A Fine-Grained Approach to Resolving Unsatisfiable Ontologies*

Joey Sik Chun Lam, Derek Sleeman, Jeff Z. Pan, and Wamberto Vasconcelos

Department of Computing Science
University of Aberdeen, AB24 3UE, UK
{slam, sleeman, jpan, wvasconc}@csd.abdn.ac.uk

Abstract. The ability to deal with inconsistencies and to evaluate the impact of possible solutions for resolving inconsistencies are of the utmost importance in real world ontology applications. The common approaches either identify the minimally unsatisfiable sub-ontologies or the maximally satisfiable sub-ontologies. However there is little work which addresses the issue of rewriting the ontology; it is not clear which axioms or which parts of axioms should be repaired, nor how to repair those axioms. In this paper, we address these limitations by proposing an approach to resolving unsatisfiable ontologies which is fine-grained in the sense that it allows parts of axioms to be changed. We revise the axiom tracing technique first proposed by Baader and Hollunder, so as to track which parts of the problematic axioms cause the unsatisfiability. Moreover, we have developed a tool to support the ontology user in rewriting problematic axioms. In order to minimise the impact of changes and prevent unintended entailment loss, both harmful and helpful changes are identified and reported to the user. Finally we present an evaluation of our interactive debugging tool and demonstrate its applicability in practice.

Keywords: Ontologies, Description Logics reasoning.

1 Introduction

Resolving inconsistencies in ontologies is a challenging task for ontology [25] modellers. Standard Description Logic (DL) [2] reasoning services can check if an ontology is unsatisfiable (i.e., if there are any unsatisfiable concepts in an ontology); however, they do not provide support for resolving the unsatisfiability. The ability to deal with inconsistencies and to evaluate the impact of the possible modifications are of the utmost importance in real world ontology applications.

Most existing approaches either identify problematic axioms (by providing the minimally unsatisfiable sub-ontologies) [22] or weaken the target unsatisfiable

---

ontology (by providing the possible maximally satisfiable sub-ontologies) [15]. However practical problems remain: it is not clear which axioms or which parts of axioms should be repaired, nor how to repair those axioms. Let us use an example to illustrate these limitations.

Example 1. Let us assume that an ontology $\mathcal{O}$ contains the following axioms:

α_1: $A \doteq C \sqcap \forall R.B \sqcap D$
α_2: $C \doteq \exists R.\neg B \sqcap B$
α_3: $G \doteq \forall R.(C \sqcap F)$

It can be shown that the concept A is unsatisfiable, by using standard DL TBox reasoning. The existing approaches [22,15] either identify the minimally unsatisfiable sub-ontologies $\mathcal{O}_1^{min} = \{\alpha_1, \alpha_2\}$ or calculate the maximally satisfiable sub-ontologies $\mathcal{O}_1^{max} = \{\alpha_1, \alpha_3\}$, and $\mathcal{O}_2^{max} = \{\alpha_2, \alpha_3\}$. In short, either α_1 or α_2 should be removed from $\mathcal{O}$. However, it is easy to see that we do not need to remove either the whole of α_1 or α_2. In order to minimise the loss of information from the ontology, we should simply remove parts of axiom α_1, i.e., (a) $A \sqsubseteq C$, or (b) $A \sqsubseteq \forall R.B$, or part of axiom α_2, i.e., (c) $C \sqsubseteq \exists R.\neg B$, and then $\mathcal{O}$ becomes satisfiable.

Schlobach et al. [22] and Kalyanpur et al. [12] have proposed approaches, which determine which parts of the asserted axioms are responsible for the unsatisfiability of concepts. We further discuss their work in Section 7. In this paper, we extend Meyer et al.'s tableaux algorithm [15]. Our algorithm traces which parts of the axioms are responsible for the unsatisfiability of a concept (this is a novel way of achieving the same result as [22,12]). Using this algorithm, we make the following two further contributions. The first is to calculate the lost entailments of named concepts due to the removal of axioms. Whenever parts of an axiom are removed, it frequently happens that indirect or implicit entailments are lost. In order to minimise the impact on the ontology, we analyse the lost entailments of named concepts which occur due to the removal of parts of axioms. The second contribution is to identify harmful and helpful changes; this is where the fine-grained tracing information is useful to facilitate rewriting the problematic axioms, rather than removing them completely. It should be noted that inappropriately revising a problematic axiom might not resolve the unsatisfiability, and could introduce additional unsatisfiable concepts into the ontology. For this purpose we define *harmful* and *helpful* changes with respect to an unsatisfiable named concept. A harmful change cannot resolve the problem, or might cause additional unsatisfiable concepts in the ontology; a helpful change resolves the problem without causing additional contradictions, and restores some lost entailments. We believe tools based on such techniques could help users to resolve unsatisfiable ontologies. To evaluate this vision, we have created a plugin in Protégé 3.2[1]. The result of our usability evaluation demonstrate that our approach helps non-expert ontology users to resolve unsatisfiable ontologies; the

---

[1] http://protege.stanford.edu/

performance results demonstrate that our algorithms provide acceptable performance when used with real world ontologies.

The rest of this paper is organised as follows. Section 2 briefly introduces ontologies and the Description Logic $\mathcal{ALC}$. Section 3 presents our fine-grained approach to pinpointing problematic parts of axioms. The impact of removing axioms is described in Section 4. The methods for identifying harmful and helpful changes are presented in Section 5. Section 6 presents the evaluation of our implementation. The paper closes with a discussion of related work and conclusion.

2   Ontology and the $\mathcal{ALC}$ DL

An ontology formally captures a shared understanding of certain aspects of a domain: it provides a common *vocabulary*, including important concepts, properties and their definitions, and *constraints* regarding the intended meaning of the vocabulary, sometimes referred to as background assumptions. Description Logics (DLs) [1] provide the underpinning of the recent W3C standard Web Ontology Language OWL DL.[2] In this paper, we use the smallest propositionally closed DL, i.e., the $\mathcal{ALC}$ DL [23], to illustrate our approach. The techniques presented here are general enough to be used as the basis for developing similar algorithms for more expressive DLs.

An ontology $\mathcal{O}$ consists of a set $\mathcal{T}$ (TBox) of concepts and role axioms and a set $\mathcal{A}$ (ABox) of individual axioms. As this paper handles satisfiabilities in ontologies, we focus on TBox reasoning. As $\mathcal{ALC}$ TBox reasoning is not influenced by ABox reasoning [16,20], without loss of generality, we assume that ontologies consist only of TBoxes in the rest of the paper. A TBox $\mathcal{T}$ consists of a set of axioms of the form $C \sqsubseteq D$ (*general concept inclusions*, GCIs); $C \doteq D$ (*concept equivalence*) is an abbreviation of $C \sqsubseteq D$ and $D \sqsubseteq C$, where C and D are (possibly complex) concept descriptions. $\mathcal{T}$ is *unfoldable* iff the left-hand side of every $\alpha \in \mathcal{T}$ contains a named concept A, there are no other αs with A on the left-hand side, and the right-hand side of α contains no direct or indirect references to A (no cycles). We divide $\mathcal{T}$ into an unfoldable part $\mathcal{T}_u$ and a general part $\mathcal{T}_g$, such that $\mathcal{T}_g = \mathcal{T} \setminus \mathcal{T}_u$.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of the domain of the interpretation $\Delta^{\mathcal{I}}$ (a non-empty set) and the interpretation function $\cdot^{\mathcal{I}}$, which maps each concept name $\mathsf{CN} \in N_C$ to a set $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $\mathsf{RN} \in N_R$ to a binary relation $\mathsf{RN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function can be extended to give semantics to concept descriptions (see Table 1). An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a TBox $\mathcal{T}$ if it satisfies all GCIs in $\mathcal{T}$; in this case, we say $\mathcal{I}$ is an interpretation of $\mathcal{T}$. A TBox $\mathcal{T}$ is consistent if there exists some interpretation that satisfies it. A concept C is satisfiable w.r.t. $\mathcal{T}$ if there exists an interpretation $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. A TBox $\mathcal{T}$ is satisfiable if all *named* concepts in $\mathcal{T}$ are satisfiable.

---

[2] More precisely, OWL DL is a key language and is a member of the family of the OWL standard languages, which also include OWL Lite and OWL Full.

Table 1. Semantics of $\mathcal{ALC}$-concepts

Constructor	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
concept name	CN	$\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
general negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
exists restriction ($\mathcal{E}$)	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y \rangle \in R^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$

Note that subsumption can be reduced to satisfiability [1]. If $\mathcal{T} \vDash C \sqsubseteq D$, then in all interpretations $\mathcal{I}$ that satisfy $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and so $C^{\mathcal{I}} \cap (\neg D)^{\mathcal{I}} = \emptyset$. Therefore, $\mathcal{T} \vDash C \sqsubseteq D$ iff $\mathcal{T} \vDash \neg(C \sqcap \neg D)$.

3   Approach

In this section, we introduce the extended tableau algorithm from Meyer et al.[15] (this kind of tracing technique was first proposed by Baader and Hollunder [4]). Instead of removing complete axioms involved in an unsatisfiability, our algorithm captures the components of axioms responsible for a concept's unsatisfiability.

3.1   Extended Tableaux Algorithm

We assume that $\mathcal{T} = \{\alpha_1, \cdots, \alpha_n\}$, with α_i referring to $C_i \doteq D_i$ or $C_i \sqsubseteq D_i$ for $i = 1, \ldots, n$. A tableau-based algorithm decides the satisfiability of a concept C_i w.r.t. $\mathcal{T}$ by trying to construct a representation of a *model* for it, called a tree **T**. The model is an interpretation $\mathcal{I}$ in which $C_i^{\mathcal{I}}$ is non-empty. Each node x in the tree is labeled with a set $\mathcal{L}(x)$ of concept or role elements. The concept elements are of the form $(a : C, I, a' : C')$, where C and C' are concepts, a and a' are individual names, and I is an index-set. This means that the individual a belongs to concept C due to an application of an expansion rule on $a' : C'$. The set of axioms, which $a : C$ comes from, is recorded in the index-set I. This is done by adding i to I, which is a set of integers in the range $1, \ldots, n$. In an element of the form $(a : C, I, a' : C')$ we frequently refer to C as "the concept", and a as "the individual" (i.e., we are referring to the first concept assertion). When a concept element $(a : C, I, a' : C')$ exists in the label of a node x, it represents an interpretation $\mathcal{I}$ that satisfies C, i.e., the individual corresponding to a is in the interpretation of C. That is, if $(a : C, -, -) \in \mathcal{L}(x)$, then $a \in C^{\mathcal{I}}$, where "$-$" stands for any value, that is, it is a place holder. Role elements are of the form $(R(a, b), I, a : \exists R.D)$, where R is a binary relationship between individual a and b; I is the index-set; the third parameter is to record the existence of

$R(a, b)$ due to an application of an expansion rule on $a : \exists R.D$. That is, if $(R(a, b), -, -) \in \mathcal{L}(x)$, then $\langle a, b \rangle \in R^{\mathcal{I}}$.

3.2   Applications of Expansion Rules

To determine the satisfiability of a concept A in $\mathcal{T}$, the algorithm initialises a tree **T** to contain a single node x, called the root node, with $\mathcal{L}(x) = \{(a : A, \emptyset, nil)\}$. The tree is then expanded by repeatedly applying a set of expansion rules which either extend node labels or add new leaf nodes. Our extended set of expansion rules for the Description Logic $\mathcal{ALC}$ is shown in Table 2, where A_i is a named concept, C, C_1, C_2, C_i, D_i are concept descriptions, R is a role name, a and b are individuals, $RHS(\alpha_i)$ is the concept at the right hand side of α_i, and the signature $Sig(\alpha_i)$ of an axiom α_i is the set of concept and role names occurring in α_i.

Table 2. Our extended tableaux expansion rules for $\mathcal{ALC}$

$U_{\doteq}^{+}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $(a : A_i, I, -) \in \mathcal{L}(x)$ and $(a : C_i, I \cup \{i\}, a : A_i) \notin \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
$U_{\doteq}^{-}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $(a : \neg A_i, I, -) \in \mathcal{L}(x)$ and $(a : \neg C_i, I \cup \{i\}, a : \neg A_i) \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \neg C_i, I \cup \{i\}, a : \neg A_i)\}$
$U_{\sqsubseteq}$-rule	if $A_i \sqsubseteq C_i \in \mathcal{T}_u$, $(a : A_i, I, -) \in \mathcal{L}(x)$ and $(a : C_i, I \cup \{i\}, a : A_i) \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
$\sqcap$-rule	if $(a : C_1 \sqcap C_2, I, -) \in \mathcal{L}(x)$, and $\{(a : C_1, I, a : C_1 \sqcap C_2), (a : C_2, I, a : C_1 \sqcap C_2)\} \nsubseteq \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_1, I, a : C_1 \sqcap C_2), (a : C_2, I, a : C_1 \sqcap C_2)\}$
$\sqcup$-rule	if $(a : C_1 \sqcup C_2, I, -) \in \mathcal{L}(x)$, and $\{(a : C_1, I, a : C_1 \sqcup C_2), (a : C_2, I, a : C_1 \sqcup C_2)\} \cap \mathcal{L}(x) = \emptyset$, then create two $\sqcup$-successor y, z of x with: $\mathcal{L}(y) := \mathcal{L}(x) \cup \{(a : C_1, I, a : C_1 \sqcup C_2)\}$ $\mathcal{L}(z) := \mathcal{L}(x) \cup \{(a : C_2, I, a : C_1 \sqcup C_2)\}$
$\exists$-rule	if $(a : \exists R.C, I, -) \in \mathcal{L}(x)$, a is not blocked (see Section 3.4), and $\{(R(a, b), I, a : \exists R.C), (b : C, I, a : \exists R.C)\} \nsubseteq \mathcal{L}(x)$, where b is an individual name not occurring in $\mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(R(a, b), I, a : \exists R.C), (b : C, I, a : \exists R.C)\}$
$\forall$-rule	if $(a : \forall R.C, I, -) \in \mathcal{L}(a)$, and $(R(a, b), J, a : \exists R.D_i) \in \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : C, I \cup J, a : \forall R.C)\}$
$\sqsubseteq$-rule	if $(C_i \sqsubseteq D_i) \in \mathcal{T}_g$, and there exists $(- : E, -, -) \in \mathcal{L}(x)$, $Sig(E) \cup Sig(C_i) \neq \emptyset$, a is not blocked, and $(a : \neg C_i \sqcup D_i, I \cup \{i\}, -) \notin \mathcal{L}(x)$, for every individual a in the node then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \neg C_i \sqcup D_i, I \cup \{i\}, a : C)\}$

During the expansion, concept descriptions are assumed to be converted to negation normal form.[3] We now explain the expansion rules. The three rules ($U_{\doteq}^{+}$-rule, $U_{\doteq}^{-}$-rule and $U_{\sqsubseteq}$-rule) describe the *unfolding* procedure. Unfolding

---

[3] A concept description is in negation normal form when negations apply only to concept names, and not to compound terms.

a concept expression is to replace defined names by their definitions, so that it does not contain names defined in the terminology. These rules are used for optimisation (also called lazy unfolding) [5]. That means only unfolding concepts as required by the progress of the satisfiability testing algorithm. The $U_{\doteq}^{+}$-rule and $U_{\doteq}^{-}$-rule reflect the symmetry of the equality relation in the non-primitive definition $A \doteq C$, which is equivalent to $A \sqsubseteq C$ and $\neg A \sqsubseteq \neg C$. The $U_{\sqsubseteq}$-rule on the other hand reflects the asymmetry of the subsumption relation in the primitive definition $A \sqsubseteq C$.

Disjunctive concept elements $(a : C_1 \sqcup C_2, -, -) \in \mathcal{L}(x)$ result in *non-deterministic* expansion. We deal with this non-determinism by creating two $\sqcup$-successors y, z of x with: $\mathcal{L}(y) := \mathcal{L}(x) \cup \{(a : C_1, \cdots)\}$, and $\mathcal{L}(z) := \mathcal{L}(x) \cup \{(a : C_2, \cdots)\}$.

For any existential role restriction concept $(a : \exists R.C, I, -) \in \mathcal{L}(x)$, the algorithm introduces a new individual b as the role filler, and this individual must satisfy the constraints expressed by the restriction. Thus, b is an individual of C, and hence $(b : C, I, a : \exists R.C)$ and $(R(a, b), I, a : \exists R.C)$ are added to the label of the node. A universal role restriction concept $(a : \forall R.D, J, -) \in \mathcal{L}(x)$ interacts with already defined role relationships to impose new constraints on individuals. That is, if $(R(a, b), I, a : \exists R.C)$ exists in $\mathcal{L}(x)$, then b is also an individual of D; new concept elements $(b : D, I \cup J, a : \forall R.D)$ and $(b : C, I \cup J, a : \exists R.C)$ are added to the label.

If there exists a concept C in the signature of the left-hand side of a GCI axiom $(\alpha_i \in \mathcal{T}_g, \alpha_i$ is $C_i \sqsubseteq D_i)$, and there is an element $(a : C, I, -) \in \mathcal{L}(x)$, and the signature of C has common elements with $Sig(C_i)$ then we apply the $\sqsubseteq$-rule to α_i. The newly added element will be $(a : \neg C_i \sqcup D_i, I \cup \{i\}, a : C)$. With this technique we are able to trace which element in the tree invokes the application of expansion rules on GCI axioms, therefore we can trace how the GCI axioms cause the concept's unsatisfiability.

The algorithm repeatedly expands the tree by applying the rules in Table 2 as many times as possible until either any one of the fully expanded leaf nodes has no clash or none of the rules is applicable to any node of the tree. A node is fully expanded when none of the rules can be applied to it. **T** is fully expanded when all of its leaf nodes are fully expanded. A node x contains an obvious *clash* when, for some individual b and some concept C, $\{(b : C, -, -), (b : \neg C, -, -)\} \subseteq \mathcal{L}(x)$.

When a clash is found in a node, the classical tableaux algorithm [4] either backtracks and selects a different leaf node, or reports the clash and terminates, if no node remains to be expanded. The main difference is that our algorithm terminates when either (1) any one of the fully expanded leaf nodes is without a clash or (2) none of the rules is applicable. Since the rules are still applicable to a node even when a clash is found, there may be more than one clash in the node, and furthermore this clash may also occur in other nodes (repeated nodes). As a result, we can obtain all the clashes in the tree and eliminate the repeated clashes. If the input of the tableaux algorithm is a concept C and a terminology $\mathcal{T}$, we have the following property: C is *unsatisfiable* iff each path from the root to the leaf node in the tree contains at least one clash. This implies

that an unsatisfiable concept becomes satisfiable if all the clashes in any one of the paths of the tree are resolved (i.e., a complete path from root to leaf). This is because whenever the non-deterministic $\sqcup$-rule is applied, two new $\sqcup$-successor nodes are created; this is the only way to create the leaf nodes. It is sufficient to resolve all clashes in either of the two branches created.

3.3   Sequences of a Clash

Figure 1 shows how the tableau algorithm is applied to Example 1 (shown in Section 1) to check for the satisfiability of A. The tree **T** contains a root node x whose label contains a clash because $\{(b : B, \{1, 2\}, a : \forall R.B), (b : \neg B, \{1, 2\}, a : \exists R.\neg B)\} \subseteq \mathcal{L}(x)$. According to Definition 1, we can obtain two sequences, Seq^+ and Seq^-(see Figure 2). Note that the union of the index sets of the first elements in the sequences of the clashes in the tree gives the set of axioms which cause A to be unsatisfiable. The above two sequences show that axiom α_1 and α_2 cause the unsatisfiability of A.

---

(1) Initialise the root node x with $\mathcal{L}(x) := \{(a : A, \emptyset, nil)\}$,

(2) Apply the $U_{\doteq}^+$ -rule to $(a : A, \emptyset, nil)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A)\}$,

(3) Apply the $\sqcap$-rule twice to $(a : C \sqcap \forall R.B \sqcap D, \{1\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D)$,
$(a : D, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : C, \{1\}, a : C \sqcap \forall R.B)$,
$(a : \forall R.B, \{1\}, a : C \sqcap \forall R.B)\}$

(4) Apply the $U_{\doteq}^+$ -rule to $(a : C, \{1\}, \cdots)$, followed by applying the $\sqcap$-rule,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C)$,
$(a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B), (a : B, \{1, 2\}, a : \exists R.\neg B \sqcap B)\}$,

(5) Apply the $\exists$-rule to $(a : \exists R.\neg B, \{1, 2\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : \neg B, \{1, 2\}, a : \exists R.\neg B), (R(a, b), \{1, 2\}, a : \exists R.\neg B)\}$

(6) Apply the $\forall$-rule to $(a : \forall R.B, \{1\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : B, \{1, 2\}, a : \forall R.B)\}$

Fig. 1. The application of expansion rules on A in Example 1

Definition 1 (Sequences of a Clash). *Given a clash in a tree, the sequences of a clash, Seq^+ and Seq^-, contain elements involved in the clash. The sequences are of the form $\langle(a_0 : C_0, I_0, a_1 : C_1), (a_1 : C_1, I_1, a_2 : C_2), \cdots, (a_{n-1} : C_{n-1}, I_n, a_n : C_n), (a_n : C_n, \emptyset, nil)\rangle$, where $I_{i-1} \subseteq I_i$ for each $i = 1, \cdots, n$. The first elements of Seq^+ and Seq^- are of the form $(a : C, I', a' : C')$ and $(a : \neg C, I'', a'' : C'')$ respectively. The last element of both sequences is the same.*

$$\mathcal{L}(x) := \{(a : A, \emptyset, nil), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),$$
$$(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : D, \{1\}, a : C \sqcap \forall R.B \sqcap D),$$
$$(a : C, \{1\}, a : C \sqcap \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),$$
$$(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C), (a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B),$$
$$(a : B, \{1, 2\}, a : \exists R.\neg B \sqcap B), (b : \neg B, \{1, 2\}, a : \exists R.\neg B),$$
$$(R(a, b), \{1, 2\}, a : \exists R.\neg B), (b : B, \{1, 2\}, a : \forall R.B)\}$$

(4). Matches (2), does not exist in Seq^+ or Seq^-

$$Seq^+ := \langle(b : B, \{1, 2\}, a : \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),$$
$$(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),$$
$$(a : A, \emptyset, nil)\rangle,$$

(2). Matches $(a{:}C, \{1\},...)$ **(3). Matches (2), but exists in Seq^-**

$$Seq^- := \langle(b : \neg B, \{1, 2\}, a : \exists R.\neg B), (a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B),$$
$$(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C), (a : C, \{1\}, a : C \sqcap \forall R.B),$$
$$(a : C \sqcap \forall R.B, \{1\}, a : a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),$$
$$(a : A, \emptyset, nil)\rangle$$ **(1). Remove $(a{:}C, \{1\},...)$**

Fig. 2. The fully expanded tree for A in Example 1

Now that clashes in the tree have been found, we want to identify the axioms which could be removed to resolve the unsatisfiability. We can identify these by looking at the nodes in the tree which contain clashes. The example in Figure 3 (on the left) shows a tree with six clashes (nodes with clashes are shaded). We now describe how Reiter's Hitting Set algorithm [19] can be adapted to make a general procedure for identifying the sets of clashes to be resolved. Firstly, for each path from the root to a leaf of the tree, we gather the set of each of the nodes on that path which has a clash. In our example the following sets are found: $\{a, b, d\}, \{a, b\}, \{a, c, e\}, \{a, c, f\}$. Now, using these sets we apply the Hitting Set algorithm; the Hitting set Tree is shown in Figure 3 on the right-hand side. Now for each leaf node n we gather the set E_n of all edge labels on the path from the root to that node. The sets thus obtained from each leaf node are gathered into one large set S. This gives a set with 13 elements; some of these $E_n \in S$ are subsets of each other; we pick out the *minimal* sets; i.e., the sets $E_i \in S$ for which there is no $E_j \in S$ such that $E_j \subset E_i$. In our example this gives $S = \{\{a\}, \{b, c\}, \{b, e, f\}, \{b, e, d\}\}$, as desired. The axioms involved in each clash from the above nodes are actually the same as the notion of minimal unsatisfiability preserving sub-TBoxes (MUPS) in [22], that is there are four sets of MUPS in the unsatisfiable concept above.

From each MUPS, we know which axioms cause the unsatisfiability. Furthermore, from the sequences of the clashes, we know which concepts within these axioms cause the unsatisfiability. We can assign a specific number to each MUPS, and annotate the problematic concepts in these axioms with a specific superscript number corresponding to the MUPS which it occurs in. Note that a concept component may be involved in more than one MUPS, therefore it may be annotated
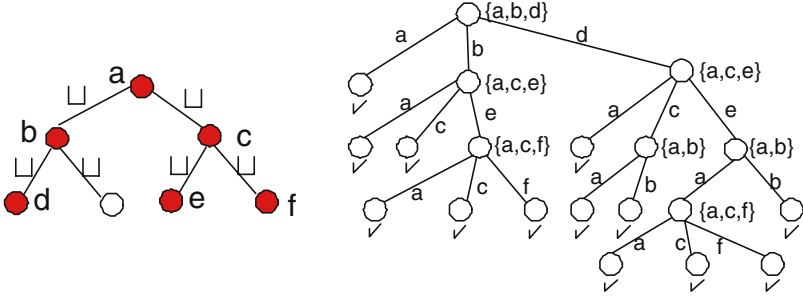
Fig. 3. Left-hand side: a fully expanded tree with six clashes; right-hand side: hitting set tree

with more than one number. We introduce the notion of arity of a concept C in an axiom α, denoted by $arity(\alpha, C)$, to count the number of times it appears in the clashes. This idea is similar to the core of MUPS in [22]. This means that removing a concept component with arity n can resolve n clashes. In order to illustrate the benefit of our fine-grained approach, we add the following axioms to Example 1:

α_4: $K \doteq C \sqcap \forall R.(P \sqcap F)$
α_5: $P \doteq \forall R.F \sqcap B$

In this case, concept K is also unsatisfiable due to the existence of a clash in a node of the tree for K. For simplicity, we do not show the sequences in the clash. We now annotate the concepts in the axioms which are involved in the two unsatisfiable concepts with superscript numbers as follows:

α_1: $A^1 \doteq C^1 \sqcap \forall R^1.B^1 \sqcap D$,
α_2: $C^{1,2} \doteq \exists R^{1,2}.(\neg B)^{1,2} \sqcap B$,
α_3: $G \doteq \forall R.(C \sqcap F)$
α_4: $K^2 \doteq C^2 \sqcap \forall R^2.(P^2 \sqcap F)$
α_5: $P^2 \doteq \forall R.F \sqcap B^2$

From the above, we can easily see which concepts in the axioms cause which concepts to be unsatisfiable. It is obvious that removing the concept $\exists R.\neg B$ in axiom α_2 can resolve two unsatisfiable concepts.

3.4   Refined Blocking

To deal with cyclic axioms, it is necessary to add cycle detection (often called *blocking*) to the preconditions of some of the expansion rules in order to guarantee termination [3,7]. We use a simple example to describe the necessity of blocking, by using the classical tableau algorithm:

Example 2. Given an ontology containing a single cyclic axiom, α_1: $A \doteq \exists R.A$, then testing the satisfiability of A leads to:

1. $\mathcal{L}(x) := \{(a_0 : A), (a_0 : \exists R.A)\}$
2. $\mathcal{L}(x_1) := \mathcal{L}(x) \cup \{R(a_0, a_1), (a_1 : A), (a_1 : \exists R.A)\}$
3. $\mathcal{L}(x_2) := \mathcal{L}(x_1) \cup \{R(a_1, a_2), (a_2 : A,), (a_2 : \exists R.A)\}$ $\cdots$

The application of the $U_{\sqsubseteq}^{+}$-rule leads to $(a_0 : \exists R.A)$ being added to $\mathcal{L}(x)$, and the application of the $\exists$-rule leads to the creation of a new individual a_1 with new elements $R(a_0, a_1), (a1 : A)$ added into $\mathcal{L}(x_1)$, the same expansion rules would be applied and the process would continue indefinitely. Since all individuals a_1, a_2, $\cdots$ receive the same concept assertions as a_0, we may say the algorithm has run into a cycle. Therefore, blocking is necessary to ensure termination. The general idea is to stop the expansion of a node whenever the same concept assertions recur in the node. Blocking imposes a condition on the $\exists$-rule: in the classical algorithm, an individual a is *blocked* by an individual b in a node label $\mathcal{L}(x)$ iff $\{D|(a : D) \in \mathcal{L}(x)\} \subseteq \{D'|(b : D') \in \mathcal{L}(x)\}$. In our example, that would mean a_1 in $\mathcal{L}(x_1)$ is blocked by a_0, because $\{A, \exists R.A\} \subseteq \{A, \exists R.A\}$ in the classical version. Intuitively, it can be seen that termination is now guaranteed because a finite terminology can only produce a finite number of different concept expressions and therefore a finite number of different labelling sets; all nodes must therefore eventually be blocked [9].

Our blocking approach is slightly different from the classical one. We define the refined blocking condition as follows: the application of the $\exists$-rule to an individual a is blocked by b iff $\{(D, I)|(a : D, I, -) \in \mathcal{L}(x)\} = \{(D', I')|(b : D', I', -) \in \mathcal{L}(x)\}$. Informally, the justification for this refinement is the following. If I is not equal to I', then we treat $(a : C, I, -)$ and $(a : C, I', -)$ as different elements; this is because the concept C in the two elements has been introduced from different axioms. Therefore, we still apply the rules to both $(a : C, I, -)$ and $(a : C, I', -)$ to expand the tree. As a result, in our approach, an individual a is blocked by b iff each of the elements in $\mathcal{L}(x)$ with individual a is exactly matched with one of the elements in $\mathcal{L}(x)$ with individual b, and vice versa; i.e. these matched elements have the same concept and index-set. In contrast, the classical tableau algorithm does not take the index-set of axioms into account when blocking is performed; the elements in the labels of nodes only have one parameter. The elements $(a : C, I, -)$ and $(a : C, I', -)$ will be presented as $(a : C)$ in the classical one, and therefore only one rule is applied to $(a : C)$ once.

Example 3. This example describes how the refined blocking works:
α_1: $A \sqsubseteq \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A$
α_2: $D \sqsubseteq C$
α_3: $E \sqsubseteq \forall R.C$
α_4: $F \sqsubseteq \forall R.\forall R.C$

We use Example 3 to illustrate why our refined blocking is necessary[4]. For simplicity, we do not show the third parameter of the elements in the node label. Figure 4 shows the fully expanded tree. In step 2, after applying the $\exists$-rule on $(a : \exists R.A, \{1\})$, we can see that, in $\mathcal{L}(x_1)$ in the classical algorithm,

---

[4] Note that despite the apparent complexity, this example is the simplest possible to illustrate the need for our refined blocking.

Step 1

$\mathcal{L}(x) := \{(a : A, \emptyset), (a : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (a : \neg C, \{1\}),$
$(a : D, \{1\}), (a : E, \{1\})(a : F, \{1\}), (a : \exists R.A, \{1\}), (a : C, \{1, 2\}),$
$(a : \forall R.C, \{1, 3\}), (a : \forall R.\forall R.C, \{1, 4\})\}$ *Clash 1*

Step 2: Apply the $\exists$-rule on $(a : \exists R.A, \{1\})$ *Clash 2*   **(1)**

$\mathcal{L}(x_1) := \mathcal{L}(x) \cup \{(b : A, \{1\}), (b : \forall R.C, \{1, 4\}), (b : C, \{1, 3\}),$
$(b : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (b : \neg C, \{1\}), (b : D, \{1\}),$
$(b : E, \{1\}), (b : F, \{1\}), (b : \exists R.A, \{1\}), (b : C, \{1, 2\}),$
$(b : \forall R.\forall R.C, \{1, 4\}), (b : \forall R.C, \{1, 3\})\}$

(2)   *Clash 3*

Step 3: Apply the $\exists$-rule on $(b : \exists R.A, \{1\})$ *Clash 4*

$\mathcal{L}(x_2) := \mathcal{L}(x_1) \cup \{(c : A, \{1\}), (c : C, \{1, 4\}), (c : C, \{1, 3\}),$
$(c : \forall R.C, \{1, 4\}), (c : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (c : \neg C, \{1\}),$
$(c : D, \{1\}), (c : E, \{1\}), (c : F, \{1\}), (c : \exists R.A, \{1\}), (c : C, \{1, 2\}),$
$(c : \forall R.\forall R.C, \{1, 4\}), (c : \forall R.C, \{1, 3\})\}$

(3)

Step 4: Apply the $\exists$-rule on $(c : \exists R.A, \{1\})$

$\mathcal{L}(x_3) := \mathcal{L}(x_2) \cup \{(d : A, \{1\}), (d : C, \{1, 4\}), (d : C, \{1, 3\}),$
$(d : \forall R.C, \{1, 4\}), (d : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (d : \neg C, \{1\}),$
$(d : D, \{1\}), (d : E, \{1\}), (d : F, \{1\}), (d : \exists R.A, \{1\}), (d : C, \{1, 2\}),$
$(d : \forall R.\forall R.C, \{1, 4\}), (d : \forall R.C, \{1, 3\})\}$

Fig. 4. The fully expanded tree of Example 3

the set of concept elements with b is a subset of the set of concept elements with a, therefore, individual b is blocked by a. The $\exists$-rule cannot be applied on $(b : \exists R.A, \{1\})$, so the algorithm would terminate. Three clashes are found in $\mathcal{L}(x_1)$; axioms α_1, α_2 and α_3 are all involved in the clashes (cf. *Clash 1, 2, 3* in Figure 4). However α_4 would be missed out by the classical algorithm, although it also triggers a clash in our tree. The reason is that the set of concept elements with a is the same as the set of concept elements with b in the classical algorithm (i.e., $\{D | (a : D, -) \in \mathcal{L}(x_1)\} = \{D | (b : D, -) \in \mathcal{L}(x_1)\}$); $(b : C, \{1, 3\})$ is the same as $(b : C, \{1, 2\})$ (cf. **(1)** in Figure 4), and $(b : \forall R.C, \{1, 4\})$ is the same as $(b : \forall R.C, \{1, 3\})$ (cf. **(2)**). In our approach, these elements are different. Therefore we still apply the $\forall$-rule to $(b : \forall R.C, \{1, 4\})$ and add a new element $(c : C, \{1, 4\})$ into the node label, which is different from the existing elements $(c : C, \{1, 2\})$ and $(c : C, \{1, 3\})$ (cf. **(3)**). The newly added element from α_4 triggers another clash (cf. *Clash 4*). Next, we keep applying the $\exists$-rule to $(c : \exists R.A, \{1\})$ and create a new individual d. Finally, each of the elements in $\mathcal{L}(x_3)$ with individual d is exactly matched with one of the elements in $\mathcal{L}(x_3)$ with individual c, and vice versa; i.e. these matched elements have the same concept

and index-set. The individual c is blocked by d, and then the application of the rules is terminated.

3.5   Complexity, Soundness and Completeness

The differences between our algorithm and the classical one are that (1) when a clash is detected, the classical algorithm either backtracks and selects a different node, or reports the clash and terminates if no more nodes remain to be expanded, whereas our algorithm will not do so; it only terminates when the tree is fully expanded or until blocking occurs, in order to find all possible clashes. Therefore, the complexity of our algorithm is the same as the classical one in the worst case [23], as both need to fully expand all nodes; (2) we add two extra parameters in each of the elements of a node label. The expansion rules do not depend on these two parameters, and hence they add only a constant amount to each expansion and do not affect the complexity and correctness of the original algorithm [4]; (3) our refined blocking condition is: the application of the $\exists$-rule to an individual a is blocked by b iff $\{(D, I) | (a : D, I, -) \in \mathcal{L}(x)\} = \{(D', I') | (b : D', I', -) \in \mathcal{L}(x)\}$. The number of elements with different concept descriptions that can be introduced in each fully expanded leaf node is finite. Also, for each of concept description C, there can be only a finite number of elements $(a : C, I_1, -)$, $(a : C, I_2, -)$,$\cdots$, $(a : C, I_n, -)$ with n bounded by the number of axioms in the ontology. The algorithm is therefore guaranteed to terminate.

3.6   Removing Clashes

Given an unsatisfiable concept A in $\mathcal{T}$, we can obtain a fully expanded tree containing a node with at least one clash. For each clash, the sequences of the clash, Seq^+ and Seq^-, are obtained as in Definition 1. We can derive the following lemma:

Lemma 2. *Let the first elements of the sequences be $(a : C, I', -)$ and $(a : \neg C, I'', -)$, and let the last element of the sequences be $(b : A, \emptyset, nil)$. We know that the set of axioms $I := I' \cup I''$ causes A to be unsatisfiable. Let $\mathcal{D}$ be the set of all concepts appearing in the elements of the sequences: removing one of the concepts in $\mathcal{D}$ from one of the axioms in I is sufficient to resolve the clash.*

Proof. For any concept picked from $\mathcal{D}$, it must occur in the sequences and have an adjacent element which is before or after. For any two adjacent elements in a sequence, e_1 and e_2, there are only two possibilities:

- e_1 and e_2 are of the form $(a : E_1, -, a : E_2)$ and $(a : E_2, -, -)$ containing the same individual, this means the concept E_1 is a superconcept of E_2. If E_1 (or E_2) is removed, the subsumption relationship between E_2 and E_1 is removed. Therefore, the individual a no longer belongs to E_1 (or E_2), nor does it belong to any of the concepts in the elements preceding the occurrence of e_1 in the sequences. That means the concept of the first element in the

sequence is not subsumed by the removed concept either, hence the clash is resolved.

– e_1 and e_2 are of the form $(a : E_1, -, b : E_2)$ and $(b : E_2, -, -)$ containing different individuals, this means the concept E_1 participates in a role relationship with E_2. If E_1 or E_2 is removed, then the role relationship will be removed, therefore there will be no such individual a participating in the role, and all the concepts in the elements preceding the occurrence of e_1 will not be related to a, and hence the clash will be resolved.   □

4   Impact of Removing Axioms

After the parts of the axioms causing the unsatisfiability of concept(s) are identified, the next step is to resolve the unsatisfiability. In this section, we discuss, with examples, the impact of removing axioms on an ontology.

The simplest way to resolve unsatisfiability is to remove parts of the problematic axioms or the whole axioms. However, in this case, it will be easy for ontology modellers to accidentally remove indirect or implicit entailments in the ontology. We use the following mad_cow[5] example to explain what we mean by the impact of removing axioms from an ontology:

Example 4. Given an ontology where Mad_Cow is unsatisfiable due to axioms $\alpha_1, \alpha_3, \alpha_4, \alpha_5$, the concepts and roles tagged with a star (*) are responsible for the unsatisfiability:

α_1: Mad_Cow* $\doteq$ $\exists$ eats*.(($\exists$part_of*.Sheep*) $\sqcap$ Brain) $\sqcap$ Cow*
α_2: ($\exists$part_of.Plant $\sqcup$ Plant) $\sqsubseteq$ $\neg$ ($\exists$part_of.Animal $\sqcup$ Animal)
α_3: Cow* $\sqsubseteq$ Vegetarian*
α_4: Vegetarian* $\doteq$ $\forall$ eats.($\neg$ Animal) $\sqcap$ Animal $\sqcap$ $\forall$eats*.($\neg\exists$part_of*.Animal*)
α_5: Sheep* $\sqsubseteq$ $\forall$ eats.Grass $\sqcap$ Animal*
α_6: Grass $\sqsubseteq$ Plant
α_7: Giraffe $\sqsubseteq$ Vegetarian

When an axiom involved in the unsatisfiability of a concept is changed, we calculate the impact of removal on the ontology in three ways: (1) *the named concepts involved in the unsatisfiability*: These concepts might lose entailments which are not responsible for the unsatisfiability. To resolve Mad_Cow, one may claim that not all cows are vegetarians if there exist mad cows, therefore, α_3 is removed. However, the indirect assertion Mad_Cow $\sqsubseteq$ Animal $\sqcap$ $\forall$eats.($\neg$ Animal) and Cow $\sqsubseteq$ Animal $\sqcap$ $\forall$eats.($\neg$ Animal) will be lost, as we know the Animal $\sqcap$ $\forall$eats.($\neg$ Animal) in α_4 is not responsible for Mad_Cow's unsatisfiability. Similarly, if α_4 is removed, the indirect assertion Vegetarian $\sqsubseteq$ Animal $\sqcap$ $\forall$eats.($\neg$ Animal) and the above two assertions will be all lost. (2) *the satisfiable concepts irrelevant to the unsatisfiability*: Other named concepts irrelevant to the satisfiability might lose entailments introduced by the axiom to be changed. The entailments we consider in this case are indirectly asserted in the ontology before

---

[5] http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/mad_cows.owl

the change. If the user removes the problematic part $\forall$eats.($\neg\exists$ part_of.Animal) from α_4, then all the subconcepts of Vegetarian will be affected. The indirect assertion all giraffes only eat something which is not part of an animal inherited from Vegetarian will be lost. Cow, which is involved in the unsatisfiability, is not considered here, as the assertion all cows only eat something which is not part of an animal will still make Mad_Cow unsatisfiable; (3) *the classification of the named concepts of the ontology*: The satisfiable named concepts might lose implicit subsumption relations due to the change of axioms. We run classification on the example, and find that Sheep is subsumed implicitly by Vegetarian due to axioms $\alpha_2, \alpha_4, \alpha_5, \alpha_6$. The change of α_4 might also remove the inferred subsumption between Sheep and Vegetarian.

We now deal with each of the above three cases.

4.1   Impact on Named Concepts Involved in the Unsatisfiability

We first describe how to calculate the impact of the removal of parts of axioms on the concepts involved in the unsatisfiability. In the following we use an example to explain how to find entailments, which are not responsible for the concept unsatisfiability in the ontology, by analysing the sequences of the clashes of an unsatisfiable concept.

Our idea is to search for any element which exists in the fully expanded tree but not in the sequences of the clashes. In Example 1, if C in axiom α_1 is going to be removed, then we have to calculate the lost entailments of A which are not responsible for A's unsatisfiability. Figure 2 shows the nodes and the sequences of the clash in A. We find that $(a : C, \{1\}, a : \forall R.B)$ exists in Seq^- (cf. 1 in Figure 2). We search for elements in the tree whose second concept assertion is $a : C$, but which do not exist in Seq^- or Seq^+. $(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C)$ matches $a : C$ but exists in Seq^- (cf. 2), so we keep searching for other elements whose second concept assertion is $a : \exists R.\neg B \sqcap B$. The matched elements are $(a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B)$ which exists in Seq^- (cf. 3) and $(a : B, \{1, 2\}, a : \exists R.\neg B \sqcap B)$ (cf. 4) which does not exist in Seq^- or Seq^+, and has the same individual as $(a : A, \emptyset, nil)$. This means, B is a superconcept of A, and hence, the lost entailment is $A \sqsubseteq B$.

Assume that A is an unsatisfiable concept, and α_i is involved in its unsatisfiability, and there exists a clash in node x in the fully expanded tree. When a concept C in α_i is to be removed, we can calculate the lost entailments of A with the algorithm shown in Figure 5.

4.2   Impact on Satisfiable Concepts Irrelevant to the Unsatisfiability

We now describe how to calculate the impact on named concepts irrelevant to the unsatisfiability. Note that when a concept is unsatisfiable, it is trivially a subconcept of all satisfiable concepts and equivalent to all unsatisfiable concepts. If an axiom $C \sqsubseteq D$ is removed, any named concept in other axioms, which refers to C, will lose entailments introduced by this axiom. In general we lose $X \sqsubseteq Y$ where X is a subconcept of C and Y is a superconcept of D. Continuing the

Given: an unsatisfiable A, the sequences of Seq^+, Seq^- of a clash,
the label of node x is $\mathcal{L}(x)$, and C is to be removed from α_i

1. let a be the individual of the last element of the Seq^+;
2. $setEle := \{\}$; $lostEnt := \{\}$;
3. $roleSeq := \langle\rangle$;
4. $ele := SearchSequence((- : C, -, -), Seq^+)$, where $ele = (a' : C, -, -)$
                //search for the element in the sequence whose first concept is C
5. if $(ele \,!= \text{null})$ then $Seq := Seq^+$;
6. else $ele := SearchSequences((- : C, -, -), Seq^-)$, where $ele = (a' : C, -, -)$
7. $Seq := Seq^-$;
8. $\mathcal{S} := SearchElement((-, -, a' : C), \mathcal{L}(x), setEle)$
9. for each $\varepsilon \in \mathcal{S}$, where $\varepsilon = (a_1 : D_1, -, -)$
10.   if $(a = a_1)$, then $lostEnt := lostEnt \cup \{A \sqsubseteq D_1\}$;
11.   else $roleSeq := SearchRoleSeq((a' : C, -, -), Seq, roleSeq, a_1)$;
12. $lostEnt := lostEnt \cup \{createSubsumption(A, roleSeq, D_1)\}$;
             //$createSubsumption$ creates a subsumption relationship for A,
             //e.g., if $roleSeq = \langle \forall R, \exists R \rangle$, then $A \sqsubseteq \forall R.(\exists R.D_1)$ is created.
13. return $lostEnt$;

14. subroutine: $SearchElement((-, -, a' : C), \mathcal{L}(x), setEle)$
15. $\mathcal{S} := search((-, -, a' : C), \mathcal{L}(x))$;
                //search for elements in $\mathcal{L}(x)$ whose second concept is C
16. for each $\varepsilon \in \mathcal{S}$, where $\varepsilon = (b : D_1, -, a' : C)$
17.   if ε exists in Seq^+ or Seq^-, then
18. $setEle := setEle \cup SearchElement((-, -, b : D_1), \mathcal{L}(x), setEle)$
19.   else $setEle := setEle \cup \{\varepsilon\}$;
20. end for
21. return $setEle$;

22. subroutine: $SearchRoleSeq((a' : C, -, -), Seq, roleSeq, a_1)$
23. $\varepsilon := searchSuccessor((a' : C, -, -), Seq)$, where $\varepsilon = (a' : -, -, b : E)$, $a' \neq b$
        //search for the first element succeeding $(a' : C,-,-)$
        //in the Seq with different individuals
24. if $(\varepsilon = \text{null})$, then
25. $\varepsilon := searchPredecessor((a' : C, -, -), Seq)$, where $\varepsilon = (b : E, -, a' : -)$, $a' \neq b$
        //search for the first element preceding $(a' : C,-,-)$
        //in the Seq with different individuals
26. if E of the form $\forall R.-$, then
27. $roleSeq := roleSeq \cdot \langle \forall R \rangle$;   // $\cdot$ means to append an element to a sequence
28. else $roleSeq := roleSeq \cdot \langle \exists R \rangle$;
29. if $(a_1 = b)$, then   return $roleSeq$;
30. else   return $SearchRoleSeq((b : E, -, -), Seq, roleSeq, a_1)$;

Fig. 5. Algorithm for Finding Lost Entailments

mad_cow example, when the problematic part $\forall$ eats.($\neg\exists$ part_of.Animal) from α_4
is removed, all the subconcepts of Vegetarian which are not responsible for the
unsatisfiability will be affected. It is obvious that the lost entailment of Giraffe
is Giraffe $\sqsubseteq \forall$ eats.($\neg\exists$ part_of.Animal).

For those named concepts which refer to a concept to be removed not just via subsumption relations, the lost entailments cannot be as easily obtained as above. For the mad_cow example, if α_4 is changed to be Vegetarian $\doteq$ $\forall$eats.Plant $\sqcap$ Animal $\sqcap$ $\forall$eats.($\neg\exists$part_of.$\bot$), then we cannot say the lost entailment is Vegetarian $\sqsubseteq$ $\forall$eats.($\neg\exists$part_of.Animal), because the definition of Vegetarian still implies that it only eats part of anything, which includes $\neg$Animal.

The lost entailment of such concepts can be computed by calculating the difference between the original and modified concepts. To do this we adapt the notion of the "*difference*" operator between concepts which is defined in [24]. The difference between C and C' (1) contains enough information to yield the information in C if added to C', i.e., it contains all information from C which is missing in C', and (2) is maximally general, i.e., it does not contain any additional unnecessary information.

Definition 3 (Difference of Concepts). *Let C and C' be the original and modified concept expressions, the difference between C and C', which is a set of concepts, is defined as*

$$difference(C, C') = \begin{cases} max_{\sqsupseteq}\{E | E \doteq C \sqcup \neg C'\} & if\ C \sqsubseteq C', \\ max_{\sqsupseteq}\{E | E \doteq C' \sqcup \neg C\} & if\ C' \sqsubseteq C \end{cases}$$

For Example 1, if concept $\exists R.\neg B$ in axiom α_2 is removed, then the modified axiom becomes $C \sqsubseteq B$. As α_3 refers to C, the lost entailment of G will be $\forall R.(\exists R.\neg B \sqcap B \sqcap F) \sqcup \neg\forall R.(B \sqcap F)$, i.e., $\forall R.(\exists R.\neg B) \sqcup \neg\forall R.(B \sqcap F)$. The disadvantage of this calculation is that the representation of lost entailments could be too complicated for human users to understand, the simplification of such representations is therefore necessary. Brandt et al. [6] introduced a syntax-oriented difference operator, but the algorithm only supports the difference between an $\mathcal{ALC}$- and an $\mathcal{ALE}$-concept description. As $\mathcal{ALE}$ does not support disjunction concepts, their difference operator is not applicable to our approach. In the future work, the approaches to updating of DLs [14,8] can be borrowed.

4.3 Impact on the Classification

Besides deciding the satisfiability of concept expressions, description logic reasoners are able to compute the classification of an ontology. Classification is the process of determining the subsumption relationship between any two named concepts in an ontology; e.g., for A and B, it determines whether $A \sqsubseteq B$ and/or $B \sqsubseteq A$. Recall that reasoners decide subsumption relationships by reducing the problem to a satisfiability test (i.e., $A \sqcap \neg B$ is unsatisfiable if $A \sqsubseteq B$ holds). Whenever an axiom is changed, the classification of the ontology might be affected. In this paper we aim to point out to the user which parts of the classification will be affected if a certain change is made to the ontology. If the classification of the entire ontology must be checked after each change, then it will involve n^2 subsumption tests for n named concepts; moreover, each subsumption test (checking for satisfiability in $\mathcal{ALC}$ w.r.t general inclusion axioms)

is EXPTIME-complete [23]. It is impractical to run this classification test after each change made to the ontology. In this section, we will describe how we make use of the sequences of clashes (satisfiability test) to check if existing subsumption relations will be affected. If it is not affected, the subsumption test can be skipped.

Due to the monotonicity of the DLs we consider in this paper, removal of (part of) axioms cannot add new entailments, and will not change any previous non-subsumption relationships. Therefore, we only need to re-check if the removal of axioms will invalidate the previously found subsumption relationships. By building a tree with the application of the expansion rules on $A \sqcap \neg B$, we can obtain the sequences of the clashes. The elements in the sequences are the cause of the unsatisfiability, that is the subsumption relationship. With the sequences of clashes in the tree, we can analyse if a certain removal/change of (part of) an axiom will affect the current subsumption relation. Therefore, we are able to predict which subsumption relationships of named concepts will be affected, and skip the subsumption tests for the unaffected named concepts.

We check if a concept component of an axiom which is going to be removed will affect the previously found subsumption as follows:

Lemma 4. *Given a terminology $\mathcal{T}$ such that $\mathcal{T} \models A \sqsubseteq B$, where A and B are named concepts, and a fully expanded tree $\mathbf{T}$ of $A \sqcap \neg B$, the sequences of clashes in the tree are obtained. Let I_u be the union of the index-set of the first element in all of the sequences. Assume that a concept component C in α_i is going to be removed, where $\alpha_i \in \mathcal{T}$, the subsumption $A \sqsubseteq B$ is unaffected if either one of the following conditions hold:*

1. *$i \notin I_u$, α_i is not invovled in the unsatisfiability,*
2. *$(- : C, I, -)$ and $(- : C', I, -)$ do not exist in any sequences of clashes where $i \in I$, $i \in I_u$ and C' is a negated form of C*

Proof. The sequences of the clashes in $\mathbf{T}$ contain the concept components and sets of axioms which are relevant to the subsumption.

1. If an axiom α_i going to be changed does not exist in the index-set of any sequence of the clashes, i.e., $i \notin I_u$, then α_i is not involved in the unsatisfiability, any change of α_i does not affect the subsumption.
2. If α_i is involved in the unsatisfiability (i.e., $i \in I_u$), but $(- : C, I, -)$ and $(- : C', I, -)$, where $i \in I$, do not appear in any sequences of clashes, then they are not responsible for the clashes in the tree, and therefore not responsible for the subsumption. Since B is negated for the satisfiability test $(A \sqcap \neg B)$, we need to check the negated form of C as well. $\square$

We use the following example to illustrate how we make use of the sequences of clashes (satisfiability test between two named concepts) to detect if some change will affect a subsumption.

$\mathcal{L}(x) := \{(a : B \sqcap \neg A, \emptyset, nil), (a : B, \emptyset, a : B \sqcap \neg A), (a : \neg A, \emptyset, a : B \sqcap \neg A),$
$(a : E \sqcap F, \{2\}, a : B), (a : \neg D, \{1\}, a : \neg A), (a : E, \{2\}, a : E \sqcap F),$
$(a : F, \{2\}, a : E \sqcap F), (a : D, \{2, 3\}, a : E)\}$ *Clash*

        to be removed, not exists in Seq⁻ or Seq⁺

Clash:
$Seq^{+} := \langle (a : D, \{2, 3\}, a : E), (a : E, \{2\}, a : E \sqcap F),$
$(a : E \sqcap F, \{2\}, a : B), (a : B, \emptyset, a : B \sqcap \neg A), (a : B \sqcap \neg A, \emptyset, nil) \rangle$

$Seq^{-} := \langle (a : \neg D, \{1\}, a : \neg A), (a : \neg A, \emptyset, a : B \sqcap \neg A), (a : B \sqcap \neg A, \emptyset, nil) \rangle$

Fig. 6. Subsumption Test on $B \sqsubseteq A$

Example 5. Given a terminology with the following axioms, we check if $B \sqsubseteq A$.

α_1: $A \doteq D$
α_2: $B \doteq E \sqcap F$
α_3: $E \sqsubseteq D$

As seen in Figure 6, $B \sqsubseteq A$ holds, because a clash exists in the label of the root node, and so $B \sqcap \neg A$ is unsatisfiable. Assume the concept component F in α_2 is to be removed, we know that $B \sqsubseteq A$ still holds, because $(a : F, \{2\}, -)$ does not exist in either of the sequences of the clash.

5   Harmful and Helpful Changes

In this section we study ways of changing problematic axioms to resolve unsatisfiability. It should be noted that improperly rewriting a problematic axiom might not resolve the unsatisfiability, and could introduce additional unsatisfiability. It is important to help ontology modellers to make changes in order not to introduce unintended contradictions. For this purpose, we define *harmful* and *helpful* changes. Harmful changes either fail to resolve the existing unsatisfiability or introduce additional unsatisfiability. Helpful changes resolve the problem without causing additional contradictions, and restore some lost entailments.

5.1   Harmful Changes

Given an unsatisfiable concept A in $\mathcal{T}$, assume a concept E on the right-hand side of a problematic axiom α_i is chosen to be replaced by some other concept. We can find the harmful concepts for the replacement of E by analysing the elements in the sequences of the clashes of concept A.

Definition 5 (Harmful Change). *A change which transforms $\mathcal{T}$ to $\mathcal{T}'$ is harmful with respect to an unsatisfiable concept A in $\mathcal{T}$, if one of the following conditions holds:*

- *$\mathcal{T}' \models A \sqsubseteq \bot$, where $\mathcal{T}'$ is the changed ontology;*
- *if some named concept A_i which is satisfiable in $\mathcal{T}$ is not satisfiable in $\mathcal{T}'$. That is, $\mathcal{T} \nvDash A_i \sqsubseteq \bot$ and $\mathcal{T}' \models A_i \sqsubseteq \bot$, for some A_i in $\mathcal{T}$.*

The following lemma identifies the changes which are harmful due to the fact that they fail to resolve the existing unsatisfiability. To identify other harmful changes (which introduce additional unsatisfiability unrelated to the original problem), the whole ontology may have to be rechecked.

Lemma 6. *Assume a concept C on the right-hand of axiom α_i is to be rewritten. Given two sequences of a clash, Seq^+ and Seq^-, involving C, if one of the elements, ε, in Seq^+, is of the form $(a : C, I, -)$ and $i \in I$, then*

1. *All the concepts in the elements in Seq^+ preceding $(a : C, I, -)$, which contain the same individual as ε, are harmful for replacing C;*
2. *The negation of all the concepts in elements in Seq^-, which contain the same individual as ε, are also harmful, because these replacements still keep the unsatisfiability.*

The lemma is analogous for the element ε in Seq^-.

Proof. Assume that a concept C in axiom α_i is to be rewritten, and two sequences of a clash, Seq^+ and Seq^-, involving C, are obtained from a node of the tree **T**. In a sequence, for every two adjacent elements, ε_1 and ε_2, which are of the form $(a : E_1, -, a : E_2)$ and $(a : E_2, -, -)$, containing the same individual, the concept E_1 is a superconcept of E_2. This extends inductively to all elements preceding ε_1, i.e., they are all superconcepts of E_2.

1. If an element ε in Seq^+, which is of the form $(a : C, I, -)$ and $i \in I$, then the concepts in all the elements, which are preceding ε and contain individual a, are harmful for replacing C. This is because they are superconcepts which are involved in the clash.
2. The elements in Seq^- lead to a negated concept, which results in a contradiction. Hence, the negation of all the concepts in elements in Seq^-, which contain the same individual a, are also harmful.   □

In Example 1, if C in axiom α_1 is going to be replaced, we know that there exists an element $(a : C, \{1\}, a : C \sqcap \forall R.B)$ in Seq^- of the clash, then the harmful concepts for the replacements will be $\exists R.\neg B \sqcap B$, $\exists R.\neg B$, $\neg(\forall R.B)$, $\neg(C \sqcap \forall R.B)$, $\neg(C \sqcap \forall R.B \sqcap D)$, and $\neg A$. The first two items are from Seq^-, the rest are from negated elements in Seq^+.

5.2   Helpful Changes

If we know which concepts are harmful to replace a concept in a problematic axiom, then all the concepts which are not harmful are candidates for replacement. However, there are many possible candidates. Our aim is to find desirable concepts for replacement in order to minimise the impact of changes. To do this we introduce *helpful* changes which cover for the lost entailments due to the removal. When an axiom $A \sqsubseteq C$ in $\mathcal{T}$ is changed to be $A \sqsubseteq C'$ (where A is a named concept), this change is helpful if (1) C' can compensate for at least one lost entailment due to the removal of C, (2) the changes are not harmful, that

means all concepts which are satisfiable in $\mathcal{T}$ are also satisfiable in the changed ontology. Note that we only change concepts in the right-hand side of axioms. We now formally define a helpful change.

Definition 7 (Helpful Change). *A helpful change is defined as the removal of an axiom followed by an addition. Assume that $\mathcal{T}$ is the original ontology and an axiom α in $\mathcal{T}$ involved in the unsatisfiability of concept A is going to be removed, resulting in intermediate ontology $\mathcal{T}_1$. A new axiom is then added to $\mathcal{T}_1$, resulting in the changed ontology $\mathcal{T}'$. The change is helpful with respect to A, if the following conditions hold:*

1. *if Ω is the set of lost entailments in going from $\mathcal{T}$ to $\mathcal{T}_1$ (i.e., due to the removal of α), such that $\forall \gamma \in \Omega$, $\mathcal{T} \vDash \gamma$, then there exists $\beta \in \Omega$, such that $\mathcal{T}_1 \nvDash \beta$ and $\mathcal{T}' \vDash \beta$;*
2. *$\mathcal{T}' \nvDash A \sqsubseteq \bot$.*

Lemma 8. *Assume C on the right-hand side of a problematic axiom (involved in the unsatisfiability of A) is going to be replaced by C', the change is helpful if C' is a superconcept of C and is not involved in the clash of A.*

Proof. It is obvious that any concept which is not involved in the clash is not harmful as a replacement for C. We now prove its superconcepts are helpful. Given that in an axiom $\alpha : E \sqsubseteq C$ in $\mathcal{T}$, concept C is going to be replaced by its superconcept C'. We divide the change into two steps:

1. Remove C from α, the changed ontology $\mathcal{T}_1 = \mathcal{T} \setminus \{E \sqsubseteq C\}$;
2. Add C' to α, the final ontology $\mathcal{T}' = \mathcal{T}_1 \cup \{E \sqsubseteq C'\}$.

As C' is a superconcept of C, $E \sqsubseteq C$ is removed in $\mathcal{T}_1$, so the indirect subsumption relationships of A with C's superconcepts are also lost, that means $\mathcal{T}_1 \nvDash E \sqsubseteq C'$, but obviously, $\mathcal{T}' \vDash E \sqsubseteq C'$. □

Lemma 9. *Given two sequences of a clash w.r.t the unsatisfiability of A obtained from a fully expanded tree $\mathbf{T}$, assume a concept C on the right-hand side of axiom α_i is to be rewritten. C' is helpful as a replacement for C, if the following conditions hold:*

1. *there exist two elements e and e' in $\mathbf{T}$, which are $(a : C, I, -)$ and $(a : C', I', -)$, and no element of the form $(a : C', I', -)$ exists in either of the two sequences;*
2. *$I \subset I'$, the index-set of the element with concept C is a proper subset of that of the element with concept C'.*

Proof. We have to prove that (1) C' is a superconcept of C, this is a sufficient condition to ensure that the first requirement for helpfulness is met; and (2) C' is not involved in the clash.

1. If elements e and e' in **T** contain the same individual, then they have a subsumption relationship (i.e., C is either a subconcept or superconcept of C'). Additionally, if the index-set of the element e is a proper subset of the index-set of the element e', then that means e' is added to $\mathcal{L}(x)$ after the addition of e (i.e., the addition of e' is triggered by e). Then we can confirm that C' is a superconcept of C.
2. If an element, which is of the form $(a : C', -, -)$, exists in **T**, but not in either of the two sequences, then C' is not involved in the clash. □

Continuing with Example 1, assume C in axiom α_1 is going to be replaced, there exists an element $(a : C, \{1\}, a : C \sqcap \forall R.B)$ in Seq^- of the clash (see Figure 2), we find that the two elements $(a : D, \{1\}, \cdots)$ and $(a : B, \{1, 2\}, \cdots)$ do not exist in either of the sequences of the clash. However, the former element does not fulfill condition (2) in Lemma 9, because the index set of $(a : C, \{1\}, \cdots)$ is not a proper subset of $(a : D, \{1\}, \cdots)$. Hence, the only helpful concept for the replacement is the concept of the latter element, B, because B is a superconcept of A, but D is not.

Overall, the helpful changes include the replacements of a concept by its superconcepts not involved in any clash (see Lemma 9), and the lost entailments irrelevant to the unsatisfiability of the ontology (see Section 4). These changes are suggested to the user to add back to the ontology in order to minimise the impact of changes.

6   Evaluation

To demonstrate the effectiveness of our proposed approach, we have built a prototype. The implementation extends the Pellet[6]1.3 reasoner to support our fine-grained approach. In this section we describe a usability evaluation to evaluate the benefits of our approach; the result is compared with existing debugging tools. Next, we present the performance evaluation of our prototype using a set of satisfiability tests and comparing it with an existing DL reasoner.

6.1   Usability Evaluation

We created a plugin in Protégé 3.2 for repairing ontologies, called 'RepairTab'. Figure 7 shows our plugin displaying the problematic axioms of mad_cow, which is an unsatisfiable concept from the Mad_Cow ontology[7]. As can be seen, the parts of axioms responsible for the unsatisfiability are highlighted. The parts of axioms or whole axioms can be removed by striking out (cf. (A) in Figure 7). If the user decides to remove vegetarian from the axiom cow ⊑ vegetarian, the lost entailments of this removal can be previewed. The harmful and helpful changes are also listed. The user can choose to add the helpful changes to the ontology to minimise the impact of the removal (cf. (B)).
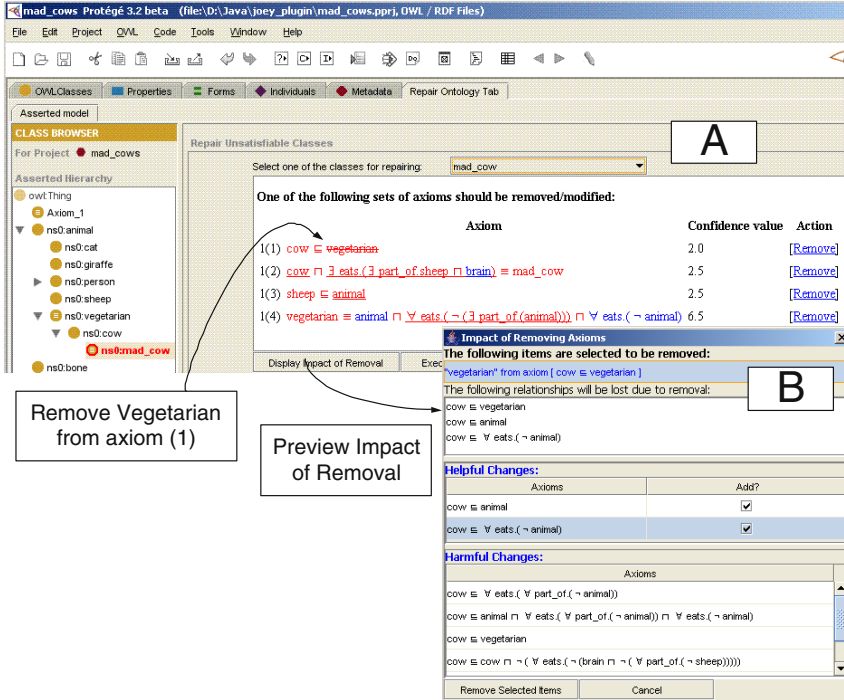
Fig. 7. (A) mad_cow is unsatisfiable. The problematic parts of the axioms are highlighted. (B) The lost entailments of the selected item can be previewed. The helpful and harmful changes are listed.

For the purpose of evaluation, we decided to compare RepairTab with OWLDebugger[8] and SWOOP[9]. OWLDebugger is another Protégé plugin, which provides explanations for unsatisfiable concepts. SWOOP is a stand-alone editor. We are interested in two main functionalities in SWOOP [12]: (1) explanations of unsatisfiability – it pinpoints the problematic axioms for unsatisfiable concepts, and is able to strike out irrelevant parts of axioms that do not contribute to the unsatisfiability; (2) ontology repair service – it displays the impact on ontologies due to the removal of axioms. When an axiom is removed, it shows the fixed and remaining unsatisfiable concepts, as well as the lost and retained entailments.

We conducted a usability-study with three ontologies and three groups of subjects. Fifteen subjects, who were postgraduate students in the Computing Science Department at the University of Aberdeen, were chosen for the evaluation. They had knowledge of OWL ontologies and Description Logics; they had experience of using both Protégé and SWOOP. None of the subjects had seen these ontologies before, and they were divided into three groups to debug

the same set of ontologies using one of these tools. Ideally, ontologies in our evaluation would fulfill the following conditions:

1. the expressivity of ontologies is in $\mathcal{ALC}$;
2. they should be interestingly axiomatised, i.e., containing axioms like disjointness, role restrictions, concept definitions, and so on, and should not be simply taxonomies;
3. the domain of the ontologies can be easily understood by subjects.
4. they are available on the Web and contain unsatisfiable concepts which could be difficult for non-expert users to debug.

The Mad_Cow[10], Bad-food.owl[11] and University.owl[12] ontologies available on the Web meet the requirements and were chosen for evaluation. Both the Mad_Cow.owl and Bad-food.owl ontologies each contain one unsatisfiable concept. University.owl was simplified into $\mathcal{ALC}$ format, and it contains 12 unsatisfiable concepts which were sorted based on the number and size of the MUPSs of the concepts. Our hypotheses for this usability study were:

1. The explanation function of RepairTab, which highlights parts of the axioms causing the unsatisfiability, helps users resolve the unsatisfiability. This is a relative advantage of RepairTab when compared with OWLDebugger and SWOOP.
2. The subjects using RepairTab will take less time to understand the source of errors and resolve them, compared with OWLDebugger and SWOOP.
3. RepairTab's list of lost entailments helps subjects decide which change(s) should be made in order to minimise the impact on the ontologies.
4. RepairTab's list of helpful changes provides useful (as rated by subjects) suggestions for subjects to add axioms back to the ontologies in order to minimise the impact of the changes on the ontologies.
5. RepairTab's list of harmful changes provides useful (as rated by subjects) guidance for subjects about which changes should not be made in order to prevent more unsatisfiable concepts being created.

The usability study was conducted as follows. Each subject was given a tutorial on the debugger they would use. A detailed walkthrough of the relevant explanation and debugging functions was given using a sample ontology. Groups A, B and C were assigned to RepairTab, OWLDebugger and SWOOP respectively. Each group was to resolve all unsatisfiable concepts in the three ontologies using their respective tools.

The subjects in the three groups were asked to answer a survey. For each task, they were asked if they understood the cause of the unsatisfiable concepts, which axioms were changed, and how many changes were made etc. At the end of the session, they were also asked to rate the usefulness of the tool used on a 5 point scale where 5 is 'very useful' and 1 corresponds to 'useless'. For Group A, the

---

[10] http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/mad_cows.owl
[11] http://www.mindswap.org/dav/ontologies/commonsense/food/foodswap.owl
[12] http://www.mindswap.org/ontologies/debugging/university.owl

subjects were also asked how useful the lost entailments, helpful and harmful changes facilities were, and how many helpful changes they had selected to add back to the three ontologies. For Group C, the subjects were also asked about the usefulness of the explanation and repairing functionalities provided by SWOOP. In addition, we also asked the subjects for their comments on the tool used, and how it could be improved. The time taken by each subject for resolving the unsatisfiability in each ontology was recorded. The modified ontologies were also recorded for analysis by the experimenter.

6.2   Analysis of Results

Table 3 shows the results for the three tools used by the subjects. We took the average of the times for each group to complete the tasks. As some tools do not provide certain functionalities, those ratings are not included in the table. As can be seen from the Table 3, firstly, the explanation function (i.e. highlighting the problematic parts of axioms) of RepairTab was rated to be more useful than SWOOP and OWLDebugger in two examples, but less useful on the Bad-food.owl ontology compared with OWLDebugger. Secondly, the subjects in Group A took less time to resolve the unsatisfiability than Group B; Group C had similar performance with Group A. Therefore, we cannot verify the first and second hypothesis currently. Both the lost entailments and helpful changes were rated to be useful overall, the ratings were in agreement with the third and fourth hypotheses. However, the harmful changes are less useful relatively, therefore the final hypothesis was falsified. We now analyse their performance for each ontology.

Table 3. Results of Debugging ontologies (A = RepairTab, B = OWLDebugger, C = SWOOP)

	Mad_Cow			Bad-food			University		
Group	A	B	C	A	B	C	A	B	C
Average Time Taken (in mins)	5	8.8	6.9	6.4	6.8	6.0	10.2	16.3	11.5
No. of subjects who understood the errors	5/5	4/5	3/5	3/5	2/5	3/5	0/5	0/5	0/5
Rating of Explanation Function	5	4	2.6	3.5	4.5	3.5	5	4	2.6
Rating of Lost & Retained Entailments (SWOOP)	-	-	3	-	-	4	-	-	5
Rating of Fixed & Remaining Unsat. Concepts (SWOOP)	-	-	3	-	-	3	-	-	5
Rating of Lost Entailments (RepairTab)	5	-	-	4	-	-	5	-	-
Rating of Helpful Changes (RepairTab)	5	-	-	4	-	-	4	-	-
Rating of Harmful Changes (RepairTab)	4	-	-	2.5	-	-	2.5	-	-

For Mad_Cow.owl, more subjects using RepairTab understood the errors in Mad_cow.owl than those using OWLDebugger or SWOOP. It is suggested this is because the problematic axioms of mad_cow were highlighted by RepairTab, and so the subjects understood the error quickly. However, it is difficult to resolve

the problem correctly. The subjects in Group A usually resolved the error by removing part of an axiom and then adding the helpful changes suggested by the plugin; the subjects in Group B had to explore changes to the definitions of concepts or add extra subconcepts for cow (e.g., to have Normal_Cow as a sibling of mad_cow), some also triggered additional unsatisfiable concepts. Two subjects in Group C failed to understand the cause of unsatisfiable mad_cow, because some irrelevant parts of axioms were not struck out, this led the subjects to think that the irrelevant parts were responsible for the unsatisfiability.

In the case of Bad-food.owl, we report two issues. Firstly, the times taken for this ontology were similar in all three tools; the subjects in Group B took relatively less time to debug this ontology than when they were debugging Mad_Cow.owl. Secondly, the explanation function of OWLDebugger was rated to be more useful than RepairTab or SWOOP. The following is our explanation for this observation. OWLDebugger explains the error was due to the disjoint axiom, when in fact all the axioms referring to this disjoint axiom are also causing the problem. As a result of OWLDebugger's recommendation most subjects immediately chose to remove this axiom without understanding the cause of the unsatisfiable concept; if they understood the precise reason they could instead have altered one component of an axiom referring to the disjoint axiom. Our plugin facilitates these fine-grained changes. However, it is not without its shortcomings: two subjects found it difficult to analyse the problematic axioms which were presented in the formal DL notation. This problem was pronounced with Bad-food.owl because the axioms are relatively complicated. Furthermore, the fine-grained approach was not applicable because all parts of the axioms are relevant to the unsatisfiability, and hence all were highlighted in red. This explanation given for this example is similar in SWOOP. As a result, the subjects using RepairTab or SWOOP found it difficult to understand the reason for the unsatisfiability and to decide which changes should be made.

For the University.owl, we report two issues. Firstly, the rating of usefulness of the explanation in RepairTab and OWLDebugger is higher than that of SWOOP. This is because, for the unsatisfiable concept Person, SWOOP strikes out the whole right-hand side of a problematic axiom; some subjects thought that the explanation was confusing. Secondly, the subjects using RepairTab and SWOOP took less time to complete the task than those using OWLDebugger. We suggest the following reasons: (1) RepairTab sorted the twelve unsatisfiable concepts in order of size of problematic axioms. The subjects were guided to debug the concept with the least problematic axioms first. SWOOP highlights the root and derived unsatisfiable concepts. When a concept was resolved, most of its subconcepts were resolved as well. However, the subjects in Group B had to explore each unsatisfiable concept one by one. (2) RepairTab and SWOOP provide previews of the impact of removal, but OWLDebugger does not. We noticed that two subjects in Group B removed the disjoint axioms which caused the problems. This removal causes many lost entailments, but the subjects did not realise this. On the other hand, the subjects in Group A or C were discouraged from this type of removal because they previewed the impact of removal. Three

subjects in Group A also changed their minds after exploring the consequences of different modifications (i.e., after seeing many lost entailments or more helpful changes provided). Helpful changes were usually added back to the ontology. The subjects in Group C tried to remove some axioms and preview the impact of the removal on the ontology. Some subjects chose to remove axioms which caused fewer lost entailments and more retained entailments. However, there is no functionality to add the lost entailments back to the ontology in SWOOP. In some cases, the displayed lost entailment is exactly the same as the axiom just removed by the subjects. Therefore, in comparison, RepairTab is able to minimise the impact on the ontologies in the case of removing parts of axioms, by providing helpful changes facility.

Interestingly, we found that some subjects claimed they understood the reasons for the unsatisfiability, but they simply deleted disjoint axioms or subclass-of relationships, particularly in University.owl. Therefore, we classified these subjects as not understanding the errors. We believe this ontology, which contains one of the most common OWL modelling errors, is very difficult for the subjects. In the case of Person in University.owl, none of the subjects realised that FrenchUniversity $\doteq$ $\forall$ offerCourse.Frenchcourse, the domain of offersCourse is University, FrenchUniversity is a subclass of University, therefore, University is defined as equivalent to owl.Thing implicitly, then Person which is disjoint with owl:Thing is unsatisfiable. Two subjects using SWOOP did not realise that SWOOP displays the implicit axiom University $\doteq$ owl:Thing, they removed the domain or disjoint axiom to resolve the problem. However, when some subjects in Group A were exploring the removal of FrenchUniversity $\doteq$ $\forall$ offerCourse.Frenchcourse, they discovered that a helpful change FrenchUniversity $\sqsubseteq$ $\forall$ offerCourse.Frenchcourse could be added back to the ontology, and they decided to make this change.

6.3   Overall Comments and Summary

We learnt some useful lessons based on the results of the study and the comments given by the subjects.

Group A − RepairTab. The subjects using RepairTab appreciated that the problematic parts of axioms are highlighted, this helped them to analyse the cause of errors. For the impact of change, for those subjects who understood the problems but had no idea what changes should be made, the impact of removal and suggested changes were rated to be very useful. On the other hand, for those subjects who already had an idea what changes should be made, the suggestions of changes were less useful; for example, if a subject wants to make complex changes, such as changing role restrictions or creating new concepts; our plugin does not support these changes. Also, the list of harmful changes was rated as 3 on average. This is because the subjects who understood the causes of problems, already knew what changes should not be made.

The overall comments on our plugin were that it is useful for resolving inconsistencies, but that the presentation of problematic axioms could be more user friendly, such as using natural language. Two subjects thought the presentation

of problematic axioms was too formal;they took longer to analyse the meaning of those axioms. For example, Protégé presents disjoint concepts in a 'Disjoints' table, but a disjoint axiom is presented in our plugin as '$C \sqsubseteq \neg D$'.

Group B – OWLDebugger. Most subjects thought the plugin was useful because it indicated which conditions contradict with each other; the clash information was also shown in quasi-natural language. Debugging steps were provided to suggest which concepts should be debugged. For example, subjects were pointed to debug CS_Student when the concept AIStudent was chosen to be debugged. However, sometimes it was not helpful because the explanation of the unsatisfiable concept was oversimplistic when the cause of the unsatisfiability was too complex to explain.

Group C – SWOOP. (i) *Explanation Function*: Two subjects thought the function was confusing, because in some cases, it does not strike out all of the irrelevant parts (mad_cow in Mad_Cow.owl is an example), sometimes, it strikes out the relevant parts of axioms. Person in University.owl is an example, in which the whole right-hand side of an axiom was struck out, this misled the subjects to think that the problematic axiom was not responsible for the unsatisfiability. This is because the implementation in the latest version of SWOOP 2.3 Beta 3 is incomplete with respect to the published algorithm [12]. Thus, this function only works in a few cases in our study.

(ii) *Repairing Function*: Most subjects thought the tool was useful because (1) it separates the root and derived unsatisfiable concepts, so that they can focus on only debugging the root ones; (2) it enables the user to try removing different axioms and preview the impact of removal before committing the change; (3) it displays the lost and retained entailments when axioms are removed. Additionally, in some cases, the tool provides (Why?) hyperlinks which explain why entailments are lost and retained. However, there is no explanation for the fixed and remaining unsatisfiable concepts.

However, two subjects thought that SWOOP's repair service is limited to the removal of the whole axioms, rather than changing certain parts of axioms. Removing whole axioms will unnecessarily cause additional information loss. For the Mad_Cow.owl example, one subject claimed the definition of mad_cow was modelled poorly. If the definition axiom of mad_cow is completely removed, then all information about mad_cow will be lost. This is not a desired change for the subject, though the ontology becomes satisfiable.

6.4   Performance Analysis

The non-determinism in the expansion rule (i.e., $\sqcup$-rule) results in poor performance of the tableau algorithm. Existing DL reasoners employ optimisation techniques. They have demonstrated that even with expressive DLs, highly optimised implementations can provide acceptable performance in realistic DL applications. For example, dependency directed backtracking is used to prune the search tree [9]. However, in our fine-grained approach, these techniques are no longer applicable, as we aim to detect all possible clashes by fully expanding

the tree. This will adversely affect the performance of the algorithm especially if there is extensive non-determinism in the ontology. Considering that the number of unsatisfiable concepts is relatively small compared to the total number of concepts in realistic ontologies, we believe it is practical to first check the consistency of ontologies using optimised reasoners to find the unsatisfiable concepts, and then run the fine-grained algorithm on those unsatisfiable concepts. For example, Sweet-JPL.owl[13] contains 1537 concepts, and one concept is unsatisfiable. Hence, our algorithm is then only applied to the unsatisfiable concept, instead of the whole ontology.

In this subsection we report an evaluation with a number of realistic ontologies. RepairTab, a plugin for Protégé, was implemented in Java. The tests were performed on a PC (Intel Pentium IV with 2.4GHz and 1GB RAM) with Windows XP SP2 as operating system. Benchmarking with real-life ontologies is obviously a convincing way to evaluate the quality of our approach. However, there is only a limited number of realistic ontologies that are both represented in $\mathcal{ALC}$ and unsatisfiable. We therefore constructed simplified $\mathcal{ALC}$ versions for a number of ontologies downloaded from the Internet. We then removed, for example, numerical constraints, role hierarchies and instance information. As some ontologies are satisfiable, we randomly changed them such that each change on its own lead to unsatisfiable concepts. For example, we added disjointness statements among sibling concepts, and introduced some common ontology modelling errors enumerated by [18]. Figure 8 (left-hand side) shows the average runtime (in seconds) of the satisfiability test of a set of ontologies. The brackets of the ontology names indicate the number of unsatisfiable concept tested. The execution time of our extended algorithm is increased by 15% on average compared with that of Pellet, because our algorithm aims to detect all possible clashes given that it requires a fully expanded tableau tree, while many optimisations are disabled. In the cases of Transportation.owl and Economy.owl, the running time for checking the satisfiability of 20 and 30 concepts is less than 0.6 second. The result shows that the performance of our algorithm is feasible in realistic ontologies which do not contain a large number of unsatisfiable concepts.

We also were interested in the GALEN ontology[14], which models medical terms and procedures. It contains over 2700 classes and about 400 GCIs. As its DL expressivity is $\mathcal{SHf}$, we constructed a simplified $\mathcal{ALC}$ version of it. Figure 8 (right-hand side) shows the average runtime (in seconds) of from 1 to 1000 satisfiability tests. Note that there is a large number of GCIs in the GALEN ontology, the optimised reasoner is able to eliminate non-determinism by absorbing them into primitive concept introduction axioms whenever possible ($\mathsf{CN} \sqsubseteq D$, where CN is a named concept, D is a concept description). Although the execution time of RepairTab dramatically increases with the number of unsatisfiable concepts with the same reason as the above, *absorption* is still applicable to our revised algorithm, because it is algorithm independent. Absorption is used to preprocess the ontology before the tableau algorithm is applied. For example, given two
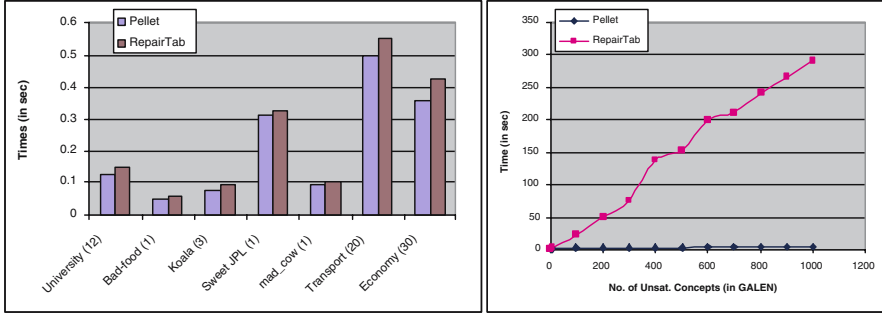
---

[13] http://www.mindswap.org/ontologies/debugging/buggy-sweet-jpl.owl
[14] http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl

Fig. 8. Performance Test of Pellet and RepairTab

axioms (1) $\mathsf{CN} \sqsubseteq \forall R.\neg C \sqcap \neg D$, (2) $\forall R.\neg C \sqsubseteq \neg \mathsf{CN} \sqcup D$. Axiom (2) will be absorbed as $\mathsf{CN} \sqsubseteq D \sqcup \exists R.C$, and can then be merged with axiom (1), the resulting axiom is $\mathsf{CN} \sqsubseteq (\forall R.\neg C \sqcap \neg D) \sqcap (D \sqcup \neg \exists R.C)$. We then apply the fine-grained algorithm to trace which parts of the axiom cause the unsatisfiability. However, the algorithm modifies the original axioms; in this example, two axioms are modified into one axiom, we can only tag the parts of the resulting axioms (modified by the algorithm) relevant to the unsatisfiability, instead of the original asserted axioms. In future work, to improve usability, it might be necessary to explain the correlation between the originally asserted axioms and the axioms (modified by the tableau algorithm) in a way that is understandable to the user.

7 Related Work

Several methods have been developed in the literature to deal with unsatisfiable ontologies. In this section, we first review three existing approaches to analysing unsatisfiable ontologies, and then describe two related fine-grained approaches used in debugging ontologies. Finally, we discuss the related work on resolving unsatisfiable concepts in ontologies.

7.1 Analysing Unsatisfiable Ontologies

We now describe three approaches to analysing unsatisfiable ontologies. One approach is to find maximally satisfiable sub-ontologies by excluding problematic axioms [4,15]. Another approach is to find minimal unsatisfiable sub-ontologies by pinpointing possible problematic axioms [22,21,10,13]. Finally there is the heuristic approach to explaining unsatisfiability [26].

Baader et al. [4] investigate the problem of finding the maximally satisfiable subsets of ABox assertions. In their approach, each element in the nodes in a tree is labeled with a propositional formula which indicates the sources of axioms, whereas Meyer et al. [15] use an index-set associated with every element of the label of nodes in a tree. The index-set is used to exclude axioms involved in the

unsatisfiability of concepts, so that maximally concept-satisfiable subontologies (so called MCSS) can be obtained.

Schlobach et al. [22,21] proposed to pinpoint the so called Minimal Unsatisfiability Preserving Sub-ontologies (MUPSs), which are sets of axioms responsible for an unsatisfiable concept. This is called *axiom pinpointing*. Roughly, a MUPS of a named concept contains only axioms that are necessary to preserve its unsatisfiability. They further exploit the minimal hitting-set algorithm described by Reiter [19] to calculate *diagnosis sets*, i.e minimal subsets of an ontology that need to be repaired/removed to make the ontology satisfiable.

Overall, the above approaches achieve the same result. A MCSS can be obtained by excluding the axioms in any one of the diagnosis sets. Moreover, they are only applicable to unfoldable terminologies $\mathcal{T}$; they simply remove problematic axioms from the ontology, and the support for rewriting the axioms is still limited.

Kalyanpur et al. [13,10] extended the axiom pinpointing technique (i.e., finding MUPS) to the more expressive description logic $\mathcal{SHOIN}$. They utilise a glass-box strategy for finding the first MUPS of an unsatisfiable concept. The description logic tableaux reasoner was modified to keep track the cause for the unsatisfiability of a concept, so that the minimal set of relevant axioms in the ontology that support the concept unsatisfiability was obtained. Their tool, SWOOP, also detects interdependencies between unsatisfiable concepts, in which root and derived unsatisfiable concepts are identified. The user can differentiate the root bugs from others which are caused by the root unsatisfiable concepts, and focus solely on the root concepts. This is a particularly effective approach to fixing a large set of derived unsatisfiable concepts. Then, they use a black-box approach, which is reasoner independent, to derive the remaining MUPSs from the first MUPS. Reiter's Hitting Set Tree (HST) algorithm [19] was adapted to find the remaining MUPSs [12]. The advantage of this approach is that it makes use of the optimisation techniques embedded in the reasoner. The disadvantage is that the complexity of generating the HST is exponential with the number of MUPSs. Their results showed the algorithm performed well in practice, because most of the satisfiability tests exhibited at most three or four MUPSs, with five to ten axioms each [11]. In comparison, RepairTab detects all MUPSs of an unsatisfiable concepts by fully expanding the tree. The disadvantage of our approach is that most optimisations are not applicable, however, its complexity is independent of the number of MUPSs.

Rector et al. [18] addressed some common problems of ontology users during the modelling of OWL ontologies. Based on these common errors, the authors developed a set of *heuristic rules* and incorporated them into a Protégé-OWL plugin. Their program is called OWLDebugger; it can detect commonly occurring error patterns in OWL ontologies. This alleviates the user from troubleshooting the unsatisfiable concepts. It helps users to track down the reasons for errors in OWL concepts. Quasi-natural language explanations for unsatisfiable OWL concepts are also generated. As the heuristic approach and pattern matching cannot determine the causes of the inconsistency in every case, the authors are

still investigating how to extend and improve the set of heuristic rules. However, the process of resolving problems is left to the user who has to run the reasoner frequently to check if consistency has been achieved.

7.2   Fine-Grained Approaches

Schlobach et al. [22] apply syntactic generalisation techniques to highlight the exact position of a contradiction within the axioms of the TBox. This is called *concept pinpointing*. Concepts are diagnosed by successive generalisation of axioms until the most general form which is still unsatisfiable is achieved. The main difference with our work is that in [22] the concepts in axioms are generalised and only these generalised axioms are shown to the user. For example, α_1: $A \sqsubseteq C \sqcap D \sqcap E$, α_2: $C \sqsubseteq \neg D \sqcap F$, then the generalised axioms $A \sqsubseteq C \sqcap D$, and $C \sqsubseteq \neg D$ are shown. It can be an additional burden on the user to correlate between the generalised axioms and the originally asserted axioms. In the case of very complicated axioms, the user might find it difficult to know which generalised axioms correspond to which of the original asserted axioms. Compared to our approach, we use a tracing technique to pinpoint problematic parts of axioms, the asserted axioms are directly displayed and faulty parts are highlighted.

Kalyanpur et al. [12] also propose a fine-grained approach, which determines which parts of the asserted axioms are responsible for the unsatisfiability of concepts. Their idea is to rewrite the axioms in an ontology in a normal form and split up conjunctions in the normalised version, e.g., $A \sqsubseteq \exists R.(C \sqcap D)$ is rewritten as $A \sqsubseteq \exists R.E$, $E \sqsubseteq C$, $E \sqsubseteq D$ and $C \sqcap D \sqsubseteq E$. In comparison, we achieve the same results as their approach, but we identify the irrelevant parts of axioms by making use of the tableaux algorithm, instead of splitting the axioms.

7.3   Resolving Unsatisfiable Concepts

Few approaches have been proposed which address the strategies for resolving unsatisfiable concepts. Plessers et al. [17] propose a set of rules to rewrite axioms to resolve the detected inconsistencies. They weaken restrictions either by removing an axiom, replacing it with its superconcepts, or changing its cardinality restriction values.

On the other hand, in SWOOP[13], the rewriting axioms suggestions are provided, but these suggestions are limited to a small number of common errors patterns. Moreover, the common error patterns may only apply for those ontologies built by non-expert users, it is insufficient to cover other applications, such as ontology merging/integration. The lost information due to their suggestions for rewriting axioms is also not considered. For example, an intersection concept $C \sqcap D$ is suggested to be changed as $C \sqcup D$, the modified concept is more generic, and hence certain information is lost.

Furthermore, Kalyanpur et al. [13] analyse the impact on an ontology when a whole axiom is removed. Currently, they only consider the subsumption/ disjointness between two named concepts (i.e., $A \sqsubseteq B$) and an instantiation

(i.e., $B(a)$) which will be lost due to axiom removal. The difference with our work is the following: (1) the lost entailments we consider in Section 4 which are not responsible for concepts' unsatisfiability can be added back to the ontology, whereas this feature is not available in their approach. (2) we calculate the lost entailments of named concepts when a part of an axiom or a whole axiom is removed; they only consider the impact when a whole axiom is removed. (3) we adapt the "difference" operator to calculate the lost entailment of a concept (see Section 4.2); their lost entailment is limited to subsumption/disjointness between two named concepts and instantiations. (Continuing our mad_cow example in Example 4, if α_4 is removed, their lost entailment is Cow $\sqsubseteq$ Animal.[15] See Section 4 to compare with our results).

8   Conclusion

In this paper we have proposed a fine-grained approach to rewriting problematic axioms in an ontology, by revising the classical tableaux algorithm. Our technique not only identifies the problematic axioms, but also captures which parts of the axioms are responsible for the unsatisfiability of concepts. Moreover, we present methods for finding harmful and helpful changes for concepts which are going to be replaced. With our approach, users are provided with support to help them to: (1) understand the reasons for the unsatisfiability of concepts, and (2) rewrite axioms in order to resolve the problems with minimal impact on the ontology. The results of our usability evaluation have demonstrated the applicability of our approach in practice. The plugin which we have developed, RepairTab, is very useful for ontology users who want to diagnose problematic axioms at a fine-grained level and achieve satisfiable ontologies. In future work, we plan to extend our algorithms to support more expressive Description Logics.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
2. Baader, F., Nutt, W.: Basic description logics. The Description Logic Handbook: Therory, Implementation, and Applications
3. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. Artif. Intell. 88(1-2), 195–213 (1996)
4. Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. Autom. Reasoning 14(1), 149–180 (1995)
5. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.-J., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems or "making KRIS get a move on". In: International Conference on the Principles of Knowledge Representation and Reasoning, San Mateo, pp. 270–281. Morgan Kaufmann, San Francisco (1992)

---

[15] The result is obtained from the latest version of SWOOP 2.3 Beta3.

6. Brandt, S., Küsters, R., Turhan, A.-Y.: Approximation and difference in description logics. In: KR 2002. Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning, pp. 203–214. Morgan Kaufmann, San Francisco (2002)

7. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. Journal of Artificial Intelligence Research 1, 109–138 (1993)

8. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proceedings of the 21st National Conference on Artificial Intelligence, pp. 1271–1276. AAAI Press, Stanford (2006)

9. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. PhD thesis, University of Manchester (1997)

10. Kalyanpur, A., Parsia, B., Grau, B.C., Sirin, E.: Justifications for entailments in expressive description logics. Technical report, University of Maryland (January 2006)

11. Kalyanpur, A.: Debugging and Repair of OWL Ontologies. PhD thesis, Dept. of Computer Science, University of Maryland (2006)

12. Kalyanpur, A., Parsia, B., Cuenca-Grau, B.: Beyond asserted axioms: Fine-grain justifications for OWL-DL entailments. In: DL 2006. International Workshop on Description Logics (June 2006)

13. Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B.: Repairing Unsatisfiable Concepts in OWL Ontologies. In: ESWC 2006. Proceedings of the Third European Semantic Web Conference (June 2006)

14. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating description logic ABoxes. In: KR. Proceedings of International Conference of Principles of Knowledge Representation and Reasoning, pp. 46–56 (June 2006)

15. Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic ALC. In: AAAI 2006. Proceedings of the 21st National Conference on Artificial Intelligence (July 2006)

16. Nebel, B.: Reasoning and Revision in Hybrid Representation Systems. Springer, Heidelberg (1990)

17. Plessers, P., De Troyer, O.: Resolving inconsistencies in evolving ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 200–214. Springer, Heidelberg (2006)

18. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS (LNAI), vol. 3257, pp. 63–81. Springer, Heidelberg (2004)

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

20. Schaerf, A.: Reasoning With Individuals in Concept Languages. Data and Knowledge Engineering 13(2), 141–176 (1994)

21. Schlobach, S., Huang, Z., Cornet, R.: Inconsistent ontology diagnosis: Evaluation. SEKT Deliverable 3.6.2, University of Karlsruhe (January 2006)

22. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: IJCAI 2003. 8th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco (2003)

23. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artifical Intelligence 48(1), 1–26 (1991)

24. Teege, G.: Making the difference: A subtraction operation for description logics. In: KR 1994. 4th International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, San Francisco (1994)
25. Uschold, M., Gruninger, M.: Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review (1996)
26. Wang, H., Horridge, M., Rector, A., Drummond, N., Seidenberg, J.: Debugging OWL-DL Ontologies: A heuristic approach. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 745–757. Springer, Heidelberg (2005)