{T}$, denoted $\mathcal{T} \models C \sqsubseteq D$, if and only if every $\mathcal{I}$ which satisfies all elements of $\mathcal{T}$, also satisfies $C \sqsubseteq D$. Given a Tbox $\mathcal{T}$, and a basic concept A, $\mathcal{T}$is *A-unsatisfiable* if and only if for all models $\mathcal{I}$ of $\mathcal{T}$, $A^{\mathcal{I}} = \emptyset$.

A DL ontology consists of both a TBox and ABox [1], only a TBox or only an ABox. This decision depends on the specific application of the ontology.

3 Ontology Debugging and Repair

In this section we discuss the standard Black-box technique for eliminating errors in the ontology based on existing work.

3.1 Debugging

The process of creating and maintaining ontologies is a dynamic one. At any given stage during development, the ontology has a set of consequences which follow from it. It is up to the *ontology engineer* (ontology developer) and *domain expert* (expert on the topic which the ontology is representing) to decide which of these consequences are desired and which are not. The undesired consequences are indications of modelling errors which were introduced during ontology development and these consequences are thus referred to as errors in the ontology.

Explanation is a service which focuses on explaining *why* selected consequences follow from an ontology. Given an ontology $\mathcal{O}$ with some axiom α such that $\mathcal{O} \models \alpha$, there exists an explanation which indicates why α follows from $\mathcal{O}$. The most widely used explanation is a set of *justifications* (also known as *minAs* [2] and *MUPSes* [11]) for the entailment. A justification for $\mathcal{O} \models \alpha$ is a minimal subset of $\mathcal{O}$ from which α logically follows.

Example 1. Consider the following ontology:

$$\mathcal{O} = \left\{ \begin{array}{ll} 1.\ \mathsf{C} \sqsubseteq \mathsf{A} & 2.\ \mathsf{C} \sqsubseteq \neg \mathsf{A} \\ 3.\ \mathsf{F} \sqsubseteq \mathsf{C} \sqcap \neg \mathsf{A} & 4.\ \mathsf{F} \sqsubseteq \mathsf{C} \end{array} \right\}$$

We represent $\mathcal{O}$ as the set $\{1, 2, 3, 4\}$ with the understanding that each number represents an axiom in $\mathcal{O}$. One can see that $\mathcal{O}$ is F-unsatisfiable: $\mathcal{O} \models F \sqsubseteq \bot$. Two justifications for this entailment are $\{1, 3\}$ and $\{1, 2, 4\}$ □

Justifications are useful for many reasons. They allow for the *pinpointing* of the causes of modelling errors. In Example 1 they show that Axioms 1 and 3 may not both occur in $\mathcal{O}$ without $\mathcal{O}$ being F-unsatisfiable. Similarly, for Axioms 1, 2, and 4. In practice it is frequently the case that justifications are significantly smaller than the Tbox as a whole.

We now describe a basic Black-box algorithm (*naïve pruning algorithm* [13]) for computing a single justification for an ontology error. It works by removing axioms from the ontology, one at a time, while monitoring how this affects the entailment under consideration.

For example, if we are computing a justification for the C-unsatisfiability of $\mathcal{O}$, we want to find the minimal subset of $\mathcal{O}$ such that this subset is C-unsatisfiable. Therefore, we remove one axiom from $\mathcal{O}$, to get $\mathcal{O}' \subset \mathcal{O}$, then we check if $\mathcal{O}'$ is C-unsatisfiable. If this is the case, we can proceed to remove another axiom from $\mathcal{O}$. If it is *not*, then we have to add the axiom back to the ontology and continue to remove a *different* axiom. This is repeated for all the axioms in $\mathcal{O}$.

The set of axioms remaining in the ontology, after this process, constitutes a justification for the C-unsatisfiability of $\mathcal{O}$. Algorithm 1 below gives the pseudo-code for such a method:

Algorithm 1: *naïve pruning algorithm* (Single justification)

Input: Ontology $\mathcal{O}$ and concept C such that $\mathcal{O}$ is C-unsatisfiable
Output: Justification J for $\mathcal{O}$ being C-unsatisfiable

```
1 J := O;
2 foreach α ∈ J do
3 |   if J\{α} is C-unsatisfiable then
4 |   |   J := J\{α};
5 |   end
6 end
7 return J;
```

Although the algorithm is correct, one can see that it is also computationally intensive. This is because it uses the same number of entailment/satisfiability checks as there are axioms in the ontology. Limiting the number of checks is crucial in Black-box methods. This is especially true for ontologies with a large

number of axioms. The computational complexity for concept satisfiability in $\mathcal{ALC}$ is PSPACE [14].

The naïve pruning algorithm provides a method for computing a *single* justification but in order to obtain a *complete* explanation for the unsatisfiability of a concept one has to determine *all* its justifications. The most well known method to do this is a variant of *Reiter's [10] hitting set algorithm*. It assumes that we have a method for computing a single justification (for example the naïve procedure presented above). This method is used to generate a *justification tree* for the unsatisfiability of the concept w.r.t. $\mathcal{O}$.

A justification tree, $T_{\mathcal{J}}$, is a set of nodes $\mathcal{V}_{\mathcal{J}}$ and edges $\mathcal{E}_{\mathcal{J}}$. Each node $j \in \mathcal{V}_{\mathcal{J}}$ has a label $j.label$ which is a justification for the unsatisfiability of a concept, say C, with respect to the ontology $\mathcal{O}$. Each edge $e \in \mathcal{E}_{\mathcal{J}}$ has label $e.label \in \bigcup_{j.label \in T_{\mathcal{J}}} j.label$, i.e., $e.label$ is an axiom of some justification. The function $P(j)$, the path function, returns the set of edge labels (representing axioms) on the path from the root node to node j.

To construct a justification tree $T_{\mathcal{J}}$ in a breadth-first fashion, the following rules are applied.

(i) The first step is to generate a root node j_{root} for $T_{\mathcal{J}}$ which is labelled with a *justification* for the unsatisfiability of C. This justification can be generated with respect to the ontology $\mathcal{O}$ using any method for computing a single justification (for example, Algorithm 1 presented above).

(ii) If a node j in the justification tree is labelled with a justification J, then for each axiom $\alpha \in J$, a successor node j_α is attached to j via an edge e_α which is labelled with α.

(iii) Each successor node j_α in the justification tree is labelled with a justification J' for C. J' is generated using the same method as in the first rule. However the difference now is that J' is computed with respect to the ontology $\mathcal{O} \backslash P(j_\alpha)$ and *not* $\mathcal{O}$. If however, it turns out that $\mathcal{O} \backslash P(j_\alpha)$ does not contain a justification for C it means that $\mathcal{O} \backslash P(j_\alpha)$ is not C-unsatisfiable and therefore we label j_α with '$\checkmark$' indicating a *terminating node* with no successors.

Construction of the justification tree continues until all leaves of the tree are terminating nodes. The distinct nodes in the final tree represent the set of all justifications for the unsatisfiability of the concept w.r.t. $\mathcal{O}$ [4, Theorem 4].

3.2 Repair

Recall Example 1. One can consider justifications $\{1, 3\}$ and $\{1, 2, 4\}$ as reasons for the F-unsatisfiability of $\mathcal{O}$. To eliminate the unsatisfiability, one has to nullify *all* its reasons. For example, suppose we remove Axiom 3 from $\mathcal{O}$. This would nullify the justification $\{1, 3\}$ for the unsatisfiability because both Axioms 1 and 3 are to be present for the unsatisfiability to hold. However, even if this justification is nullified, there is still another "reason" for the unsatisfiability to hold, i.e., the justification $\{1, 2, 4\}$. Therefore the common strategy is to remove

a single axiom from each justification thereby nullifying each "reason" for the unsatisfiability to hold and thus eliminating this unsatisfiability.

For example, if we remove the set $\{2, 3\}$ from $\mathcal{O}$ then the unsatisfiability of F is eliminated. The resulting ontology $\mathcal{O}\backslash\{2, 3\}$ is a *repair* [8] for the F-unsatisfiability of $\mathcal{O}$ and the set $\{2, 3\}$ is a *diagnosis* [10] for the F-unsatisfiability of $\mathcal{O}$. A key requirement of computing repairs (diagnoses) is to find *maximal* repairs (*minimal* diagnoses).

In this case of a set of unsatisfiable concepts, $\mathcal{C}$, in the ontology, the issues we then want to address are (i) finding the causes of the unsatisfiabilities, and (ii) repairing the ontology $\mathcal{O}$ by replacing it with an ontology $\mathcal{O}'$ which is C-*satisfiable* for every $C \in \mathcal{C}$.

Kalyanpur et al. [6] discuss an approach for eliminating this set of unsatisfiable concepts in the ontology. This approach separates the unsatisfiable concepts into *root* unsatisfiable concepts and *derived* unsatisfiable concepts. Intuitively, a root unsatisfiable concept is a concept whose unsatisfiability is not caused by that of another concept in the ontology. A derived unsatisfiable concept is one which is *not* root unsatisfiable.

Example 2. Consider the following ontology:

$$\mathcal{O} = \left\{ \begin{array}{l} 1.\ \mathsf{C} \sqsubseteq \mathsf{A} \\ 2.\ \mathsf{C} \sqsubseteq \neg\mathsf{A} \\ 3.\ \mathsf{F} \sqsubseteq \mathsf{C} \end{array} \right\}$$

$\mathcal{O}$ is F-unsatisfiable and C-unsatisfiable. One cause for the F-unsatisfiability of $\mathcal{O}$ is the set of axioms $\{1, 2, 3\}$. One cause for the C-unsatisfiability of $\mathcal{O}$ is the set of axioms $\{1, 2\}$. One can see that $\{1, 2\} \subset \{1, 2, 3\}$ and therefore the C-unsatisfiability of $\mathcal{O}$ *causes* the F-unsatisfiability of $\mathcal{O}$. F is thus a derived unsatisfiable concept w.r.t. $\mathcal{O}$. C is a root unsatisfiable concept w.r.t. $\mathcal{O}$ because there is no other unsatisfiable concept in $\mathcal{O}$ which causes C to be unsatisfiable. □

A useful property of a root unsatisfiable concept, C, is that if one repairs the unsatisfiability of C then all other concepts in the ontology whose unsatisfiability is caused by that of C are automatically repaired [6]. This principle is demonstrated in the following example.

Example 3. Considering the ontology in Example 2:

We know that a justification for the C-unsatisfiability of $\mathcal{O}$ is $\{1, 2\}$. If Axiom 1 *or* Axiom 2 is removed from $\mathcal{O}$ we find that $\mathcal{O}$ becomes C-satisfiable. That is, $\mathcal{O}\backslash\{1\}$ and $\mathcal{O}\backslash\{2\}$ are both C-satisfiable. In addition to this, we find that $\mathcal{O}\backslash\{1\}$ and $\mathcal{O}\backslash\{2\}$ are also both F-satisfiable. Therefore resolving the unsatisfiability of C (root unsatisfiable concept) also resolves the unsatisfiability of F (derived unsatisfiable concept). □

Using the property described in Example 3, one can resolve a set of unsatisfiable concepts in some ontology as follows. Initially one identifies all the

root unsatisfiable concepts in the set. The unsatisfiabilities of these concepts are then eliminated using the repair procedure described above (eliminating axioms from the ontology which cause the unsatisfiabilities). After this, one has to re-compute/re-identify which of the *remaining* concepts in the set are root unsatisfiable (some concepts which were derived unsatisfiable may become root unsatisfiable after the initial repair).

This process has to be repeated until there are no more unsatisfiable concepts in the set. The drawbacks to this repair strategy are that (*i*) it is only applicable to one type of ontology error (unsatisfiable concepts) and (*ii*) it does not eliminate the entire set of unsatisfiable concepts simultaneously. Rather, it uses an iterative approach as described above.

4 Repair using Root Justifications

In this section, we discuss our approach to ontology repair. This approach can be applied to a set of *unwanted axioms*. We start by observing that the most prominent types of ontology error (unsatisfiable concepts and ontology inconsistency) can be generalized to some unwanted axiom.

For example, when a concept A is unsatisfiable w.r.t. an ontology $\mathcal{O}$, $A \sqsubseteq \bot$ is an unwanted axiom in $\mathcal{O}$. Removing $A \sqsubseteq \bot$ from $\mathcal{O}$ will make A satisfiable. When the ontology is inconsistent, then $\top \sqsubseteq \bot$ is the unwanted axiom. Removing such an axiom delivers a consistent ontology as output. Therefore, concept unsatisfiability and ontology inconsistency can both be reduced to the presence of unwanted axioms in the ontology.

Thus, the key differences from the method discussed in the previous section are (*i*) We are dealing with *other* types of errors as well, not just unsatisfiable concepts and (*ii*) We are eliminating a *set* of such errors *simultaneously*.

We first define some terminology that will be used in the remainder of this section. Given an ontology $\mathcal{O}$ and and some unwanted axiom α_U, a justification J for $\mathcal{O} \models \alpha_U$ is a *α_U-justification* for $\mathcal{O}$. The set of all α_U-justifications for $\mathcal{O}$ is denoted by $\mathcal{J}_\mathcal{O}(\alpha_U)$. Given an ontology $\mathcal{O}$ and a *set* of unwanted axioms $\mathcal{U}$, the set $\mathcal{J}_\mathcal{O}(\mathcal{U}) = \bigcup_{\alpha_U \in \mathcal{U}} \mathcal{J}_\mathcal{O}(\alpha_U)$. Using this terminology we characterize a root justification as follows.

Definition 1 ($\mathcal{U}$-root justification). *Given an ontology $\mathcal{O}$ and a* set *of unwanted axioms $\mathcal{U}$, a set RJ is a* $\mathcal{U}$-root justification *for $\mathcal{O}$ if and only if it is a α_U-justification for $\mathcal{O}$ for some $\alpha_U \in \mathcal{U}$ (i.e. $RJ \in \mathcal{J}_\mathcal{O}(\mathcal{U})$), and there is no $J \in \mathcal{J}_\mathcal{O}(\mathcal{U})$ such that $J \subset RJ$. We denote the set of all $\mathcal{U}$-root justifications for $\mathcal{O}$ by $\mathcal{RJ}_\mathcal{O}(\mathcal{U})$.*

In the work by Kalyanpur et al. [6] on root and derived unsatisfiable concepts, root justifications are used (implicitly) as a means to identify the root and derived unsatisfiable concepts. For our approach to ontology repair, the notion of a root justification is central and thus we highlight this principle here.

Example 4. For Example 1, let $\mathcal{U} = \{F \sqsubseteq \bot, C \sqsubseteq \bot\}$. We have already seen that $\mathcal{J}_{\mathcal{O}}(F \sqsubseteq \bot) = \{\{1,3\},\{1,2,4\}\}$. It is easy to see that $\mathcal{J}_{\mathcal{O}}(C \sqsubseteq \bot) = \{\{1,2\}\}$ and therefore that $\mathcal{J}_{\mathcal{O}}(\mathcal{U}) = \{\{1,3\},\{1,2,4\},\{1,2\}\}$. Therefore, according to Definition 1, the set of $\mathcal{U}$-root justifications for $\mathcal{O}$ is $\mathcal{RJ}_{\mathcal{O}}(\mathcal{U}) = \{\{1,2\},\{1,3\}\}$. □

We now show how root justifications can be computed for a set of unwanted axioms. Given an ontology $\mathcal{O}$ and a set of unwanted axioms $\mathcal{U}$, the following algorithm computes a *single* root justification $RJ \in \mathcal{RJ}_{\mathcal{O}}(\mathcal{U})$:

Algorithm 2: (Single root justification)

Input: Ontology $\mathcal{O}$, unwanted axiom set $\mathcal{U}$ ($|\mathcal{U}| \geq 1$)
Output: $\mathcal{U}$-root justification, RJ, for $\mathcal{O}$
Uses: entailedAxioms($\mathcal{O}, \mathcal{U}$), which returns $\{\alpha \in \mathcal{U} \mid \mathcal{O} \models \alpha\}$

```
1 RJ := O;
2 foreach α ∈ RJ do
3 |   if |entailedAxioms(RJ\{α}, U)| ≥ 1 then
4 |   |   RJ := RJ\{α};
5 |   end
6 end
7 return RJ;
```

The key difference between Algorithm 2 and the naïve pruning algorithm is that we are now considering the entailment of *all* unwanted axioms in a set (in procedure *entailedAxioms*(.)), rather than just a single axiom. It is clear that Algorithm 2 terminates for all finite inputs of $\mathcal{O}$ and $\mathcal{U}$. This follows from Line 2 of the algorithm which shows that the loop only considers the axioms in the finite set RJ. We now give an example to demonstrate how Algorithm 2 computes a root justification for a set of unwanted axioms.

Example 5. Consider an ontology $\mathcal{O}$ with ten axioms, i.e., $\mathcal{O} = \{1,\ldots,10\}$. Let $\mathcal{O}$ have a set of unwanted axioms: $\mathcal{U} = \{\gamma_1, \gamma_2, \gamma_3\}$. Let us assume that $\mathcal{J}_{\mathcal{O}}(\gamma_1) = \{\{1,2,3\},\{4,5\}\}$, $\mathcal{J}_{\mathcal{O}}(\gamma_2) = \{\{1,3\}\}$ and $\mathcal{J}_{\mathcal{O}}(\gamma_3) = \{\{4,5,7\},\{6,7,8\}\}$. Therefore $\mathcal{J}_{\mathcal{O}}(\mathcal{U}) = \{\{1,2,3\},\{4,5\},\{1,3\},\{4,5,7\},\{6,7,8\}\}$ and the set of all $\mathcal{U}$-root justifications for $\mathcal{O}$ is $\mathcal{RJ}_{\mathcal{O}}(\mathcal{U}) = \{\{4,5\},\{1,3\},\{6,7,8\}\}$.

We now show how Algorithm 2 computes *one* of these root justifications. We begin with the input $\mathcal{O}$ with ten axioms $\{1,\ldots,10\}$ and the unwanted axiom set $\mathcal{U} = \{\gamma_1, \gamma_2, \gamma_3\}$ such that $\mathcal{O} \models \alpha_U$ for each $\alpha_U \in \mathcal{U}$. The algorithm starts by assigning the set of axioms in $\mathcal{O}$ to the initial set RJ which will constitute the $\mathcal{U}$-root justification when the algorithm terminates. At this point, $RJ = \{1,\ldots,10\}$. In Lines 3 and 4, the algorithm loops through each axiom α in RJ, removing α from RJ if and only if $RJ\backslash\{\alpha\}$ entails *at least one* axiom from $\mathcal{U}$. When Axioms 1, 2 and 3 are removed it is still the case that $RJ \models \{\gamma_1, \gamma_3\}$. This is because the justification $\{4,5\}$ still holds for $\mathcal{O} \models \gamma_1$ and the justifications

$\{4, 5, 7\}$ and $\{6, 7, 8\}$ still hold for $\mathcal{O} \models \gamma_3$. Through similar reasoning, after removing Axioms 4 and 5 we find that $RJ \models \{\gamma_3\}$. At last, when we remove Axiom 6 from RJ we find that RJ does not entail any of the unwanted axioms in $\mathcal{U}$. Therefore, Axiom 6 *remains* in RJ. The same holds for Axioms 7 and 8. Finally, after removing Axioms 9 and 10, it is the case that $RJ \models \{\gamma_3\}$ and the result is that $RJ = \{6, 7, 8\}$ constitutes a *$\mathcal{U}$-root justification* for $\mathcal{O}$. □

Of course, Algorithm 2 is computationally intensive because we only consider a single axiom at a time in the ontology and for each consideration we require between 1 and $|\mathcal{U}|$ entailment tests. We use a more optimized version of this algorithm in practice (based on a sliding window technique [4]). Details of this algorithm [9] can be found in the reference provided.

In order to ensure that we are able to generate *all* the repairs for an unwanted axiom set in an ontology, it is necessary to know *all* the root justifications for the unwanted axiom set. We have given a brief description of a variant of Reiter's Hitting Set Algorithm which computes *all* the "regular" justifications for a single entailment [4]. This variant algorithm can also be used (with some slight modifications) to compute all *root* justifications for a *set* of unwanted axioms. The algorithm [9] can be found in the reference provided.

The significance of root justifications is that they can be used to generate precisely the $\mathcal{U}$-repairs for $\mathcal{O}$.

Definition 2 ($\mathcal{U}$-repair). *A subset R of $\mathcal{O}$ is a* $\mathcal{U}$-repair *for $\mathcal{O}$ if and only if $R \not\models \alpha_U$ for every $\alpha_U \in \mathcal{U}$, and for every R' for which $R \subset R' \subseteq \mathcal{O}$, $R' \models \alpha_U$ for some $\alpha_U \in \mathcal{U}$.*

We denote the set of $\mathcal{U}$-repairs for $\mathcal{O}$ by $\mathcal{R}_\mathcal{O}(\mathcal{U})$. For Example 1 it can be verified that $\mathcal{RJ}_\mathcal{O}(\{\mathsf{C} \sqsubseteq \bot, \mathsf{F} \sqsubseteq \bot\}) = \{\{1, 3\}, \{1, 2\}\}$ and thus $\mathcal{R}_\mathcal{O}(\{\mathsf{C} \sqsubseteq \bot, \mathsf{F} \sqsubseteq \bot\}) = \{\{2, 3, 4\}, \{1, 4\}\}$. The above $\mathcal{U}$-repairs can be generated from *$\mathcal{U}$-diagnoses*.

Definition 3 ($\mathcal{U}$-diagnosis). *A subset D of $\mathcal{O}$ is a* $\mathcal{U}$-diagnosis *for $\mathcal{O}$ if and only if $D \cap RJ \neq \emptyset$ for every $RJ \in \mathcal{RJ}_\mathcal{O}(\mathcal{U})$. D is a* minimal $\mathcal{U}$*-diagnosis for $\mathcal{O}$ if and only if there is no $\mathcal{U}$-diagnosis D' (for $\mathcal{O}$) such that $D' \subset D$.*

The set of minimal $\mathcal{U}$-diagnoses for $\mathcal{O}$ is denoted by $\mathcal{D}_\mathcal{O}(\mathcal{U})$. We then have the following theorem showing that the $\mathcal{U}$-repairs for $\mathcal{O}$ can be obtained from the $\mathcal{U}$-diagnoses for $\mathcal{O}$:

Theorem 1. $\mathcal{R}_\mathcal{O}(\mathcal{U}) = \{\mathcal{O} \setminus D \mid D \in \mathcal{D}_\mathcal{O}(\mathcal{U})\}$.

For Example 1 we have already seen that $\mathcal{RJ}_\mathcal{O}(\{\mathsf{C} \sqsubseteq \bot, \mathsf{F} \sqsubseteq \bot\}) = \{\{1, 3\}, \{1, 2\}\}$. From this it follows that $\mathcal{D}_\mathcal{O}(\{\mathsf{C} \sqsubseteq \bot, \mathsf{F} \sqsubseteq \bot\}) = \{\{1\}, \{2, 3\}\}$ and therefore, as indicated by Theorem 1, that $\mathcal{R}_\mathcal{O}(\{\mathsf{C} \sqsubseteq \bot, \mathsf{F} \sqsubseteq \bot\}) = \{\{2, 3, 4\}, \{1, 4\}\}$.

5 Implementation and Evaluation

We have implemented a Protégé 4 plugin for computing root justifications for sets of unwanted axioms (`http://krr.meraka.org.za/software/ontorepair`). We have extended it to also compute the $\mathcal{U}$-repairs. We have performed some preliminary experiments to compare this approach for ontology repair with the naïve sequential approach described in Section 3.2.

In this section we present and analyse the results of these experiments. We use three sample ontologies of varying size, structure, and application. There are three test cases, one for each ontology and in each case we select four different unwanted axiom sets from the ontology. Each test case has four experiments (one for each unwanted axiom set). In each experiment, we perform a *control* computation. This control computation uses the *naïve approach* to identify all *regular* justifications for each axiom in the unwanted axiom list. We record the *timing* for this as well as the total *number* of regular justifications computed.

Thereafter we compute all *root* justifications for the unwanted axiom list and record the same data of *timing* and total *number* of root justifications computed. The results of the latter approach are then compared to the results of the control computation. The x-axis of the performance graphs represents time in milliseconds. All the experiments were conducted on an Intel(R) Core(TM)2 Duo processor (2.66 GHz) running Ubuntu 8.04 with 3.2GB of memory.

The results depicted in Figures 1, 2 and 3 show that *OntoRepair* generally performs much better than the naïve approach in the cases where the total number of root justifications for the unwanted axiom set is less than the total number of regular justifications for all the axioms in the set. The only notable exception is Axiom set 1 in Test case 2. In the cases where the total number of root justifications is the same as the total number of regular justifications the *OntoRepair* approach performs worse than the naïve approach (Axiom sets 2 and 4 in Test case 1 and Axiom set 3 in Test case 3).

The overall best performance by the *OntoRepair* approach is observed in Test case 2. This is also the only test case in which the number of root justifications is less than the number of regular justifications in all the experiments. The overall worst performance was observed in Test case 3 where there are two out of four instances in which the number of root justifications and regular justifications are equal. Possibly the main reason for the performance being worse in this case is that we have to keep track of the entailment of *each* of the axioms in the set when computing a single *root* justification, whereas we only have to keep track of the entailment of *one* axiom when computing a single *regular* justification. We call this overhead inherent in computing root justifications the *root-of-set* overhead. Recall also that an entailment check is computationally expensive and many such checks (depending on the size of the ontology) are needed to compute a (regular or root) justification.

The worst performance in a *single* experiment is for Axiom set 4 in Test case 1. The reason for this is likely a combination of three things: (a) *root-of-set* overhead (the number of root justifications are equal to the number of regular justifications), (b) the justifications for the axioms in Axiom set 4 are *similar*,

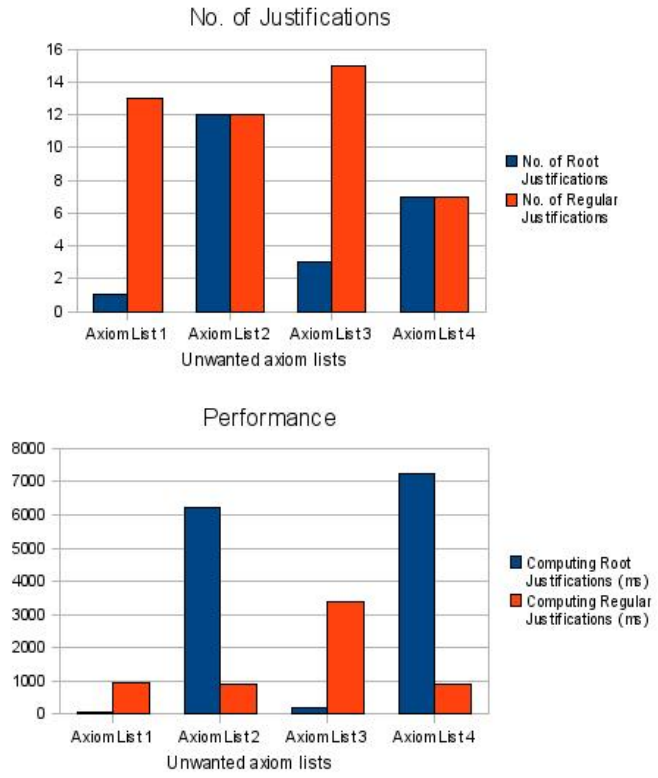

Fig. 1. Results for Test Case 1.

meaning that they share a significant amount of axioms, and (c) the number of axioms in some justifications is large. In particular, one axiom in Axiom set 4 in this test case, has four justifications, each of which has at least eleven axioms. Factors (b) and (c) can cause a considerable performance hit because if two justifications share a significant amount of axioms (or if they have a large amount of axioms) then more entailment checks are required because computing both justifications requires a fine-grained look at which axioms are *unique* to each justification.

The best performance in a *single* experiment is for Axiom set 3 in Test case 2. The most likely reasons for this are that (a) the number of root justifications are far less than the number of regular justifications, and (b) the justifications for the axioms in Axiom set 3 do not share many axioms.

In conclusion, the performance of *OntoRepair* depends on the kinds of axioms contained in the unwanted axiom set. This is because the specific axioms in the set influence the *justifications* which are computed and thus also the number of root justifications for the set versus the number of regular justifications for the

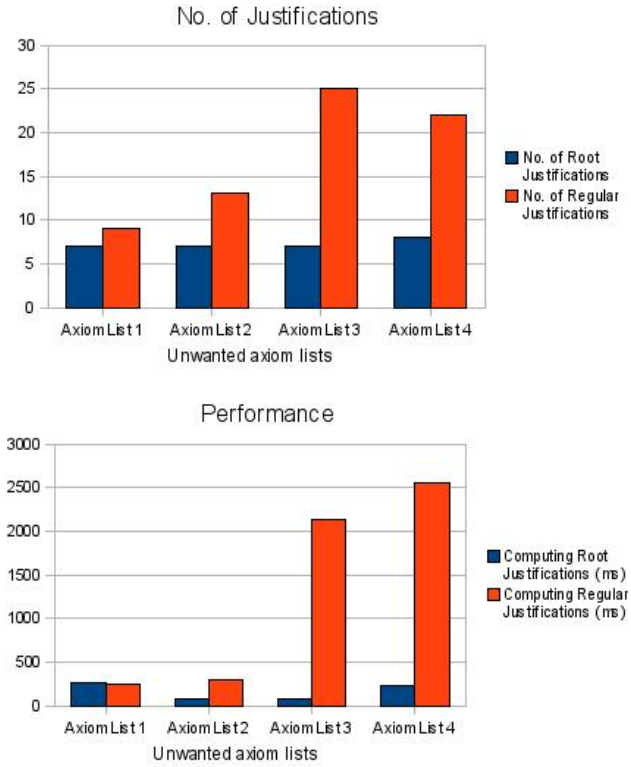

Fig. 2. Results for Test Case 2.

axioms in the set. Nine out of twelve of our experiments show instances where the number of root justifications is less than the number of regular justifications. This frequency justifies the use of *OntoRepair* which provides improved overall performance (over the naïve approach) in all these instances. However, there is scope for optimizing the computation of root justifications in *OntoRepair* to be comparable to the performance of the naïve approach, for those cases in which the number of root and regular justifications are equal. Finally, it is important to mention that since the empirical analysis was conducted on the basis of just three examples, the results are unlikely to be statistically significant and conclusive.

6 Conclusion

We have presented an alternative approach for ontology repair using the notion of root justifications. As we have seen, this approach may be used to eliminate a set of unwanted axioms from a particular ontology. We have implemented a Protégé 4 plugin, *OntoRepair*, to demonstrate this repair strategy. Some pre-

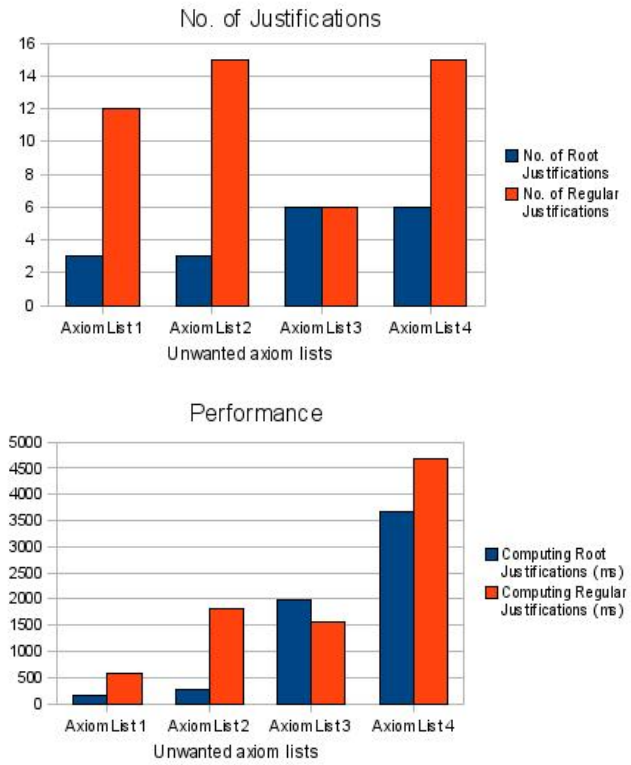

Fig. 3. Results for Test Case 3.

liminary experiments show overall better performance than a naïve approach to ontology repair. However, there are some special cases in which the naïve approach performs far better than the root justification approach and vice versa. More experiments will be performed to characterize the circumstances in which the root justification approach gains a clear advantage. Various performance optimizations for the *OntoRepair* tool are also in the pipeline.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge, 2 edition, 2007.

[2] F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the Description Logic $\mathcal{EL}$. In *Proc. KI*, pages 52–67. Springer, 2007.

[3] M. Horridge, B. Parsia, and U. Sattler. Explaining inconsistencies in OWL ontologies. In *Proc. SUM*, pages 124–137. Springer-Verlag, 2009.

[4] A. Kalyanpur. *Debugging and repair of OWL ontologies*. PhD thesis, University of Maryland, 2006.

[5] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proc. ISWC*, pages 267–280, 2007.

[6] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable classes in OWL ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):268–293, 2005.

[7] T. Meyer, K. Lee, R. Booth, and J. Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proc. AAAI*, 2006.

[8] T. Meyer, K. Moodley, and I. Varzinczak. First steps in the computation of root justifications. In *Proc. ARCOE*, 2010.

[9] K. Moodley. Debugging and repair of Description Logic ontologies. Master's thesis, University of KwaZulu-Natal, 2011.

[10] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[11] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. IJCAI*, pages 355–360. Morgan Kaufmann Publishers, 2003.

[12] E. Sirin, B. Parsia, B. Cuenca-Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2), 2007.

[13] B. Suntisrivaraporn, G. Qi, Q. Ji, and P. Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In *Proc. ASWC*, pages 1–15, 2008.

[14] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[15] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System description. In *Proc. IJCAR*, pages 292–297. Springer-Verlag, 2006.

[16] H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg. Debugging OWL-DL ontologies: A heuristic approach. In *Proc. ISWC*, pages 745–757. Springer, 2005.