Tableau-based Forgetting in $\mathcal{ALC}$ Ontologies

Zhe Wang[1] and **Kewen Wang**[1] and **Rodney Topor**[1] and **Xiaowang Zhang**[2]

Abstract. In this paper, we propose two new and different approaches to forgetting variables for $\mathcal{ALC}$ based on the well-known tableau algorithm. The first approach computes the result of forgetting via rolling up tableaux, which also provides insights of the decidability of existence of forgetting in $\mathcal{ALC}$. When the result of forgetting does not exist, we provide an incremental method for computing approximations of forgetting. The second approach uses variable substitution to refine approximations of forgetting and eventually obtain the result of forgetting. This approach is capable of preserving structural information of the original ontologies and thus renders readability. As both approaches are based on the tableau algorithm, their implementations can make use of the mechanisms and optimization techniques of the existing description logic reasoners.

1 Introduction

An ontology is a formal definition for a common vocabulary (or signature) that are used to describe and represent an application domain. Ontologies can be used by automated tools to provide advanced services such as more accurate web search, intelligent software agents and knowledge management. An example of large biomedical ontology is SNOMED CT [15]. Ontology editing and maintaining tools, such as Protégé [14], are supported by efficient reasoners based on tableau algorithms [10] for description logics (DLs). However, as shown in [3], the existing reasoners provide limited reasoning support for ontology modifications, which largely restricts the wide use of ontologies in the Semantic Web.

For ontology modifications, an essential task is to reduce the vocabulary of an ontology $\mathcal{T}$, that is, to *forget* a subvocabulary $\mathcal{S}$ of the vocabulary of $\mathcal{T}$ and transform $\mathcal{T}$ into a new ontology $\mathcal{T}'$ containing no symbol in $\mathcal{S}$ and sharing with $\mathcal{T}$ the same logical consequences that do not use $\mathcal{S}$.

Two typical scenarios of ontology vocabulary reduction discussed in [13] are *ontology extraction* and *ontology summary*. An example of ontology extraction is as follows. Suppose we had an ontology of medical terms, such as SNOMED, but were only interested in infectious diseases. Rather than building a new ontology from scratch, it is preferable to reuse the existing ontology (SNOMED) and to forget all other terms in the ontology irrelevant to infectious diseases.

In AI and mathematical logic, *forgetting* or *uniform interpolation* has been well investigated in classical logics [7, 8], modal logics [11], logic programming [4], and more recently, in lightweight DLs DL-Lite [12, 6], $\mathcal{EL}$ [5] and more expressive DL $\mathcal{ALC}$ [13]. However, forgetting in expressive DL like $\mathcal{ALC}$ is shown to be difficult, and the result of forgetting in $\mathcal{ALC}$ TBoxes may not exist [13]. Also, the forgetting algorithm based on $\mathcal{ALC}$ concept normal form [13] is not satisfactory, for the following reasons. (1) The computation cannot make use of the off-the-shelf DL reasoners which are based on tableau algorithms. (2) When the result of forgetting does not exist, an incremental algorithm for computing approximations of forgetting is missing. (3) As the algorithm transform each TBox into a normal form, the structural information of the original TBox is lost after forgetting.

In this paper, we propose two different approaches for forgetting in $\mathcal{ALC}$ based on the well-known tableau algorithm. We first introduce a calculus called *rolling up* of tableaux, and show its applications in computing forgetting in concept descriptions and TBoxes. As the result of forgetting in an $\mathcal{ALC}$ may not exist, we provide an algorithm for deciding the existence of forgetting in TBoxes, and to compute the result of forgetting whenever it exists. The algorithm can also be used to compute approximations of forgetting in an incremental manner. However, this approach is unable to preserve structural information of the original TBox. Inspired by a method used for computing least common subsumers in [2], we provide a different approach for computing forgetting in concept descriptions and TBoxes, by introducing a general tableau-based calculus via substitutions on concept terms containing concept variables. To this end, a set of substitution rules are introduced, and then algorithms are developed for forgetting in both concept descriptions and TBoxes. When the result of TBox forgetting exists, the algorithm is capable of computing the result, in which the original structural information is well preserved.

Due to space limitation, only proof sketches for some major results are given in the paper.

2 Preliminary

In this section, we briefly recall some preliminaries of $\mathcal{ALC}$, the tableau algorithm for $\mathcal{ALC}$, and the definitions of concept and TBox forgetting.

2.1 Description Logic $\mathcal{ALC}$

A *concept description* (or concept) in $\mathcal{ALC}$ is built up with concept names and role names. We use N_C to denote the set of concept names and N_R the set of role names. The syntax of $\mathcal{ALC}$ -concept descriptions is defined inductively as follows.

$$B \longleftarrow A \mid \top \mid \bot$$
$$C, D \longleftarrow B \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \forall r.C$$

where $A \in N_C$ and $r \in N_R$. B is called an *atomic concept*, and C, D are called *complex concepts*. A *literal* is an atomic concept or its negation. Given a literal L, we use L^+ to denote its concept name.

Let N_X be the set of *concept variables* (or variables), which can be quantified over and can be substituted with concepts in $\mathcal{ALC}$. A *concept term* is defined in the same way as a concept description but

[1] Griffith University, Brisbane, Australia
[2] Peking University, Beijing, China

allowing concept variables to be used as atomic concepts. We use E, F to denote concept terms.

A concept (term) is in *negation normal form* (NNF) if negations occur only in front of concept names (and variables). A concept (term) can be transformed into its NNF in linear time of its size. In the rest of the paper, without loss of generality, we assume all concept descriptions and concept terms are in NNF.

A *TBox* is a set of axioms of the form $C \sqsubseteq D$ (C is *subsumed* by D). $C \equiv D$ is the abbreviation of both $C \sqsubseteq D$ and $D \sqsubseteq C$. Again, we generalize the syntax of TBoxes to allow variables and axioms of the form $E \sqsubseteq F$.

The signature of a concept description C, written $\mathsf{sig}(C)$, is the set of all concept and role names in C. Similarly, we can define $\mathsf{sig}(\mathcal{T})$ for a TBox $\mathcal{T}$.

A *substitution* σ is a set of pairs of the form $X \mapsto E$ with $X \in N_X$ and E a concept term. A substitution is ground if every E contains no variable. We say σ is *over* a signature $\mathcal{S}$, denoted $\sigma_{\mathcal{S}}$, if E contains only concept and role name in $\mathcal{S}$. Given a concept term F or a TBox $\mathcal{T}$ containing variables, $\sigma(F)$ and $\sigma(\mathcal{T})$ can be defined in a natural way.

2.2 Tableau rules for $\mathcal{ALC}$

A *tableau* $\mathbf{T}$ is a set of trees $\{\mathfrak{T}_1, \ldots, \mathfrak{T}_n\}$. Each node x in $\mathfrak{T}_i$ is labeled with a set of concepts $\mathcal{L}_i(x)$, and each edge $\langle x, y\rangle$ is labeled with a set of roles $\mathcal{L}_i(\langle x, y\rangle)$. In case $\langle x, y\rangle$ is labeled with a set containing role name r, we say x is a *r-predecessor* of y, and y a *r-successor* of x.

The initial state of the tableau algorithm is a labeled root node x, denoted $\mathcal{L}_0(x) = \{C_1, \ldots, C_n\}$, which is then expanded by tableau expansion rules (T-rules, ref. Table 1) [1].

Table 1. Tableau expansion rules (T-rules)

$\sqcap$-rule:	if	$C_1 \sqcap C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
	then	set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if	$C_1 \sqcup C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	create a copy of the tree with label function$\mathcal{L}'$, and set $\mathcal{L}'(x) = \mathcal{L}(x) \cup \{C_1\}$ and $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$
$\exists$-rule:	if	$\exists r.C \in \mathcal{L}(x)$, and x has no r-successor y with $C \in \mathcal{L}(y)$
	then	create a new node y, and set $\mathcal{L}(\langle x, y\rangle) = \{r\}$ and $\mathcal{L}(y) = \{C\}$
$\forall$-rule:	if	$\forall r.C \in \mathcal{L}(x)$, and there is an r-successor y of x with $C \notin \mathcal{L}(y)$
	then	set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\sqsubseteq$ -rule:	if	$C_1 \sqsubseteq C_2 \in \mathcal{T}$, and $\neg C_1 \sqcup C_2 \notin \mathcal{L}(x)$
	then	set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\neg C_1 \sqcup C_2\}$

Note that $\sqcup$-rule splits a tree $\mathfrak{T}$ into two, $\mathfrak{T}$ and $\mathfrak{T}'$. For convenience of marking corresponding nodes and edges in these two trees, we denote corresponding nodes using the same names (*e.g.*, x in the table). We assume each new node created other than by $\sqcup$-rule to be with a new name. We call a subtree whose root is named by x a *x-rooted* subtree.

2.3 Forgetting

Two notions of forgetting are defined in [13], one for concept descriptions and the other for TBoxes.

Definition 1 (C-Forgetting) *Let C be an $\mathcal{ALC}$ -concept description and $\mathcal{S}$ a set of concept and role names. A concept description C' on the signature $\mathsf{sig}(C) - \mathcal{S}$ is a result of* c-forgetting *about $\mathcal{S}$ in C if the following conditions are satisfied:*

(CF1) $\models C \sqsubseteq C'$.
(CF2) $\models C \sqsubseteq D$ *implies* $\models C' \sqsubseteq D$ *for any $\mathcal{ALC}$ -concept D with* $\mathsf{sig}(D) \subseteq \mathsf{sig}(C) - \mathcal{S}$.

Given a signature $\mathcal{S}$, the result of c-forgetting about $\mathcal{S}$ in any $\mathcal{ALC}$ -concept C always exists and is unique up to concept equivalence, which we denote as $\mathsf{cforget}(C, \mathcal{S})$. Here we only consider finite concept descriptions, but the result can be extended to infinite concepts.

Definition 2 (TBox Forgetting) *Let $\mathcal{T}$ be an $\mathcal{ALC}$ -TBox and $\mathcal{S}$ be a set of concept and role names. A TBox $\mathcal{T}'$ over the signature $\mathsf{sig}(\mathcal{T}) - \mathcal{S}$ is a result of* TBox forgetting *about $\mathcal{S}$ in $\mathcal{T}$ if the following conditions are satisfied:*

(TF1) $\mathcal{T} \models \mathcal{T}'$;
(TF2) $\mathcal{T} \models C \sqsubseteq D$ *implies* $\mathcal{T}' \models C \sqsubseteq D$ *for any $\mathcal{ALC}$ -concepts C, D s.t.* $\mathsf{sig}(C \sqcup D) \subseteq \mathsf{sig}(\mathcal{T}) - \mathcal{S}$.

It is shown in [13] that some $\mathcal{ALC}$ -TBoxes may not have finite result of forgetting, even if $\mathcal{S}$ contains only concept names. If the result of forgetting about $\mathcal{S}$ in an $\mathcal{ALC}$ -TBox $\mathcal{T}$ is expressible as a (finite) $\mathcal{ALC}$ TBox, the result of forgetting is unique up to TBox equivalence, which we denote as $\mathsf{forget}(\mathcal{T}, \mathcal{S})$. In this case, we say $\mathcal{S}$ is *forgettable* in $\mathcal{T}$.

The *existence problem* of forgetting is, for given $\mathcal{S}$ and $\mathcal{T}$, to decide whether $\mathcal{S}$ is forgettable in $\mathcal{T}$.

3 Forgetting via Rolling Up

In this section, we introduce a forgetting algorithm based on tableau expansion and *rolling up* techniques. We first introduce rolling up of tableaux for $\mathcal{ALC}$ concept descriptions, and show its application in computing c-forgetting. After that, we introduce the algorithm for computing TBox forgetting based on the rolling up techniques for c-forgetting. With the help of tableau theory and rolling up, we also show the decidability of the existence of forgetting for $\mathcal{ALC}$.

3.1 Forgetting in Concept Descriptions

The tableau algorithm for $\mathcal{ALC}$ expands a (collection of the conjuncts of) concept C into a tableau $\mathbf{T}$. An observation is that $\mathbf{T}$ contains the complete information about C, such that we can restore the concept C up to concept equivalence from $\mathbf{T}$. We call such a process the *rolling up* of a tableau.

Before presenting the formal definition of rolling up, we need to first introduce an additional expansion rule. It is because $\forall$-rule is only applicable when the node has a successor, and the universal quantified concept C in $\forall r.C$ cannot be expanded otherwise. In order to fully expand a tableau including all universal quantified concepts, we introduce for each role r a new role $r_\forall$, and a new expansion rule $\forall*$-rule as follows.

$\forall*$-rule If $\forall r.C \in \mathcal{L}(x)$, and (1) if x has no $r_\forall$-successor, then create a new node y, and set $\mathcal{L}(\langle x, y\rangle) = \{r_\forall\}$ and $\mathcal{L}(y) = \{C\}$; or (2) if y is the $r_\forall$-successor of x and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$.

A *clash* in a label is of the form $\{A, \neg A\}$ or $\bot$. A tree is called *clash free* if it contains no clash or clashes only occur in subtrees whose roots are $r_\forall$-successors. We call a tableau *complete* if no expansion rule or $\forall*$-rule applies to it. A complete tableau $\mathbf{T}$ is *open* if at least one tree in $\mathbf{T}$ is clash free. Otherwise, $\mathbf{T}$ is said to be *closed*. We also say a tree $\mathfrak{T}$ is closed or open, seen as a singleton tableau.

Now we introduce the definition of rolling up for any tableau.

Definition 3 (Rolling Up) *Given a signature $\mathcal{S}$ and a tableau $\mathbf{T}_x = \{\mathfrak{T}_1, \ldots, \mathfrak{T}_n\}$ where each $\mathfrak{T}_i$ is x-rooted, we define the* rolling up *of $\mathbf{T}_x$ over $\mathcal{S}$ to be a concept*

$$\mathsf{roll}(\mathbf{T}_x, \mathcal{S}) = \bigsqcup_{1\le i\le n} \Big(\bigsqcap_{L^+\in\mathcal{S}, L\in\mathcal{L}_i(x)} L \sqcap \bigsqcap_{r\in\mathcal{S}, r\in\mathcal{L}_i(\langle x,y\rangle)} \exists r.\mathsf{roll}(\mathbf{T}_y, \mathcal{S}) \sqcap \bigsqcap_{r\in\mathcal{S}, r_\forall\in\mathcal{L}_i(\langle x,z\rangle)} \forall r.\mathsf{roll}(\mathbf{T}_z, \mathcal{S}) \Big)$$

where L is a literal with L^+ its concept name, and $\mathbf{T}_y$ and $\mathbf{T}_z$ are the sets of y-rooted and z-rooted subtrees in $\mathbf{T}_x$, respectively.

Example 1 *Let $\mathbf{T}$ be the complete tableau of expanding $\mathcal{L}_0(x) = \{ (A \sqcup \exists r.\neg B) \sqcap \forall r.(B \sqcup C) \}$, then $\mathsf{roll}(\mathbf{T}, \{A, B, C, r\})$ is $\big(A \sqcap \forall r.(B \sqcup C) \big) \sqcup \big(\exists r.(\neg B \sqcap C) \sqcap \forall r.(B \sqcup C) \big)$, and $\mathsf{roll}(\mathbf{T}, \{A, C, r\})$ is $A \sqcup \exists r.C$.*

The following result states the correctness of rolling up. That is, the rolling up of an arbitrary tableau of a concept is equivalent to the concept itself.

Proposition 1 *Given an $\mathcal{ALC}$ concept C, let $\mathbf{T}$ be an arbitrary complete tableau obtained by expanding $\mathcal{L}_0(x) = \{C\}$. Then we have $\models C \equiv \mathsf{roll}(\mathbf{T}, \mathsf{sig}(C))$.*

We can define an equivalence relation over all complete tableaux: two complete tableaux $\mathbf{T}_1, \mathbf{T}_2$ are *equivalent* over signature $\mathcal{S}$ if $\models \mathsf{roll}(\mathbf{T}_1, \mathcal{S}) \equiv \mathsf{roll}(\mathbf{T}_2, \mathcal{S})$. Given an $\mathcal{ALC}$ -concept C, define $\mathbf{T}(C)$ to be a tableau obtained by expanding $\mathcal{L}_0(x) = \{C\}$ and removing all closed trees. We can show that $\mathbf{T}(C)$ is unique up to equivalence, and we call $\mathbf{T}(C)$ the *tableau of C*. Note that $\mathbf{T}(C) = \emptyset$ if $\models C \equiv \bot$. The fact that $\mathbf{T}(C)$ contains no closed trees is necessary for the correctness of the following theorem.

Theorem 1 *Given an $\mathcal{ALC}$ concept C and a set $\mathcal{S}$ of concept and role names, we have $\mathsf{cforget}(C, \mathcal{S}) = \mathsf{roll}(\mathbf{T}(C), \mathsf{sig}(C) - \mathcal{S})$.*

3.2 Forgetting in TBoxes

To generalize the c-forgetting method to TBox forgetting, we note that the tableau algorithm *w.r.t.* TBoxes requires an addition rule, the $\sqsubseteq$-rule. The involvement of $\sqsubseteq$-rule may cause the tableaux to be infinite. An example is $\mathcal{T} = \{A \sqsubseteq \exists r.B\}$. As $\neg A \sqcup \exists r.B$ is added into the label of each node, $\exists$-rule may apply indefinitely. A *blocking* condition is used to ensure termination in the tableau algorithm. However, for the tableau of concept C *w.r.t.* a TBox $\mathcal{T}$ to capture the complete information of C and $\mathcal{T}$, in this subsection we assume blocking is not applied.

We generalize the classical definition of concept description to allow *infinite concepts*, which are concept descriptions with infinite expressions. The rolling up of an infinite tableau is an infinite concept description. However, we note that the results for c-forgetting in the previous subsection apply to infinite concepts. However, the result of c-forgetting of an infinite concept may also be an infinite concept.

In what follows, we first show that TBox forgetting can be characterized by c-forgetting in infinite cases. We show this by introducing a concept encoding for the $\sqsubseteq$-rule.

Given a TBox $\mathcal{T}$, define concept $con(\mathcal{T}) = \bigsqcap_{C\sqsubseteq D\in\mathcal{T}}(\neg C \sqcup D)$. Note that each TBox $\mathcal{T}$ can be transformed into an equivalent TBox of the form $\{\top \sqsubseteq con(\mathcal{T})\}$, and thus can be uniquely characterized by the concept $con(\mathcal{T})$.

Given a concept C and a number $n \ge 0$, define

$$C^{(n)} = \prod_{k=0}^{n} \bigsqcap_{r_1,\ldots,r_k\in\mathcal{R}} \forall r_1 \cdots \forall r_k.C,$$

where $\mathcal{R}$ is the set of role names in C. Note that $C^0 = C$, and $\models C^{(n+1)} \sqsubseteq C^{(n)}$ for any $n \ge 0$. $C^{(\infty)}$ is the limit of $C^{(n)}$ when n goes ∞, which is an infinite concept description.

Intuitively, $con(\mathcal{T})^{(n)}$ (together with $\forall$-rule) functions during tableaux expansion by adding $con(\mathcal{T})$ to all the nodes within depth n, as exactly $\sqsubseteq$-rule does. Thus, for any concept C, the expansion of $\mathcal{L}_0(x) = \{C, con(\mathcal{T})^{(n)}\}$ without $\sqsubseteq$-rule imitates the expansion of $\mathcal{L}'_0(x) = \{C\}$ with $\sqsubseteq$-rule applied to all the nodes within depth n. In what follows, we use $\mathbf{T}(C \sqcap con(\mathcal{T})^{(\infty)})$ to denote the tableau of concept C *w.r.t.* $\mathcal{T}$ (with $\sqsubseteq$-rule), and $\mathsf{cforget}(C \sqcap con(\mathcal{T})^{(\infty)}, \mathcal{S})$ the rolling up of $\mathbf{T}(C \sqcap con(\mathcal{T})^{(\infty)})$ over $\mathsf{sig}(C) \cup \mathsf{sig}(\mathcal{T}) - \mathcal{S}$.

The following lemma is indeed a extension of Lemma 9 in [9] to the infinite case.

Lemma 1 *Let $\mathcal{T}$ be a satisfiable $\mathcal{ALC}$ -TBox, and C_1 and C_2 be two $\mathcal{ALC}$ -concepts. Then $\mathcal{T} \models C_1 \sqsubseteq C_2$ iff $\models con(\mathcal{T})^{(\infty)} \sqcap C_1 \sqsubseteq C_2$.*

We have the following result connecting c-forgetting and TBox forgetting.

Theorem 2 *Let $\mathcal{T}$ be an $\mathcal{ALC}$ -TBox and $\mathcal{S}$ be a set of concept and role names. Then*

(1) $\mathcal{S}$ is forgettable in $\mathcal{T}$ iff there exists a number $N \ge 0$ such that $\mathsf{forget}(\mathcal{T}, \mathcal{S}) = \{ \top \sqsubseteq \mathsf{cforget}(con(\mathcal{T})^{(N)}, \mathcal{S}) \}$.

(2) The existence problem of TBox forgetting is decidable.

To see the correctness of Theorem 2, we first show that TBox forgetting can be equally defined using concept relations in infinite case. From Lemma 1, (TF1) is $\mathcal{T} \models \top \sqsubseteq con(\mathcal{T}')$ and is equivalent to $\models con(\mathcal{T})^{(\infty)} \sqsubseteq con(\mathcal{T}')$. And since each concept inclusion can be transformed into the form of $\top \sqsubseteq D$, (TF2) is equivalent to say that $\models con(\mathcal{T})^{(\infty)} \sqsubseteq D$ implies $\models con(\mathcal{T}')^{(\infty)} \sqsubseteq D$ for any $\mathcal{ALC}$ -concept D with $\mathsf{sig}(D) \subseteq \mathsf{sig}(\mathcal{T}) - \mathcal{S}$.

Note that $C = con(\mathcal{T}')$ must be a finite concept description. From (CF1) and (CF2) of c-forgetting, we have the following result.

Proposition 2 *Given an $\mathcal{ALC}$ -TBox $\mathcal{T}$ and a set $\mathcal{S}$ of concept and role names, $\mathcal{S}$ is forgettable in $\mathcal{T}$ iff there exists a finite concept description C over $\mathsf{sig}(\mathcal{T}) - \mathcal{S}$ satisfying*

(E1) $\models \mathsf{cforget}(con(\mathcal{T})^{(\infty)}, \mathcal{S}) \sqsubseteq C$, and

(E2) $\models C^{(\infty)} \sqsubseteq \mathsf{cforget}(con(\mathcal{T})^{(\infty)}, \mathcal{S})$.

In this case, $\mathsf{forget}(\mathcal{T}, \mathcal{S}) = \{\top \sqsubseteq C\}$.

(TF1) and (TF2) correspond to (E1) and (E2), respectively. To see (E2), note that (TF2) requires $\models con(\mathcal{T}')^{(\infty)} \sqsubseteq \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{S})$ for all $n \ge 0$.

From Proposition 2, the existence of TBox forgetting is reduced to the existence of a concept C satisfying (E1) and (E2). The following result shows that the form of such a concept can be further restricted.

Lemma 2 *Let $\mathcal{T}$ and $\mathcal{S}$ be as in Proposition 2. If there exists a finite concept satisfying (E1) and (E2), then there is a number $n \geq 0$ such that $C = \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{S})$ satisfies (E1) and (E2).*

Note that $C = \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{S})$ is finite for any n and always satisfies (E1). The existence of TBox forgetting is further reduced to the existence of a number $n \geq 0$ such that $C = \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{S})$ satisfies (E2).

We use $sub(C)$ to denote the negation closure of the set of all sub-concepts occurring in C. And $sub(\mathcal{T})$ is $sub(con(\mathcal{T}))$. It is not hard to verify that each label $\mathcal{L}(x)$ in tableau $\mathbf{T}(C^{(\infty)} \sqcap D)$ satisfies $\mathcal{L}(x) \subseteq sub(C \sqcup D)$, bearing in mind that $\mathbf{T}(C^{(\infty)} \sqcap D)$ is the tableau of expanding $\mathcal{L}_0(x) = \{D\}$ *w.r.t.* $\{\top \sqsubseteq C\}$ and $C^{(\infty)}$ is not in any label. Thus, there are at most $2^{|sub(C \sqcup D)|}$ different labels in $\mathbf{T}(C^{(\infty)} \sqcap D)$.

Lemma 3 *Let C, D be two finite concepts, $\mathcal{S}$ a set of concept and role names, and $N = 2^{|sub(C \sqcup D)|}$. Then $\models C^{(\infty)} \sqsubseteq \mathsf{cforget}(D^{(\infty)}, \mathcal{S})$ iff the tableau of $C^{(N)} \sqcap \neg\mathsf{cforget}(D^{(N)}, \mathcal{S})$ is closed with in each tree a clash occurs within depth N.*

The above lemma show that it is decidable for each n whether $C = \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{S})$ satisfies (E2). And it also suggests that we only need to check some large enough n.

Lemma 4 *Let C, D be two finite concepts, $\mathcal{S}$ a set of concept and role names, and $N = 2^{|sub(C \sqcup D)|}$. Then $\models (\mathsf{cforget}(C^{(N+k)}, \mathcal{S}))^{(N)} \sqsubseteq D$ iff $\models (\mathsf{cforget}(C^{(N)}, \mathcal{S}))^{(N)} \sqsubseteq D$ for any $k > 1$.*

The correctness Theorem 2 of is clear from Proposition 2 and Lemmas 2 – 4. These results also provide a fixed number $N = 2^{|sub(\mathcal{T})|}$ to Theorem 2 for deciding the existence and computing TBox forgetting.

Now we introduce an algorithm for deciding existence of forgetting and computing $\mathsf{forget}(\mathcal{T}, \mathcal{S})$ if it exists (ref. Figure 1).

Algorithm 1
Input: An $\mathcal{ALC}$ -TBox $\mathcal{T}$, a set $\mathcal{S}$ of concept and role names.
Output: $\mathsf{forget}(\mathcal{T}, \mathcal{S})$ if $\mathcal{S}$ is forgettable in $\mathcal{T}$; otherwise "$\mathcal{S}$ is not forgettable in $\mathcal{T}$".
Method: Initially, let $n = 0$, $\mathbf{T} = \{\mathcal{L}_0(x) = \emptyset\}$, and $\mathcal{T}_0 = \emptyset$. Let $N = 2^{|sub(\mathcal{T})|}$.

Repeat the following steps until $n > N$:

1. Apply $\sqsubseteq$-rule to the nodes of depth n in $\mathbf{T}$, and complete $\mathbf{T}$ with other T-rules and $\forall*$-rule.
2. Compute $C = \mathsf{roll}(\mathbf{T}, \mathsf{sig}(\mathcal{T}) - \mathcal{S})$ and let $\mathcal{T}_{n+1} = \{\top \sqsubseteq C\}$. If $\mathcal{T}_n \models \mathcal{T}_{n+1}$ and C satisfies (E2) (decided using Lemma 3), then return $\mathcal{T}_n$ as $\mathsf{forget}(\mathcal{T}, \mathcal{S})$.
3. Assign $n = n + 1$.

Return "$\mathcal{S}$ is not forgettable in $\mathcal{T}$".

Figure 1. Compute and approximate TBox forgetting via rolling up.

The correctness of Algorithm 1 is easily seen based on previous discussions.

Theorem 3 *Given an $\mathcal{ALC}$ -TBox $\mathcal{T}$ and a set $\mathcal{S}$ of concept and role names, then Algorithm 1 returns $\mathsf{forget}(\mathcal{T}, \mathcal{S})$ if it exists, and returns "$\mathcal{S}$ is not forgettable in $\mathcal{T}$" otherwise.*

Algorithm 1 is also a anytime approximation algorithm for TBox forgetting, as each $\mathcal{T}_n$ obtained is an approximation of the result of forgetting. We have the following results for $\mathcal{T}_n$.

Proposition 3 *Given an $\mathcal{ALC}$ -TBox $\mathcal{T}$ and a set $\mathcal{S}$ of concept and role names, for any $n \geq 0$, we have*

1. $\mathcal{T} \models \mathcal{T}_{n+1} \models \mathcal{T}_n$.
2. $\mathcal{T} \models C \sqsubseteq D$ *implies* $\mathcal{T}_n \models C \sqsubseteq D$ *for any $\mathcal{ALC}$ -concepts C, D with $2^{|sub(C \sqcup D)|} \leq n$ s.t.* $\mathsf{sig}(C \sqcup D) \cap \mathcal{S} = \emptyset$.

Each $\mathcal{T}_{n+1}$ is a better approximation of result of forgetting than $\mathcal{T}_n$, and $\mathcal{T}_{n+1}$ can be computed by further expanding the tableau of $\mathcal{T}_n$. Thus, Algorithm 1 is also an incremental approximation algorithm for TBox forgetting.

4 Forgetting via Variable Substitution

As easily seen, the forgetting algorithm in the previous section cannot preserve the original structure of the TBox. However, in many applications, preservation of the original structure is desired, as it is important for TBox readability. By introducing concept variables into the TBox, our following approach is capable of preserving structural information in the original TBox during forgetting.

4.1 Forgetting in Concept Descriptions

In this approach, we start with a general concept D over $\mathsf{sig}(C) - \mathcal{S}$ such that $\models C \sqsubseteq D$, and replace D with stronger and stronger concepts (*w.r.t.* subsumption) to approximate $\mathsf{forget}(C, \mathcal{S})$.

A natural way of strengthening concepts is by introducing new conjuncts. In [2], a notion of *concept decoration* is defined. Given an $\mathcal{ALC}$ -concept description C in NNF, $dec(C)$ is a concept term defined inductively as follows:

- if C is a literal, $dec(C) = C$;
- if C is $C_1 \sqcap C_2$, $dec(C) = dec(C_1) \sqcap dec(C_2)$; if C is $C_1 \sqcup C_2$, $dec(C) = dec(C_1) \sqcup dec(C_2)$;
- if C is $\exists r.E$, $dec(C) = \exists r.(X \sqcap E)$ with X a new variable; if C is $\forall r.E$, $dec(C) = \forall r.(X \sqcap E)$ with X a new variable.

Define $C_{dec} = X_0 \sqcap dec(C)$ with X_0 a new variable.

The decoration process simply adds one variable conjunct under each quantifier in the concept description. For example, the decoration of concept description $(A \sqcup \exists r.\neg B) \sqcap \forall r.(B \sqcup C)$ is $X_0 \sqcap \big(A \sqcup \exists r.(X_1 \sqcap \neg B)\big) \sqcap \forall r.\big(X_2 \sqcap (B \sqcup C)\big)$.

Since variables are added as conjuncts, it is easy to see that $\models \sigma(C_{dec}) \sqsubseteq C$ for any concept C and ground substitution σ. Moreover, for any concept D with $\models D \sqsubseteq C$, there exists a ground substitution σ such that $\models \sigma(C_{dec}) \equiv D$. Such a σ can be simply constructed as $\{X_0 \mapsto D\} \cup \{X \mapsto \top \mid X \neq X_0, X \text{ in } C_{dec}\}$.

With the notion of decoration and substitution, we can present an equivalent characterization of c-forgetting as follows.

Proposition 4 *Given $\mathcal{ALC}$ -concepts C, D and a set $\mathcal{S}$ of concept and role names, we have $\mathsf{cforget}(C, \mathcal{S}) = D$ iff (1) $\mathsf{sig}(D) \subseteq \mathsf{sig}(C) - \mathcal{S}$, (2) $\models C \sqsubseteq D$, and (3) the following formula is false:*

$$\exists\sigma.\ \{\ \sigma \text{ is ground and is over } \mathsf{sig}(C) - \mathcal{S},\ \text{s.t. } \models C \sqsubseteq \sigma(D_{dec}) \text{ and } \not\models D \sqsubseteq \sigma(D_{dec})\ \}. \qquad (*)$$

To check whether $\mathsf{cforget}(C, \mathcal{S}) = D$, we want to decide whether such a substitution σ satisfying (*) exists or not, and to construct

σ if it exists. Using the tableau algorithm, we need to expand two tableaux, $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$, which are defined to be $\mathcal{L}_0(x) = \{C, \neg D_{dec}\}$ and $\mathcal{L}'_0(x) = \{D, \neg D_{dec}\}$, respectively.

In order to construct σ, we introduce a set of *Substitution rules* (S-rules, ref. Table 2). All the rules are applicable only if $\mathcal{L}$ is the label of an open tree in $\mathbf{T}_{cl}$. As disjunctions need to be handled, the S-rules proposed here are more complex than those in [2]. We use $\sigma(\mathbf{T})$ to denote the global substitution of tableau $\mathbf{T}$. X, Y (with subscripts) are concept variables, and I, J, K are mutual disjoint sets of numbers.

Table 2. Substitution rules (S-rules)

U-rule:	if	$\{\neg X, \neg Y\} \subseteq \mathcal{L}(x)$
	then	apply $\sigma = \{X \mapsto Y\}$ to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$
$\top$-rule:	if	$\neg X \in \mathcal{L}(x)$
	then	apply $\sigma = \{X \mapsto \top\}$ to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$
L-rule:	if	$\{\neg X, L_i\} \subseteq \mathcal{L}_i(x)$ with $i \in I$ and each $L_i^+ \notin \mathcal{S}$
	then	apply $\sigma = \{X \mapsto \bigsqcup_{i\in I} L_i\}$ to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$
Q-rule:	if	$\{\neg X, L_i\} \subseteq \mathcal{L}_i(x)$ with $i \in I$ and each $L_i^+ \notin \mathcal{S}$, $\{\neg X, \exists r_j.C_j\} \subseteq \mathcal{L}_j(x)$ with $j \in J$ and $r_j \notin \mathcal{S}$, $\{\neg X, \forall r_k.C_k\} \subseteq \mathcal{L}_k(x)$ with $k \in K$ and $r_k \notin \mathcal{S}$
	then	apply to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ substitution $\sigma = \{X \mapsto \bigsqcup_{i\in I} L_i \sqcup \bigsqcup_{j\in J} \exists r_j.Y_j \sqcup \bigsqcup_{k\in K} \forall r_k.Y_k\}$ where each Y_j, Y_k are new variables

Each variable X only occurs negated in $\langle \mathbf{T}_{cl}, \mathbf{T}_{op} \rangle$, and thus we only consider $\neg X$ in each label. When both T-rules and S-rules are applicable, T-rules have always precedence over S-rules. U-rule unifies any two variables in a label, and have precedence over all other S-rules, to ensure that other S-rules apply at most once in each label $\mathcal{L}(x)$. Also, the $\top$-rule and L-rule have precedence over the Q-rule, as we want to introduce as few new variables as possible. We call a tableau *S-complete* if no T-rule or S-rule is applicable, and we talk about openness and closeness only for S-complete tableaux.

The following theorem states the soundness and completeness of the S-rules.

Proposition 5 *Given $\mathcal{ALC}$ -concept descriptions C, D and a set $\mathcal{S}$ of concept and role names with $\models C \sqsubseteq D$ and $\mathsf{sig}(D) \subseteq \mathsf{sig}(C) - \mathcal{S}$, then*

(1) The application of T-rules and S-rules to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ will terminate; and

(2) Formula (*) holds iff there is a way of applying S-rules to obtain a ground substitution σ, such that $\sigma(\mathbf{T}_{cl})$ is closed and $\sigma(\mathbf{T}_{op})$ is open.

Now with T-rules and S-rules, we are able to show an approximating algorithm for computing c-forgetting (ref. Figure 2). We call a concept description an *$\mathcal{S}$-literal* if it is of the form A or $\neg A$ with $A \in \mathcal{S}$, or $\exists r.C$ or $\forall r.C$ with $r \in \mathcal{S}$.

In Step 1 of Algorithm 2, D is taken as the first approximation of $\mathsf{cforget}(C, \mathcal{S})$, as it is not hard to verify that $\models C \sqsubseteq D$ and $\mathsf{sig}(D) \subseteq \mathsf{sig}(C) - \mathcal{S}$. And in Step 2, D is refined by $\sigma(D_{dec})$ repeatedly until D is the strongest *w.r.t.* subsumption, which is the result of c-forgetting. Since c-forget always exists for any $\mathcal{ALC}$ -concept description C, Algorithm 2 always terminates. The correctness of Algorithm 2 is stated as follows.

Theorem 4 *Given an $\mathcal{ALC}$ -concept description C and a set $\mathcal{S}$ of concept and role names, then Algorithm 2 always terminates and returns $\mathsf{cforget}(C, \mathcal{S})$.*

Algorithm 2
Input: An $\mathcal{ALC}$ -concept C and a set $\mathcal{S}$ of concept and role names.
Output: $\mathsf{cforget}(C, \mathcal{S})$.
Method:
Step 1. Let D be the concept obtained from C by replacing all $\mathcal{S}$-literals in C with $\top$.

Step 2. Repeat the following steps until D does not change:

1. Assign $\mathbf{T}_{cl}$ to be $\mathcal{L}_0(x) = \{C, \neg D_{dec}\}$ and $\mathbf{T}_{op}$ to be $\mathcal{L}'_0(x) = \{D, \neg D_{dec}\}$.
2. Complete $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ with T-rules and S-rules.
   If a substitution σ is found s.t. $\sigma(\mathbf{T}_{cl})$ is closed and $\sigma(\mathbf{T}_{op})$ is open, then assign $D = \sigma(D_{dec})$.

Step 3. Return D as $\mathsf{cforget}(C, \mathcal{S})$.

Figure 2. Compute c-forgetting via variable substitution

4.2 Forgetting in TBoxes

To generalize the approach in the previous subsection to TBox forgetting, we want to start with a weak TBox $\mathcal{T}'$ over $\mathsf{sig}(\mathcal{T}) - \mathcal{S}$ such that $\mathcal{T} \models \mathcal{T}'$, and approximate the result of forgetting by replacing $\mathcal{T}'$ with stronger and stronger TBoxes (*w.r.t.* logical consequence).

Intuitively, a TBox axiom is strengthened via introducing new disjuncts into the left-hand side of each axiom, and/or new conjuncts into the right-hand side. We define *TBox decoration* with the help of concept decoration.

Definition 4 (TBox Decoration) *Given an $\mathcal{ALC}$ -TBox $\mathcal{T}$, $dec(\mathcal{T})$ is obtained from $\mathcal{T}$ by replacing each axiom $C \sqsubseteq D$ in $\mathcal{T}$ with $\neg dec(\neg C) \sqsubseteq X \sqcap dec(D)$, where each X is a new variable.*

Define $\mathcal{T}_{dec} = dec(\mathcal{T}) \cup \{\top \sqsubseteq X_0\}$ with X_0 a new variable.

For example, the decoration of TBox $\{\ A \sqcap \exists r.B \sqsubseteq \forall r.C,\ C \sqsubseteq D\ \}$ is $\{\ \top \sqsubseteq X_0,\ A \sqcap \exists r.(\neg X_1 \sqcup B) \sqsubseteq X_2 \sqcap \forall r.(X_3 \sqcap C),\ C \sqsubseteq X_4 \sqcap D\ \}$.

Lemma 5 *Given an $\mathcal{ALC}$ -TBox $\mathcal{T}$, we have (1) $\sigma(\mathcal{T}_{dec}) \models \mathcal{T}$ for any ground substitution σ; and (2) for any $\mathcal{ALC}$ -TBox $\mathcal{T}'$ with $\mathcal{T}' \models \mathcal{T}$, there exists a ground substitution σ such that $\sigma(\mathcal{T}_{dec}) \equiv \mathcal{T}'$.*

Similar to Proposition 4, we have the following characterization for TBox forgetting.

Proposition 6 *Given $\mathcal{ALC}$ -TBoxes $\mathcal{T}, \mathcal{T}'$ and a set $\mathcal{S}$ of concept and role names, then $\mathsf{forget}(\mathcal{T}, \mathcal{S}) = \mathcal{T}'$ iff (1) $\mathsf{sig}(\mathcal{T}') \subseteq \mathsf{sig}(\mathcal{T}) - \mathcal{S}$, (2) $\mathcal{T} \models \mathcal{T}'$, and (3) the following formula is false:*

$$\exists\sigma.\ \{\ \sigma \text{ is ground and is over } \mathsf{sig}(\mathcal{T}) - \mathcal{S},\ \text{s.t. } \mathcal{T} \models \sigma(\mathcal{T}'_{dec}),\ \text{and } \mathcal{T}' \not\models \sigma(\mathcal{T}'_{dec})\ \}. \qquad (**)$$

To check whether $\mathsf{forget}(\mathcal{T}, \mathcal{S}) = \mathcal{T}'$, in contrast to c-forgetting, TBox tableaux expansion requires the additional $\sqsubseteq$-rule, and the classical tableau blocking condition, called T-blocking, is needed here. $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ both are initialized to be $\mathcal{L}_0(x) = \{\bigsqcup_{E\sqsubseteq F\in\mathcal{T}'_{dec}} E \sqcap \neg F\}$. The difference is that they are expanded *w.r.t.* different TBoxes. $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ are expanded *w.r.t.* $\mathcal{T}$ and $\mathcal{T}'$, respectively. By applying T-rules (including the $\sqsubseteq$-rule) and the same set of S-rules to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$, the algorithm tries to construct substitution σ such that $\sigma(\mathbf{T}_{cl})$ is closed and $\sigma(\mathbf{T}_{op})$ is open.

However, because of the inter-operation of $\sqsubseteq$-rule and S-rules, termination is violated. In particular, when $\mathcal{T}'$ is already the result of forgetting, $\sqsubseteq$-rule may keep adding concepts of the form $\exists r.E$ into the labels and introducing new nodes which trigger the application of Q-rule. As labels are changed after applying Q-rule, T-blocking may fail.

A blocking condition is needed here, which is similar to that used in [2]. Application of Q-rule with substitution $X \mapsto E$ is *S-blocked* in $\mathcal{L}(x)$ if in a previous state, Q-rule has been applied with $X' \mapsto E'$ in $\mathcal{L}'(x')$ such that: (1) $E = E'$, (2) $\mathcal{L}(x) = \mathcal{L}'(x')$, and (3) for each r occurring in E and each r-successor y of x, there is a r-successor y' of x' with $\mathcal{L}(y) = \mathcal{L}(y')$. The equations in (1) – (3) are regardless of variable name variations.

The following result states the termination, soundness and completeness of S-rules regarding TBoxes.

Proposition 7 *Given $\mathcal{ALC}$ -TBoxes $\mathcal{T}, \mathcal{T}'$ and a set $\mathcal{S}$ of concept and role names with $\mathcal{T} \models \mathcal{T}'$ and $\mathsf{sig}(\mathcal{T}') \subseteq \mathsf{sig}(\mathcal{T}) - \mathcal{S}$, then*

(1) The application of T-rules and S-rules to $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ will terminate (with T-blocking and S-blocking); and

(2) Formula (**) holds iff there is a way of applying S-rules to obtain a ground substitution σ, s.t. $\sigma(\mathbf{T}_{cl})$ is closed and $\sigma(\mathbf{T}_{op})$ is open.

Now we present the algorithm for computing TBox forgetting based on T-rules, S-rules, and blocking conditions (ref. Figure 3). We assume all concepts in the TBox are in NNF.

Algorithm 3
Input: An $\mathcal{ALC}$ -TBox $\mathcal{T}$ and a set $\mathcal{S}$ of concept and role names.
Output: $\mathsf{forget}(\mathcal{T}, \mathcal{S})$.
Method:
Step 1. $\mathcal{T}'$ is obtained from $\mathcal{T}$ by replacing all $\mathcal{S}$-literals on the left-hand sides of the axioms with $\bot$, and those on the right-hand sides with $\top$.

Step 2. Repeat the following steps until $\mathcal{T}'$ does not change:

1. Assign both $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ to be $\mathcal{L}_0(x) = \{\bigsqcup_{E \sqsubseteq F \in \mathcal{T}'_{dec}} E \sqcap \neg F\}$.
2. Complete $\mathbf{T}_{cl}$ and $\mathbf{T}_{op}$ by T-rules and S-rules *w.r.t.* , $\mathcal{T}$ and $\mathcal{T}'$, respectively.
3. If a substitution σ is found *s.t.* $\sigma(\mathbf{T}_{cl})$ is closed and $\sigma(\mathbf{T}_{op})$ is open, then assign $\mathcal{T}' = \sigma(\mathcal{T}'_{dec})$.

Step 3. Return $\mathcal{T}'$ as $\mathsf{forget}(\mathcal{T}, \mathcal{S})$.

Figure 3. Compute TBox forgetting via variable substitution.

Note that Algorithm 3 may not terminate, even with the blocking conditions. In particular, when the result of forgetting does not exist, there is an infinite sequence of stronger and stronger TBoxes derived as better approximations of the result of forgetting. That is, although blocking condition guarantees each execution of Step 2 to be within finite steps, Step 2 can be repetitively executed infinite times.

The following example shows the performance of Algorithm 3 in a case where TBox forgetting does not exist.

Example 2 *Let $\mathcal{T} = \{\ A \sqsubseteq B,\ B \sqsubseteq \exists r.B \sqcap C\ \}$ and $\mathcal{S} = \{B\}$. Then Algorithm 3 starts with $\mathcal{T}' = \{\ A \sqsubseteq \top,\ \bot \sqsubseteq \exists r.\top \sqcap C\ \}$ and $\mathcal{T}'_{dec} = \{\ \top \sqsubseteq X_0,\ A \sqsubseteq X_1,\ \bot \sqsubseteq X_2 \sqcap \exists r.(X_3 \sqcap \top) \sqcap C\ \}$.*

Denote σ_i and $\mathcal{T}'_i$ to be the substitution and resulting TBox, respectively, in the i-th iteration of Step 2. We omit the pairs in each σ_i of the form $X \mapsto \top$ and trivial axioms in each $\mathcal{T}'_i$ of the form $\bot \sqsubseteq C$ or $C \sqsubseteq \top$. Then we have

$\sigma_1 = \{X_1 \mapsto C\}$, *and* $\mathcal{T}'_1 = \{A \sqsubseteq C\}$ *with* $\mathcal{T}'_{dec} = \{\ \top \sqsubseteq X_0,\ A \sqsubseteq X_1 \sqcap C\ \}$;

$\sigma_2 = \{X_1 \mapsto \exists r.Y,\ Y \mapsto C\}$, *and* $\mathcal{T}'_2 = \{A \sqsubseteq C \sqcap \exists r.C\}$ *with* $\mathcal{T}'_{dec} = \{\ \top \sqsubseteq X_0,\ A \sqsubseteq X_1 \sqcap C \sqcap \exists r.(X_2 \sqcap C)\ \}$;

$\sigma_3 = \{X_1 \mapsto \top,\ X_2 \mapsto \exists r.Y,\ Y \mapsto C\}$, *and* $\mathcal{T}'_3 = \{A \sqsubseteq C \sqcap \exists r.(C \sqcap \exists r.C)\}$;

$\cdots$

$\mathcal{T}'_n = \{A \sqsubseteq \underbrace{C \sqcap \exists r.\big(C \sqcap \exists r.(C \cdots \exists r.C)\big)}_{n\ C's}\}$ *for* $n \geq 1$.

However, whenever the result of forgetting exists, the termination and correctness of Algorithm 3 are guaranteed.

Theorem 5 *Given $\mathcal{ALC}$ -TBox $\mathcal{T}$ and a set $\mathcal{S}$ of concept and role names, if $\mathcal{S}$ is forgettable in $\mathcal{T}$, then Algorithm 3 always terminates and returns* $\mathsf{forget}(\mathcal{T}, \mathcal{S})$.

5 Conclusion

We have presented two approaches for computing the result of forgetting in both $\mathcal{ALC}$ concept descriptions and TBoxes, based on the tableau algorithm for $\mathcal{ALC}$. The first approach is based on the technique of rolling up tableaux. Compared to the algorithm introduced in [13], this new method allows to compute the result of forgetting in an incremental way, which is key to efficient implementation of forgetting. An important application of this method is to show that the existence problem of forgetting in $\mathcal{ALC}$ TBoxes is decidable, However, the first method cannot guarantee that the structural information of the original ontology (TBox) is preserved after forgetting. As a result, we have developed quite different method for forgetting in $\mathcal{ALC}$, which is inspired by a technique used for computing least common subsumers in [2]. This method consists of running two tableau-based procedures in parallel. The second new method possesses several advantages: (1) it is an incremental computation algorithm; (2) it can be implemented using an off-the-shelf reasoner for $\mathcal{ALC}$; and (3) it is capable of preserving the structural information of the original ontologies (TBoxes). For future research, it would be useful to find lower bounds on the complexity of forgetting. It would be also useful to generalize the forgetting algorithms for more expressive DLs than $\mathcal{ALC}$. It would be interesting to implement our algorithms and incorporate them into ontology editors for some experiments.

Acknowledgements: The authors would like to thank the referees for their helpful and constructive comments. This work was partially supported by the Australia Research Council (ARC) under DP0666107 and DP1093652.

REFERENCES

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. (2003).
[2] F. M. Donini, S. Colucci, T. Di Noia, and E. Di Sciascio, 'A tableaux-based method for computing least common subsumers for expressive description logics', in *Proc. 21st IJCAI*, pp. 739–745, (2009).
[3] M. Dzbor, E. Motta, C. Buil, J. M. Gomez, O. Görlitz, and H. Lewen, 'Developing ontologies in OWL: an observational study', in *Proc. Workshop on OWL: Experiences and Directions*, (2006).
[4] T. Eiter and K. Wang, 'Semantic forgetting in answer set programming', *Artificial Intelligence*, **14**, 1644–1672, (2008).
[5] B. Konev, F. Wolter, and M. Zakharyaschev, 'Forgetting and uniform interpolation in large-scale description logic terminologies', in *Proc. 20th IJCAI*, pp. 830–835, (2009).

[6] R. Kontchakov, F. Wolter, and M. Zakharyaschev, 'Can you tell the difference between DL-Lite ontologies?', in *Proc. 11th KR*, pp. 285–295, (2008).
[7] J. Lang, P. Liberatore, and P. Marquis, 'Propositional independence: Formula-variable independence and forgetting.', *J. Artif. Intell. Res.*, **18**, 391–443, (2003).
[8] F. Lin and R. Reiter, 'Forget it', in *Proc. AAAI Fall Symposium on Relevance*, pp. 154–159. New Orleans (LA), (1994).
[9] B. ten Cate, W. Conradie, M. Marx, and Y. Venema, 'Definitorially complete description logics', in *Proc. 10th KR*, pp. 79–89, (2006).
[10] D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider, 'Optimizing terminological reasoning for expressive description logics', *J. Autom. Reasoning*, **39**(3), 277–316, (2007).
[11] Albert Visser, 'Uniform interpolation and layered bisimulation', in *Proc. Gödel'96*, pp. 139–164, (1996).
[12] Z. Wang, K. Wang, R. Topor, and J. Z. Pan, 'Forgetting concepts in DL-Lite', in *Proc. 5th ESWC*, pp. 245–257, (2008).
[13] K. Wang, Z. Wang, R. Topor, J. Z. Pan, and G. Antoniou, 'Concept and role forgetting in ALC-ontologies', in *Proc. 8th ISWC*, pp. 666–681, (2009).
[14] 'Protégé', `http://protege.stanford.edu`, (2010).
[15] 'SNOMED CT', `http://www.fmrc.org.au/snomed/`, (2007).