Finding Maximally Satisfiable Terminologies for the Description Logic $\mathcal{ALC}$

Thomas Meyer and **Kevin Lee** *
National ICT Australia and
University of New South Wales
Kensington, 2052, Australia
{tmeyer,kevinl}@cse.unsw.edu.au

Richard Booth
Faculty of Informatics
Mahasarakham University
Mahasarakham 44150, Thailand
richard.b@msu.ac.th

Jeff Z. Pan
Department of Computing Science
The University of Aberdeen
jpan@csd.abdn.ac.uk

Abstract

For ontologies represented as Description Logic Tboxes, optimised DL reasoners are able to detect logical errors, but there is comparatively limited support for resolving such problems. One possible remedy is to weaken the available information to the extent that the errors disappear, but to limit the weakening process as much as possible. The most obvious way to do so is to remove just enough Tbox sentences to eliminate the errors. In this paper we propose a tableau-like procedure for finding maximally concept-satisfiable terminologies represented in the description logic $\mathcal{ALC}$. We discuss some optimisation techniques, and report on preliminary, but encouraging, experimental results.

Introduction

Description Logics (DLs) are widely accepted as an appropriate class of knowledge representation lanaguages to represent and reason about ontologies. Reasoners that perform satisfiability and consistency checking have grown increasingly powerful and sophisticated in the last decade. Optimised reasoners such as RACER (Haarslev & Möller 2001), FaCT (Horrocks 1998) and FaCT++ (Tsarkov & Horrocks 2004) are able to handle reasonably large and complex ontologies. However, there is comparatively little research effort that has gone into resolving logical errors in ontologies. Clearly the development of debugging tools to help rectify such errors will greatly facilitate the construction of high-quality ontologies.

Approaches to resolving such errors can roughly be divided into two categories. One approach is to identify possible sources of the problem and then leave it up to the modeller to rectify them. This approach typically involves the pinpointing of the possible problematic statements. Examples of this include the work of (Kalyanpur, Parsia, & Sirin 2005; Kalyanpur *et al.* 2005) and (Schlobach & Cornet 2003). A second, more proactive approach is to suggest possible resolutions to the problem, obtained by weakening the available information to the extent that the errors are eliminated. The most obvious way to do this is to remove just enough sentences to eliminate all errors. Examples of this include the work of (Baader & Hollunder 1995), as well as (Schlobach 2005) which is based on model-based diagnosis (Reiter 1987), and that of (Meyer, Lee, & Booth 2005).

The approach we follow in this paper falls into the second category. We propose a basic tableau-like algorithm for identifying the maximally concept-satisfiable subterminologies of an unfoldable terminology represented in the description logic $\mathcal{ALC}$. We discuss some optimisations and report on preliminary, but encouraging, experimental results.

Description Logics

The logic of interest to us in this paper is the well-known DL $\mathcal{ALC}$ (Schmidt-Schauß & Smolka 1991). We don't provide a formal introduction to DLs, but rather point the reader to (Baader & Nutt 2003). A DL knowledge base (Γ, Ω) consists of two finite and mutually disjoint sets. A *Tbox* Γ which introduces the *terminology*, and an *Abox* Ω which contains facts about particular objects in the application domain. Tbox statements have the form $C \doteq D$ (*equalities*) where C and D are (possibly complex) concept descriptions. The Abox contains statements of the form $a{:}C$ where C is a concept and a is an individual name, and *role assertions* which we don't need to elaborate on in this paper. A DL interpretation I contains a domain and a mapping which interprets a concept C as a subset C^I of the domain. An interpretation I is a model of a Tbox axiom $C \doteq D$ iff $C^I = D^I$. Given a Tbox Γ and a concept name A, Γ is A-*satisfiable* iff there is a model I of Γ such that $A^I \neq \emptyset$. Γ is *concept-satisfiable* iff it is A-satisfiable for every concept name A occurring in Γ. Γ is *unfoldable* iff the left-hand side of every $\gamma \in \Gamma$ contains a concept name A, there are no other γs with A on the left-hand side, and the right-hand side of γ contains no direct or indirect references to A (i.e. there are no cyclic definitions).

Calculating A-MSSs and MCSSs

In this section we describe a specialised tableau-like algorithm for finding the maximally A-satisfiable subsets (A-

*National ICT Australia is funded by the Australia Government's Department of Communications, Information and Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program. It is supported by its members the Australian National University, University of NSW, ACT Government, NSW Government and affiliate partner University of Sydney.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

MSSs) of an *unfoldable* terminology Γ represented in $\mathcal{ALC}$, for any concept name A occurring in Γ. A subset Γ' of Γ is an A-*MSS* of Γ iff it is A-satisfiable, and every Γ'' such that $\Gamma' \subset \Gamma'' \subseteq \Gamma$ is A-unsatisfiable. The algorithm generates a tree $\mathcal{T}$ in much the same style as classical $\mathcal{ALC}$ tableau algorithms, and with similar expansion rules. But instead of closing a branch when it detects a clash in a leaf node, it employs an additional non-deterministic expansion rule which breaks clashes by excluding axioms involved in the clash. As a result, we can show that the set of Tbox axioms *not* excluded in the process of getting to a fully expanded (and hence clash-free) leaf node is A-satisfiable, and indeed, that every A-MSS of Γ can be obtained from some fully expanded leaf node in this way. The algorithm can also be used to find the maximally concept-satisfiable subsets (MCSSs) of Γ. A subset Γ' of Γ is an *MCSS* of Γ iff it is concept-satisfiable, and every Γ'' such that $\Gamma' \subset \Gamma'' \subseteq \Gamma$ is concept-unsatisfiable.

We assume that Γ contains the *defined* concept names $A_1, \ldots, A_n$ with $n \geq 1$ (those concept names occurring on the left-hand sides of the axioms), and that it contains the *primitive* concept names $B_1, \ldots, B_m$, with $m \geq 1$ (those concept names occurring only on the right-hand sides of the axioms). It is easily established that Γ is B_i-satisfiable for $i = 1, \ldots, m$. For the rest of the paper we let $\Gamma = \{ax_1, \ldots, ax_n\}$, with ax_i referring to the axiom $A_i \doteq C_i$ for $i = 1, \ldots, n$. We assume that, during expansion, all concepts are converted to negation normal form. As is the case for some classical tableau algorithms, every node x of the tree $\mathcal{T}$ is labelled with a set of concept assertions. Additionally, we also associate with every such concept assertion $a{:}C$ a set of integers I in the range $1, \ldots, n$, called an *index-set*. The purpose of I is to maintain information about the axioms that were used in order for $a{:}C$ to be associated with the node x. So, each x is labelled with a set $L(x)$ containing elements of the form $(a{:}C, I)$, where C is a concept, a is an individual name, and I is an index-set. We frequently abuse notation by referring to elements of the form $(a{:}C, I)$ as concept assertions. We associate with each node x an *exclusion-set* $E(x)$ containing indices in the range $1, \ldots, n$. $E(x)$ contains the indices of the axioms that are excluded when applying the additional clash-breaking expansion rule mentioned above. It is from the information contained in the exclusion-sets of the leaf nodes of $\mathcal{T}$ that we will construct the A_i-MSSs of Γ.

Let us now assume that we want to determine the A_j-MSSs of Γ for some $j = 1, \ldots, n$. The first step is to create a root node r for $\mathcal{T}$, with $L(r) = \{(a{:}A_j, \emptyset)\}$ and $E(r) = \emptyset$. We shall refer to $\mathcal{T}$ as a tree for the DL knowledge base $(\Gamma, \{a{:}A_j\})$. Then we repeatedly expand the tree $\mathcal{T}$ by applying the six rules in Figure 1 to the leaf nodes. A node is fully expanded when none of the rules can be applied to it. The tree $\mathcal{T}$ is fully expanded when all of its leaf nodes are fully expanded.

The rules need some explanation and clarification. Rules 1 and 2 perform lazy unfolding. The D^+-rule adds the definition C_i of a defined concept name A_i, as prescribed by the axiom $A_i \doteq C_i$, then adds i to the index-set of C_i to indicate that this occurrence of C_i is due to an application of axiom i, and then tags A_i to indicate that the D^+-rule has already been applied to it. The D^--rule is involved in the same type of replacement, but is applied to the concept $\neg A_i$ to add $\neg C_i$.

1. D^+-rule	If $(a{:}A_i, I)$ is in $L(x)$ and has not been tagged, then Tag $(a{:}A_i, I)$ and let $L(x) := L(x) \cup \{(a{:}C_i, I \cup \{i\})\}$
2. D^--rule	If $(a{:}\neg A_i, I)$ is in $L(x)$ and has not been tagged, then Tag $(a{:}\neg A_i, I)$ and let $L(x) := L(x) \cup \{(a{:}\neg C_i, I \cup \{i\})\}$
3. $\sqcap$-rule	If $(a{:}C \sqcap D, I) \in L(x)$ then $L(x) := L(x) \setminus \{(a{:}C \sqcap D, I)\} \cup \{(a{:}C, I), (a{:}D, I)\}$
4. $\sqcup$-rule	If $(a{:}C \sqcup D, I) \in L(x)$ then Create two children y and z of x; $L(y) := L(x) \setminus \{(a{:}C \sqcup D, I)\} \cup \{(a{:}C, I)\}$; $L(z) := L(x) \setminus \{(a{:}C \sqcup D, I)\} \cup \{(a{:}D, I)\}$; $E(y) := E(x)$; $E(z) := E(x)$
5. $\exists$-rule	If $(a{:}\exists R.C, I) \in L(x)$ and rules 1-4 can't be applied then $X := \{(b{:}C, I)\} \cup \{(b{:}D, I \cup J) \mid (a{:}\forall R.D, J) \in L(x)\}$ (b is a new unique individual name not used before) $L(x) := (L(x) \setminus \{(a{:}\exists R.C, I)\}) \cup X$;
6. $\bot$-rule	If $(a{:}A, I) \in L(x)$ and $(a{:}\neg A, J) \in L(x)$ then For every $i \in I \cup J$ do: Create a new child y of x; $L(y) := L(x) \setminus \{(b{:}D, K) \mid i \in K\}$; $E(y) := E(x) \cup \{i\}$

Figure 1: The six expansion rules of our algorithm.

Rules 3 and 4 are similar to the classical $\sqcap$- and $\sqcup$-rules for $\mathcal{ALC}$. The main difference is that our rules allow for duplicate concept assertions (with different associated index-sets) to occur. This is necessary to ensure the completeness of the algorithm (i.e. that all A_j-MSSs will be identified) as demonstrated in Example 1.

Rule 5 is similar to a combination of the classical $\exists$- and $\forall$-rules for $\mathcal{ALC}$. Given the concept assertion $(a{:}\exists R.C, I)$, it removes $(a{:}\exists R.C, I)$ from $L(x)$ and creates a new concept assertion $(b{:}C, I)$ where b is a new individual name. In addition, for every concept assertion of the form $(a{:}\forall R.D, J)$, it creates a new concept assertion $(b{:}D, I \cup J)$. Note that the index-set J associated with every such $b{:}D$ is enlarged to contain the index set I associated with $\exists R.C$ as well. This is necessary to ensure that all A_j-MSSs are found. It is included to deal with cases where, for example, a node contains the concept assertions $a{:}\forall R.D$, $a{:}\exists R.C$ and $a{:}\forall R.\neg D$, where a direct clash will be detected only between the two $\forall$-concept assertions, even though it is because of the presence of the $\exists$-concept assertion that the clash occurs at all. Example 2 provides more details about such cases.

Observe that rule 5 may only be applied when rules 1-4 cannot be applied. This is to avoid situations where a concept assertion of the form $(a{:}\forall R.D, J)$ appears in $L(x)$ only after the rule has been applied to a concept assertion of the form $(a{:}\exists R.C, I)$. For example, suppose that $L(x)$ contains $(a{:}\exists R.C, I)$ and $(a{:}(\forall R.D) \sqcap E, J)$. If the $\exists$-rule is now applied before the $\sqcap$-rule, the set X calculated as part of the rule will not include $(b{:}D, I \cup J)$, as it should. Note that it is only rules 1-4, and not rule 6, that have to be applied before the $\exists$-rule. So it is permissible to apply rule 5 even if rule 6 is applicable at the same time.

Rule 6 is the new non-deterministic rule added to break clashes. The intended use of the rule is easy to understand. Whenever a *clash* is detected in a node x (i.e. a concept assertion $(a{:}A, I)$ and its negation $(a{:}\neg A, J)$ both occur in $L(x)$), the idea is to branch by first excluding $(a{:}A, I)$ and then excluding $(a{:}\neg A, J)$, thereby resolving the clash. The exclusion of a concept assertion actually amounts to the exclusion of the Tbox axiom responsible for the concept assertion being in $L(x)$. The index-set I associated with $a{:}A$ contains the information of which axiom is responsible for $a{:}A$ being in $L(x)$. In general, more than one axiom may bear this responsibility. This is reflected by the fact that I is an index-set. We therefore have to branch on each of the indices in I. The same argument goes for $(a{:}\neg A, J)$ as well. So, what the $\bot$-rule does is to create a new child y of x for every index i occurring in one of I or J. In line with the understanding that node y corresponds to the case where axiom i is excluded, y is labelled only with the concept assertions in $L(x)$ whose index-sets do not contain i, and the exclusion-set $E(y)$ is obtained by adding the index i to the indices in $E(x)$.

We can obtain the A_j-MSSs of Γ from the leaf nodes of a fully expanded $\mathcal{T}$ as follows. For any set of indices X, let $\Gamma(X) = \{A_i \doteq C_i \in \Gamma \mid i \in X\}$. So $\Gamma(X)$ contains those axioms in Γ whose indices occur in X. For every leaf node x, let $\Delta_x = \{1, \ldots, n\} \setminus E(x)$, and $\Delta_{\mathcal{T}} = \{\Delta_x \mid x$ is a leaf node of $\mathcal{T}\}$. So $\Gamma(\Delta_x)$ contains the Tbox axioms *not* identified for exclusion in x. We abuse notation slightly to let $\Gamma(\Delta_{\mathcal{T}})$ denote the set $\{\Gamma(\Delta_x) \mid \Delta_x \in \Delta_{\mathcal{T}}\}$. The maximal elements of $\Gamma(\Delta_{\mathcal{T}})$ are the A_j-MSSs of Γ.

Examples

The first example demonstrates a simple application of the rules, and shows why it is necessary to maintain duplicate concept assertions with different associated index-sets.

Example 1 *Let* $\Gamma = \{A_1 \doteq A_2 \sqcap A_3 \sqcap \neg A,\ A_2 \doteq A,\ A_3 \doteq A\}$. *To check for* A_1*-satisfiability we create the root node* r *with* $L(r) = \{(a{:}A_1, \emptyset)\}$. *An application of the* D^+*-rule to* $(a{:}A_1, \emptyset)$*, followed by two applications of the* $\sqcap$*-rule give* $L(r) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}\neg A, \{1\})\}$. *Applications of the* D^+*-rule to* $(a{:}A_2, \{1\})$ *and* $(a{:}A_3, \{1\})$ *give* $L(r) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A, \{1, 2\}), (a{:}A, \{1, 3\}), (a{:}\neg A, \{1\})\}$. *An application of the* $\bot$*-rule creates two children* y *and* z *of* r *with* $L(y) = \{(a{:}A_1, \emptyset)\}$, $E(y) = \{1\}$, $L(z) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A, \{1, 3\}), (a{:}\neg A, \{1\})\}$*, and* $E(z) = \{2\}$. *Another application of the* $\bot$*-rule, to* z*, creates two children* z_1 *and* z_2 *of* z*, with* $L(z_1) = \{(a{:}A_1, \emptyset)\}$, $E(z_1) = \{1, 2\}$, $L(z_2) = L(z) \setminus \{(a{:}A, \{1, 3\})\}$*, and* $E(z_2) = \{2, 3\}$. *From the exclusion-sets of* y*,* z_1 *and* z_2 *we see that* $\Delta_{\mathcal{T}} = \{\{2, 3\}, \{3\}, \{1\}\}$. *Observe that only two of these,* $\{2, 3\}$ *and* $\{1\}$*, are maximal, and so the* A_1*-MSSs of* Γ *are* $\{A_2 \doteq A, A_3 \doteq A\}$ *and* $\{A_1 \doteq A_2 \sqcap A_3 \sqcap \neg A\}$.

Our algorithm currently does not guarantee that every leaf node corresponds to an A-MSS, as is clearly demonstrated in Example 1, where one of the leaf nodes corresponds to the A_1-satisfiable subset $\{A_3 \doteq A\}$ which is not maximal.

The next example demonstrates why the $\exists$-rule enlarges the associated index-sets of concept assertions of the form $b{:}\forall R.D$ to include the indices in I.

Example 2 *Let* $\Gamma = \{A_1 \doteq A_2 \sqcap A_3 \sqcap A_4,\ A_2 \doteq \forall R.D,\ A_3 \doteq \exists R.C,\ A_4 \doteq \forall R.\neg D\}$. *To check for* A_1*-satisfiability we create the root node* r *with* $L(r) = \{(a{:}A_1, \emptyset)\}$. *An application of the* D^+*-rule to* $(a{:}A_1, \emptyset)$*, followed by two applications of the* $\sqcap$*-rule give* $L(r) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\})\}$. *Three applications of the* D^+*-rule, one to* $(a{:}A_2, \{1\})$*, one to* $(a{:}A_3, \{1\})$*, and one to* $(a{:}A_4, \{1\})$*, give* $L(r) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\}), (a{:}\forall R.D, \{1, 2\}), (a{:}\exists R.C, \{1, 3\}), (a{:}\forall R.\neg D, \{1, 4\})\}$. *Now suppose that we apply the* $\exists$*-rule, but do not enlarge the index-sets of the* $\forall$*-concept assertions. Firstly, this adds* $(b{:}D, \{1, 2\})$, $(b{:}C, \{1, 3\})$*, and* $(b{:}\neg D, \{1, 4\})$ *to* $L(r)$. *An application of the* $\bot$*-rule now creates three children,* x*,* y*, and* z*, of* r*, with* $L(x) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\}), (b{:}D, \{1, 2\}), (b{:}C, \{1, 3\})\}$, $E(x) = \{4\}$, $L(y) = \{(a{:}A_1, \emptyset)\}$, $E(y) = \{1\}$, $L(z) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\}), (b{:}\neg D, \{1, 4\}), (b{:}C, \{1, 3\})\}$*, and* $E(z) = \{2\}$. *From the exclusion-sets of* x*,* y *and* z *we see that* $\Delta_{\mathcal{T}} = \{\{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 3\}\}$ *which all happen to be maximal. So, according to this version these are the sets of indices of all the* A_1*-MSSs of* Γ. *But note that* $\{A_1 \doteq A_2 \sqcap A_3 \sqcap A_4,\ A_2 \doteq \forall R.D,\ A_4 \doteq \forall R.\neg D\}$ *is also an* A_1*-MSS.*

Now consider the correct application of the $\exists$*-rule in which the index-sets of the* $\forall$*-concept assertions are enlarged. The indexed concept assertions now added to* $L(r)$ *are* $(b{:}D, \{1, 2, 3\})$, $(b{:}C, \{1, 3\})$*, and* $(b{:}\neg D, \{1, 3, 4\})$. *And the application of the* $\bot$*-rule now creates four children,* x*,* y*,* z*, and* v *of* r *with with* $L(x) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\}), (b{:}D, \{1, 2, 3\}), (b{:}C, \{1, 3\})\}$, $E(x) = \{4\}$, $L(y) = \{(a{:}A_1, \emptyset)\}$, $E(y) = \{1\}$, $L(z) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\}), (b{:}\neg D, \{1, 3, 4\}), (b{:}C, \{1, 3\})\}$, $E(z) = \{2\}$, $L(v) = \{(a{:}A_1, \emptyset), (a{:}A_2, \{1\}), (a{:}A_3, \{1\}), (a{:}A_4, \{1\})\}$*, and* $E(v) = \{3\}$. *That is, the* $\bot$*-rule now produces a fourth leaf node* v *with 3 as the only element in its exclusion-set, which means that* $\Delta_v = \{1, 2, 4\}$ *and that* $\Gamma(\Delta_v)$ *is the missing* A_1*-MSS.*

Correctness and Complexity

Checking for satisfiability in $\mathcal{ALC}$ is PSPACE-complete. However, the maintenance of the index-sets associated with concepts means that our algorithm yields EXP-TIME as upper bound.

To prove the algorithm is correct we need to show that: (1) The algorithm always terminates; (2) For every leaf node x of a fully expanded $\mathcal{T}$, $\Gamma(\Delta_x)$ is A_j-satisfiable; (3) Every A_j-MSS of Γ is equal to $\Gamma(\Delta_x)$ for some leaf node x of a fully expanded $\mathcal{T}$ for $(\Gamma, a{:}A_j)$. Below we provide outlines of how to do so. Note firstly that the order in which rules are applied (other than the requirement specified in the $\exists$-rule) does not affect the leaf nodes of a fully expanded tree.

For (1), it suffices to show that $\mathcal{T}$ will be fully expanded after a finite number of steps. This follows from the following observations: (i) The D^+- and D^--rules can only be applied a finite number of times; (ii) The $\sqcap$-, $\sqcup$- and $\exists$-rules can only be applied once to a concept assertion; (iii) The $\sqcap$-, $\sqcup$- and $\exists$-rules all create a finite number of new concept assertions that are strictly smaller than the concept assertions they were applied to; (iv) The $\perp$-rule creates a finite number of new leaf nodes, all with fewer concept assertions than the node it was applied to.

The key point is that rules 1-5 will only add concept assertions to $\mathcal{T}$ and there can only be a finite number of expansions if we restrict ourselves to these rules. The additional $\perp$-rule, unlike rules 1-5, will only remove concept assertions. However, this does not mean the same concept assertion can be added and removed indefinitely (i.e. the yoyo effect cannot occur). To avoid this problem, we have employed two book-keeping techniques (i.e. tagging for rules 1-2 and removal of triggering source for rules 3-5). Both of these techniques will ensure expansion of a particular concept assertion can only happen once, regardless of whether any concept assertion is removed at a later stage.

For (2), observe firstly that A_j-satisfiability is equivalent to the DL knowledge base $K = (\Gamma, \{a{:}A_j\})$ being satisfiable. Let $\mathcal{T}$ be a fully expanded tree for $(\Gamma, \{a{:}A_j\})$ obtained by applying the $\perp$-rule only after all other rules have been exhausted, and let x be any leaf node of $\mathcal{T}$.[1] We show that $K' = (\Gamma(\Delta_x), \{a{:}A_j\})$ is satisfiable. To do so, it is sufficient to show that a fully expanded tree $\mathcal{T}'$ for K' contains a clash-free leaf node y. The path from the root node of $\mathcal{T}'$ to y can be constructed from the path from the root node of $\mathcal{T}$ to x as follows. Let $\rho_1, \ldots, \rho_n$ be the sequence of rule applications to obtain the path with x as leaf node. Remove all those rule applications involving any of the axioms with indices in $E(x)$. It follows readily that the remaining rules in the sequence can be used to generate the required path from the root node of $\mathcal{T}'$ to y. Node y has to be clash-free since we have removed the axioms responsible for the clashes encountered on the path to x, and so the $\perp$-rule cannot be applied to it. Also, all other rules appearing in $\rho_1, \ldots, \rho_n$, and not involving any axioms with indices in $E(x)$ will still be applied, which means that y has to be a leaf node of $\mathcal{T}'$.

For (3), consider again the case where the $\perp$-rule is applied only after all other rules have been exhausted. Now consider the clashes contained in the leaf nodes of the tree obtained before we start applying the $\perp$-rule. It can be shown that every possible source of A_j-unsatisfiability correspond to one of these clashes (even though some clashes need not correspond to any source of A_j-unsatisfiability). Observe furthermore that the $\perp$-rule branches by resolving one clash at a time, and therefore, excluding one axiom at a time. From this it can be shown that every A_j-MSS of Γ is equal to $\Gamma(\Delta_x)$ for some leaf node x of the tree obtained by applying the $\perp$-rule until it is fully expanded.

Calculating MCSSs

Calculating MCSSs can be done in two ways, using the algorithm described above. The first method is to first calculate the set of A_j-MSSs for $j = 1, \ldots, n$. From this, the set of MCSSs can be calculated in the following way. Denote by M_j the set of A_j-MSSs, for $j = 1, \ldots, n$. Then the set of MCSSs are the maximal elements of the set $\left\{\bigcap_{i=1}^{i \leq n} X_i \mid X_i \in M_i \text{ for } i = 1, \ldots n\right\}$. But MCSSs can also be calculated directly using the algorithm above. Set $(L(r) = \{a_i{:}A_j, \emptyset) \mid i = 1, \ldots, n\}$ where r is the root node of the tree and the a_is are all distinct individual names, and then expand the tree exactly as before. The maximal elements of $\Gamma(\Delta_{\mathcal{T}})$ are precisely the MCSSs of Γ. Using the algorithm in this way makes use of the fact that concept-satisfiability is equivalent to the DL knowledge base $K = (\Gamma, \{(a_i{:}A_i) \mid i = 1, \ldots, n\})$ being satisfiable.

[1]The choice of applying the $\perp$-rule only after all other rules is useful in this context, but note that this is *not* a general requirement.

Modifications and Optimisations

Since the tableau procedure described above is an extension of standard tableau procedures, it is possible to incorporate into it some standard optimisation techniques (Horrocks 1997) such as normalisation, encoding, caching and ordering heuristics. However, our purpose in this section is to focus on a modification and an optimisation technique specific to the finding of A_j-MSSs. The optimisation technique is based on the observation that the duplication of concept assertions (with differing associated index-sets) in the same node may lead to inefficient behaviour. This can be illustrated by considering Example 1 again. Observe that the first application of the $\perp$-rule in Example 1 involves the detection of a clash involving $a{:}A$ and $a{:}\neg A$. But note that there are two such clashes; one between $(a{:}A, \{1, 2\})$ and $(a{:}\neg A, \{1\})$, and another between $(a{:}A, \{1, 3\})$ and $(a{:}\neg A, \{1\})$. In the example, the $\perp$-rule is applied to the first of these two clashes, and after branching on the index 2, the clash between $(a{:}A, \{1, 3\})$ and $(a{:}\neg A, \{1\})$ still remains, and the $\perp$-rule has to be applied again. It would clearly be more efficient if these two applications of the $\perp$-rule can be rolled into one.

To do so, it is useful to introduce a more parsimonious representation of the duplicated concept assertions in which each concept assertion in a node is associated with a *set* of index-sets. For example, in Example 1, just before the applications of the $\perp$-rule, we would have $(a{:}A, \{\{1, 2\}, \{1, 3\}\})$ replace the two concept assertions $(a{:}A, \{1, 2\})$ and $(a{:}A, \{1, 3\})$. To do so, we need to make a number of modifications to the algorithm. We replace the initial concept assertion $(a{:}A_j, \emptyset)$ to be added to $L(r)$ with $(a{:}A_j, \{\emptyset\})$, and assume that every application of a rule is followed by a consolidation phase in which any duplicate concept assertions $(a{:}C, \mathcal{I}_1), \ldots, (a{:}C, \mathcal{I}_m)$ are replaced with a single concept assertion $(a{:}C, \cup_{i=1}^{i \leq m} \mathcal{I}_i)$. Below we'll see that the consolidation phase also has to remove concept assertions of the form $(a{:}C, \emptyset)$.

We also need to make some changes to the six rules. Firstly, in every one one of the six rules, replace every index-set I and J with the set of index-sets $\mathcal{I}$ and $\mathcal{J}$ respectively. Next, in the D^+-rule we replace the second line with:

$$L(x) := L(x) \cup \{(a{:}C_i, \{I \cup \{i\} \mid I \in \mathcal{I}\})\}$$

(where $\mathcal{I}$ is the set of index-sets associated with $a{:}A_i$). The change to the D^--rule is similar. We replace the second line with:

$$L(x) := L(x) \cup \{(a{:}\neg C_i, \{I \cup \{i\} \mid I \in \mathcal{I}\})\}$$

Thus, the index i is simply added to every index-set occurring in $\mathcal{I}$.

In the $\exists$-rule the second line is changed to the following:

$$X := \{(b{:}C, \mathcal{I})\} \cup \{(b{:}D, \mathcal{K}_D) \mid (a{:}\forall R.D, \mathcal{J}) \in L(x)\}$$

where $\mathcal{K}_D = \{I \cup J \mid I \in \mathcal{I} \;\&\; J \in \mathcal{J}\}$. This simply ensures that we associate with the concept assertion $b{:}D$ those index-sets constructed by combining every index-set in $\mathcal{I}$ with every index-set in $\mathcal{J}$.

This brings us to the $\bot$-rule, where the optimisation technique takes its effect. We give the modified rule and then explain it.

6. $\bot$-rule	If $(a{:}A, \mathcal{I}) \in L(x)$ and $(a{:}\neg A, \mathcal{J}) \in L(x)$ then
	For every $K \in (MH(\mathcal{I}) \cup MH(\mathcal{J}))$ do:
	Create a new child y of x;
	$L(y) := \{(b{:}D, \mathcal{I}_K) \mid (b{:}D, \mathcal{I}') \in L(x)\}$;
	$E(y) := E(x) \cup K$

Observe that we define $\mathcal{I}_K$ as $\{I \in \mathcal{I}' \mid K \cap I = \emptyset\}$, while $MH(\mathcal{K})$ denotes the *minimal hitting sets* of the set of index-sets $\mathcal{K}$. A *hitting set* of $\mathcal{K}$ is an index-set $H \subseteq \cup\mathcal{K}$ such that, for every $K \in \mathcal{K}$, $|H \cap K| = 1$.

Now for the explanation. Recall that the $\bot$-rule is applicable when there is a clash between two concept assertions of the form $(a{:}A, \mathcal{I}))$ and $(a{:}\neg A, \mathcal{J})$, both occurring in $L(x)$. In order to remove the clash, it is necessary to have two branches. One in which $a{:}A$ is removed, and one in which $a{:}\neg A$ is removed. To ensure the removal of *all* duplicates of $a{:}A$, we have to exclude, simultaneously, one of the axioms in each of index-sets occurring in $\mathcal{I}$. That is the same as excluding a hitting set. But since we are interested in removing as few axioms as possible, we only need to exclude the minimal hitting sets of $\mathcal{I}$. And similarly, we need to exclude the minimal hitting sets of $\mathcal{J}$. All in all then, we need to branch on each index-set K occurring in either $MH(\mathcal{I})$ or $MH(\mathcal{J})$.

This brings us to the elements to be assigned to $L(y)$ for each of the newly created nodes y. Line 4 of the new $\bot$-rule labels node y with exactly the concept assertions occurring in $L(x)$, but retains only those index-sets which do not intersect K (the index-set to be excluded). Put differently, every concept assertion $(b{:}D, \mathcal{I}')$ in $L(x)$ is added to $L(y)$ after $\mathcal{I}'$ is modified so that it retains only those index-sets which do not intersect K. Observe that $L(y)$ might end up with elements of the form $(b{:}D, \emptyset)$. Such elements indicate that all the axioms accounting for the presence of $b{:}D$ in $L(y)$ have been excluded using applications of the $\bot$-rule, which means that these elements have to be removed from $L(y)$. This accounts for the part of the consolidation phase described above which removes elements of precisely this kind after the application of each rule. Consider again Example 1 just before the two applications of the $\bot$-rule. With the modified rules we have $L(r) = \{(a{:}A_1, \{\emptyset\}), (a{:}A_2, \{\{1\}\}), (a{:}A_3, \{\{1\}\}), (a{:}A, \{\{1,2\},\{1,3\}\}), (a{:}\neg A, \{\{1\}\})\}$. An application of the new $\bot$-rule now creates two children y and z of r, with $L(y) = \{(a{:}A_1, \{\emptyset\})\}$, $E(y) = \{1\}$, $L(z) = L(r) \setminus \{(a{:}A, \{\{1,2\},\{1,3\}\})\}$, and $E(z) = \{2,3\}$. It is easily established that this modication and optimisation do not affect the correctness and termination conditions of the algorithm.

Related Work

Most of the work on dealing with the debugging of terminologies have focused on *concept-unsatisfiability*. These approaches frequently focus on identifying the possible sources of the unsatisfiability. This includes the work of (Kalyanpur, Parsia, & Sirin 2005; Kalyanpur *et al.* 2005) and (Schlobach & Cornet 2003). The approach closest to ours is the work of Sclobach (Schlobach 2005) which applies techniques from model-based diagnosis to find A-MSSs and MCSSs, and we shall consider this in some detail. The idea here is to first find the minimal unsatisfiability preserving sub-terminologies for a concept name A (or A-MUPSs) of a Tbox Γ. A subset Γ' of Γ is an A-MUPS iff Γ' is A-unsatisfiable, and every strict subset of Γ' is A-satisfiable. He uses a specialised algorithm described in (Schlobach & Cornet 2003) to find the A-MUPSs of an $\mathcal{ALC}$ TBox, and then applies Reiter's hitting set algorithm (Reiter 1987) to find the minimal hitting sets of the set of A-MUPSs. The subsets of Γ obtained by excluding the minimal hitting sets are precisely the A-MSSs of Γ. For example, for the Tbox in Example 1, the A_1-MUPSs are $\{ax_1, ax_2\}$ and $\{ax_1, ax_3\}$, and the minimal hitting sets are $\{ax_1\}$ and $\{ax_2, ax_3\}$. The first five of our rules are very similar to the rules described in (Schlobach & Cornet 2003). Indeed, if we removed the $\bot$-rule, we would have an algorithm which identifies all the sources of A-unsatisfiability, in a way that is very similar to the algorithm in (Schlobach & Cornet 2003). The introduction of the $\bot$-rule thus replaces the need for computing the A-MUPSs from the sources of A-unsatisfiability as is done in (Schlobach & Cornet 2003), as well as for applying the hitting set algorithm, as is done in (Schlobach 2005).

Some more related work which predates that of Schlobach can be found in (Risch & Schwind 1994) and (Baader & Hollunder 1995). The latter investigates the problem of finding the maximally satisfiable subsets of Abox assertions. They attach a propositional formula to each of the sentences in the Abox and propagate these formulas to other sentences while completing the Abox with the expansion rules. The complete Aboxes (with labelled concept assertions) are then used to construct a propositional formula called the clash formula. Each model of the clash formula corresponds to an unsatisfiable subset of the original Abox assertions and with each minimal model corresponds to a minimally unsatisfiable subset of the original Abox assertions.

Experimental Results

We have implemented the algorithm without the suggested optimisations in the previous section, and have performed some preliminary experiments. It was run on three ontologies: the Camera ontology with 12 axioms and 14 concepts, the Koala ontology with 29 concepts and 19 axioms, and a simplified version of the DICE terminology with 527 concepts and 536 axioms, and with the 21 disjointness axioms disabled.[2] In each case the task was to find the A-MSSs for every concept name occurring in the ontology Tests were performed on a standard Linux (Debian) machine with a 2.53GHz Intel Pentium 4 processor, 512KB cache and 512MB of physical memory. Our implementation was developed in Java (JDK 1.5.0) without using any of the existing reasoners. The algorithm is optimised using only ordering heuristics. The order in which expansion rules are applied can be defined manually by the user, and this can have significant effects on the performance. For the purpose of this paper, all experiments were done based on the following fixed ordering of expansion: (1) $\sqcap$-rule; (2) D^+- and D^--rules; (3) $\sqcup$-rule; (4) $\exists$-rule; (5) $\perp$-rule.

Parsing the Camera ontology took 83ms, and so did finding the A-MSSs. Three of the 14 concepts are unsatisfiable, with all three excluding just a single axiom. Parsing the Koala ontology took 86ms and finding the A-MSSs took 131 ms. Three of the 29 concepts were unsatisfiable, with all three excluding just a single axiom. Parsing the DICE ontology took 282 ms and finding all MSSs 7.017 seconds. Of the 527 concepts, 109 were unsatisfiable, with all of them excluding no more than two axioms.

Compare this with the use of FACT++ as a black box for finding the A-MSSs of DICE. It takes 527 satisfiability checks to determine that 109 concepts are satisfiable, then another 109×515 checks to determine all A-MSSs for all concepts in which exactly one axiom is excluded, and then $2 \times (515 \times 514/2)$ to find the A-MSSs in which two axioms are excluded. All in all, this gives 321372 satisfiability checks. It takes about 0.2ms for FACT++, implemented on the same machine as our algorithm, to perform a satisfiability check for one of the DICE concepts. This means it takes FACT++ 64.274 seconds to find all A-MSSs, but without a guarantee that all A-MSSs have been found.

At this stage it is too early to draw any meaningful conclusions, but the results obtained so far merit a more detailed investigation.

Conclusion and Future Work

We have described a specialised algorithm for finding the maximally A-concept satisfiable sub-terminologies of a terminology represented in the description logic $\mathcal{ALC}$. We also showed that the same algorithm can be applied to identify the maximally concept-satisfiable subsets of an $\mathcal{ALC}$ terminology, and discussed some ways of making the algorithm more efficient. We have obtained some promising but preliminary experimental results. For future work we plan to conduct a more detailed experimental evaluation of the implementation.

At present the algorithm can only handle unfoldable terminologies. We are currently working on a tableau-based algorithm for computing maximally satisfiable terminologies in $\mathcal{ALC}$ with cyclic definitions, using a refined blocking condition which ensures that termination is achieved at the right point during the expansion process. A topic for further research is to extend the algorithm in order to deal with more expressive description logics as well.

As mentioned in the section on related work, our algorithm with the $\perp$-rule removed identifies the sources of A-concept unsatisfiability. This information can be used to find the minimally unsatisfiable sub-terminologies. We plan to incorporate this into the algorithm and compare this experimentally with the algorithm in (Schlobach & Cornet 2003).

[2]The Camera and Koala ontologies were obtained from http://protege.stanford.edu/plugins/owl/owl-library/index.html. We are indebted to Stefan Schlobach for providing us with an anonomised version of DICE.

References

Baader, F., and Hollunder, B. 1995. Embedding Defaults into Terminological Knowledge Representational Formalisms. *Journal of Automated Reasoning* 14:149–180.

Baader, F., and Nutt, W. 2003. Basic description logics. In Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P., eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Haarslev, V., and Möller, R. 2001. Racer system description. In Goré, R.; Leitsch, A.; and Nipkow, T., eds., *IJCAR 2001*, volume LNAI 2100.

Horrocks, I. 1997. *Optimising Tableaux Decision Procedures for Description Logics*. Ph.D. Dissertation, University of Manchester.

Horrocks, I. 1998. The FaCT system. In de Swart, H., ed., *Tableaux '98*, volume LNAI 1397, 307–312.

Kalyanpur, A.; Parsia, B.; Sirin, E.; and Hendler, J. 2005. Debugging unsatisfiable classes in OWL ontologies. In *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*. (To Appear).

Kalyanpur, A.; Parsia, B.; and Sirin, E. 2005. Black box techniques for debugging unsatisfiable concepts. In *International Workshop on Description Logics*.

Meyer, T.; Lee, K.; and Booth, R. 2005. Knowledge integration for description logics. In *Proceedings of AAAI05*, 645–650.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Risch, V., and Schwind, C. 1994. Tableaux-Based Characterization and Theorem Proving for Default Logic. *Journal of Automated Reasoning* 13(2):223–242.

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

Schlobach, S. 2005. Diagnosing terminologies. In *Proceedings of AAAI05*, 670–675.

Schmidt-Schauß, M., and Smolka, G. 1991. Attributive concept descriptions with complements. *Artificial Intelligence* 48:1–26.

Tsarkov, D., and Horrocks, I. 2004. Efficient reasoning with range and domain constraints. In *Proceedings of the Description Logics Workshop (DL2004)*, 41–50. Available from ceur-ws.org.