View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Griffith Research Online

Concept and Role Forgetting in $\mathcal{ALC}$ Ontologies

Paper Number: 423

No Institute Given

Abstract. Forgetting is a useful tool for tailoring ontologies by reducing the number of concepts and roles, while preserving sound and complete reasoning. Some attempts have been made to address the problem of forgetting in some relatively simple description logics (DLs) such as DL-Lite and extended EL . Ontologies in those works are mostly expressed as TBoxes rather than general knowledge bases (KBs). However, the issue of forgetting for general KBs in more expressive ontology languages, such as $\mathcal{ALC}$ and OWL DL, is largely unexplored. In particular, the problem of characterizing and computing forgetting is still open. In this paper, we first define semantic forgetting about concepts and roles in $\mathcal{ALC}$ ontologies and show several important properties of forgetting in this setting. Unlike the case of DL-Lite, the result of forgetting in an $\mathcal{ALC}$ ontology may not exist in general, which makes the problem of how to compute forgetting in $\mathcal{ALC}$ more challenging. As a result, we tackle the non-existence of the result of forgetting in $\mathcal{ALC}$ ontologies by first investigating forgetting in concept descriptions and then defining and studying a series of approximations to the result of forgetting in $\mathcal{ALC}$ ontologies. We show that forgetting in $\mathcal{ALC}$ ontologies can be approximated through computing forgetting for concept descriptions. Our algorithms can be embedded into an ontology editor to enhance its ability to manage and apply (large) ontologies.

Keywords: Forgetting, description logics, update, nonmonotonic reasoning.

1 Introduction

There are more and more semantically annotated data available in the Web. For example, so far there are about 5 billion linked data[1] available online. Accordingly, the Web is rapidly emerging as a large scale platform for publishing and sharing formalised knowledge models [**?**]. As more ontologies become available for annotating data on the Web and as the populated ontologies become larger and more comprehensive, it becomes more crucial for the Semantic Web [4] to construct and manage ontologies. Examples of large ontologies include the Systematised Nomenclature of Medicine Clinical Terms (SNOMED CT) containing 380K concepts, GALEN, the Foundational Model of Anatomy (FMA), the National Cancer Institute (NCI) Thesaurus containing over 60K axioms, and the OBO Foundry containing about 80 biomedical ontologies.

While it is expensive to construct large ontologies, much higher costs of maintenance would also occur for hosting and running a large and comprehensive ontology

[1] `http://linkeddata.org/`

than a trimmed-down version of that ontology. Tools to reduce an ontology to one that better fits certain needs can greatly aid and encourage reusing existing ontologies. However, as the tool evaluation study in [5] shows, existing tools, such as Protégé [27], NeOn [28] and TopBraid [29], are far from satisfactory.

In several areas of ontology management, the ontology engineers face tasks of obtaining a small ontology from an existing (large) ontology by hiding/forgetting some irrelevant concepts and roles while still preserving certain forms of reasoning. These areas include ontology extraction, ontology summary, ontology integration, and ontology evolving. Let us consider two scenarios in ontology extraction and ontology summary, respectively.

Ontology extraction: To reduce the high cost of building ontologies by hand, it has been the focus of some research to construct ontologies automatically. One promising approach to constructing new ontologies is to search and reuse ontologies that already exist on the Web. In many cases, large ontologies need to be tailored first and only relevant parts are reused. Consider a scenario discussed in [9]: suppose we want to design an ontology *Pets* describing properties of domestic animals such as cats and dogs. Rather than starting from scratch we would first search the Web and try to find similar ontologies that can be reused. Suppose that we found a large ontology *Animals* on the Web describing domestic animals as well as wild animals such as lions and tigers. In this case we can forget about those terms of animals in the ontology *Animal* that are not considered as pets and obtain a smaller portion of *Animal*.

Ontology summary: Compared to ontology extraction, existing tools are even more limited in providing support for navigating and making sense of the ontologies. As argued in [1, 18], a key problem faced by an ontology engineer is so-called ontology summary. When considering the reuse of a large ontology, it is important to obtain a view of the ontology in making decisions about the suitability of the ontology in question for the current ontology engineering development project. In general, a process of ontology summary consists of two stages: The first stage is to identify the *key concepts* in the large ontology. There have been some algorithms for accomplishing this task [1, 18]. After a set of key concepts are found, the next stage in ontology summary is to hide/forget the concepts that are not key concepts.

However, an ontology is often represented as a logical theory, and the removal of one term may influence other terms in the ontology. Thus, more advanced methods for dealing with large ontologies and reusing existing ontologies are desired.

Forgetting (or, *uniform interpolation*) developed in logics [15, 16, 6] provides a useful tool for obtaining small ontologies from a large ontology by discarding irrelevant concepts, roles and axioms while preserving sound and complete reasoning. Informally, forgetting is a particular form of reasoning that allows a piece of information (say, p) in an ontology to be discarded or hidden in such a way that future reasoning on information irrelevant to p will not be affected.

It is well-known that some ontology languages, like OWL, are built on description logics (DLs) [3]. By an ontology we mean a knowledge base (KB) in a description logic. A terminology box (TBox) is considered as a special form of ontology in which the assertion box (ABox) is empty. Although most description logics are fragments of first order logic, the forgetting introduced in [16] does not help much with description logics for at least two reasons: First, the correspondence between DLs and FOL is useless in investigating forgetting for DLs because the result of forgetting in a theory of the first order logic (FOL) may be a theory in second order logic. Second, one would like to directly perform forgetting in description logics rather than transforming an ontology into a first order theory and then back. Indeed, some attempts have been made to address the issue of forgetting in relatively simple DLs such as DL-Lite [20, 14] and extended EL [12]. Moreover, ontologies in these works are mostly expressed as TBoxes rather than general knowledge bases (KBs). Forgetting generalizes *conservative extensions* [9, 7, 17] and the *modularity* defined in [8, 10, 13].

While a definition of forgetting for TBoxes in $\mathcal{ALC}$ is briefly mentioned [7], the problem of forgetting for ontologies expressed as $\mathcal{ALC}$ knowledge bases is largely unexplored. In particular, the problem of characterizing and computing forgetting is still open.

In this paper we first define semantic forgetting for ontologies in the description logic $\mathcal{ALC}$ and show several important properties of forgetting. We choose $\mathcal{ALC}$ to study in this paper because it allows all boolean operations and most expressive DLs are based on it. Unlike the case of DL-Lite, the result of forgetting about concepts and roles in an $\mathcal{ALC}$ ontology may not exist. We tackle the non-existence of the result of forgetting in $\mathcal{ALC}$ ontologies by first investigating forgetting in concept descriptions and then defining and studying a series of approximations to the result of forgetting in $\mathcal{ALC}$ ontologies. Based on our results, we show that forgetting in $\mathcal{ALC}$ ontologies can be approximated through computing forgetting for concept descriptions. Our algorithms can be embedded into an ontology editor to enhance its ability to manage and apply (large) ontologies.

It is worth pointing out that our work significantly extended previous works in at least two ways: (1) We made the first attempt to study forgetting for an expressive description logic, instead of DL-Lite and variants of EL. (2) Ontologies in this paper are expressed as general KBs rather than only TBoxes as in previous works. In addition, our definitions and results hold for forgetting about both concepts and roles.

2 Description Logic $\mathcal{ALC}$

In this section, we briefly recall some preliminaries of $\mathcal{ALC}$, the basic description logic which contains all boolean operators. Further details of $\mathcal{ALC}$ and other DLs can be found in [3].

First, we introduce the syntax of *concept descriptions* for $\mathcal{ALC}$. To this end, we assume that N_C is a set of *concept names* (or *concept*), N_R is a set of *role names* (or *roles*) and N_I is a set of individuals.

Elementary concept descriptions consist of both *concept names* and *role names*. So a concept name is also called *atomic concept* while a role name is also called *atomic role*. Complex concept descriptions are built inductively as follows: A (atomic concept); $\top$ (universal concept); $\bot$ (empty concept); $\neg C$ (negation); $C \sqcap D$ (conjunction); $C \sqcup D$ (disjunction); $\forall R.C$ (universal quantification) and $\exists R.C$ (existential quantification). Here, A is an (atomic) concept, C and D are concept descriptions, and R is a role.

An interpretation $\mathcal{I}$ of $\mathcal{ALC}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain* and $\cdot^{\mathcal{I}}$ is an interpretation function which associates each (atomic) concept A with a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and each atomic role R with a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ can be naturally extended to complex descriptions:

$$\begin{array}{ll}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & \bot^{\mathcal{I}} = \emptyset \\
(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} - C^{\mathcal{I}} & (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} & \\
(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \ : \ \forall b.(a,b) \in R^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\} & \\
(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \ : \ \exists b.(a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} &
\end{array}$$

An $\mathcal{ALC}$ *assertional box* (or *ABox*) is a finite set of *assertions*. An assertion is a *concept assertion* of the form $C(a)$ or a *role assertion* of the form $R(a, b)$, where a and b are individuals, C is a concept name, R is a role name.

An interpretation $\mathcal{I}$ *satisfies* a concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, a role assertion $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. If an assertion α is satisfied by $\mathcal{I}$, it is denoted $\mathcal{I} \models \alpha$. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, if it satisfies all assertions in $\mathcal{A}$.

A *inclusion axiom* (simply *inclusion*, or *axiom*) is of the form $C \sqsubseteq D$ (C is *subsumed* by D), where C and D are concept descriptions. The inclusion $C \equiv D$ (C is *equivalent* to D) is an abbreviation of two inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$. A *terminology box*, or *TBox*, is a finite set of inclusions. An interpretation $\mathcal{I}$ satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ is a model of a TBox $\mathcal{T}$, denoted $\mathcal{I} \models \mathcal{T}$, if $\mathcal{I}$ satisfies every inclusion of $\mathcal{T}$. $\mathcal{T} \models C \sqsubseteq D$ if for any $\mathcal{I}$, $\mathcal{I} \models \mathcal{T}$ implies $\mathcal{I} \models C \sqsubseteq D$.

Formally, a knowledge base (KB) is a pair $(\mathcal{T}, \mathcal{A})$ of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. An interpretation $\mathcal{I}$ is a model of $\mathcal{K}$ if $\mathcal{I}$ is a model of both $\mathcal{T}$ and $\mathcal{A}$, denoted $\mathcal{I} \models \mathcal{K}$. If α is an axiom or an assertion, $\mathcal{K} \models \alpha$ if every model of $\mathcal{K}$ is also a model of α. Two KBs $\mathcal{K}$ and $\mathcal{K}'$ are equivalent, written $\mathcal{K} \equiv \mathcal{K}'$, if they have the same models. "$\equiv$" can be similarly defined for ABoxes and TBoxes.

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 $\mathsf{sig}(\mathcal{A})$ for an ABox $\mathcal{A}$, $\mathsf{sig}(\mathcal{T})$ for a TBox $\mathcal{T}$, and $\mathsf{sig}(\mathcal{K})$ for a KB $\mathcal{K}$.

3 Forgetting in $\mathcal{ALC}$ Ontologies

In this section, we will first give a semantic definition of what it means to forget about a set of variables in a $\mathcal{ALC}$ KB, and then discuss several interesting properties of the forgetting operation.

As explained earlier, given an ontology $\mathcal{K}$ on signature $\mathcal{S}$ and $\mathcal{V} \subset \mathcal{S}$, in ontology engineering it is often desirable to obtain a new ontology $\mathcal{K}'$ on $\mathcal{S} - \mathcal{V}$ such that reasoning tasks on $\mathcal{S} - \mathcal{V}$ are still preserved in $\mathcal{K}'$. As a result, $\mathcal{K}'$ is weaker than $\mathcal{K}$ in general. This intuition is formalized in the following definition.

Definition 3.1 (KB-forgetting). *Let $\mathcal{K}$ be a KB in $\mathcal{ALC}$ and $\mathcal{V}$ be a set of variables. A KB $\mathcal{K}'$ over the signature* $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ *is a* result of forgetting *about $\mathcal{V}$ in $\mathcal{K}$ if*

(KF1) $\mathcal{K} \models \mathcal{K}'$*;*

(KF2) *for each concept inclusion $C \sqsubseteq D$ in $\mathcal{ALC}$ not containing any variables in $\mathcal{V}$, $\mathcal{K} \models C \sqsubseteq D$ implies $\mathcal{K}' \models C \sqsubseteq D$;*

(KF3) *for each membership assertion $C(a)$ or $R(a, b)$ in $\mathcal{ALC}$ not containing any variables in $\mathcal{V}$, $\mathcal{K} \models C(a)$ (resp., $\mathcal{K} \models R(a, b)$) implies $\mathcal{K}' \models C(a)$ (resp., $\mathcal{K}' \models R(a, b)$);*

To illustrate the above definition of semantic forgetting and how forgetting can be used in ontology extraction, consider the following example of designing an $\mathcal{ALC}$ ontology about flu.

Example 3.1. Suppose we have searched the Web and found an ontology about human diseases (such a practical ontology could be very large):

$Disease \sqsubseteq \forall attacks.Human$,
$Human \equiv Male \sqcup Female$,
$Human \sqcap Infected \sqsubseteq \exists shows.Symptom$,
$Disease \equiv Infectious \sqcup Noninfectious$,
$Influenza \sqcup HIV \sqcup TB \sqsubseteq Infectious$.

We want to construct a (smaller) ontology only about flu by reusing the above ontology. This is done by forgetting about the undesired concepts $\{Disease, Noninfectious, HIV, TB\}$. As a result, the following ontology is obtained:

$Influenza \sqsubseteq Infectious$,
$Infectious \sqsubseteq \forall attacks.Human$,
$Human \equiv Male \sqcup Female$,
$Human \sqcap Infected \sqsubseteq \exists shows.Symptom$,

The next example shows that the result of forgetting in an $\mathcal{ALC}$ ontology may not exist in some cases.

Example 3.2. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALC}$ KB where $\mathcal{T} = \{\, A \sqsubseteq B,\ B \sqsubseteq C,\ C \sqsubseteq \forall R.C,\ C \sqsubseteq D \,\}$, and $\mathcal{A} = \{\, B(a),\ R(a, b) \,\}$.

Take $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ where $\mathcal{T}_1 = \{\, A \sqsubseteq C,\ C \sqsubseteq \forall R.C,\ C \sqsubseteq D \,\}$ and $\mathcal{A}_1 = \{\, C(a),\ R(a, b) \,\}$. Then $\mathcal{K}_1$ is a result of forgetting about concept B in $\mathcal{K}$.

However, there does not exist a result of forgetting about $\{B, C\}$ in $\mathcal{K}$. To understand this, we note that the result of forgetting about $\{B, C\}$ in $\mathcal{K}$ should include the following inclusions:

$A \sqsubseteq D$, $A \sqsubseteq \forall R.D$, $A \sqsubseteq \forall R.\forall R.D$, ... , and
$D(a)$, $(\forall R.D)(a)$, $(\forall R.\forall R.D)(a)$, ... , and
$R(a, b)$, $D(b)$, $(\forall R.D)(b)$, $(\forall R.\forall R.D)(b)$, ...

Thus, there is no finite $\mathcal{ALC}$ KB which is equivalent to the above infinite set of inclusions.

If the result of forgetting about $\mathcal{V}$ in $\mathcal{K}$ is expressible as an $\mathcal{ALC}$ KB, we say $\mathcal{V}$ is *forgettable* from $\mathcal{K}$.

In the rest of this section, we present some desirable properties of forgetting in KBs for $\mathcal{ALC}$.

Proposition 3.1. *Let $\mathcal{K}$ be a KB in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. If both $\mathcal{K}'$ and $\mathcal{K}''$ in $\mathcal{ALC}$ are results of forgetting about $\mathcal{V}$ in $\mathcal{K}$, then $\mathcal{K}' \equiv \mathcal{K}''$.*

This proposition says that the result of forgetting in $\mathcal{ALC}$ is unique up to KB equivalence. Given this result, we write $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ to denote any result of forgetting about $\mathcal{V}$ in $\mathcal{K}$ in $\mathcal{ALC}$. In particular, $\mathsf{forget}(\mathcal{K}, \mathcal{V}) = \mathcal{K}'$ means that $\mathcal{K}'$ is a result of forgetting about $\mathcal{V}$ in $\mathcal{K}$.

In fact, forgetting in TBoxes is independent of ABoxes as the next result shows.

Proposition 3.2. *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and $\mathcal{V}$ a set of variables. Then, for any $\mathcal{ALC}$ ABox $\mathcal{A}$, $\mathcal{T}'$ is the TBox of $\mathsf{forget}((\mathcal{T}, \mathcal{A}), \mathcal{V})$ iff $\mathcal{T}'$ is the TBox of $\mathsf{forget}((\mathcal{T}, \emptyset), \mathcal{V})$.*

For simplicity, we write $\mathsf{forget}(\mathcal{T}, \mathcal{V})$ for $\mathsf{forget}((\mathcal{T}, \emptyset), \mathcal{V})$ and call it the result of TBox-forgetting about $\mathcal{V}$ in $\mathcal{T}$.

The following result, which generalizes Proposition 3.1, shows that forgetting preserves implication and equivalence relations between KBs.

Proposition 3.3. *Let $\mathcal{K}_1, \mathcal{K}_2$ be two KBs in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. Then*

- *$\mathcal{K}_1 \models \mathcal{K}_2$ implies $\mathsf{forget}(\mathcal{K}_1, \mathcal{V}) \models \mathsf{forget}(\mathcal{K}_2, \mathcal{V})$;*
- *$\mathcal{K}_1 \equiv \mathcal{K}_2$ implies $\mathsf{forget}(\mathcal{K}_1, \mathcal{V}) \equiv \mathsf{forget}(\mathcal{K}_2, \mathcal{V})$.*

However, the converse of Proposition 3.3 is not true in general. Consider $\mathcal{K}$ and $\mathcal{K}_1$ in Example 3.2, it is obvious that $\mathsf{forget}(\mathcal{K}, \{B\}) \equiv \mathsf{forget}(\mathcal{K}_1, \{B\})$. However, $\mathcal{K}$ and $\mathcal{K}_1$ are not equivalent.

Consistency and query answering are two major reasoning tasks in description logics. It is a key requirement for a reasonable definition of forgetting to preserve these two reasoning forms.

Proposition 3.4. *Let $\mathcal{K}$ be a KB in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. Then*

1. *$\mathcal{K}$ is consistent iff $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ is consistent;*
2. *for any inclusion or assertion α not containing variables in $\mathcal{V}$, $\mathcal{K} \models \alpha$ iff $\mathsf{forget}(\mathcal{K}, \mathcal{V}) \models \alpha$.*

The next result shows that the forgetting operation can be divided into steps, with a part of the signature forgotten in each step.

Proposition 3.5. *Let $\mathcal{K}$ be a KB in $\mathcal{ALC}$ and $\mathcal{V}_1, \mathcal{V}_2$ two sets of variables. Then we have*

$$\mathsf{forget}(\mathcal{K}, \mathcal{V}_1 \cup \mathcal{V}_2) \equiv \mathsf{forget}(\mathsf{forget}(\mathcal{K}, \mathcal{V}_1), \mathcal{V}_2).$$

To compute the result of forgetting about $\mathcal{V}$ in $\mathcal{K}$, it is equivalent to forget the variables in $\mathcal{V}$ one by one.

4 Forgetting in $\mathcal{ALC}$ Concept Descriptions

Forgetting in a concept description has been investigated under the name of *uniform interpolation* in [19]. In this section, we reformulate the definition of the forgetting about concept and role names in $\mathcal{ALC}$ concept descriptions (briefly, c-forgetting) and introduce some results that will be used in the next section. From the view point of ontology management, the issue of forgetting in concept descriptions is less important than that for KBs and TBoxes. However, we will show later that c-forgetting can be used to provide an approximation algorithm for KB-forgetting in $\mathcal{ALC}$, as well as its theoretical importance.

Intuitively, the result C' of forgetting about a set of variables from a concept description C should be weaker than C but as close to C as possible. For example, after the concept Male is forgotten from a concept description for "Male Australian student" $\mathit{Australians} \sqcap \mathit{Students} \sqcap \mathit{Male}$, then we should obtain a concept description $\mathit{Australians} \sqcap \mathit{Students}$ for "Australian student". More specifically, C' should be a concept description that defines a minimal concept description among all concept descriptions that subsumes C and is syntactically irrelevant to $\mathcal{V}$ (i.e. variables in $\mathcal{V}$ do not appear in the concept description).

Definition 4.1 (c-forgetting). *Let C be a concept description in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. A concept description C' on the signature* $\mathsf{sig}(C) - \mathcal{V}$ *is a* result of c-forgetting *about $\mathcal{V}$ in C if the following conditions are satisfied:*

(CF1) $\models C \sqsubseteq C'$.

(CF2) *For every $\mathcal{ALC}$ concept description C'' with* $\mathsf{sig}(C'') \cap \mathcal{V} = \emptyset$, $\models C \sqsubseteq C''$ *implies* $\models C' \sqsubseteq C''$.

The above (CF1) and (CF2) correspond to the conditions (2) and (3) of Theorem 8 in [19]. A fundamental property of c-forgetting in $\mathcal{ALC}$ concept descriptions is that the result of c-forgetting is unique under concept description equivalence.

Proposition 4.1. *Let C be a concept description in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. If two concept descriptions C' and C'' in $\mathcal{ALC}$ are results of c-forgetting about $\mathcal{V}$ in C, then $\models C' \equiv C''$.*

As all results of c-forgetting are equivalent, we write $\mathsf{forget}(C, \mathcal{V})$ to denote an arbitrary result of c-forgetting about $\mathcal{V}$ in C.

Example 4.1. Suppose the concept "Research Student" is defined by $C = \mathit{Student} \sqcap (\mathit{Master} \sqcup \mathit{PhD}) \sqcap \exists \mathit{supervised}.\mathit{Professor}$ where "Master", "PhD" and "Professor" are all concepts; "supervised" is a role and $\mathit{supervised}(x, y)$ means that x is supervised by y. If the concept description C is used only for students, we may wish to forget about $\mathit{Student}$: $\mathsf{forget}(C, \mathit{Student}) = (\mathit{Master} \sqcup \mathit{PhD}) \sqcap \exists \mathit{supervised}.\mathit{Professor}$. If we do not require that a supervisor for a research student must be a professor, then the filter "Professor" can be forgotten: $\mathsf{forget}(C, \mathit{Professor}) = \mathit{Student} \sqcap (\mathit{Master} \sqcup \mathit{PhD}) \sqcap \exists \mathit{supervised}.\top$.

A concept description C is *satisfiable* if $C^{\mathcal{I}} \neq \emptyset$ for some interpretation $\mathcal{I}$ on $\mathsf{sig}(C)$. C is unsatisfiable if $\models C \equiv \bot$. By Definition 4.1, c-forgetting also preserves satisfiability of concept descriptions.

Proposition 4.2. *Let C be a concept description in $\mathcal{ALC}$, and $\mathcal{V}$ be a set of variables. Then C is satisfiable iff $\mathsf{forget}(C, \mathcal{V})$ is satisfiable.*

Similar to forgetting in KB, the c-forgetting operation can be divided into steps.

Proposition 4.3. *Let C be a concept description in $\mathcal{ALC}$ and $\mathcal{V}_1, \mathcal{V}_2$ two sets of variables. Then we have*

$$\models \mathsf{forget}(C, \mathcal{V}_1 \cup \mathcal{V}_2) \equiv \mathsf{forget}(\mathsf{forget}(C, \mathcal{V}_1), \mathcal{V}_2).$$

Given the above result, when we want to forget about a set of variables, they can be forgotten one by one. Also, the ordering of c-forgetting operation is irrelevant to the result.

Corollary 4.1. *Let C be a concept description in $\mathcal{ALC}$ and let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a set of variables. Then, for any permutation $(i_1, i_2, \ldots, i_n)$ of $\{1, 2, \ldots, n\}$,*

$$\models \mathsf{forget}(\mathsf{forget}(\mathsf{forget}(C, V_{i_1}), V_{i_2}), \ldots), V_{i_n}) \equiv \\ \mathsf{forget}(\mathsf{forget}(\mathsf{forget}(C, V_1), V_2), \ldots), V_n).$$

The following result, which is not obvious, shows that c-forgetting distributes over union $\sqcup$.

Proposition 4.4. *Let $C_1, \ldots, C_n$ be concept descriptions in $\mathcal{ALC}$. For any set $\mathcal{V}$ of variables, we have*
$\models \mathsf{forget}(C_1 \sqcup \cdots \sqcup C_n, \mathcal{V}) \equiv \mathsf{forget}(C_1, \mathcal{V}) \sqcup \cdots \sqcup \mathsf{forget}(C_n, \mathcal{V}).$

However, c-forgetting for $\mathcal{ALC}$ does not distribute over intersection $\sqcap$. For example, if the concept description $C = A \sqcap \neg A$, then $\mathsf{forget}(C, A) = \bot$, since $\models C \equiv \bot$. But $\mathsf{forget}(A, A) \sqcap \mathsf{forget}(\neg A, A) \equiv \top$.

An important reason for this is that c-forgetting does not distribute over negation. Actrually, we have

$$\models \neg\mathsf{forget}(C, \mathcal{V}) \sqsubseteq \neg C \sqsubseteq \mathsf{forget}(\neg C, \mathcal{V}).$$

The next result shows that c-forgetting distributes over quantifiers. Since c-forgetting does not distribute over negation, the two statements in the following proposition do not necessarily imply each other. The proof uses tableau reasoning for $\mathcal{ALC}$ and is surprisingly complex.

Proposition 4.5. *Let C be a concept description in $\mathcal{ALC}$, R be a role name and $\mathcal{V}$ be a set of variables. Then*

– $\mathsf{forget}(\forall R.C, \mathcal{V}) = \top$ *for* $R \in \mathcal{V}$, *and* $\mathsf{forget}(\forall R.C, \mathcal{V}) = \forall R.\mathsf{forget}(C, \mathcal{V})$ *for* $R \notin \mathcal{V}$;

– $\mathsf{forget}(\exists R.C, \mathcal{V}) = \top$ *for* $R \in \mathcal{V}$*, and* $\mathsf{forget}(\exists R.C, \mathcal{V}) = \exists R.\mathsf{forget}(C, \mathcal{V})$. *for* $R \notin \mathcal{V}$*;*

These results suggest a way of computing c-forgetting about set $\mathcal{V}$ of variables in a complex $\mathcal{ALC}$ concept description C. That is, to forget about each variable V in $\mathcal{V}$ one after another, and to distribute the c-forgetting computation to subconcepts of C.

In what follows, we introduce an algorithm for computing the result of c-forgetting through rewriting of concept descriptions (syntactic concept transformations) [19]. This algorithm consists of two stages: (1) C is first transformed into an equivalent disjunctive normal form (DNF), which is a disjunction of conjunctions of simple concept descriptions; (2) the result of c-forgetting about $\mathcal{V}$ in each such simple concept description is obtained by removing some parts of the conjunct.

Before we introduce disjunctive normal form (DNF), some notation and definitions are in order. We call an (atomic) concept A or its negation $\neg A$ a *literal concept* or simply *literal*. An *pseudo-literal* with role R is a concept description of the form $\exists R.F$ or $\forall R.F$, where R is a role name and F is an arbitrary concept. A *generalized literal* is either a literal or a pseudo-literal.

First, every arbitrary concept description can be transformed into an equivalent disjunction of conjunctions of generalized literals. This is a very basic DNF for $\mathcal{ALC}$.

Definition 4.2. *A concept description D is in* disjunctive normal form *(DNF) if $D = \bot$ or $D = \top$ or D is a disjunction of conjunctions of generalized literals $D = D_1 \sqcup \cdots \sqcup D_n$, where each $D_i \not\equiv \bot$ $(1 \leq i \leq n)$ is a conjunction $\bigsqcap L$ of literals, or of the form*

$$\bigsqcap L \sqcap \bigsqcap_{R \in \mathcal{R}} \left[\forall R.U_R \sqcap \bigsqcap_k \exists R.(E_R^{(k)} \sqcap U_R) \right]$$

where $\mathcal{R}$ is the set of role names that occur in D_i, and each U_R and each $E_R^{(k)} \sqcap U_R$ is a concept description in DNF.

We note that, to guarantee the correctness of the algorithm, the above DNF for $\mathcal{ALC}$ is more complex than we have in classical logic and DL-Lite.

Each concept description in $\mathcal{ALC}$ can be transformed into an equivalent one in DNF by the following two steps: (1) first transform the given concept description into a disjunction of conjunctions of pseudo-literals using De Morgan's laws, distributive laws and necessary simplifications, and then (2) for each conjunction in the resulting concept description, perform the following three laws in order:

$$\begin{aligned} C &\leadsto \forall R.\top \sqcap C, \quad \text{for } C = \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_m, m > 0 \\ \forall R.C_1 \sqcap \exists R.C_2 &\leadsto \forall R.C_1 \sqcap \exists R.(C_1 \sqcap C_2) \\ \forall R.C_1 \sqcap \cdots \sqcap \forall R.C_n &\leadsto \forall R.(C_1 \sqcap \cdots \sqcap C_n). \end{aligned}$$

The first transformation above is to transform a concept description containing only existential quantifier into the normal form. For example, if C is concept name, $\exists R.C$, which is not in normal form, can be transformed into the normal form $\forall R.\top \sqcap \exists R.C$. While the second is to assemble several quantifications with the same role name into a single one, the third is crucial for guaranteeing the correctness of our algorithm.

Algorithm 1 (Compute C-Forgetting)
Input: An $\mathcal{ALC}$ concept description C and a set $\mathcal{V}$ of variables in C.
Output: $\mathsf{forget}(C, \mathcal{V})$.
Method:
Step 1. Transform C into its DNF D. If D is $\top$ or $\bot$, return D; otherwise, let $D = D_1 \sqcup \cdots \sqcup D_n$ as in Definition 4.2.
Step 2. For each conjunct E in each D_i, perform the following transformations:

- if E is a literal of the form A or $\neg A$ with $A \in \mathcal{V}$, replace E with $\top$;
- if E is a pseudo-literal in the form of $\forall R.F$ or $\exists R.F$ with $R \in \mathcal{V}$, replace E with $\top$;
- if E is a pseudo-literal in the form of $\forall R.F$ or $\exists R.F$ where $R \notin \mathcal{V}$, replace F with $\mathsf{forget}(F, \mathcal{V})$, and replace each resulting $\forall R.(\top \sqcup F)$ with $\top$.

Step 3. Return the resulting concept description as $\mathsf{forget}(C, \mathcal{V})$.

Fig. 1. Forgetting in concept descriptions.

Once an $\mathcal{ALC}$ concept description D is in the normal form, the result of c-forgetting about a set $\mathcal{V}$ of variables in D can be obtained from D by simple symbolic manipulations (ref. Algorithm 1).

According to Algorithm 1, an input concept description must first be transformed into the normal form before the steps for forgetting are applied. For instance, if we want to forget A in the concept description $D = A \sqcap \neg A \sqcap B$, D is transformed into the normal form, which is $\bot$, and then obtain $\mathsf{forget}(D, A) = \bot$. We note that B is not a result of forgetting about A in D.

Example 4.2. Given a concept $D = (A \sqcup \exists R.\neg B) \sqcap \forall R.(B \sqcup C)$, we want to forget about concept name B in D. In Step 1 of Algorithm 1, D is firstly transformed into its DNF $D' = [A \sqcap \forall R.(B \sqcup C)] \sqcup [\forall R.(B \sqcup C) \sqcap \exists R.(\neg B \sqcap C)]$. Note that $\exists R.(\neg B \sqcap C)$ is transformed from $\exists R.[\neg B \sqcap (B \sqcup C)]$. Then in Step 2, each occurrence of B in D' is replaced by $\top$, and $\forall R.(\top \sqcup F)$ is replaced with $\top$. We obtain $\mathsf{forget}(D, \{B\}) = A \sqcup \exists R.C$. To forget about role R in D, Algorithm 1 replaces each pseudo-literals in D' of the form $\forall R.F$ or $\exists R.F$ with $\top$, and returns $\mathsf{forget}(D, \{R\}) = \top$.

Obviously, the major cost of Algorithm 1 is from transforming the given concept description into its DNF. For this reason, the algorithm is exponential time in the worst case. However, if the concept description C is in DNF, Algorithm 1 takes only linear time (w.r.t. the size of C) to compute the result of c-forgetting about $\mathcal{V}$ in C. And the result of c-forgetting is always in DNF.

Theorem 4.1. *Let $\mathcal{V}$ be a set of concept and role names and C a concept description in $\mathcal{ALC}$. Then Algorithm 1 always returns* $\mathsf{forget}(C, \mathcal{V})$.

Using the Tableau for $\mathcal{ALC}$, we have established a proof for Theorem 4.1.

5 Approximate Forgetting in $\mathcal{ALC}$ Ontologies

The basic idea of forgetting about a signature $\mathcal{V}$ in a KB $\mathcal{K}$ is to obtain a new KB $\mathcal{K}'$ such that $\mathcal{K}'$ and $\mathcal{K}$ are equivalent w.r.t. all consequences (assertions and inclusions) that are irrelevant to $\mathcal{V}$. An important observation is that the sizes of those consequences in an application often have an upper bound N. The problem of computing forgetting in KBs of expressive DLs including $\mathcal{ALC}$ is quite hard in general. Instead of computing the result $\mathcal{K}'$ of KB-forgetting, we propose to compute an approximation $\mathcal{K}''$ to $\mathcal{K}'$, i. e. $\mathcal{K}''$ and $\mathcal{K}$ are equivalent w.r.t. consequences that are irrelevant to $\mathcal{V}$ and whose sizes are bounded by N. To this end, we introduce a non-decreasing sequence of KBs (the n-th KB is called n-forgetting) and the result of KB-forgetting is the limit of this sequence. Moreover, each KB in the sequence in turn can be computed using Algorithm 1 for computing c-forgetting. So the results in this section essentially provide a novel way of approximating the result of KB-forgetting. Another advantage of n-forgetting is that there always exists a result of n-forgetting and thus its existence does not dependent on the existence of KB-forgetting.

As a special case, we first introduce an approximation to TBox-forgetting. Example 3.2 shows that, for some TBox $\mathcal{T}$, $\mathsf{forget}(\mathcal{T},\mathcal{V})$ may not be expressible as a finite $\mathcal{ALC}$ TBox. Thus, it is natural to consider a sequence of (finite) TBoxes that approximate the result of forgetting in $\mathcal{T}$ in the sense that the sequence is non-decreasing in terms of logical implication and the limit of the sequence is the result of forgetting. Such a consequence is constructed by using results developed for c-forgetting in Section 4.

We note that, for an inclusion $C \sqsubseteq D$ in $\mathcal{T}$, $\mathsf{forget}(C,\mathcal{V}) \sqsubseteq \mathsf{forget}(D,\mathcal{V})$ may not be a logical consequence of $\mathcal{T}$ and thus may not be in $\mathsf{forget}(\mathcal{T},\mathcal{V})$. However, if we transform $\mathcal{T}$ into an equivalent singleton TBox $\{\top \sqsubseteq C_{\mathcal{T}}\}$, where $C_{\mathcal{T}} = \bigsqcap_{C\sqsubseteq D\in\mathcal{T}}(\neg C \sqcup D)$, then inclusion α_0 of the form $\top \sqsubseteq \mathsf{forget}(C_{\mathcal{T}},\mathcal{V})$ is a logical consequence of $\mathcal{T}$. In general, the singleton TBox $\{\alpha_0\}$ is not necessarily equivalent to $\mathsf{forget}(\mathcal{T},\mathcal{V})$. However, it can be a starting point of a sequence whose limit is $\mathsf{forget}(\mathcal{T},\mathcal{V})$. Note that $\mathcal{T}$ is also equivalent to $\{\top \sqsubseteq C_{\mathcal{T}} \sqcap \forall R.C_{\mathcal{T}}\}$ for an arbitrary role name R in $\mathcal{T}$. Hence, inclusion α_1 of the form $\top \sqsubseteq \mathsf{forget}(C_{\mathcal{T}} \sqcap \forall R.C_{\mathcal{T}},\mathcal{V})$ is a logical consequence of $\mathcal{T}$, and it can be shown that TBox $\{\alpha_1\}$ is logically stronger then $\{\alpha_0\}$. That is, $\mathsf{forget}(\mathcal{T},\mathcal{V}) \models \{\alpha_1\} \models \{\alpha_0\}$. Let α_2 be $\top \sqsubseteq \mathsf{forget}(C_{\mathcal{T}} \sqcap \forall R.C_{\mathcal{T}} \sqcap \forall R.\forall R.C_{\mathcal{T}},\mathcal{V})$, then we have $\mathsf{forget}(\mathcal{T},\mathcal{V}) \models \{\alpha_2\} \models \{\alpha_1\} \models \{\alpha_0\}$. In this way, we can construct a sequence of TBoxes with increasing logical strength, whose limit is $\mathsf{forget}(\mathcal{T},\mathcal{V})$.

For $n \geq 0$, define

$$C_{\mathcal{T}}^{(n)} = \bigsqcap_{k=0}^{n} \; \bigsqcap_{R_1,\ldots,R_k\in\mathcal{R}} \forall R_1 \cdots \forall R_k.C_{\mathcal{T}}$$

where $C_{\mathcal{T}} = \bigsqcap_{C\sqsubseteq D\in\mathcal{T}}(\neg C \sqcup D)$ and $\mathcal{R}$ is the set of role names in $\mathcal{K}$.

We now define a sequence of TBoxes, which essentially provides an approximation to the result of TBox-forgetting.

Definition 5.1. *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and $\mathcal{V}$ be a set of variables. For each $n \geq 0$, the TBox*

$$\mathsf{forget}^n(\mathcal{T},\mathcal{V}) = \{\, \top \sqsubseteq \mathsf{forget}(C_{\mathcal{T}}^{(n)},\mathcal{V}) \,\}$$

*is called the n-*forgetting *about $\mathcal{V}$ in $\mathcal{T}$.*

Note that the above n-forgetting for TBoxes is defined in terms of forgetting in concept descriptions (c-forgetting).

Example 5.1. Consider the TBox $\mathcal{T}$ in Example 3.2, we have $C_{\mathcal{T}} = (\neg A \sqcup B) \sqcap (\neg B \sqcup C) \sqcap (\neg C \sqcup \forall R.C) \sqcap (\neg C \sqcup D)$, and $C_{\mathcal{T}}^{(0)} = C_{\mathcal{T}}$, $C_{\mathcal{T}}^{(1)} = C_{\mathcal{T}} \sqcap \forall R.C_{\mathcal{T}}, \ldots,$ $C_{\mathcal{T}}^{(n)} = C_{\mathcal{T}} \sqcap \forall R.C_{\mathcal{T}}^{(n-1)}$ $(n \geq 2)$.

Let $\mathcal{V} = \{B, C\}$. For each $n \geq 0$, the $\mathsf{forget}^n(\mathcal{T}, \mathcal{V})$ can be computed as follows.
$\mathsf{forget}^0(\mathcal{T}, \mathcal{V}) = \{\, \top \sqsubseteq \neg A \sqcup D \,\}$, which is equivalent to $\{\, A \sqsubseteq D \,\}$.
$\mathsf{forget}^1(\mathcal{T}, \mathcal{V}) = \{\, \top \sqsubseteq \neg A \sqcup (D \sqcap \forall R.D) \,\}$, which is $\{\, A \sqsubseteq D,\ A \sqsubseteq \forall R.D \,\}$.
......
$\mathsf{forget}^n(\mathcal{T}, \mathcal{V}) = \{\, A \sqsubseteq D,\ A \sqsubseteq \forall R.D,\ \ldots,\ A \sqsubseteq \underbrace{\forall R.\forall R \cdots \forall R}_{n\ Rs}.D \,\}$.

We call $(\bigsqcap_{i=1}^{n} C_i)(a)$ the *conjunction* of assertions $C_1(a), \ldots, C_n(a)$ while $(\bigsqcup_{i=1}^{n} C_i)(a)$ is called the *disjunction* of these assertions.

Before we can perform forgetting on a given ABox, we need to preprocess it and thus transform it into a normal form. To this end, we give the following definition.

Definition 5.2. *An ABox $\mathcal{A}$ in $\mathcal{ALC}$ is* complete *if for any individual name a in $\mathcal{A}$ and assertion $C(a)$ with $\mathcal{A} \models C(a)$, we have $\models C' \sqsubseteq C$, where $C'(a)$ is the conjunction of all the concept assertions about a in $\mathcal{A}$.*

For example, ABox $\mathcal{A} = \{\forall R.A(a), R(a, b)\}$ is incomplete, because $\mathcal{A} \models A(b)$ whereas no such assertion $C(b)$ exists in $\mathcal{A}$ that $\models C \sqsubseteq A$. After adding assertions $A(b)$, $\exists R.A(a)$ and $\top(a)$ into $\mathcal{A}$, the resulting ABox is complete.

In complete ABoxes, concept assertion entailment can be reduced to concept subsumption, and is independent of role assertions.

However, there exist incomplete ABoxes that are not equivalent any (finte) complete ABox. For example, the ABox $\{R(a, b), R'(b, a)\}$ has infinitely many logical consequences of the form $(\exists R.\exists R'.C \sqcup \neg C)(a)$ where C is an arbitrary concept description. This kind of situations are caused by certain cycles in ABoxes. We say an ABox is *acyclic* if there exists no cycle of the form $R_1(a, a_1), R_2(a_1, a_2), \ldots, R_i(a_i, a)$ in $\mathcal{A}$ in the ABox.

Note all the role assertions in an acyclic ABox form tree-shape relations between individuals. We call an individual without any predecessor a *root* individual, and that without any successor a *leaf* individual.

Algorithm 2 is developed to transform a given acyclic $\mathcal{ALC}$ ABox into an equivalent complete ABox. The correctness of the algorithm shows that any acyclic ABox can be transformed to an equivalent complete ABox in $\mathcal{ALC}$.

Note that Algorithm 2 always terminates.

Lemma 5.1. *Given an acyclic $\mathcal{ALC}$ ABox $\mathcal{A}$, Algorithm 2 always returns a complete ABox $\mathcal{A}'$ that is equivalent to $\mathcal{A}$.*

With the notion of complete ABox, we can extend n-forgetting in TBoxes and define n-forgetting for an $\mathcal{ALC}$ KB as follows.

Algorithm 2 (Complete an acyclic ABox)
Input: An acyclic $\mathcal{ALC}$ ABox $\mathcal{A}$.
Output: An equivalent complete $\mathcal{ALC}$ ABox $\mathcal{A}'$.
Method:
Step 1. Starting from root individuals, for each individual a and each role assertion $R(a,b)$ in $\mathcal{A}$, let $C(a)$ be the conjunction of all the concept assertions about a in $\mathcal{A}$.
Transform C into its DNF $C = \bigsqcup_{i=1}^{n} D_i$ as in Definition 4.2. Let $\forall R.U_i$ be the universal quantified conjunct of R in D_i. Add $(\bigsqcup_{i=1}^{n} U_i)(b)$ to $\mathcal{A}$.
Step 2. For each individual a in $\mathcal{A}$, add $\top(a)$ to $\mathcal{A}$.
Step 3. Starting from leaf individuals, for each individual b and each role assertion $R(a,b)$ in $\mathcal{A}$, add assertion $(\exists R.E)(a)$ to $\mathcal{A}$, where $E(b)$ is the conjunction of all the concept assertions about b in $\mathcal{A}$.
Step 4. Return the resulting ABox.

Fig. 2. Transform an acyclic ABox into a complete ABox.

Definition 5.3. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALC}$ KB with $\mathcal{A}$ being an acyclic ABox and $\mathcal{V}$ a set of variables. For each $n \geq 0$, the KB $\mathsf{forget}^n(\mathcal{K}, \mathcal{V}) = (\mathcal{T}', \mathcal{A}')$ is called the result of n-*forgetting *about $\mathcal{V}$ in $\mathcal{K}$, where $\mathcal{T}' = \mathsf{forget}^n(\mathcal{T}, \mathcal{V}) = \{ \top \sqsubseteq \mathsf{forget}(C_{\mathcal{T}}^{(n)}, \mathcal{V}) \}$ and $\mathcal{A}'$ is obtained from $\mathcal{A}$ through the following steps:*

1. *For each individual name a in $\mathcal{A}$, add $C_{\mathcal{T}}^{(n)}(a)$ to $\mathcal{A}$.*
2. *Apply Algorithm 2 to obtain a complete ABox, still denoted $\mathcal{A}$.*
3. *For each individual name a in $\mathcal{A}$, replace $C(a)$ with $(\mathsf{forget}(C, \mathcal{V}))(a)$, where $C(a)$ is the conjunction of all the concept assertions about a in $\mathcal{A}$.*
4. *Remove each $R(a,b)$ from $\mathcal{A}$ where $R \in \mathcal{V}$.*

The basic idea behind Definition 5.3 is to transform the given KB into a new KB such that forgetting can be done in its ABox and TBox, separately, in terms of c-forgetting for individual assertions and inclusions.

Example 5.2. Consider the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ in Example 3.2 and let $\mathcal{V} = \{B, C\}$.

For each $n \geq 0$, let $\mathcal{A}_n$ be the ABox of $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$. We will only elabourate the computation of $\mathcal{A}_0$ as follows: Note that $\mathcal{A}$ is acyclic. First $C_{\mathcal{T}}^{(0)}(a)$ and $C_{\mathcal{T}}^{(0)}(b)$ are added into $\mathcal{A}$, where $C_{\mathcal{T}}^{(n)}$ is the same as in Example 5.1. After applying Algorithm 2 to $\mathcal{A}$, the resulting ABox is equivalent to
$\{ (B \sqcap C \sqcap \forall R.C \sqcap D)(a),\ R(a,b),\ ((\neg A \sqcup B) \sqcap C \sqcap \forall R.C \sqcap D)(b),$
$\exists R.((\neg A \sqcup B) \sqcap C \sqcap \forall R.C \sqcap D)(a) \}$

By applying c-forgetting to the conjunctions of concept assertions about a and b, we obtain $\mathcal{A}_0 = \{ D(a),\ R(a,b),\ D(b) \}$.

Similarly, we can compute $\mathcal{A}_1, \ldots, \mathcal{A}_n$ as:

$$\mathcal{A}_1 = \{ D(a),\ (\forall R.D)(a),\ R(a,b),\ D(b),\ (\forall R.D)(b) \}.$$
$$\cdots\cdots$$
$$\mathcal{A}_n = \{ D(a),\ (\forall R.D)(a),\ \ldots,\ (\underbrace{\forall R.\forall R \cdots \forall R}_{n\ Rs}.D)(a),\ R(a,b),$$
$$D(b),\ (\forall R.D)(b),\ \ldots,\ (\underbrace{\forall R.\forall R \cdots \forall R}_{n\ Rs}.D)(b) \}.$$

The following result shows that n-forgetting preserves logical consequences of the original KB.

Given a concept description C, let $|C|$ be the number of all different subconcepts of C. For a TBox $\mathcal{T}$, define $|\mathcal{T}| = \sum_{C \sqsubseteq D \in \mathcal{T}}(|C| + |D|)$. Similarly, for an ABox $\mathcal{A}$, define $|\mathcal{A}| = \sum_{C(a) \in \mathcal{A}} |C|$. Then for a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $|\mathcal{K}| = |\mathcal{T}| + |\mathcal{A}|$.

Proposition 5.1. *Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ be a set of variables. Then* $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$ *satisfies the following conditions:*

1. $\mathcal{K} \models \mathsf{forget}^n(\mathcal{K}, \mathcal{V})$.
2. *Let C and D be two concept descriptions containing no variable in $\mathcal{V}$. If $n \geq 2^{|C|+|D|+|\mathcal{K}|}$, then $\mathcal{K} \models C \sqsubseteq D$ iff* $\mathsf{forget}^n(\mathcal{K}, \mathcal{V}) \models C \sqsubseteq D$.
3. *Let C be a concept description containing no variable in $\mathcal{V}$, and a an individual name in $\mathcal{K}$. If $n \geq 2^{|C|+|\mathcal{K}|}$, then $\mathcal{K} \models C(a)$ iff* $\mathsf{forget}^n(\mathcal{K}, \mathcal{V}) \models C(a)$.
4. *Let R be a role name not in $\mathcal{V}$, and a, b two individual names in $\mathcal{K}$. Then $\mathcal{K} \models R(a, b)$ iff* $\mathsf{forget}^n(\mathcal{K}, \mathcal{V}) \models R(a, b)$.

Recall from the definition of KB-forgetting that, with respect to inclusions and assertions not containing variables in $\mathcal{V}$, $\mathcal{K}$ is logically equivalent to $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. Proposition 5.1 tells us that if we know *which* inclusions and assertions not containing variables in $\mathcal{V}$ we wish to reason about in advance, then we can derive a value for n, compute $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$, and use the fact that, with respect to these inclusions and assertions, $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$ is logically equivalent to $\mathcal{K}$ and hence to $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. In this way, we can use $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$ as a practical approximation to $\mathsf{forget}(\mathcal{K}, \mathcal{V})$.

The above proposition shows that, for any $n \geq 0$, each $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$ is logically weaker than $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. Also, as the number n is sufficiently large, $\mathsf{forget}^n(\mathcal{K}, \mathcal{V})$ preserves more and more consequences of $\mathcal{K}$. Therefore, the sequence of KBs $\{\mathsf{forget}^n(\mathcal{K}, \mathcal{V})\}_{n \geq 0}$ is non-decreasing w.r.t. semantic consequence as the next proposition shows.

Proposition 5.2. *Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ a set of variables. Then, for any $n \geq 0$, we have* $\mathsf{forget}^{n+1}(\mathcal{K}, \mathcal{V}) \models \mathsf{forget}^n(\mathcal{K}, \mathcal{V})$.

Based on the above two results, we can show the main theorem of this section as follows, which states that the limit of the sequence of n-forgettings captures the result of forgetting.

Theorem 5.1. *Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ a set of variables. Then*

$$\mathsf{forget}(\mathcal{K}, \mathcal{V}) = \bigcup_{n=0}^{\infty} \mathsf{forget}^n(\mathcal{K}, \mathcal{V}).$$

So, by Theorem 5.1, we can compute $\mathsf{forget}(\mathcal{K}, \mathcal{V})$, if it exists, using algorithms introduced in the paper.

Corollary 5.1. *Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ be a set of variables. $\mathcal{V}$ is forgettable from $\mathcal{K}$ if and only if there exists $N \geq 0$ such that* $\mathsf{forget}^n(\mathcal{K}, \mathcal{V}) \equiv \mathsf{forget}^N(\mathcal{K}, \mathcal{V})$ *for all $n \geq N$. In this case,* $\mathsf{forget}(\mathcal{K}, \mathcal{V}) = \mathsf{forget}^N(\mathcal{K}, \mathcal{V})$.

As we can see from Example 3.2, the sizes of consequences (assertions and inclusions) of $\mathcal{K}$ not containing variables in $\mathcal{V}$ do not have an upper bound. If it does not exist, we can always choose n large enough to "approximate" $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. However, two issues are still unclear to us: First, the computation of $\mathsf{forget}^{n+1}(\mathcal{K}, \mathcal{V})$ is not based on $\mathsf{forget}^{n}(\mathcal{K}, \mathcal{V})$. Next, it would be interesting to find a way to measure how close $\mathsf{forget}^{n}(\mathcal{K}, \mathcal{V})$ is from $\mathsf{forget}(\mathcal{K}, \mathcal{V})$.

6 Conclusion

In this paper, we have proposed a theory of forgetting for ontologies represented as knowledge bases in $\mathcal{ALC}$ and shown several important properties of forgetting, which together demonstrate the suitability of our approach. Unlike the case of DL-Lite, forgetting for $\mathcal{ALC}$ KBs may not exist in general because the result of forgetting is an infinite KB in some cases. For this reason, we have developed an algorithm for approximating the result of forgetting in ontologies and shown that the algorithm is sound and complete w.r.t. our semantic definition of forgetting. This new approximation is achieved by employing forgetting for concept descriptions. In our approach, both concepts and roles can be forgotten in $\mathcal{ALC}$ ontologies. The notion of forgetting is a semantic one and thus many properties can be formally proven. The technique of forgetting for DLs has applications in reuse of large ontologies and many other tasks in ontology engineering.

There are still several interesting issues that we are working on: (1) Currently we are working on an implementation prototype of forgetting. Such a forgetting component can be used by ontology editors to enhance their ability of partially reusing large ontologies. The forgetting component is planning to be embedded into Protégé [27]. (2) We are also working on extending the results in this paper to more expressive DLs.

References

1. H. Alani, S. Harris, and B. O'Neil. Winnowing ontologies based on application use. In *Proc. 3rd ESWC*, pp.185–199, 2006.
2. G. Antoniou and F. Harmelen. *A Semantic Web Primer*. MIT Press (2nd Edition), 2008.
3. F. Baader, D. Calvanese, D.McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2002.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pp. 29–37, May, 2001.
5. 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.
6. T. Eiter and K. Wang. Semantic forgetting in answer set programming. *Artificial Intelligence*, 172(14): 1644–1672, 2008.
7. S. Ghilardi, C. Lutz, and F. Wolter. Did i damage my ontology? a case for conservative extensions in description logics. In *Proc. KR'06*, pp.187–197, 2006.
8. B. Grau, Y. Kazakov, I. Horrocks, and U. Sattler. A logical framework for modular integration of ontologies. In *Proc. IJCAI'07*, pp.298–303, 2007.
9. B. Grau, B. Parsia, and E. Sirin. Combining OWL ontologies using e-connections. *Journal of Web Semantics*, 4(1):40–59, 2006.

10. B. Cuenca Grau, Y. Kazakov, I. Horrocks, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. WWW'07*, pp.717–726, 2007.
11. R. Kontchakov, D. Walther, and F. Wolter. The logical difference problem for description logic terminologies. In *Proc. IJCAR'08*, pp.259-274, 2008.
12. R. Kontchakov, D. Walther, and F. Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In *Proc. IJCAI'09*, 2009.
13. R. Kontchakov, F. Wolter, and M. Zakharyaschev. Modularity in DL-Lite. In *Proc. DL'07*, 2007.
14. R. Kontchakov, F. Wolter, and M. Zakharyaschev. Can you tell the difference between DL-Lite ontologies? In *Proc. KR'08*, pp.285–295, 2008.
15. J. Lang, P. Liberatore, and P. Marquis. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res.*, 18:391–443, 2003.
16. F. Lin and R. Reiter. Forget it. In *Proc. AAAI Fall Symposium on Relevance*, pp.154–159, 1994.
17. C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proc. IJCAI'07*, pp. 453–458, 2007.
18. S. Peroni, E. Motta, and M. d'Aquin. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. In *Proc. 3rd ASWC*, pages 242–256, 2008.
19. B. ten Cate, W. Conradie, M. Marx, and Y. Venema. Definitorially complete description logics. In *Proc. KR'06*, pp.79–89, 2006.
20. Z. Wang, K. Wang, R. Topor, and J. Z. Pan. Forgetting concepts in DL-Lite. In *Proc. ESWC'08*, pp.245–257, 2008.
21. Z. Wang, K. Wang, and R. Topor. Forgetting for Knowledge Bases in DL-Lite$_{bool}$. In *Proc. ARCOE'09 (IJCAI'09 Workshop)*, 2009.
22. `http://www.fmrc.org.au/snomed/`
23. `http://www.openclinical.org/prj_galen.html`
24. `http://fma.biostr.washington.edu/`
25. `http://ncit.nci.nih.gov/`
26. `http://www.obofoundry.org/`
27. `http://protege.stanford.edu`
28. `http://www.neon-toolkit.org`
29. `http://www.topquadrant.com/products/TB_Composer.html`