Towards Efficient Reasoning for Description Logics with Inverse Roles

Yu Ding and Volker Haarslev
Concordia University, Montreal, Quebec, Canada
{ding_yu|haarslev}@cse.concordia.ca

Abstract

This paper presents a first proposal for improving the efficiency of modern description logic (DL) reasoners that are known to be less efficient for DLs with inverse roles. The current loss of performance is usually caused by the missing applicability of well-known optimization techniques such as caching the satisfiability status of modal successors. In order to improve this situation we propose a first version of a modified tableau algorithm for ALCI that can be considered as a basis for integrating sound caching techniques into modern reasoners supporting DLs with inverse roles.

1 Introduction

Caching satisfiability results of modal successors is important for speeding up tableau decision procedures for description logics (DLs). In propositional logics, semantic tableau procedures augmented by unsatisfiable lemma generation were reported to be significantly faster than resolution based decision procedure [11]. In DLs, ExpTime worst-case tableau procedures for $\mathcal{ALC}$ [2] also rely on caching. Several sophisticated caching-related techniques have been reported and analyzed in [5, 6]. Besides the caveat about caching satisfiable results in $\mathcal{ALC}$ [2], it is also known that caching techniques are subject to unsoundness in the presence of inverse roles [7]. In this paper, we discuss tableau procedures with an integrated caching functionality for the DLs $\mathcal{ALC}$ and $\mathcal{ALCI}$ respectively. We also propose for $\mathcal{ALCI}$ a polynomial-time procedure for estimating back-propagated information [9, 3] in the presence of unfoldable cyclic TBoxes. This could be considered as a first step to alleviate the gap between the unsatisfactory performance of tableau procedures for DLs with inverse roles and a long-known fact that inverse roles alone do not add to the worst-case complexity of the underlying logics. Finally, we report on an axiom transformation technique with a great potential to switch between the use of roles and their inverses. This transformation technique could be employed in the process of axiom absorption, which is a standard optimization technique used in modern DL reasoners.

2 Tableau Rules for $\mathcal{ALC}$ Augmented by Caching

As already analyzed in [5], caching unsatisfiability results of modal successors does not cause any major problems. However, this changes completely for caching satisfiability results because additional bookkeeping is required to guarantee the soundness of this technique. One possible strategy is to use only local caches. In this proposal, we to cache satisfiable results in a global cache, and use subset caching because of its wide acceptance in real systems [4, 8, 6].

In our tableau procedure, we use twin two labels as in [9]: $\mathcal{L}(x)$ is the normal label as used in the literature; and an additional label $\mathcal{C}(x)$, which is used for caching. In $\mathcal{ALC}$ there is only one type of clash trigger, $\{A, \neg A\}$ for some concept name A [1] ($\perp$ is used as an abbreviation for $A \sqcap \neg A$). To integrate caching into the tableau rules for $\mathcal{ALC}$, a second clash trigger is needed. When creating a successor node y from a node x, if there exists some label $\mathcal{L}(z)$ known to be unsatisfiable and $\mathcal{C}(z) \subseteq \mathcal{C}(y)$, then a clash trigger is applicable. We call a clash of this kind an *unsat-cached clash.*

Definition 2.1 (sat-cached) *Let node x have a label $\{\mathcal{C}(x), \mathcal{L}(x)\}$, and y be a newly generated successor node of x with $\{\mathcal{C}(y), \mathcal{L}(y)\}$. If there exists a node z such that $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $\mathcal{L}(z)$ does not contain a clash, then node y is sat-cached by node z.*[1]

Below is a set of tableau rules for $\mathcal{ALC}$ with caching integrated. It consists of the standard tableau rules for $\mathcal{ALC}$ except for the combined $\exists\forall$-rule.[2] The precedence of the rules is in the order of $\sqcap$-rule, $\sqcup$-rule, $\exists\forall$-rule.

$\sqcap$-rule	if	1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) \neq \{C_1, C_2\}$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule	if	1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$
$\exists\forall$-rule	if	1. $\exists R.C \in \mathcal{L}(x)$
		2. x is not sat-cached
		3. x has no R-successor y with $C \in \mathcal{C}(y)$
	then	create a label $\mathcal{C}(y) = \{C\} \cup \{D \mid \forall R.D \in \mathcal{L}(x)\}$
		set $\mathcal{L}(y) = \mathcal{C}(y)$, $\mathcal{L}(\langle x, y\rangle) = R$

List 1: Tableau rules for $\mathcal{ALC}$ with an integrated generalized cache.

Definition 2.2 *Let E be an $\mathcal{ALC}$ concept in negation normal form (NNF) and R_E be the set of roles occurring in E. A tableau structure for E is a triple $(\mathcal{S}, \mathcal{L}, \mathcal{E})$ where*

[1]We dismiss the possibility of a chain of $x_1, ..., x_n$ for $n \geq 3$, such that x_i is sat-cached by x_{i+1}, where $i = 1, ..., n-1$.

[2]See [7] for a discussion of $\exists\forall$-rule without caching conditions.

$\mathcal{S}$ is a finite set of individuals, $\mathcal{L} : \mathcal{S} \rightarrow 2^{sub(E)}$, $\mathcal{E} : R_E \rightarrow 2^{S \times S}$, and the following properties hold:

(1) $\bot \notin \mathcal{L}(s)$, and if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$,
(2) if $C \sqcap D \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ and $D \in \mathcal{L}(s)$,
(3) if $C \sqcup D \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ or $D \in \mathcal{L}(s)$,
(4) if $\forall R.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$,
(5) if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathcal{S}$ s.t. $\langle s, t\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$.

Proposition 2.3 *The tableau structure introduced above is sound and complete for testing the satisfiability of $\mathcal{ALC}$ concepts.*

Proof. We acknowledge that this proof is almost a $\mathcal{ALC}$ variant of the one for $\mathcal{ALCI}_{R^+}$ proven in [9], if one reads 'blocked' as 'sat-cached'. Additionally, the tableau structure $T = (\mathcal{S}, \mathcal{L}, \mathcal{E})$ is composed from the tableau tree $\mathbf{T}$ as follows:

$\mathcal{S} = \{x \mid x$ is a node in $\mathbf{T}$, and x is not sat-cached$\}$,
$\mathcal{L} =$ the restriction of the labeling $\mathcal{L}$ in $\mathbf{T}$ to $\mathcal{S}$,
$\mathcal{E}(R) = \{(x, y) \in \mathcal{S} \times \mathcal{S} \mid 1.\, y$ is an R-successor of x, or
$2.\, \mathcal{L}(\langle x, z\rangle) = R$ and z is sat-cached by $y\}$.

If $\mathcal{L}(\langle x, z\rangle) = R$, $\forall R.C \in \mathcal{L}(x)$, and z is sat-cached by y, then $C \in \mathcal{C}(z)$. Since $\mathcal{C}(z) \subseteq \mathcal{C}(y) \subseteq \mathcal{L}(y)$, so $C \in \mathcal{L}(y)$. It is guaranteed for $\mathcal{L}(y)$ that no clash is introduced in the final tableau structure. Summing up, $\exists\forall$-rule does not violate or change the property of a tableau structure and its corresponding tableau tree. For the remaining proof of termination and completeness we refer to the relevant parts in [9]. □

3 Tableau Rules for $\mathcal{ALCI}$

In order to extend the previous ideas to $\mathcal{ALCI}$, we need to consider in the tableau tree the two-way computations, i.e., the back-propagation due to $\forall$-restrictions, and accordingly reflect the satisfiability of $\mathcal{ALCI}$ concepts. Please refer to [9, 3] for a discussion on how to handle this back-propagation. We again use the notions of an *unsat-cached clash* and a $\mathcal{C}$-label as given in the previous section. In addition, we introduce a $\mathcal{U}$-label, and require that $\mathcal{C}(x) \cup \mathcal{U}(x) \subseteq \mathcal{L}(x)$. We restrict a $\mathcal{C}$-label such that it keeps the information propagated down the tree, a $\mathcal{U}$-label keeps the information propagated up the tree, while $\mathcal{L}(x)$ contains all information.

Definition 3.1 *Given a node x with $\{\mathcal{C}(x), \mathcal{U}(x), \mathcal{L}(x)\}$ and its R-successor y with $\{\mathcal{C}(y), \mathcal{U}(y), \mathcal{L}(y)\}$, we define the function $up(y, R) = \{C \mid \forall R^-.C \in \mathcal{L}(y)\}$. It is easy to see that $up(y, R) \subseteq \mathcal{U}(x)$ holds.*

Definition 3.2 *Given a node x with $\{\mathcal{C}(x), \mathcal{U}(x), \mathcal{L}(x)\}$, a set of back-propagation edges for x is defined as $\{R^- \mid \forall R^-.C \in \mathcal{L}(x)\}$ and denoted by a function $bedge(x)$.*

Definition 3.3 (sat-cached) *Given a node x with $\{\mathcal{C}(x), \mathcal{U}(x), \mathcal{L}(x)\}$ and its R-successor y with $\{\mathcal{C}(y), \mathcal{U}(y), \mathcal{L}(y)\}$, and some node z with $\{\mathcal{C}(z), \mathcal{U}(z), \mathcal{L}(z)\}$ such that $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $\mathcal{L}(z)$ does not cause a clash.*
IF:
(case-1) $R \notin bedge(z)$; or
(case-2) $R \notin bedge(y)$; or
(case-3) $up(z, R) \subseteq \mathcal{L}(x)$; or
(case-4) z has a predecessor w s.t $\mathcal{L}(w) \subseteq \mathcal{L}(x)$, and $\mathcal{L}(w)$ does not cause a clash;
THEN: node y is sat-cached by node z.

Below is a set of tableau rules for $\mathcal{ALCI}$ with caching integrated.

$\sqcap$-rule	if	1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) \neq \{C_1, C_2\}$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule	if	1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$
$\forall$-rule	if	1. $\forall R.C \in \mathcal{L}(x)$
		2. there is an R-predecessor y of x with $C \notin \mathcal{U}(y)$
	then	$\mathcal{U}(y) = \mathcal{U}(y) \cup \{C\}, \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
		2'. there is an R-successor y of x with $C \notin \mathcal{C}(y)$
	then	$\mathcal{C}(y) = \mathcal{C}(y) \cup \{C\}, \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\exists\forall$-rule	if	1. $\exists R.C \in \mathcal{L}(x)$
		2. x is not sat-cached
		3. x has no R-neighbor y with $C \in \mathcal{C}(y)$
	then	create $\mathcal{C}(y) = \{C\} \cup \{D \mid \forall R.D \in \mathcal{L}(x)\}$ with $\mathcal{L}(\langle x, y\rangle) = R$
		set $\mathcal{L}(y) = \mathcal{C}(y)$

List 2: Tableau rules for $\mathcal{ALCI}$ with an integrated generalized cache.

Definition 3.4 *A tableau structure for an $\mathcal{ALCI}$-concept E is defined as in Definition 2.2 plus an additional condition:*
(6) $\langle s, t\rangle \in \mathcal{E}(R)$ iff $\langle t, s\rangle \in \mathcal{E}(R^-)$.

Proposition 3.5 The above tableau procedure is sound and complete for the testing satisfiability of $\mathcal{ALCI}$ concepts.

Proof. It is easy to see that case-1 is a situation already covered by proposition 2.3; case-4 is a special case of case-3. Therefore, only case-2 and case-3 remain to be proven:

(case 3) Given $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and a known label $\mathcal{L}(z)$, there must exist an $\mathcal{L}$-label for y s.t. $\mathcal{L}(y) \subseteq \mathcal{L}(z)$. So, for any $\forall R^-.C$, if $\forall R^-.C \in \mathcal{L}(y)$, it must be the case that $\forall R^-.C \in \mathcal{L}(z)$. Given $up(z, R) \subseteq \mathcal{L}(x)$, it means that for any $\forall R^-.C \in \mathcal{L}(z)$ it holds $C \in \mathcal{L}(x)$. So, for any $\forall R^-.C$, if $\forall R^-.C \in \mathcal{L}(y)$, then $C \in \mathcal{L}(x)$. To construct the tableau structure T out of the tableau tree $\mathbf{T}$, we need $\mathcal{E}(R) \supseteq \{(x, z) \in \mathcal{S} \times \mathcal{S} \mid \mathcal{L}(\langle x, y\rangle) = R$ and y is sat-cached by z (case-3)$\}$.

(case 2) It is only necessary to discuss $R \notin bedge(y) \wedge R \in bedge(z)$ and not $up(z, R) \subseteq \mathcal{L}(x)$. We need to construct the sub-tableau T' for y in way that no clash will be introduced to $\mathcal{L}(x)$ due to extra back-propagated information. And $T'(y)$ indeed can be copied from a subset of $T(z)$ s.t. no back-propagation to x will be introduced. For the remaining proof for termination and completeness we refer to [9]. □

4 Estimating Back-Propagation Edges

The correct recognition of back-propagation edges for a cached satisfiable label is critical to the soundness of this caching technique. In this section, we discuss a relevant and generalized topic, i.e., how to find back-propagated information [9, 3] for a given concept w.r.t. a cyclic unfoldable $\mathcal{ALCI}$ TBox.[5] We first introduce a method to represent $\mathcal{ALCI}$ concepts in a graph whose size is linear to the size of the given concept. Then, an extension of this representation to cyclic unfoldable TBoxes is introduced. A method is sketched for finding the possibly back-propagated information.[6]

Definition 4.1 *Given an $\mathcal{ALCI}$ expression C in NNF and its directed multi-edge graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$[7], there exists a one-to-one mapping $f : \mathcal{V} \to sub(C)$, and it satisfies ($M_i$ stands for $\forall R$ or $\exists R$ for some role name R, or if it is empty then denoted as ϵ; $\oplus$ denotes $\sqcap$ or $\sqcup$):*
(1) There is exactly one $v_0 \in \mathcal{V}$ s.t. $f(v_0) = C$;
(2) There is exactly one $v_i \in \mathcal{V}$ s.t. $f(v_i) = C_i$ for each concept name C_i occurring in C;
(3) For any $v_1 \in \mathcal{V}$, $f(v_1) = M_1.C_1 \oplus M_2.C_2$ iff there are $v_{21}, v_{22} \in \mathcal{V}$ s.t. $f(v_{2i}) = C_i$, and $M_i \in \mathcal{E}(\langle v_1, v_{2i}\rangle)$ for i=1,2.

Definition 4.2 *A directed multi-edge graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a graph for a TBox $\mathcal{T}$ in NNF[8] if and only if:*
(1) There is exactly one $v_i \in \mathcal{V}$ such that $f(v_i) = a$ for each concept name a in $\mathcal{T}$; and

[5]Due to lack of space we will not discuss the corresponding tableau rules.

[6]This method can be easily adapted to estimate back-propagation edges, and can also serve as heuristics for tableau procedure to refine sat-cached conditions.

[7](1) Multi-edge means $\mathcal{E} : \mathcal{V} \times \mathcal{V} \to 2^S$ for some alphabet S; (2) The graph could be cyclic for unfoldable cyclic TBox; (3) If simply pushing modality prefixes inside, quadratic space is needed in the worst case.

[8]We assume the unfoldable cyclic TBox consists of definitions like $C_N \sqsubseteq D$ only.

(2) There is exactly one sub-graph $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ of $\mathcal{G}(\mathcal{V}, \mathcal{E})$[9], s.t. $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ is the graph of b as by definition 4.1 for each axiom $a \sqsubseteq b \in \mathcal{T}$; and
(3) There is a sub-graph $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ iff $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ is a graph of $sub(b)$ for some axiom $a \sqsubseteq b \in \mathcal{T}$.

Informally, a combined directed multi-edge graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for a concept expression C w.r.t. a TBox $\mathcal{T}$ consists of exactly two, no more and no less, components: (1) a sub-graph $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for TBox $\mathcal{T}$; and (2) a sub-graph $\mathcal{G}''(\mathcal{V}'', \mathcal{E}'')$ in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for concept C. In addition, $\mathcal{V} = \mathcal{V}' \cup \mathcal{V}''$, and $\mathcal{E}(x, y) = \mathcal{E}'(x, y) \cup \mathcal{E}''(x, y)$ for each $x, y \in \mathcal{V}$. By the definitions, it is easy to see such a graph is minimal w.r.t. the number of its nodes and edges.[10]

Example: Given a TBox $\mathcal{T}_0 = \{C \sqsubseteq \exists R.A \sqcap \forall R.B; B \sqsubseteq \forall R^-.A\}$. The sub-graph for TBox $\mathcal{T}_0$ is shown as the triangle in Figure 1. Let concept $D = \exists S.(C \sqcap A)$, a sub-graph for D is shown to the left of the triangle. Figure 1 shows the combined graph for D w.r.t $\mathcal{T}_0$, in which: $f(2) = C \sqcap A$, $f(1) = \exists S.(C \sqcap A)$, $f(5) = A$, $f(4) = \forall R^-.A$.

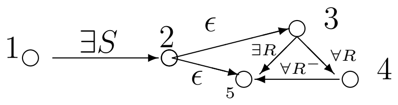

Figure 1: An example of a combined graph.

Definition 4.3 *A multi-valued function match is defined as $match(\exists R) = \{\forall R^-\}$, $match(\forall R) = \{\forall R^-, \exists R^-\}$.*

Proposition 4.4 *Given a combined graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for a unfoldable cyclic TBox $\mathcal{T}$ and a concept C, there exists a polynomial-time bounded procedure for estimating the back-propagated information.*

Proof. (only sketch of the algorithm and its proof and analysis)
The correctness of this estimating method is based on the correspondences between the syntactic structure and the tableau proof tree for $\mathcal{ALCI}$ w.r.t. a unfoldable (cyclic) TBox. The idea of a polynomial-time bounded estimation procedure is based on the following algorithm working on the combined graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$:
The size of graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, i.e., $\|\mathcal{V}\| + \|\mathcal{E}\|$, is of size $O(n)$, where n is the total size of the given TBox and concept. It takes $O(n^3)$ steps for line 1 to initialize a transitive closure for ϵ. For each node $v \in \mathcal{V}$, it is reasonable to assume the number of incoming and outgoing edges are some constant. So, line 4 takes a fixed cost. The loop between line 3 and line 6 loops at most n^2 times. It takes a cost of $O(n^3)$ for line 8 to compute transitive closure. Thus, it takes $O(n^3)$ for each pair $(a, b) \in R$. Since $\|R\| \leq n^2$, the loop between line 2 and line 9 loops at most n^2. So, we get a total upper bound of $O(n^5)$. The final R records back-propagated information, for any $x \in \mathcal{V}$, $\{y | (x, y) \in R\}$ is the back-propagated information for x. □

[9] I.e., $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{E}' : \mathcal{V}' \times \mathcal{V}' \to 2^S$ for the alphabet S, and for each $x, y \in \mathcal{V}'$ it holds that $\mathcal{E}'(x, y) \subseteq \mathcal{E}(x, y)$.

[10] I.e., no node and no edge can be deleted, and there exists no pair of nodes that can be identified.

```
(01) initialize $L = \emptyset$, $R = \{(a,a) | a \in \mathcal{V}\} \cup \{(a,b) | \langle a, b, \epsilon \rangle \in \mathcal{V}\}^*$
(02) for each $(a,b) \in R \wedge (a,b) \notin L$
(03)     for each $x_i \in \mathcal{V} \wedge y_j \in \mathcal{V}$
(04)       if $\exists e_k \exists e_l$ s.t. $(x_i, a, e_k) \in \mathcal{E} \wedge (b, y_j, e_l) \in \mathcal{E} \wedge e_k \in match(e_l)$
(05)       then $R = R \cup \{(x_i, y_j)\}$
(06)     endfor
(07)     $L = L \cup \{(a,b)\}$
(08)     $R = R^*$
(09) endfor
```

5 Elimination of Inverse Roles through Axiom Transformation

Due to the current lack of appropriate optimization techniques in the presence of inverse roles, DL tableau procedures are very sensitive in their performance in this situation but resolution-based approaches to DL reasoning are unaffected. Hence, it might be beneficial if we look at the fragment of first order logic (FO) translated from expressions or axioms of description logics with inverse roles.

For an axiom $C \sqsubseteq \forall R.D$, based on the standard semantics, a translation into FO looks like $\forall x(c(x) \rightarrow (\forall y(r(x,y) \rightarrow d(y))))$. Its prenex form is $\forall x \forall y(\neg c(x) \sqcup \neg r(x,y) \sqcup d(y))$ (axiom 1). On the other hand, the corresponding FO for $\neg D \sqsubseteq \forall R^-.\neg C$ is $\forall z(\neg d(z) \rightarrow (\forall w(r^-(z,w) \rightarrow \neg c(w))))$, and its prenex form is $\forall z \forall w(d(z) \sqcup \neg r^-(z,w) \sqcup \neg c(w))$ (axiom 2). It is easy to see that (2) $\Leftrightarrow \forall z \forall w(d(z) \sqcup \neg r(w,z) \sqcup \neg c(w)) \Leftrightarrow \forall z \forall w(\neg c(w) \sqcup \neg r(w,z) \sqcup d(z))$ (axiom 2.1).

Based on the variable substitution $\{x/w, y/z\}$, (1) becomes identical to (2.1). In this sense, we can conclude that the two axioms are equivalent, i.e. $C \sqsubseteq \forall R.D \Leftrightarrow \neg D \sqsubseteq \forall R^-.\neg C$, w.r.t. the above mentioned translation into first-order logic.

With respect to DL tableau procedures $C \sqsubseteq \forall R.D \Leftrightarrow \neg D \sqsubseteq \forall R^-.\neg C$ also holds because they implies each other.

6 Conclusion

We presented a refined caching technique for the DLs $\mathcal{ALC}$ and $\mathcal{ALCI}$. The proposed technique uses an additional label $\mathcal{C}$ especially designated for the purpose of caching intermediate satisfiable computation results during the execution of tableau proving procedures. Furthermore, we presented a polynomial-time bounded algorithm, which is able to estimate the back-propagated information in tableau procedures for satisfiability tests of the DL $\mathcal{ALCI}$ under unfoldable cyclic TBoxes. We accomplish this by constructing a graph for the underlying unfoldable TBox and the given concept in such a way to abstract away the interaction between the $\exists$-rule and $\forall$-rule, and to treat the $\sqcap$-rule and $\sqcup$-rule equally. This algorithm is based on a simulation of the behavior of the $\forall$-rule in a tableau tree with an edge-matching procedure on the graph.

We also presented a transformation for axioms that can be used to eliminate the use of inverse roles and possibly enhance the absorption of global axioms.

Our plans for future work include, but are not limited to, narrowing down the estimation of back-propagation, sharpening the *sat-cached* conditions, and re-formulating new tableau or pre-test procedures exploiting back-propagated information. These could be added to well-established optimization techniques [7] in tableau procedures for languages having inverse roles, and will possibly lead to an optimized implementation of DL systems [10]. Our immediate plan is to add this new caching technique to Racer [4] and evaluate its effectiveness.

References

[1] F. Baader, D. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2001.

[2] Francesco M. Donini and Fabio Massacci. EXPTime Tableaux for ALC. *Rapporto Tecnico 32/99*, page 50 pages, 12 1999.

[3] Giuseppe De Giacomo and Fabio Massacci. Combining Deduction and Model Checking into Tableaux and Algorithms for Converse-PDL. *Information and Computation,162*, pages 117–137, 2000.

[4] Volker Haarslev and R. Möller. RACER System Description. *International Joint Conference on Automated Reasoning (IJCAR2001)*, 2001.

[5] Volker Haarslev and Ralf Möller. Consistency Testing: The RACE Experience. *Proc. of TABLEAUX'2000, International Conference, Automated Reasoning with Analytic Tableaux and Related Methods*, pages 57–61, july 2000.

[6] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? *KR98*, pages 636–647, 1998.

[7] Ian Horrocks. Implementation and Optimisation Techniques. *Chapter 9, Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346, 12 2003.

[8] Ian Horrocks and Peter F. Patel-Schneider. FaCT and DLP. *In Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of Tableaux'98*, pages 27–30, may 1998.

[9] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. A PSPACE-Algorithm for Deciding $\mathcal{ALCNI}_{\mathcal{R}^+}$-Satisfiability. *LTCS-Report 9808*, page 35, august 1998.

[10] Ralf Möller and Volker Haarslev. *Description Logics Systems.* Chapter 8 in Description Logic Handbook, 2001.

[11] David A. Plaisted. Mechanical Theorem Proving. *Formal Techniques in Artificial Intelligence: A Sourcebook, edited by R. B. Banerji*, pages 269–315, 1990.