Computational Intelligence, Volume 30, Number 2, 2014

ELIMINATING CONCEPTS AND ROLES FROM ONTOLOGIES IN EXPRESSIVE DESCRIPTIVE LOGICS

KEWEN WANG,[1] ZHE WANG,[2] RODNEY TOPOR,[3] JEFF Z. PAN,[4] AND GRIGORIS ANTONIOU[5]

[1]*Griffith University Brisbane, Queensland, Australia*
[2]*University of Oxford, Oxford, United Kingdom*
[3]*Griffith University, Brisbane, Queensland, Australia*
[4]*University of Aberdeen, Aberdeen, United Kingdom*
[5]*University of Huddersfield, Huddersfield, United Kingdom*

Forgetting is an important tool for reducing ontologies by eliminating some redundant concepts and roles while preserving sound and complete reasoning. Attempts have previously been made to address the problem of forgetting in relatively simple description logics (DLs), such as DL-Lite and extended $\mathcal{EL}$. However, the issue of forgetting for ontologies in more expressive DLs, such as $\mathcal{ALC}$ and OWL DL, is largely unexplored. In particular, the problem of characterizing and computing forgetting for such logics is still open. In this paper, we first define semantic forgetting about concepts and roles in $\mathcal{ALC}$ ontologies and state several important properties of forgetting in this setting. We then define the result of forgetting for concept descriptions in $\mathcal{ALC}$, state the properties of forgetting for concept descriptions, and present algorithms for computing the result of forgetting for concept descriptions. Unlike the case of DL-Lite, the result of forgetting for an $\mathcal{ALC}$ ontology does not exist in general, even for the special case of forgetting in TBoxes. This makes the problem of computing the result of forgetting in $\mathcal{ALC}$ more challenging. We address this problem by defining a series of approximations to the result of forgetting for $\mathcal{ALC}$ ontologies and studying their properties. Our algorithms for computing approximations can be directly implemented as a plug-in of an ontology editor to enhance its ability of managing and reasoning in (large) ontologies.

Received 9 November 2010; Revised 10 April 2012; Accepted 18 April 2012; Published online 26 June 2012

Key words: DLs, forgetting, ontology.

1. INTRODUCTION

The amount of semantically annotated data available on the Web is growing rapidly. For example, it is estimated that there are currently 5 billion linked data items available online (Heath 2009). Accordingly, the Web is rapidly emerging as a large-scale platform for publishing and sharing knowledge using formal models (Peroni, Motta, and d'Aquin 2008). Ontologies have been widely used by automated tools to provide advanced services, such as more accurate web search, intelligent software agents, and knowledge management. An ontology is a formal specification for a common set of terms that are used to describe and represent an application domain. Because more ontologies are used for annotating data on the Web, and because the populated ontologies become larger and more comprehensive, it becomes increasingly important for the future Web to provide abilities for constructing and managing such ontologies. Examples of large ontologies currently in use include the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT)[1] containing 380K concepts, the National Cancer Institute (NCI) Thesaurus[2] containing over 60K axioms, and the OBO Foundry[3] containing about 80 biomedical ontologies.

Although it is expensive to construct large ontologies, it is even more expensive to host, manage, and use a large, comprehensive ontology when a module of the ontology on a

Address correspondence to Kewen Wang, School of Information and Communication Technology, Griffith University, Brisbane, QLD 4111, Australia; e-mail: k.wang@griffith.edu.au

[1] http://www.fmrc.org.au/snomed/

[2] http://ncit.nci.nih.gov/

[3] http://www.obofoundry.org/

smaller alphabet would suffice. Therefore, tools to reduce large ontologies to new ontologies on a smaller alphabet that meet the needs of specific applications aid and encourage the use of existing ontologies. However, because the tool evaluation study in Dzbor et al. (2006) shows, existing tools, such as Protégé,[4] NeOn,[5] and TopBraid,[6] are far from satisfactory for this purpose.

Ontology engineers, thus, face the task of reducing existing, large ontologies to smaller (in terms of alphabet), better focused ontologies by hiding or forgetting irrelevant concepts and roles while preserving required reasoning capabilities. Such ontology reductions can be applied to ontology extraction, ontology summary, ontology integration, and ontology evolution.

We consider two typical 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 to be reused. Consider a scenario discussed in Cuenca Grau, Parsia, and Sirin (2005): 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 ontology in terms of alphabet.

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 Alani, Harris, and O'Neil (2006) and Peroni et al. (2008), 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 (Alani et al. 2006; Peroni et al. 2008). 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 has previously been studied for propositional logic, first-order logic (FOL), and logic programming (Lin and Reiter 1994; Lang, Liberatore, and Marquis 2003; Eiter and Wang 2008), where it has proved a useful technique for reducing a logical theory while preserving sound and complete reasoning in the resulting smaller theory (in terms of alphabet).

However, description logic (DL; Baader et al. 2002) is a different and important knowledge representation framework, which is the basis for ontology languages such as OWL, that are widely used in the Semantic Web.

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

[5] http://www.neon-toolkit.org

[6] http://www.topquadrant.com/products/TB_Composer.html

Although most DLs are equivalent to fragments of FOL, the forgetting for FOL introduced in Lin and Reiter (1994) is not directly applicable to DLs for at least two reasons. First, the correspondence between DLs and FOL does not help much in investigating forgetting for DLs because the result of forgetting in a theory of the FOL may not be expressible in FOL. Second, it is preferable to perform forgetting in DLs directly rather than transforming an ontology into a first-order theory, forgetting and then transforming back to an ontology, because in DLs, we may not need a form of forgetting to preserve all consequences in the FOL and also we want to utilize some advantages of DLs, such as tractability and decidability, in implemting forgetting for DLs.

The issue of forgetting for ontologies in expressive DLs is largely unexplored. In particular, the problem of characterizing and computing forgetting for such logics is still open.

In this paper, we first give a semantic definition of forgetting for ontologies in the DL $\mathcal{ALC}$ and state several important properties of forgetting. We choose $\mathcal{ALC}$ to study in this paper because it allows all boolean operations and quantifiers, and most expressive DLs are based on it. Moreover, some researchers, e.g., Rector, Brandt, and Kola (2008), have argued that practical ontologies, such as, SNOMED CT would benefit from a more expressive DL. Unlike forgetting in DL-Lite (Wang et al. 2010), the result of forgetting for an $\mathcal{ALC}$ ontology does not exist in general, even for the special case of forgetting concepts. This makes the problem of computing forgetting for $\mathcal{ALC}$ ontologies even more challenging. We address this problem by introducing a way of approximating the result of forgetting for $\mathcal{ALC}$ ontologies. This technique of approximation is based on forgetting for concept descriptions in $\mathcal{ALC}$ and algorithms for computing the result of forgetting for concept descriptions. Our algorithms for computing approximations can be embedded into an ontology editor to enhance its ability to manage and reason in large $\mathcal{ALC}$ ontologies.

The contribution of this work can be summarized as follows:

(i) This is the first attempt to investigate the theory of forgetting for TBoxes and KBs, while a definition of interpolation for $\mathcal{ALC}$ TBoxes is mentioned in Definition 17 of Ghilardi, Lutz, and Wolter (2006). However, they did not provide any further results on the concept of interpolation.

(ii) In our paper, forgetting (KB-forgetting) is defined for general KBs and properties of KB-forgetting are investigated. In particular, the properties of KB-forgetting are new, even just for TBoxes.

(iii) We provide a novel way of dealing with the issue of approximating KB-forgetting (n-forgetting). Especially, we present an algorithm for computing n-forgetting for TBoxes and its correctness is proven.

(iv) Although a similar algorithm for computing interpolant for concept descriptions (c-forgetting) was given by ten Cate et al., the proof sketch in their proof does not provide much detail. In this sense, we provide the first detailed proof for the correctness of Algorithm 1.

The work presented here appeared in an abridged form in two conference papers (Wang et al. 2009a,b). Most proofs appear here for the first time as well as some formal properties of forgetting. A few long proofs are included in the Appendix at the end of the paper.

2. RELATED WORK

In ontology engineering, there have been many efforts to deal with large and complex ontologies that are represented in logic-based ontology languages, such as DLs, and the latest

W3C standard (Calvanese et al. 2009). One stream of this research area is to obtain certain modules of a given ontology that is usually large and complex.

In the last few years, *conservative extensions* have been identified as a crucial technique for formalizing modularity in ontology design and reuse (e.g., Antoniou and Kehagias 2000; Cuenca Grau et al. 2005, 2007a,b; Ghilardi et al. 2006; Kontchakov, Wolter, and Zakharyaschev 2007; Lutz, Walther, and Wolter 2007; Cuenca Grau et al. 2008). Informally, an ontology O is a (deductive) conservative extension of another ontology O' with regard to an alphabet Γ (equivalently, O' is a *module* of O) if $O' \subseteq O$ and they are "equivalent" on Γ. For some ontology application domains such as ontology summary, it is required that a module of an ontology should contain only the symbols from a given alphabet Γ (Kontchakov, Wolter, and Zakharyaschev 2008, 2010; Konev, Walther, and Wolter 2009). Although the concept of modules based on conservative extensions is simple and intuitive, in some cases, for a given ontology O, there may not exist such a module containing only symbols in an alphabet Γ. One source for this phenomenon is the requirement that a module must be a subset of the original ontology. For example, consider $O = \{Penguin \sqsubseteq Bird, Bird \sqsubseteq Animal\}$ and $\Gamma = \{Penguin, Animal\}$. Then the ontology O does not have a module containing only symbols in $\{Penguin, Animal\}$. Intuitively, it would be reasonable to treat $O'' = \{Penguin \sqsubseteq Animal\}$ as a module of O, whereas O is not a conservative extension of O'' because O'' is not a subset of O. This simple example demonstrates that the notion of modularity based on deductive conservative extensions is probably insufficient for addressing some issues of modularity and reuse in logic-based ontology languages.

Forgetting generalizes the notion of conservative extensions and the corresponding definition of modules in the sense that a result of forgetting does not necessarily a subset of the original ontology. In the above example, the ontology $O'' = \{Penguin \sqsubseteq Animal\}$ is a result of forgetting about $Bird$ in $O = \{Penguin \sqsubseteq Bird, Bird \sqsubseteq Animal\}$ as we will see later. Attempts have been made to address the issue of forgetting in relatively simple DLs, such as DL-Lite (Wang et al. 2008) and $\mathcal{EL}$ (Konev et al. 2009). In Wang et al. (2008), a definition of forgetting for DL-Lite is introduced, which is based on the classical equivalence of two logical theories, and an algorithm for computing the forgetting is developed. In some practical applications, different notions of forgetting might be needed. For example, one may need only to guarantee that two ontologies are equivalent with regard to a given set of queries (for instance, if the set of queries consists of only concept inclusions, the resulting equivalence is weaker than the classical equivalence in the sense that the new equivalence preserves fewer consequences than classical equivalence). Based on this observation, (Kontchakov et al. 2008, 2010) introduced two variants of forgetting for DL-Lite. (Wang et al. 2010) further developed these results and proposed the notion of *parameterized forgetting*. An approach to forgetting for ontologies in $\mathcal{EL}$ is investigated in (Konev et al. 2009) and it has been used to extract modules from realistic medical ontologies. In our view, the concept of forgetting provides a more suitable approach to modular ontologies. However, as explained in the last section, previous approaches to forgetting for DLs are limited in that forgetting is investigated for only very simple ontology languages and DLs instead of expressive ones such as $\mathcal{ALC}$ and OWL DL.

Another concept that is well known in classical logic and closely related to forgetting is *interpolation*. Originally, it was proposed and investigated in pure mathematical logic, specifically, in proof theory. Given a theory T, the *interpolation property* for T says that if $T \vdash \phi \rightarrow \psi$ for two formulas ϕ and ψ, then there is a formula $I(\phi, \psi)$ in the language containing only the shared symbols, say S, such that $T \vdash \phi \rightarrow I(\phi, \psi)$ and $T \vdash I(\phi, \psi) \rightarrow \psi$. Such a $I(\phi, \psi)$ is referred as an *interpolant*. The notion of *uniform interpolation* is a strengthening of interpolation property such that the interpolant can be obtained from either ϕ and S or from ψ and S. The uniform interpolation for various propositional modal

logics has been investigated, e.g., Visser (1996) and Ghilardi (1995). A definition of uniform interpolation for the DL $\mathcal{ALC}$ is given in ten Cate et al. (2006) and it is used in investigating the definability of TBoxes for $\mathcal{ALC}$. We note that the concept of interpolation for $\mathcal{ALC}$ is only defined for concept descriptions rather than $\mathcal{ALC}$ ontologies.

Forgetting and uniform interpolation have different intuitions behind them and are introduced by different communities. Uniform interpolation is originally investigated as a syntactic concept and forgetting is a semantic one. However, if the axiom system is sound and complete, they can be characterized by each other. In particular, as we will see later, the uniform interpolation is equivalent to the forgetting for *concept descriptions*.

3. DESCRIPTION LOGIC $\mathcal{ALC}$

DLs are a family of concept-based knowledge representation languages. They are equivalent to fragments of FOL. A DL knowledge base has two components: a TBox and an ABox. The TBox specifies the terminology of an application domain, including concepts and roles and their relations. The ABox contains assertions about membership of named individuals. For example, we may have a concept named $Area$, which specifies a set of areas in computer science, and another concept $Expert$ which is a set of names of experts in computer science. Also, we can have a role $expertIn$ which relates $Expert$ to $Area$. The TBox $\{Expert \sqsubseteq \exists expertIn.Area\}$ means that each expert must have expertize in an area of computer science; the set $\{Expert(John), expertIn(John, AI)\}$ is an ABox.

Different DLs have different languages. In this section, we briefly recall some preliminaries of $\mathcal{ALC}$, a basic DL which contains all boolean operators. Further details of $\mathcal{ALC}$ and other DLs can be found in (Baader et al. 2002).

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 *concepts*), N_R is a set of *role names* (or *roles*), and N_I is a set of individuals. In this paper, a *variable* is either a concept name or a role name.

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, $A \in N_C$, C and D are concept descriptions, and $R \in N_R$.

An interpretation I is a pair $(\Delta^I, \cdot^I)$ where Δ^I is a nonempty set called the *domain* and $\cdot^I$ is an interpretation function which associates each individual a with an element a^I of Δ^I, each (atomic) concept A with a subset A^I of Δ^I and each atomic role R with a binary relation $R^I \subseteq \Delta^I \times \Delta^I$. The function $\cdot^I$ can be naturally extended to complex descriptions

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

A concept description C is *satisfiable* if $C^I \neq \emptyset$ for some interpretation I. Otherwise, we say C is unsatisfiable.

An *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. $C \equiv D$ (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 I *satisfies* an inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$. I is a *model*

of a TBox $\mathcal{T}$, denoted $I \models \mathcal{T}$, if I satisfies every inclusion of $\mathcal{T}$. $\mathcal{T} \models C \sqsubseteq D$ if for any I, $I \models \mathcal{T}$ implies $I \models C \sqsubseteq D$. We note that C is unsatisfiable if $\models C \equiv \bot$.

An *assertion* is a *concept assertion* of the form $C(a)$ or a *role assertion* of the form $R(a, b)$, where $a, b \in N_I$, C is a concept description, and $R \in N_R$. An *assertion box*, or *ABox*, is a finite set of *assertions*. $\mathcal{A}_c$ and $\mathcal{A}_r$ denote the set of concept assertions and the set of role assertions, respectively.

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

Formally, a knowledge base (KB) $\mathcal{K}$ is a pair $(\mathcal{T}, \mathcal{A})$ of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. An interpretation I is a model of $\mathcal{K}$ if I is a model of both $\mathcal{T}$ and $\mathcal{A}$, denoted $I \models \mathcal{K}$. A KB is *consistent* if it has at least one model. If α is an axiom or an assertion, $\mathcal{K} \models \alpha$ if every model of $\mathcal{K}$ satisfies 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 TBoxes and ABoxes.

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

We use $\mathsf{sub}(C)$ to denote the set of all sub-concepts occurring in C. And $\mathsf{sub}(\mathcal{K})$ is defined similarly.

4. 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 (i.e., concepts and roles) in an $\mathcal{ALC}$ KB, and then state and discuss several important properties of forgetting that justify the definition chosen. This is the first study of forgetting both concepts and roles for arbitrary knowledge bases in $\mathcal{ALC}$.

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 the results of some reasoning tasks on $\mathcal{S} - \mathcal{V}$ are still preserved in $\mathcal{K}'$ (e.g., query answering). As a result, $\mathcal{K}'$ is logically weaker than or as strong as $\mathcal{K}$ in general. Unlike the concept of conservative extensions, $\mathcal{K}'$ may not be a subset of $\mathcal{K}$. This intuition is formalized in the following definition.

Definition 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

- $\mathcal{K} \models \mathcal{K}'$;
- $\mathcal{K} \models C \sqsubseteq D$ implies $\mathcal{K}' \models C \sqsubseteq D$ for each concept inclusion $C \sqsubseteq D$ in $\mathcal{ALC}$ over $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$;
- $\mathcal{K} \models C(a)$ implies $\mathcal{K}' \models C(a)$ for each concept assertion $C(a)$ in $\mathcal{ALC}$ over $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$;
- $\mathcal{K} \models R(a, b)$ implies $\mathcal{K}' \models R(a, b)$ for each role assertion $R(a, b)$ in $\mathcal{ALC}$ such that $R \in \mathsf{sig}(\mathcal{K}) - \mathcal{V}$.

Conditions (KF3) and (KF4) extend two corresponding conditions in Ghilardi et al. (2006) that are used in the definition of *uniform interpolants* for (only) TBoxes. We note that if $\mathcal{K}$ is a TBox (with empty ABox), then we need only the conditions (KF1) and (KF2) in the above definition. And if $\mathcal{K}$ is an ABox (with empty TBox), condition (KF2) is unnecessary. 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 1. Suppose we have searched the Web and found an ontology about human diseases (such a practical ontology could be very large)

(1) Disease $\sqsubseteq \forall$attacks.Human,
(2) Human $\sqcap$ Infected $\sqsubseteq \exists$ shows.Symptom,
(3) Disease $\equiv$ Infectious $\sqcup$ Noninfectious,
(4) Influenza $\sqsubseteq$ Infectious,
(5) Influenza(H1N1), (6) $attacks$(H1N1, P1) and (7) Infected(P1).

We want to construct an ontology only about flu by reusing the above ontology (i.e., smaller in terms of alphabet). This is done by forgetting about the undesired concepts $\{Disease, Infectious, Noninfectious\}$. The result is obtained by replacing (1), (3), and (4) with

$Influenza \sqsubseteq \forall attacks.Human.$

To forget about concept $Infected$, the result can be obtained by removing (2), and replacing (7) with

$\exists shows.Symptom(P1)$.

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

Example 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 $\mathcal{K}$.

But 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 and assertions:

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

However, 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 state and discuss some important consequences of this definition of forgetting for KBs in $\mathcal{ALC}$. These properties provide an evidence that the definition is appropriate.

Proposition 1. [Uniqueness] Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB 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. We will leave the proof till a more general result (Proposition 2) is introduced. Given this result, we write 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 $\mathcal{K}$. For convenience, we assume that the result of forgetting about $\mathcal{V}$ in $\mathcal{K}$ exists when $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ is mentioned.

The following result, which generalizes Proposition 1, shows that forgetting preserves implication and equivalence relations between KBs.

Proposition 2. [Implication Invariance] 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 $\mathsf{forget}(\mathcal{K}_1, \mathcal{V}) \equiv \mathsf{forget}(\mathcal{K}_2, \mathcal{V})$.

Proof. We need only to show the first statement. For each inclusion or assertion α in $\mathsf{forget}(\mathcal{K}_2, \mathcal{V})$, we have $\mathcal{K}_2 \models \alpha$, and thus $\mathcal{K}_1 \models \alpha$. Because α does not contain any variable in $\mathcal{V}$, by the definition of forgetting, we have $\mathsf{forget}(\mathcal{K}_1, \mathcal{V}) \models \alpha$. Thus, we have shown $\mathsf{forget}(\mathcal{K}_1, \mathcal{V}) \models \mathsf{forget}(\mathcal{K}_2, \mathcal{V})$. ■

However, the converse of Proposition 2 is not true in general. Consider $\mathcal{K}$ and $\mathcal{K}_1$ in Example 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 checking and entailment are two major reasoning tasks in DLs. It is a key requirement for a reasonable definition of forgetting to preserve these two reasoning forms.

Proposition 3. Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ a set of variables. Then,

(i) Consistency: $\mathcal{K}$ is consistent iff $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ *is consistent;*
(ii) Entailment Invariance: for any inclusion or assertion α over $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$, $\mathcal{K} \models \alpha$ *iff* $\mathsf{forget}(\mathcal{K}, \mathcal{V}) \models \alpha$.

Proof. (1): Because $\mathcal{K} \models \mathsf{forget}(\mathcal{K}, \mathcal{V})$, the consistency of $\mathcal{K}$ implies the consistency of $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. If $\mathcal{K}$ is inconsistent, we have $\mathcal{K} \models (\top \sqsubseteq \bot)$, which implies that $\mathsf{forget}(\mathcal{K}, \mathcal{V}) \models (\top \sqsubseteq \bot)$. That is, $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ is also inconsistent.

2) It follows from Definition 1. ■

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

Proposition 4. [Variable Splitting] Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}_1, \mathcal{V}_2$ two sets of variables. Then, we have

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

Proof. For each inclusion or assertion α in $\mathsf{forget}(\mathsf{forget}(\mathcal{K}, \mathcal{V}_1), \mathcal{V}_2)$, we have $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1) \models \alpha$, and thus $\mathcal{K} \models \alpha$. Because α contains no variable from $\mathcal{V}_1$ or $\mathcal{V}_2$. By the definition of forgetting, we have $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1 \cup \mathcal{V}_2) \models \alpha$. We have shown $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1 \cup \mathcal{V}_2) \models \mathsf{forget}(\mathsf{forget}(\mathcal{K}, \mathcal{V}_1), \mathcal{V}_2)$.

For each inclusion or assertion α in $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1 \cup \mathcal{V}_2)$, we have $\mathcal{K} \models \alpha$ and α contains no variable from $\mathcal{V}_1 \cup \mathcal{V}_2$. Again, according to the definition of forgetting, $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1) \models \alpha$, and thus $\mathsf{forget}(\mathsf{forget}(\mathcal{K}, \mathcal{V}_1), \mathcal{V}_2) \models \alpha$. That is $\mathsf{forget}(\mathsf{forget}(\mathcal{K}, \mathcal{V}_1), \mathcal{V}_2) \models \mathsf{forget}(\mathcal{K}, \mathcal{V}_1 \cup \mathcal{V}_2)$. ■

Thus, computing the result of forgetting about $\mathcal{V}$ in $\mathcal{K}$ is equivalent to forgetting the variables in $\mathcal{V}$ one by one, *i.e.*, forgetting can be computed incrementally. By Proposition 4, it is easy to see that, as more variables are forgotten, the results are getting logically weaker.

Corollary 1. Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}_1, \mathcal{V}_2$ two sets of variables. Then $\mathcal{V}_1 \subseteq \mathcal{V}_2$ implies $\mathsf{forget}(\mathcal{K}, \mathcal{V}_1) \models \mathsf{forget}(\mathcal{K}, \mathcal{V}_2)$ *if they are treated as two KBs on* $\mathsf{sig}(\mathcal{K})$.

Another property useful for the computation of forgetting is that, forgetting in TBoxes is independent of ABoxes.

Proposition 5. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and $\mathcal{V}$ a set of variables. Then, for any $\mathcal{ALC}$ ABox $\mathcal{A}$ such that $(\mathcal{T}, \mathcal{A})$ is consistent, $\mathcal{T}'$ is the TBox of $\mathsf{forget}((\mathcal{T}, \mathcal{A}), \mathcal{V})$ *iff $\mathcal{T}'$ is the TBox of* $\mathsf{forget}((\mathcal{T}, \emptyset), \mathcal{V})$.

Proof. Observe that $(\mathcal{T}, \mathcal{A}) \models C \sqsubseteq D$ iff $(\mathcal{T}, \emptyset) \models C \sqsubseteq D$ for any inclusion $C \sqsubseteq D$, TBox $\mathcal{T}$ and ABox $\mathcal{A}$ such that $(\mathcal{T}, \mathcal{A})$ is consistent. Thus,

- $(\mathcal{T}, \mathcal{A}) \models \mathcal{T}'$ iff $(\mathcal{T}, \emptyset) \models \mathcal{T}'$; and
- for each concept inclusion $C \sqsubseteq D$ in $\mathcal{ALC}$ not containing any variables in $\mathcal{V}$, $(\mathcal{T}, \mathcal{A}) \models C \sqsubseteq D$ iff $(\mathcal{T}, \emptyset) \models C \sqsubseteq D$ iff $\mathcal{T}' \models C \sqsubseteq D$. ■

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 $\mathcal{T}$. Similarly, we use $\mathsf{forget}(\mathcal{A}, \mathcal{V})$ when the TBox is empty.

5. FORGETTING IN $\mathcal{ALC}$ CONCEPT DESCRIPTIONS

Uniform interpolation in $\mathcal{ALC}$ is proposed for studying the definability of concepts in ten Cate et al. (2006). It is also reformulated and investigated in terms of variable forgetting (briefly, c-forgetting) in Wang et al. (2009b) but only one variable can be forgotten each time. In this section, we reformulate the definition of the forgetting in $\mathcal{ALC}$ concept descriptions to allow forgetting a set of variables at the same time and present some results that will be used in the next section. From the viewpoint of ontology management, the issue of forgetting in concept descriptions might be less important than that for KBs and TBoxes. However, we will see later that the problem of computing KB-forgetting in $\mathcal{ALC}$ TBoxes can be reduced to that of computing c-forgetting.

Intuitively, the result C' of forgetting about a set of variables from a concept description C should not be logically stronger than C and at the same time, semantically as close to C as possible. For example, after the concept $Male$ is forgotten from a concept description for a "Male Australian student," $Australian \sqcap Student \sqcap Male$, then we should obtain a concept description $Australian \sqcap Student$ for an "Australian student". More specifically, C' should be a concept description that defines a minimal (w.r.t. subsumption) concept description

among all concept descriptions that subsume C and are syntactically irrelevant to $\mathcal{V}$ (i.e., variables in $\mathcal{V}$ do not appear in the concept description).

Definition 2. [c-forgetting] Let C be an $\mathcal{ALC}$ concept description and $\mathcal{V}$ a set of variables. An $\mathcal{ALC}$ 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:

- $\models C \sqsubseteq C'$.
- $\models C \sqsubseteq C''$ implies $\models C' \sqsubseteq C''$, for every $\mathcal{ALC}$ concept description C'' with $\mathsf{sig}(C'') \subseteq \mathsf{sig}(C) - \mathcal{V}$.

The above (CF1) and (CF2) correspond to the conditions (2) and (3) of Theorem 8 in ten Cate et al. (2006). 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 6. [Uniqueness] Let C be an $\mathcal{ALC}$ concept description and $\mathcal{V}$ a set of variables. If two $\mathcal{ALC}$ concept descriptions C' and C'' are results of c-forgetting about $\mathcal{V}$ in C, then $\models C' \equiv C''$.

Proof. Because both C', C'' are on $\mathsf{sig}(C) - \mathcal{V}$, by the definition of c-forgetting, we have $\models C' \sqsubseteq C''$ and $\models C'' \sqsubseteq C'$. Thus $\models C' \equiv C''$. ■

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

We use the following examples of concept descriptions to illustrate our semantic definition of c-forgetting for $\mathcal{ALC}$. We will introduce an algorithm later and explain how we can compute a result of c-forgetting through a series of syntactic transformations of concept descriptions.

Example 3. Suppose the concept "Influenza Carrier" is defined by $C = Human \sqcap (Male \sqcup Female) \sqcap \exists infected.Influenza$ where $Human$, $Male$, $Female$, and $Influenza$ are all concepts; $infected$ is a role and $infected(x, y)$ means that x is infected by y.

- *If the concept description C is used only for humans, we may wish to forget about $Human$:* $\mathsf{cforget}(C, Human) = (Male \sqcup Female) \sqcap \exists infected.Influenza$.
- $\mathsf{cforget}(C, Male) = Human \sqcap \exists infected.Influenza$. *This means that if it is not necessary to distinguish men from women, then it does not make any sense to keep either $Male$ or $Female$ here.*
- *If we want to generalize the concept to any virus carrier, then the filter $Influenza$ can be forgotten:* $\mathsf{cforget}(C, Influenza) = Human \sqcap (Male \sqcup Female) \sqcap \exists infected.\top$.
- *If we want to use the concept to describe all possible affected persons, then he or she may not be really infected:* $\mathsf{cforget}(C, infected) = Human \sqcap (Male \sqcup Female)$.

C-forgetting for $\mathcal{ALC}$ concept descriptions possesses several important properties.

First of all, c-forgetting for $\mathcal{ALC}$ preserves the subsumption and equivalence relation between concept descriptions.

Proposition 7. [Subsumption Invariance] Let C_1, C_2 be two concept descriptions in $\mathcal{ALC}$ and $\mathcal{V}$ a set of variables. Then,

- $\models C_1 \sqsubseteq C_2$ implies $\models \mathsf{cforget}(C_1, \mathcal{V}) \sqsubseteq \mathsf{cforget}(C_2, \mathcal{V})$;
- $\models C_1 \equiv C_2$ implies $\models \mathsf{cforget}(C_1, \mathcal{V}) \equiv \mathsf{cforget}(C_2, \mathcal{V})$.

Proof. We need only to show the first assertion: Because $\models C_1 \sqsubseteq C_2$ and $\models C_2 \sqsubseteq \mathsf{forget}(C_2, \mathcal{V})$, we have $\models C_1 \sqsubseteq \mathsf{forget}(C_2, \mathcal{V})$. By the definition of c-forgetting, we have $\models \mathsf{forget}(C_1, \mathcal{V}) \sqsubseteq \mathsf{forget}(C_2, \mathcal{V})$. ■

The converses of the above statements are not true in general. As we can see in Example 3, let $D = \mathsf{cforget}(C, Human)$, we have $\models \mathsf{cforget}(C, Human) \equiv \mathsf{cforget}(D, Human)$, but $\not\models C \equiv D$.

Recall that a concept description C is *satisfiable* iff $C^I \neq \emptyset$ for some interpretation I. By Definition 2, c-forgetting also preserves satisfiability of concept descriptions.

Proposition 8. [Satisfiability Invariance] Let C be an $\mathcal{ALC}$ concept description, and $\mathcal{V}$ be a set of variables. Then C is satisfiable iff $\mathsf{cforget}(C, \mathcal{V})$ *is satisfiable.*

Proof. If C is satisfiable, then $\mathsf{forget}(C, \mathcal{V})$ is satisfiable, because $\models C \sqsubseteq \mathsf{forget}(C, \mathcal{V})$. If C is unsatisfiable, that is $\models C \equiv \bot$. Obviously, $\mathsf{forget}(C, \mathcal{V}) = \bot$. ■

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

Proposition 9. [Variable Splitting] Let C be an $\mathcal{ALC}$ concept description and $\mathcal{V}_1, \mathcal{V}_2$ two sets of variables. Then we have

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

Proof. Denote $C_1 = \mathsf{forget}(C, \mathcal{V}_1)$, $C_2 = \mathsf{forget}(C_1, \mathcal{V}_2)$ and $C_3 = \mathsf{forget}(C, \mathcal{V}_1 \cup \mathcal{V}_2)$. We want to show that $\models C_2 \equiv C_3$.
Because $\models C \sqsubseteq C_1$ and $\models C_1 \sqsubseteq C_2$, we have $\models C \sqsubseteq C_2$. Together with $\mathsf{sig}(C_2) \cap (\mathcal{V}_1 \cup \mathcal{V}_2) = \emptyset$, by the definition of c-forgetting, we have $\models C_3 \sqsubseteq C_2$.
On the other hand, if $\models C_2 \sqsubseteq C_3$, $\models C \sqsubseteq C_3$ and $\mathsf{sig}(C_3) \cap \mathcal{V}_1 = \emptyset$ imply $\models C_1 \sqsubseteq C_3$, which is based on the definition of c-forgetting. Together with $\mathsf{sig}(C_3) \cap \mathcal{V}_2 = \emptyset$, again, by the definition of c-forgetting, we have $\models C_2 \sqsubseteq C_3$. ■

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.

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

Proposition 10. [Disjunction Distributivity] Let $C_1, \ldots, C_n$ be concept descriptions in $\mathcal{ALC}$. For any set $\mathcal{V}$ of variables, we have

$$\models \mathsf{cforget}(C_1 \sqcup \cdots \sqcup C_n, \mathcal{V}) \equiv \mathsf{cforget}(C_1, \mathcal{V}) \sqcup \cdots \sqcup \mathsf{cforget}(C_n, \mathcal{V}).$$

Proof. We only need to show that for any two concepts C_1 and C_2, $\mathsf{forget}(C_1, \mathcal{V}) \sqcup \mathsf{forget}(C_2, \mathcal{V})$ is a result of forgetting about $\mathcal{V}$ in concept $C_1 \sqcup C_2$. We show that the conditions in the definition of c-forgetting are satisfied.

It is easy to see that (CF1) is satisfied: Because $\models C_i \sqsubseteq \mathsf{forget}(C_i, \mathcal{V})$ for $i = 1, 2$, we have $\models C_1 \sqcup C_2 \sqsubseteq \mathsf{forget}(C_1, \mathcal{V}) \sqcup \mathsf{forget}(C_2, \mathcal{V})$.

To show that (CF2) is true, suppose C' is a concept such that $\mathsf{sig}(C') \cap \mathcal{V} = \emptyset$ and $\models C_1 \sqcup C_2 \sqsubseteq C'$. We have $\models C_i \sqsubseteq C'$ for $i = 1, 2$. Thus, by the definition of c-forgetting, $\models \mathsf{forget}(C_i, \mathcal{V}) \sqsubseteq C'$, which implies $\models \mathsf{forget}(C_1, \mathcal{V}) \sqcup \mathsf{forget}(C_2, \mathcal{V}) \sqsubseteq C'$. ■

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

An important reason for this is that c-forgetting does not distribute over negation. Actually, by the definition of forgetting

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

These subsumptions may be strict, e.g., if C is $A \sqcap B$ and $\mathcal{V}$ is $\{A\}$, then $\neg\mathsf{cforget}(C, \mathcal{V})$ is $\neg B$, but $\mathsf{cforget}(\neg C, \mathcal{V})$ is $\top$.

The next result shows that c-forgetting distributes over quantifiers. Because c-forgetting does not distribute over negation, the two statements in the following proposition do not necessarily imply each other.

Proposition 11. [Quantifier Distributivity] Let C be an $\mathcal{ALC}$ concept description such that $\not\models C \equiv \bot$, R be a role name and $\mathcal{V}$ be a set of variables. Then,

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

These results suggest a way of computing c-forgetting about a set $\mathcal{V}$ of variables in a complex $\mathcal{ALC}$ concept description C. That is, distribute the c-forgetting computation to subconcepts of C. The proof of this result is tedious and thus we put it in the Appendix at the end of the paper.

In what follows, we introduce an algorithm for computing the result of c-forgetting through rewriting of concept descriptions (syntactic concept transformations; ten Cate et al. 2006). 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 DNF, some notation and definitions are in order. We call an (atomic) concept A or its negation $\neg A$ a *literal concept* or simply a *literal*. A *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. Every arbitrary concept description can be transformed into an equivalent disjunction of conjunctions of generalized literals, using De Morgan's laws, distributive laws and simplifications. First, we define a very basic DNF for $\mathcal{ALC}$.

Basic DNF: A concept D is in *basic disjunctive normal form* or *basic 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 $D_i \not\equiv \bot$ and D_i is of the form

$$\bigsqcap L \sqcap \bigsqcap_{R \in \mathcal{R}} \left(\bigsqcap_j \forall R.U_{R,j} \sqcap \bigsqcap_k \exists R.E_{R,k} \right)$$

where L is a set of literals, $\mathcal{R}$ is the set of role names that occur in D_i, and each $U_{R,j}$ and each $E_{R,k}$ is a concept description in basic DNF.

Note that each conjunction in basic DNF can be an empty conjunction, which is equivalent to $\top$.

The reason for transforming a concept into its disjunctive normal form is that c-forgetting distributes over $\sqcup$ (Proposition 10). Now we only need to consider the computation of the result of c-forgetting in a conjunction of generalized literals. It can be shown that the result of c-forgetting about an (atomic) concept A in a conjunction of literals can be obtained just by extracting A (or $\neg A$) from the conjuncts (extracting a conjunct equals replacing it by $\top$). However, when C is a conjunction containing pseudo-literals, it is more complicated to compute the result of c-forgetting about A in C. For example, let $C = \forall R.A \sqcap \forall R.\neg A$. Through simple transformation we can see $C \equiv \forall R.\bot$ is the result of c-forgetting about A in C, while simply extracting A and $\neg A$ results in $\top$. A similar example is when $C = \forall R.(A \sqcup B) \sqcap \exists R.(\neg A \sqcup B)$, the result of c-forgetting about $\{A\}$ in C is $\exists R.B$ rather than $\exists R.\top$. It is worth noting that $\forall R.C_1 \sqcap \exists R.C_2 \sqsubseteq \exists R.(C_1 \sqcap C_2)$.

According to above concerns, a key step in our algorithm is to further transform each of the conjunctions through the following laws:

$$\forall R.C_1 \sqcap \forall R.C_2 \rightsquigarrow \forall R.(C_1 \sqcap C_2)$$
$$\forall R.C_1 \sqcap \exists R.C_2 \rightsquigarrow \forall R.C_1 \sqcap \exists R.(C_1 \sqcap C_2).$$

By applying the above transformations to the concept description and to the subconcepts in the scopes of value and existential restrictions recursively, we can obtain the following disjunctive normal form of the concept description. Since the number of conjuncts in each conjunction and the depth of value (existential) restrictions are finite, it is easy to see the transformation always terminates.

Definition 3. A concept description D is in 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 $\sqcap L$ 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.

Note that $U_R = \top$ if any D_i contains no universal quantification as its conjunct. For convenience, each D_i is called a *normal conjunction* in this paper. 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. The concepts U_R and $E_{R,k} \sqcap U_R$ need to be transformed into DNF separately and their lengths can be exponential in the length of the input concept. For instance, the concept description $\sqcap_{i=1}^{n} \forall R.(A_i \sqcup B_i)$ will produce a U_R with the disjunction of all 2^n combinations of A_i and B_i.

Algorithm 1 (Compute c-forgetting)
Input: An $\mathcal{ALC}$ concept description C and a set $\mathcal{V}$ of variables in C.
Output: $\mathsf{cforget}(C, \mathcal{V})$.
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 3.
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 $\mathsf{cforget}(F, \mathcal{V})$, and replace each resulting $\forall R.(\top \sqcup F)$ with $\top$.

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

FIGURE 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 $\mathsf{cforget}(D, A) = \bot$. We note that B is not a result of forgetting about A in D.

Example 4. 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 first 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 $\mathsf{cforget}(D, \{B\}) = A \sqcup \exists R.C$. To forget about role R in D, Algorithm 1 replaces each pseudo-literal in D' of the form $\forall R.F$ or $\exists R.F$ with $\top$, and returns $\mathsf{cforget}(D, \{R\}) = \top$.

Obviously, the major cost of Algorithm 1 is in 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 (with regard to 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 1. Let $\mathcal{V}$ be a set of concept and role names and C be an $\mathcal{ALC}$ concept description. Then Algorithm 1 always returns $\mathsf{cforget}(C, \mathcal{V})$.

It can be seen from the results in this and last sections that they have several similar properties although KB-forgetting and c-forgetting differ in several ways. We summarize these differences and similarities in Table 1 according to their properties. A major difference is that the result of c-forgetting always exist but a result of forgetting may not exist for some KBs and some sets of variables. We note that *Subsumption Invariance* for c-forgetting corresponds to *Implication Invariance* for KB-forgetting whereas *Satisfiability* for c-forgetting is a counterpart of *Consistency* for KB-forgetting.

TABLE 1. Summary for Properties of Forgetting.

Property	KB-forgetting	c-forgetting
Existence	N	Y
Uniqueness invariance	Y	Y
Implication/Subsumption invariance	Y	Y
Consistency/Satisfiability invariance	Y	Y
Entailment invariance	Y	N/A
Variable splitting	Y	Y
Disjunction distributivity	N/A	Y
Quantifier distributivity	N/A	Y

6. APPROXIMATE FORGETTING FOR $\mathcal{ALC}$ TBOXES

As we have shown in Section 4, the result of forgetting for an $\mathcal{ALC}$ KB (or just TBox) may not exist. Even if it exists, given the inherent complexity of KB-forgetting in $\mathcal{ALC}$, it is hard to design efficient algorithms for computing the (whole) result of KB-forgetting. As a result, two interesting issues arise: (1) When the result of KB-forgetting exists, can we compute only its part that is sufficient for ontology applications of interest? (2) When the result of KB-forgetting does not exist, can we still find a new KB so that it can act as a result of KB-forgetting for our ontology applications?

It is possible address these two issues in a uniform way. The basic idea of our approach is to provide an approximation to the result of KB-forgetting, called *n-forgetting*, no matter whether the result of KB-forgetting exists or not. Such an approach has several advantages: (1) It resolves the nonexistence problem of forgetting in a certain sense; (2) When we compute the approximation, we do not need to determine if the result of KB-forgetting exists or not; and (3) Since only a part of the ontology $\mathsf{forget}(\mathcal{K}, \mathcal{V})$ is computed in the approximation, it is possible to develop efficient algorithms. In this section, we first introduce the concept of n-forgetting as an approximation to KB-forgetting and then develop an algorithm for computing n-forgetting by employing the algorithm for forgetting in concept descriptions.

Because it is still very difficult to compute n-forgetting for a general KB, we are only able to provide an algorithm for n-forgetting in TBoxes (ontologies with empty ABoxes), whereas the concept of n-forgetting is formulated for general KBs. We note that many practical applications of forgetting in ontologies need only TBoxes instead of general KBs (e.g., extracting ontology summary). So the algorithm is still general enough for such practical applications. Recall that the motivation for forgetting about $\mathcal{V}$ in an ontology $\mathcal{K}$ is to find a new ontology $\mathcal{K}'$ on a smaller alphabet $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ than $\mathsf{sig}(\mathcal{K})$ such that $\mathcal{K}'$ is equivalent to $\mathcal{K}$ with regard to entailment against *all assertions and axioms on* $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$. The concept of n-forgetting is to find a new ontology $\mathcal{K}''$ on $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ such that $\mathcal{K}''$ is equivalent to $\mathcal{K}$ with regard to entailment against only assertions and axioms, whose sizes are bounded by a given value, on $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$.

Definition 4. [n-Forgetting] Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB, $\mathcal{V}$ be a set of variables, and n be a natural number. A KB $\mathcal{K}'$ over the signature $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ is a result of *n-forgetting* about $\mathcal{V}$ in $\mathcal{K}$ if,

- (NF1) $\mathcal{K} \models \mathcal{K}'$.
- (NF2) $\mathcal{K} \models C \sqsubseteq D$ iff $\mathcal{K}' \models C \sqsubseteq D$, for any concepts C, D over $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ *s.t.*$|\mathsf{sub}(C) \cup \mathsf{sub}(D) \cup \mathsf{sub}(\mathcal{K})| \leq n$.
- (NF3) $\mathcal{K} \models C(a)$ iff $\mathcal{K}' \models C(a)$, for any concept C over $\mathsf{sig}(\mathcal{K}) - \mathcal{V}$ *s.t.*$|\mathsf{sub}(C) \cup \mathsf{sub}(\mathcal{K})| \leq n$, and any individual name a in $\mathcal{K}$.
- (NF4) $\mathcal{K} \models R(a, b)$ iff $\mathcal{K}' \models R(a, b)$ for any role name $R \in \mathsf{sig}(\mathcal{K}) - \mathcal{V}$ and individuals $a, b \in \mathcal{K}$.

A key idea behind n-forgetting is to partially satisfy the conditions (KF2), (KF3), and (KF4) but in a parameterized way. In particular, these conditions will be fully satisfied when n is large enough. For this reason, we are more interested in relatively large n.

In general, the results of n-forgetting about $\mathcal{V}$ in $\mathcal{K}$ may not be logically equivalent. We use $\mathsf{Forget}_n(\mathcal{T}, \mathcal{V})$ to denote the set of all results of n-forgetting about $\mathcal{V}$ in a TBox $\mathcal{T}$. So, $\mathcal{K}' \in \mathsf{Forget}_n(\mathcal{K}, \mathcal{V})$ means that $\mathcal{K}'$ is a result of n-forgetting about $\mathcal{V}$ in $\mathcal{K}$.

It is not hard to see that a result of n-forgetting is always logically weaker than or equivalent to a result of forgetting. In particular, if the result of KB-forgetting about $\mathcal{V}$ in $\mathcal{K}$ exists, then it is also an n-forgetting about $\mathcal{V}$ in $\mathcal{K}$ for any $n \geq 0$.

Proposition 12. Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and $\mathcal{V}$ a set of variables. Then, for any $n \geq 0$ and any $\mathcal{K}' \in \mathsf{Forget}_n(\mathcal{K}, \mathcal{V})$,

(i) $\mathsf{forget}(\mathcal{K}, \mathcal{V}) \models \mathcal{K}'$.
(ii) $\mathsf{forget}(\mathcal{K}, \mathcal{V}) \in \mathsf{Forget}_n(\mathcal{K}, \mathcal{V})$.

Recall from the definition of KB-forgetting that, with respect to entailment against axioms and assertions not containing variables in $\mathcal{V}$, $\mathcal{K}$ is equivalent to $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. Definition 4 tells us that if we know what types of axioms and assertions not containing variables in $\mathcal{V}$ we wish to reason about in advance, then we can determine a value for n, find a KB $\mathcal{K}' \in \mathsf{Forget}_n(\mathcal{K}, \mathcal{V})$, and use $\mathcal{K}'$ as a replacement for the result of KB-forgetting about $\mathcal{V}$ in $\mathcal{K}$. In this way, n-forgetting is a useful approximation to KB-forgetting.

In the rest of this section, we introduce a novel method to compute a result of n-forgetting using algorithms for c-forgetting. We observe that each TBox $\mathcal{T}$ corresponds to a concept

$$con(\mathcal{T}) = \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D).$$

Thus, each TBox $\mathcal{T}$ can be transformed into a TBox of the form $\{\top \sqsubseteq con(\mathcal{T})\}$ that is equivalent to $\mathcal{T}$. In this sense, a TBox $\mathcal{T}$ is completely determined by the concept $con(\mathcal{T})$. We note that $con(\mathcal{T})$ is always finite.

In general, the result of forgetting in a TBox cannot be immediately obtained by performing c-forgetting on each side of the axioms. In fact, for an axiom $C \sqsubseteq D$ in $\mathcal{T}$, $\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq \mathsf{cforget}(D, \mathcal{V})$ may not even be a logical consequence of $\mathsf{forget}(\mathcal{T}, \mathcal{V})$ in general. For example, let $\mathcal{T} = \{A \sqsubseteq B\}$. Then $\mathsf{forget}(\mathcal{T}, A) = \{\top \sqsubseteq \top\} \equiv \emptyset$ and $\mathsf{cforget}(A, A) \sqsubseteq \mathsf{cforget}(B, A)$ is the axiom $\top \sqsubseteq B$. It is obvious that $\mathsf{forget}(\mathcal{T}, A) \not\models \top \sqsubseteq B$.

However, when $\mathcal{T}$ is transformed into the singleton TBox $\{\top \sqsubseteq con(\mathcal{T})\}$, we note that the axiom $\top \sqsubseteq \mathsf{cforget}(con(\mathcal{T}), \mathcal{V})$, denoted α_0, is a logical consequence of $\mathcal{T}$. The singleton TBox $\{\alpha_0\}$ is not necessarily equivalent to $\mathsf{forget}(\mathcal{T}, \mathcal{V})$ but it can be a starting point for constructing a sequence of TBoxes whose limit is $\mathsf{forget}(\mathcal{T}, \mathcal{V})$. Because $\mathcal{T}$ is also equivalent to $\{\top \sqsubseteq con(\mathcal{T}) \sqcap \forall R.con(\mathcal{T})\}$ for an arbitrary role name R in $\mathcal{T}$, the axiom $\top \sqsubseteq \mathsf{cforget}(con(\mathcal{T}) \sqcap \forall R.con(\mathcal{T}), \mathcal{V})$, denoted α_1, is a logical consequence of $\mathcal{T}$. Similarly,

we define α_2 to be $\top \sqsubseteq \mathsf{cforget}(con(\mathcal{T}) \sqcap \forall R.con(\mathcal{T}) \sqcap \forall R.\forall R.con(\mathcal{T}), \mathcal{V}), \ldots$. It can be seen that each TBox $\{\alpha_{i+1}\}$ $(i \geq 0)$ is logically stronger than or equivalent to $\{\alpha_i\}$. That is, $\mathsf{forget}(\mathcal{T}, \mathcal{V}) \models \{\alpha_{i+1}\}$ and $\{\alpha_{i+1}\} \models \{\alpha_i\}$ for $i \geq 0$. In this way, we can construct a sequence of TBoxes with increasing logical strength, whose limit is $\mathsf{forget}(\mathcal{T}, \mathcal{V})$.

Formally, given a finite concept C and a number $n \geq 0$, define

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

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 $n \geq 0$.

The following result shows that a result of n-forgetting always exists for TBoxes.

Theorem 2. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and $\mathcal{V}$ be a set of variables. For each $n = 2^k$ with $k \geq 0$, the TBox

$$\mathcal{T}_{\mathcal{V}}^{(n)} = \{\, \top \sqsubseteq \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}) \,\}$$

is a result of n-forgetting about $\mathcal{V}$ in $\mathcal{T}$.

To prove Theorem 2, we need a lemma in ten Cate et al. (2006).

Lemma 1. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox, C_1 and C_2 are two concepts. Then $\mathcal{T} \models C_1 \sqsubseteq C_2$ iff $\models (C_1 \sqcap con(\mathcal{T})^{(n)}) \sqsubseteq C_2$ where $n \geq 2^{|\mathsf{sub}(C_1) \cup \mathsf{sub}(C_2) \cup \mathsf{sub}(\mathcal{K})|}$.

Proof of Theorem 2. (NF1). To prove that $\mathcal{T} \models \mathcal{T}_{\mathcal{V}}^{(n)}$, we need to show that $I \models \mathcal{T}_{\mathcal{V}}^{(n)}$ for each model I of $\mathcal{T}$.

For each axiom $C \sqsubseteq D$ in $\mathcal{T}$, $(\neg C \sqcup D)^I = \Delta^I$. Thus $(\sqcap_{C \sqsubseteq D \in \mathcal{T}}(\neg C \sqcup D))^I = \Delta^I$. For any number $i \geq 0$ and arbitrary roles $R_1, \cdots, R_i$, $(\forall R_1 \cdots \forall R_i.\, \sqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D))^I = \Delta^I$. Hence, $(con(\mathcal{T})^{(n)})^I = \Delta^I$. Because $(con(\mathcal{T})^{(n)})^I \subseteq (\mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}))^I$, we have $\Delta^I \subseteq (\mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}))^I$. That is, I satisfies $\mathcal{T}_{\mathcal{V}}^{(n)}$.

(NF2). We only need to show that $\mathcal{T} \models C \sqsubseteq D$ implies $\mathcal{T}_{\mathcal{V}}^{(n)} \models C \sqsubseteq D$. By Lemma 1, if $\mathcal{T} \models C \sqsubseteq D$, then

$$\models C \sqcap \bigsqcap_{i=0}^{n} \bigsqcap_{R_1,\ldots,R_i \in \mathcal{R}} \forall R_1 \cdots \forall R_i.\mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}) \sqsubseteq D.$$

This implies $\models C \sqcap (con(\mathcal{T})^{(n)}) \sqsubseteq D$, which is equivalent to $\models con(\mathcal{T})^{(n)} \sqsubseteq \neg C \sqcup D$. Because neither C nor D contains any variables in $\mathcal{V}$, by the definition of c-forgetting, we have $\models \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}) \sqsubseteq \neg C \sqcup D$, which implies $\models C \sqcap \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}) \sqsubseteq D$. Thus,

$$\models C \sqcap \bigsqcap_{i=0}^{n} \bigsqcap_{R_1,\ldots,R_i \in \mathcal{R}} \forall R_1 \cdots \forall R_i.\mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}) \sqsubseteq D. \quad \blacksquare$$

By Theorem 2, the TBoxes $\mathcal{T}_{\mathcal{V}}^{(n)}$ $(n = 2^k, k \geq 0)$ actually provide a sequence of results of n-forgetting Because approximations to the result of forgetting in TBoxes. Note that the above n-forgetting for TBoxes is computed in terms of forgetting in concept descriptions (c-forgetting).

Example 5. Consider the TBox $\mathcal{T}$ in Example 2. Then,
$con(\mathcal{T}) = (\neg A \sqcup B) \sqcap (\neg B \sqcup C) \sqcap (\neg C \sqcup \forall R.C) \sqcap (\neg C \sqcup D)$,
$con(\mathcal{T})^{(0)} = con(\mathcal{T})$, $con(\mathcal{T})^{(1)} = con(\mathcal{T}) \sqcap \forall R.con(\mathcal{T})$,

and for $n = 2^k$ *with* $k \geq 1$,
$con(\mathcal{T})^{(n)} = con(\mathcal{T}) \sqcap \forall R.con(\mathcal{T})^{(n-1)}$.
Let $\mathcal{V} = \{B, C\}$. For each $n \geq 0$, $\mathcal{T}_{\mathcal{V}}^{(n)}$ can be computed Because follows.
$\mathcal{T}_{\mathcal{V}}^{(0)} = \{ \top \sqsubseteq \neg A \sqcup D \}$, which is equivalent to $\{ A \sqsubseteq D \}$.
$\mathcal{T}_{\mathcal{V}}^{(1)} = \{ \top \sqsubseteq \neg A \sqcup (D \sqcap \forall R.D) \}$, which is equivalent to $\{ A \sqsubseteq D,\ A \sqsubseteq \forall R.D \}$.
. . . . . .
$\mathcal{T}_{\mathcal{V}}^{(n)} = \{ A \sqsubseteq D,\ A \sqsubseteq \forall R.D,\ \ldots,\ A \sqsubseteq \underbrace{\forall R.\forall R \cdots \forall R}_{nRs}.D \}$.

Proposition 12 shows that, for any $n \geq 0$, each $\mathcal{T}_{\mathcal{V}}^{(n)}$ is logically weaker than $\mathsf{forget}(\mathcal{T}, \mathcal{V})$. Also, because the number n is sufficiently large, $\mathcal{T}_{\mathcal{V}}^{(n)}$ preserves more and more consequences of $\mathcal{T}$. Therefore, the sequence of TBoxes $\{\mathcal{T}_{\mathcal{V}}^{(n)}\}_{n\geq 0}$ is nondecreasing with respect to semantic consequence as the next proposition shows.

Proposition 13. Let $\mathcal{T}$ *be an* $\mathcal{ALC}$ *TBox and* $\mathcal{V}$ *a set of variables. Then, for any* $n \geq 0$, *we have* $\mathcal{T}_{\mathcal{V}}^{(n+1)} \models \mathcal{T}_{\mathcal{V}}^{(n)}$.

Proof. Note that $con(\mathcal{T})^{(n+1)} = con(\mathcal{T})^{(n)} \sqcap D$where $D = \sqcap_{R_1,\ldots,R_{n+1}\in\mathcal{R}} \forall R_1 \cdots \forall R_{n+1}.con(\mathcal{T})$.

By the definition of c-forgetting, we have $\models \mathsf{cforget}(con(\mathcal{T})^{(n+1)}, \mathcal{V}) \sqsubseteq \mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V})$. For any model I of $\mathcal{K}_{\mathcal{V}}^{(n+1)}$, because I satisfies $\mathcal{T}_{\mathcal{V}}^{(n+1)}$, we have $\Delta^I \subseteq (\mathsf{cforget}(con(\mathcal{T})^{(n+1)}, \mathcal{V}))^I$. Thus, $\Delta^I \subseteq (\mathsf{cforget}(con(\mathcal{T})^{(n)}, \mathcal{V}))^I$. That is, I satisfies $\mathcal{T}_{\mathcal{V}}^{(n)}$. ■

Based on Propositions 12 and 13, we can show the main theorem of this section because follows, which states that the limit of the sequence of $\{\mathcal{T}_{\mathcal{V}}^{(n)}\}_{n\geq 0}$ captures the result of forgetting.

Theorem 3. Let $\mathcal{T}$ *be an* $\mathcal{ALC}$ *TBox and* $\mathcal{V}$ *a set of variables. Then*

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

Proof. By Definition 4, the limit of n-forgetting satisfies the conditions in Definition 1. ■

So, by Theorem 3, we can compute $\mathsf{forget}(\mathcal{T}, \mathcal{V})$, if it exists, using the above algorithms. As we can see from Example 2, given a TBox $\mathcal{T}$, the sizes of its consequences on $\mathsf{sig}(\mathcal{T}) - \mathsf{sig}(\mathcal{V})$ may not have a finite upper bound. In this case, we can always choose n large enough to provide an approximation to TBox-forgetting that is sufficient for the ontology application at hand. Although the computation of approximations uses DNF-based algorithm for computing c-forgetting, our approach indeed provides a general framework which can approximate forgetting using arbitrary c-forgetting methods.

7. CONCLUSION

We have presented a theory and methods for forgetting in knowledge bases in the expressive DL $\mathcal{ALC}$. This is the first work that deals with forgetting about concepts and

roles in $\mathcal{ALC}$ knowledge bases. Because the result of KB-forgetting may not exist for $\mathcal{ALC}$ knowledge bases, we have defined a sequence of finite TBoxes that approximate the result of TBox-forgetting and serve as a basis for query answering over the ontology with forgotten concepts and roles. We have provided algorithms for computing these approximations, using algorithms for forgetting in concept descriptions, and proved their correctness. We note that when this paper was under review, the decidability of the existence of $\mathcal{ALC}$ forgetting was proved in Lutz and Wolter (2011) and in particular, their proof was based on our method of approximating forgetting developed in the last section.

There are still a few interesting issues for future research. First, we plan to extend the results of this paper to even more expressive DLs. Second, the approximation algorithm is not incremental in the sense that the computation of $\mathcal{K}_{\mathcal{V}}^{(n+1)}$ is not based on $\mathcal{K}_{\mathcal{V}}^{(n)}$. It is unclear if an incremental approximation can be developed for KB-forgetting. Another related issue is to find a way to measure how close $\mathcal{K}_{\mathcal{V}}^{(n)}$ is to $\mathsf{forget}(\mathcal{K}, \mathcal{V})$. Last, it would be useful to implement our algorithms and incorporate them into ontology editors such as Protégé (see http://www.ict.griffith.edu.au/ kewen/DLForget for our progress on this task).

ACKNOWLEDGMENTS

The authors would like to thank the three anonymous referees for their detailed and helpful comments, which have greatly helped in improving the quality of the paper. This work was supported by the Australia Research Council (ARC) Discovery Projects DP1093652 and DP110101042. A major part of this work was done when Zhe Wang was at Griffith University.

REFERENCES

ALANI, H., S. HARRIS, and B. O'NEIL. 2006. Winnowing ontologies based on application use. *In* Proceedings of the 3rd European Semantic Web Conference (ESWC-06), Budva, Montenegro, pp. 185–199.

ANTONIOU, G., and K. KEHAGIAS. 2000. A note on the refinement of ontologies. International Journal of Intelligent Systems, **15**:623–632.

BAADER, F., D. CALVANESE, D. MCGUINNESS, D. NARDI, and P. PATEL-SCHNEIDER. 2002. The Description Logic Handbook. Cambridge University Press: Cambridge, UK.

BAADER, F., S. BRANDT, and C. LUTZ. 2005. Pushing the EL envelope. *In* Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, UK, pp. 364–369.

BIZER, C., T. Health, and T. BERNERS-LEE. 2009. Linked data - The story so far. International Journal Semantic Web Information Systems, **5**(3): 1–22.

CALVANESE, D., G. De GIACOMO, D. LEMBO, M. LENZERINI, and R. ROSATI. 2005. DL-Lite: tractable description logics for ontologies. *In* Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, PA, pp. 602–607.

CALVANESE, D., G. De GIACOMO, D. LEMBO, M. LENZERINI, and R. ROSATI. 2007. Tractable reasoning and efficient query answering in description logics: the DL-Lite family. Journal of Automated Reasoning, **39**(3):385–429.

CALVANESE, D., J. CARROLL, G. De GIACOMO, J. HENDLER, I. HERMAN, B. PARSIA, P. PATEL-SCHNEIDER, A. RUTTENBERG, U. SATTLER, and M. SCHNEIDER. 2009. OWL 2 web ontology language. W3C Recommendation, October 2009. Available at: http://www.w3.org/TR/owl2-overview/. Accessed May 26, 2012.

CUENCA GRAU, B., I. HORROCKS, Y. KAZAKOV, and U. SATTLER. 2007a. A logical framework for modular integration of ontologies. *In* Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, pp. 298–303.

CUENCA GRAU, B., I. HORROCKS, Y. KAZAKOV, and U. SATTLER. 2008. Modular reuse of ontologies: theory and practice. Journal of Artificial Intelligence Research, **31:**273–318.

CUENCA GRAU, B., B. PARSIA, and E. SIRIN. 2006. Combining OWL ontologies using E-connections. Journal of Web Semantics, **4**(1): 40–59.

CUENCA GRAU, B., I. HORROCKS, and U. SATTLER. 2007b. Just the right amount: Extracting modules from ontologies. *In* Proceedings of the 16th International World Wide Web Conference (WWW-07), Banff, Canada, pp. 717–726.

DZBOR, M., E. MOTTA, C. BUIL, J. M. GOMEZ, O. GÖRLITZ, and H. LEWEN. 2006. Developing ontologies in OWL: An observational study. *In* Proceedings of the OWLED-06 Workshop on OWL: Experiences and Directions, Athens, GA.

EITER, T., and K. WANG. 2008. Semantic forgetting in answer set programming. Artificial Intelligence, **14:** 1644–1672.

GHILARDI, S. 1995. An algebraic theory of normal forms. Annals of Pure and Applied Logic, **71**(3):189–245.

GHILARDI, S., C. LUTZ, and F. WOLTER. 2006. Did I damage my ontology? A case for conservative extensions in description logics. *In* Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06), Lake District, UK, pp. 187–197.

KONEV, B., D. WALTHER, and F. WOLTER. 2009. Forgetting and uniform interpolation in large-scale description logic terminologies. *In* Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, CA, pp. 830–835.

KONTCHAKOV, R., F. WOLTER, and M. ZAKHARYASCHEV. 2007. Modularity in DL-Lite. *In* Proceedings of the 2007 International Workshop on Description Logics (DL-07), Brixen-Bressanone, Italy.

KONTCHAKOV, R., F. WOLTER, and M. ZAKHARYASCHEV. 2008. Can you tell the difference between DL-Lite ontologies? *In* Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-08), Sydney, Australia, pp. 285–295.

KONTCHAKOV, R., F. WOLTER, and M. ZAKHARYASCHEV. 2010. Logic-based ontology comparison and module extraction, with an application to DL-Lite. Artificial Intelligence, **174**(15):1093–1141.

LANG, J., P. LIBERATORE, and P. MARQUIS. 2003. Propositional independence: formula variable independence and forgetting. Journal of Artificial Intelligence Research, **18:**391–443.

LIN, F., and R. REITER. 1994. Forget it. *In* Proceedings of the AAAI Fall Symposium on Relevance, New Orleans, LA, pp. 154–159.

LUTZ, C., D. WALTHER, and F. WOLTER. 2007. Conservative extensions in expressive description logics. *In* Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, pp. 453–458.

LUTZ, C., and F. WOLTER. 2011. Foundations for uniform interpolation and forgetting in expressive description logics. *In* Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11), Barcelona, Spain, pp. 989–995.

PERONI, S., E. MOTTA, and M. d'AQUIN. 2008. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. *In* Proceedings of the 3rd Asian Semantic Web Conference (ASWC-08), Pathumthani, Thailand, pp. 242–256.

RECTOR, A. L., S. BRANDT, and J. KOLA. 2008. Why do it the hard way? The case for an expressive description logic for SNOMED. Journal of the American Medical Informatics Association, **15**(6): 744–751.

tEN CATE, B., W. CONRADIE, M. MARX, and Y. VENEMA. 2006. Definitorially complete description logics. *In* Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06), Lake District, UK, pp. 79–89.

VISSER, A. 1996. Uniform interpolation and layered bisimulation. *In* Proceedings of Gödel'96: Logical Foundations of Mathematics, Computer Science, and Physics—Kurt Gödel's Legacy. *Edited by* P. Hájek. Brno, Czech Republic, **Vol**. 6 of Lecture Notes Logic, pp. 139–164.

WANG, K., Z. WANG, R. TOPOR, J. Z. PAN, and G. ANTONIOU. 2009a. Eliminating concepts and roles from ontologies in description logic. *In* Proceedings of the 8th International Semantic Web Conference (ISWC-09), Washington, DC, pp. 666–681.

WANG, Z., K. WANG, R. TOPOR, and J. Z. PAN. 2008. Forgetting concepts in DL-Lite. *In* Proceedings of the 5th European Semantic Web Conference (ESWC-08), Canary Islands, Spain, pp. 245–257.

WANG, Z., K. WANG, R. TOPOR, and J. Z. PAN. 2010. Forgetting for knowledge bases in dl-lite. Annals of Mathematics and Artificial Intelligence, **58**(1–2):117–151.

WANG, Z., K. WANG, R. TOPOR, J. Z. PAN, and G. ANTONIOU. 2009b. Uniform interpolation for ALC revisited. *In* Proceedings of the 22nd Australasian Conference on Artificial Intelligence, Melbourne, Australia, pp. 528–537.

APPENDIX

The tableau-based approach for DL reasoning is well established, and is the basis for most DL reasoners (Baader et al. 2002). Some proofs in this section are heavily based on the tableau algorithm for $\mathcal{ALC}$, and thus, we first briefly introduce it.

Before we explain how the tableau algorithm works, we first introduce some basic definitions. Let C be an $\mathcal{ALC}$-concept description in negation normal form (NNF), i.e., the negation occurs only directly in front of concept names. Each concept description can be equivalently transformed into NNF using De Morgan's laws.

Given an $\mathcal{ALC}$ concept C, the tableau algorithm (checking satisfiability of) C tries to construct a finite interpretation I that satisfies concept C, which contains an element x such that $x^I \in C^I$. The algorithm starts with the ABox $\mathcal{A} = \{C(x)\}$, where C is in NNF, and applies tableau expansion rules (or simply, T-rules, Table A1; Baader et al. 2002) to the ABox until a contradiction is found or no more rules are applicable. If a finite interpretation can be successfully constructed in this way (without a contradiction), then $\mathcal{A}$ is consistent and C is satisfiable. Otherwise, $\mathcal{A}$ is inconsistent and C is unsatisfiable.

Note that $\sqcup$-rule splits an ABox $\mathcal{A}$ into two, $\mathcal{A}$ and $\mathcal{A}'$. Thus, the tableau algorithm starts with a single ABox $\mathcal{A}$, and by applying T-rules, expands $\mathcal{A}$ into a set of ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$, each of which represents a possible finite interpretation.

An ABox $\mathcal{A}$ is *complete* if none of the T-rules applies to it.

ABox $\mathcal{A}$ contains a *clash* if $\{A(x), \neg A(x)\} \subseteq \mathcal{A}$ for some individual x and concept name A. An ABox is called *closed* if it contains a clash, and *open* otherwise.

If a complete and open ABox can be generated, then the algorithm terminates and returns "C is satisfiable." Otherwise, if all the ABoxes generated from expanding $\mathcal{A}$ is closed, the algorithm returns "C is unsatisfiable."

TABLE A1. Tableau Expansion Rules for $\mathcal{ALC}$ (T-Rules).

$\sqcap$-rule:	if	$(C_1 \sqcap C_2)(x) \in \mathcal{A}$, and $\{C_1(x), C_2(x)\} \not\subseteq \mathcal{A}$
	then	set $\mathcal{A}= \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
$\sqcup$-rule:	if	$(C_1 \sqcup C_2)(x) \in \mathcal{A}$, and $\{C_1(x), C_2(x)\} \cap \mathcal{A} = \theta$
	then	create a copy $\mathcal{A}'$ of the ABox and set $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$ and $\mathcal{A} = \mathcal{A} \cup \{C_2(x)\}$.
$\exists$-rule:	if	$\exists R.C(x) \in \mathcal{A}$, and x has no R -successor y with $C(y) \in \mathcal{A}$
	then	set $\mathcal{A}= \mathcal{A} \cup \{R(x, y), C(y)\}$ where y is a new individual.
$\forall$-rule:	if	$\forall R.C(x) \in \mathcal{A}$, and there is an R-successor y of x with $C(y) \notin \mathcal{A}$
	then	set $\mathcal{A}= \mathcal{A} \cup \{C(y)\}$.

To check subsumption $\models C \sqsubseteq D$ by the tableau algorithm, the algorithm starts with ABox $\mathcal{A} = \{C(x), \neg D(x)\}$. Then, $\models C \sqsubseteq D$ holds if all the ABoxes generated from expanding $\mathcal{A}$ is closed.

Proposition 14. Let C be an $\mathcal{ALC}$ concept description such that $\not\models C \equiv \bot$, R be a role name and $\mathcal{V}$ be a set of variables. Then,

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

To prove Proposition 11, we first show some properties of the subsumption relation between concept descriptions.

The following lemma presents two useful concept transformation rules.

Lemma 2. Let C and C_i's be concepts and R a role name. Then,

$$\left\{ \begin{array}{l} \models \bigsqcap_i \forall R.C_i \sqcup \exists R.C \equiv \bigsqcap_i \forall R.(C \sqcup C_i) \sqcup \exists R.C \\ \models \forall R.C \sqcap \bigsqcap_i \exists R.C_i \equiv \forall R.C \sqcap \bigsqcap_i \exists R.(C \sqcap C_i) \end{array} \right\}$$

The above two statements follow immediately from the definitions and thus their proofs are omitted here.

Lemma 3. Let C be a concept *s.t.* $\not\models C \equiv \bot$ and R a role name. Then, for any concept D,

- $\models \forall R.C \sqsubseteq D$ implies $\models D \equiv \top$ if $R \notin \mathsf{sig}(D)$, and otherwise, there always exists a concept C' such that $\mathsf{sig}(C') \subseteq \mathsf{sig}(D)$, $\models C \sqsubseteq C'$, and $\models \forall R.C \sqsubseteq \forall R.C'$;
- $\models \exists R.C \sqsubseteq D$ implies $\models D \equiv \top$ if $R \notin \mathsf{sig}(D)$, and otherwise, there always exists a collection of concepts C'_k such that $\mathsf{sig}(C'_k) \subseteq \mathsf{sig}(D)$ and $\models C \sqsubseteq C'_k$ for each k, and $\models \exists R.C \sqsubseteq \sqcap_k \exists R.C'_k$.

Note that this result is not obvious because in general $\mathsf{sig}(C)$ is not a subset of $\mathsf{sig}(D)$.

Proof. (1): The concept D can be transformed into an equivalent conjunctive form $D = \sqcap_k D_k$, using distributive laws for $\sqcup$ over $\sqcap$ and $\exists R$, where each D_k is of the form

$$\bigsqcup_{1 \le i \le n} \forall R.U_{k,i} \sqcup \exists R.E_k \sqcup F_k,$$

with E_k and each $U_{k,i}$ being a concept, and F_k being a disjunction of literals and non-R-quantifications. Note that $n = 0$ if D_k does not have any universal R-quantification as a disjunct. And $E_k = \bot$ if D_k does not have any existential R-quantification as a disjunct.

(1) Because $\models \forall R.C \sqsubseteq D$, we have $\models \forall R.C \sqsubseteq D_k$ for each k. By the completeness of the tableau algorithm, each ABox generated by expanding $\mathcal{A} = \{\forall R.C(x), \neg D_k(x)\}$ must be closed. Consider two possible cases

Case 1. If $D_k = F_k$, then $\mathcal{A} = \{\forall R.C(x),\ \neg F_k(x)\}$. Note that $\neg F_k$ does not contain any R-quantification. Then, no R-successor of x can be generated, and $\forall R.C(x)$ cannot be further expanded. In this case, for each ABox generated from expanding $\mathcal{A}$, a clash must be introduced by $\neg F_k$, and thus we have $\models D_k \equiv \top$. In this case, D_k can be removed from the conjunction of D.

In particular, if $R \notin \mathsf{sig}(D)$ (that is, D does not contain any occurrence of R), then $\models D_k \equiv \top$ for each k and thus, $\models D \equiv \top$.

Case 2. If R appears in D_k, then $\mathcal{A} = \{\forall R.C(x),\ \neg D_k(x)\}$ can be expanded by T-rules into

$$\{\forall R.C(x),\ \exists R.\neg U_{k,1}(x),\ \ldots,\ \exists R.\neg U_{k,n}(x),\ \forall R.\neg E_k(x),\ \neg F_k(x)\}.$$

In the rest of this section we will not list all the concepts in a label when no ambiguity is caused. In particular, for simplicity, only those concepts that can be further expanded and will possibly introduce clashes are explicitly listed.

By further expansion with $\exists$- and $\forall$-rules, we generate n R-successors y_i of x with $\neg U_{k,i}(y_i)$, $C(y_i)$, and $\neg E_k(y_i)$ added into $\mathcal{A}$. In fact, each concept $\exists R.\neg U_{k,i}$ produces a new individual y_i.

Because $\neg F_k$ does not contain any R-quantification, no new role assertion for R can be added into an ABox that has been derived so far. And there is no way to generate any additional new assertions about y_i from $\neg F_k$.

If a clash is introduced by $\neg F_k$, then again, we have $\models D_k \equiv \top$, and D_k can be removed from the conjunction of D. Otherwise, for each ABox generated from expanding $\mathcal{A}$, a clash must occur among the assertions about y_i for some y_i. As each y_i is expanded independently from the other y_j $(j \neq i)$, a clash must occur at the same y_i for each ABox generated. That is, by tableau algorithm, $\models C \sqsubseteq U_{k,i_k} \sqcup E_k$ for some i_k with $1 \leq i_k \leq n$. By Lemma 2, $\models D_k \equiv \bigsqcup_{1\leq i\leq n}[\forall R.(U_{k,i} \sqcup E_k)] \sqcup \exists R.E_k \sqcup F_k$. Thus, we have $\models \forall R.C \sqsubseteq \forall R.(U_{k,i_k} \sqcup E_k)$ and $\models \forall R.C \sqsubseteq D_k$.

Therefore,

$$\models \forall R.C \sqsubseteq \bigsqcap_k [\forall R.(U_{k,i_k} \sqcup E_k)].$$

As $\models \sqcap_k[\forall R.(U_{k,i_k} \sqcup E_k)] \equiv \forall R.[\sqcap_k(U_{k,i_k} \sqcup E_k)]$, we take $C' = \sqcap_k(U_{k,i_k} \sqcup E_k)$. We have proved ph(1).

ph(2) Because $\models \exists R.C \sqsubseteq D$, we have $\models \exists R.C \sqsubseteq D_k$ for each k.

Then all the ABoxes generated from expanding $\mathcal{A} = \{\exists R.C(x),\ \neg D_k(x)\}$ must be closed. Consider two cases:

Case 1. If $D_k = \neg F_k$, then $\mathcal{A} = \{\exists R.C(x),\ \neg F_k(x)\}$. Then the $\exists$-rule generates an R-successor y of x with assertion $C(y)$ added into $\mathcal{A}$, but no assertion about y can be generated from from $\neg F_k$. A clash cannot occur among assertions about y, because otherwise we would have $\models C \sqsubseteq \bot$. Thus, a clash must be introduced by $\neg F_k$. We have shown $\models D \sqsubseteq \top$ in the case $R \notin \mathsf{sig}(D)$.

Case 2. If D_k contains R, then $\mathcal{A}$ is expanded by T-rules into

$$\{\exists R.C(x),\ \exists R.\neg U_{k,1}(x),\ \ldots,\ \exists R.\neg U_{k,n}(x),\ \forall R.\neg E_k(x),\ \neg F_k(x)\}.$$

By further expansion with $\exists$- and $\forall$-rules, we generate an R-successor y of x with assertions $C(y)$ and $\neg E_k(y)$, and n other R-successors z_i of x with assertions $\neg U_{k,i}(z_i)$ and $\neg E_k(z_i)$ for each z_i.

Again, there is no way to add any new assertions about y or z_i $(1 \leq i \leq n)$ from $\neg F_k$. Neither can any new R-successor be generated.

For each ABox generated from expanding $\mathcal{A}$, if there is a clash occurring among the assertions about z_i for some z_i, or be introduced by $\neg F_k$, then we have $\models D_k \equiv \top$, and D_k can be removed from the conjunction of D. Otherwise, a clash must occur among the assertions about y, for each ABox generated. That is, by tableau algorithm, $\models C \sqsubseteq E_k$. Then $\models \exists R.C \sqsubseteq \exists R.E_k$ and $\models \exists R.C \sqsubseteq D_k$.

Therefore, $\models \exists R.C \sqsubseteq \sqcap_k(\exists R.E_k)$ and $\models \exists R.C \sqsubseteq \sqcap_k D_k$. Let $C'_k = E_k$ for each k. We have proved ph(2). Now we are ready to show Proposition 11.

Proof of Proposition 11. To prove ph(CF1) for ph(1) and ph(2), we note that $\models C \sqsubseteq \mathsf{cforget}(C, \mathcal{V})$ implies $\models \forall R.C \sqsubseteq \forall R.\mathsf{cforget}(C, \mathcal{V})$ and $\models \exists R.C \sqsubseteq \exists R.\mathsf{cforget}(C, \mathcal{V})$. For ph(CF2), let D be an arbitrary concept such that $\models \forall R.C \sqsubseteq D$ and $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$. If $R \in \mathcal{V}$ and $R \notin \mathsf{sig}(D)$, by Lemma 3, we have $\models \top \sqsubseteq D$. Otherwise, if $R \notin \mathcal{V}$, we want to show that $\models \forall R.\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq D$ for the statement ph(1), and $\models \exists R.\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq D$ for the statement ph(2). ■

(1) By Lemma 3, if $\models \forall R.C \sqsubseteq D$, then there exists a concept description C' such that $\mathsf{sig}(C') \subseteq \mathsf{sig}(D)$, $\models C \sqsubseteq C'$ and $\models \forall R.C \sqsubseteq \forall R.C'$. Because $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$, it is obvious that $\mathsf{sig}(C') \cap \mathcal{V} = ptyset$. By the definition of c-forgetting, $\models \mathsf{cforget}(C, \mathcal{V}) \models C'$. Thus $\models \forall R.\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq \forall R.C'$ and $\models \forall R.\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq D$.

(2) By Lemma 3, if $\models \exists R.C \sqsubseteq D$, then $\models C \sqsubseteq C'_k$ and $\models \exists R.C \sqsubseteq \sqcap_k(\exists R.C'_k)$ for a collection of concepts C'_k such that $\mathsf{sig}(C'_k) \subseteq \mathsf{sig}(D)$. Hence, $\models \exists R.C \sqsubseteq D$. Observe that $\mathsf{sig}(C'_k) \cap \mathcal{V} = ptyset$. By the definition of c-forgetting, $\models \mathsf{cforget}(C, \mathcal{V}) \models C'_k$ for each k and thus, $\models \exists R.\mathsf{cforget}(C, \mathcal{V}) \sqsubseteq \sqcap_k(\exists R.C'_k)$ and $\exists R.\mathsf{cforget}(C, \mathcal{V}) \models\sqsubseteq D$.

Before we can show Theorem 1, we need the following three results (Lemma 4, Propositions 14 and 15). These results themselves are interesting because they reveal some insightful relationships between concept subsumption and c-forgetting.

Lemma 4. Let R be a role name,

$$C = \forall R.U \sqcap \bigsqcap_{1 \leq i \leq m} \exists R.E_i \sqcap F$$

such that $\not\models C \equiv \bot$, and

$$D = \bigsqcup_{1 \leq j \leq n} (\forall R.U'_j) \sqcup \exists R.E' \sqcup F',$$

where U, E_i's, U'_j's and E' are concepts, F is a conjunction, and F' is a disjunction of literals and non-R-quantifications.

If $\models C \sqsubseteq D$, then at least one of the following four conditions holds:

(1) $\models D \equiv \top$.
(2) $\models F \sqsubseteq F'$, and $\models C \sqsubseteq F'$.
(3) if $\not\models E' \equiv \bot$, then $\models E_i \sqcap U \sqsubseteq E'$ for some i, and $\models C \sqsubseteq \exists R.E'$.
(4) if $n > 0$, then $\models U \sqsubseteq E' \sqcup U'_j$, and $\models C \sqsubseteq \forall R.(E' \sqcup U'_j)$ for some j.

Proof. By the completeness of tableau algorithm, each ABox generated from expanding $\mathcal{A} = \{ C(x) \sqcap \neg D(x) \}$ must be closed. By T-rules, $\mathcal{A}$ can be expanded into

$$\{\forall R.U(x), \exists R.E_1(x), \ldots, \exists R.E_m(x), F(x)$$
$$\exists R.\neg U'_1(x), \ldots, \exists R.\neg U'_n(x), \forall R.\neg E'(x), \neg F'(x)\}.$$

Again, we do not list all the concepts in the label, for the sake of convenience. In particular, only those concepts that can be further expanded and will possibly introduce clashes are listed.

By further expansion with $\exists$- and $\forall$-rules, we generate m R-successors y_i of x with assertions $E_i(y_i)$, $U(y_i)$ and $\neg E'(y_i)$ added to $\mathcal{A}$ for each y_i. Also, n other R-successors z_j of x are generated with assertions $\neg U'_j(z_j)$, $U(z_j)$ and $\neg E'(z_j)$ added to $\mathcal{A}$ for each z_j. Note that F and $\neg F'$ do not contain any R-quantification, and no other R-successor of x can be generated. Also, there is no way to add any additional new assertions about y_i $(1 \leq i \leq m)$ or z_j $(1 \leq j \leq n)$ from F or $\neg F'$.

As $\not\models C \equiv \bot$, a clash cannot be introduced only by F. Thus, there are four possible cases, of which at least one must hold for all the ABoxes generated from expanding $\mathcal{A}$.

(1) A clash is introduced by $\neg F'$ only: in this case we have $\models \top \sqsubseteq F'$ and thus $\models D \equiv \top$.
(2) A clash is jointly caused by F and $\neg F'$: By the tableau algorithm, we have $\models F \sqsubseteq F'$.
(3) A clash occurs among the assertions about y_i for some $1 \leq i \leq m$. Note that such an i is fixed for all the ABoxes generated. That is, by tableau algorithm, $\models E_i \sqcap U \sqsubseteq E'$. By Lemma 2, $\models C \equiv \forall R.U \sqcap \sqcap_{1 \leq i \leq m}[\exists R.(E_i \sqcap U)] \sqcap F$. Thus, we have $\models C \sqsubseteq \exists R.E'$.
(4) A clash occurs among the assertions about z_j for some $1 \leq j \leq n$. Again, such a j is fixed for all the ABoxes generated. That is, by tableau algorithm, $\models U \sqsubseteq E' \sqcup U'_j$. By Lemma 2, $\models D \equiv \bigsqcup_{1 \leq j \leq n}[\forall R.(U'_j \sqcup E')] \sqcup \exists R.E' \sqcup F'$. Thus, we have $\models C \sqsubseteq \forall R.(U'_j \sqcup E')$. ■

Now we can show a more general property of forgetting with respect to quantifiers than Proposition 11.

Proposition 15. Let $\mathcal{V}$ be a set of variables and $C = \forall R.U \sqcap \sqcap_{1 \leq i \leq m} \exists R.E_i \sqcap F$ with $\not\models C \equiv \bot$, where R is a role name, U, E_i's are concepts, and F is a conjunction of literals and non-R-quantifications. Then

$$\mathsf{cforget}(C, \mathcal{V}) = \mathsf{cforget}(F, \mathcal{V}) \quad \text{if} \quad R \in \mathcal{V}, \quad \text{and} \quad \text{otherwise,}$$

$$\mathsf{cforget}(C, \mathcal{V}) = \forall R.\mathsf{cforget}(U, \mathcal{V}) \sqcap \bigsqcap_{1 \leq i \leq m} [\exists R.\mathsf{cforget}(E_i \sqcap U, \mathcal{V})] \sqcap \mathsf{cforget}(F, \mathcal{V}).$$

Proof. Consider two cases

Case1. $R \in \mathcal{V}$: We want to show that $\mathsf{cforget}(C, \mathcal{V}) = \mathsf{cforget}(F, \mathcal{V})$.

(CF1): By the assumption, we have $\models C \sqsubseteq F$ and thus, by $F \sqsubseteq \mathsf{cforget}(F, \mathcal{V})$, $\models C \sqsubseteq \mathsf{cforget}(F, \mathcal{V})$ holds.
(CF2): Suppose that D is a concept such that $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$ and $\models C \sqsubseteq D$. We need to prove that $\models \mathsf{cforget}(F, \mathcal{V}) \sqsubseteq D$ if $R \in \mathcal{V}$.

First, the concept D can be equivalently transformed into a conjunctive normal form $D = \sqcap_{1\le k\le n} F_k$, where each F_k is a disjunction of literals and non-R-quantifications. Without loss of generality, we assume that $\not\models F_k \equiv \top$ for all k. Note that $\models C \sqsubseteq D$ iff $\models C \sqsubseteq F_k$ for each k.

By Lemma 4, we have $\models F \sqsubseteq F_k$ for each k, which implies $\models F \sqsubseteq \sqcap_{1\le k\le n} F_k$. Thus, $\models F \sqsubseteq D$, and $\models \mathsf{cforget}(F, \mathcal{V}) \sqsubseteq D$.

Case 2. $R \notin \mathcal{V}$: Suppose that D is a concept such that $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$ and $\models C \sqsubseteq D$. Denote

$$C' = \forall R.\mathsf{cforget}(U, \mathcal{V}) \sqcap \bigsqcap_{1\le i\le m} [\exists R.\mathsf{cforget}(E_i \sqcap U, \mathcal{V})] \sqcap \mathsf{cforget}(F, \mathcal{V}).$$

We want to show that $\mathsf{cforget}(C, \mathcal{V}) = C'$.

(CF1): We want to show that $\models C \sqsubseteq C'$. In fact, by Lemma 2, $\models C \equiv \forall R.U \sqcap \sqcap_{1\le i\le m}[\exists R.(E_i \sqcap U)] \sqcap F$. We note that $\models U \sqsubseteq \mathsf{cforget}(U, \mathcal{V})$, $\models (E_i \sqcap U) \sqsubseteq \mathsf{cforget}(E_i \sqcap U, \mathcal{V})$, and $\models F \sqsubseteq \mathsf{cforget}(F, \mathcal{V})$. Therefore, we have $\models C \sqsubseteq C'$.

(CF2): Suppose that D is a concept such that $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$ and $\models C \sqsubseteq D$, we want to show that $\models C' \sqsubseteq D$ holds if $R \notin \mathcal{V}$.

Still, we assume that the concept D is of the conjunctive normal form $D = \sqcap_{1\le k\le n} D_k$, where each D_k is of the form

$$\bigsqcup_{1\le i\le n_k} \forall R.U_{k,i} \sqcup \exists R.G_k \sqcup F_k,$$

where G_k and $U_{k,i}$ $(1 \le i \le n_k)$ are concepts, F_k is a disjunction of literals and non-R-quantifications. Note that $n_k = 0$ if D_k does not have any universal R-quantification because a disjunct. Also, $G_k = \bot$ if D_k does not have any existential R-quantification as a disjunct.

Again, we assume that $\not\models D_k \equiv \top$ for each $1 \le k \le n$.

From $\models C \sqsubseteq D$, it follows that $\models C \sqsubseteq D_k$ for each k $(1 \le k \le n)$. By Lemma 4, $\models C \sqsubseteq D_k$ implies at least one of the following three cases:

(1) $\models F \sqsubseteq F_k$ and $\models C \sqsubseteq F_k$; or

(2) $\not\models G_k \equiv \bot$, $\models E_i \sqcap U \sqsubseteq G_k$ for some i $(1 \le i \le m)$, and $\models C \sqsubseteq \exists R.G_k$; or

(3) $n_k > 0$, $\models U \sqsubseteq U_k$ with $U_k = G_k \sqcup U_{k,j}$ for some $1 \le j \le n_k$, and $\models C \sqsubseteq \forall R.U_k$. We can divide the collection of D_k's $(1 \le k \le n)$ into three disjoint classes with $\{1, 2, \ldots, n\} = K_1 \cup K_2 \cup K_3$ such that $k \in K_j$ iff D_k satisfies the above condition (j) for $j = 1, 2, 3$. If D_k satisfies more than one of the conditions, we put it in only one class.

Construct

$$D' = \bigsqcap_{k\in K_1} F_k \sqcap \bigsqcap_{k\in K_2} (\exists R.G_k) \sqcap \bigsqcap_{k\in K_3} (\forall R.U_k).$$

Obviously, D' does not contain any variable from $\mathcal{V}$, and $\models C \sqsubseteq D'$ and $\models D' \sqsubseteq D$.

Also, for each $k \in K_1$, $\models F \sqsubseteq F_k$ holds, which implies $\models \mathsf{cforget}(F, \mathcal{V}) \sqsubseteq F_k$. For each $k \in K_2$, there exists E_i $(1 \leq i \leq m)$ in C such that $\models E_i \sqcap U \sqsubseteq G_k$ for each k and thus, $\models \mathsf{cforget}(E_i \sqcap U, \mathcal{V}) \sqsubseteq G_k$. That is, for each $k \in K_2$, there always exists some $1 \leq i \leq m$ such that $\models \exists R.\mathsf{cforget}(E_i \sqcap U, \mathcal{V}) \sqsubseteq \exists R.G_k$. This implies

$$\models \bigsqcap_{1 \leq i \leq m} \exists R.\mathsf{cforget}\left(E_i \sqcap U, \mathcal{V}\right) \sqsubseteq \bigsqcap_{k \in K_2} (\exists R.G_k\Big).$$

Similarly, for each $k \in K_3$, we have $\models U \sqsubseteq U_k$, which implies $\models \mathsf{cforget}(U, \mathcal{V}) \sqsubseteq U_k$. Thus

$$\models \forall R.\mathsf{cforget}(U, \mathcal{V}) \sqsubseteq \forall R.(\bigsqcap_{k \in K_3} U_k).$$

As $\models \sqcap_{k \in K_3}(\forall R.U_k) \equiv \forall R.(\sqcap_{k \in K_3} U_k)$, we have

$$\models \forall R.\mathsf{cforget}(U, \mathcal{V}) \sqsubseteq \bigsqcap_{k \in K_3}(\forall R.U_k).$$

Combining the above arguments, we can conclude that $\models C' \sqsubseteq D'$. Therefore, $\models C' \sqsubseteq D$. ■

To prove Theorem 1, it is enough to consider the correctness of Algorithm 1 in each disjunct, as c-forgetting distributes over disjunction by Proposition 10.

Proposition 16. Let $\mathcal{V}$ be a set of variables and let C be a concept of the form

$$C = \bigsqcap_{1 \leq i \leq m} L_i \bigsqcap \bigsqcap_{R \in \mathcal{R}} C_R,$$

where each L_i is a literal, $\mathcal{R}$ is the set of role names in C, and concept C_R is a conjunction of R-quantifications.

Then

$$\mathsf{cforget}(C, \mathcal{V}) = \bigsqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i \sqcap \bigsqcap_{R \in \mathcal{R}} \mathsf{cforget}(C_R, \mathcal{V}).$$

Here L_i^+ is the concept of the literal L_i.

Proof. Note that for each $R \in \mathcal{R}$, C_R can be equivalently transformed into a form of $\forall R.U \sqcap \sqcap_k \exists R.E_k$. By Proposition 15, given a conjunction F of literals and non-R-quantifications, $\mathsf{cforget}(C_R \sqcap F, \mathcal{V}) = \mathsf{cforget}(C_R, \mathcal{V}) \sqcap \mathsf{cforget}(F, \mathcal{V})$. Based on this observation, by a simple induction on the number of roles in $\mathcal{R}$, we can show that

$$\mathsf{cforget}(C, \mathcal{V}) = \mathsf{cforget}\left(\bigsqcap_{1 \leq i \leq m} L_i, \mathcal{V}\right) \sqcap \bigsqcap_{R \in \mathcal{R}} \mathsf{cforget}(C_R, \mathcal{V}).$$

Thus, we need only to show that

$$\mathsf{cforget}\left(\bigsqcap_{1 \leq i \leq m} L_i, \mathcal{V}\right) = \bigsqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i.$$

It is obvious that $\models \sqcap_{1 \leq i \leq m} L_i \sqsubseteq \sqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i$. Suppose $\models \sqcap_{1 \leq i \leq m} L_i \sqsubseteq D$ and $\mathsf{sig}(D) \cap \mathcal{V} = ptyset$. Let $D = \sqcap D_k$ and each D_k is of the form $\bigsqcup_{R \in \mathcal{R}'} C'_R \sqcup \bigsqcup_{1 \leq j \leq n} L'_j$, where each C'_R is a disjunction of quantifications about R and each L'_j a literal. Because $\models \sqcap_{1 \leq i \leq m} L_i \sqsubseteq D_k$ for each k, we can show through the tableau that for each k there must exist some i $(1 \leq i \leq m)$ and some j $(1 \leq j \leq n)$ such that $L_i = L'_j$. This implies that there

exists $M \subseteq \{1, \ldots, m\}$ such that $\models \sqcap_{k \in M} L_k \sqsubseteq D$ and for each $k \in M$, $L_k^+ \notin \mathcal{V}$. Obviously, $\models \sqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i \sqsubseteq \sqcap_{k \in M} L_k$. Thus $\models \sqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i \sqsubseteq D$.

We have shown that $\mathsf{cforget}(\sqcap_{1 \leq i \leq m} L_i, \mathcal{V}) = \sqcap_{1 \leq i \leq m, L_i^+ \notin \mathcal{V}} L_i$. ■

Proof of Theorem 1. The correctness of Algorithm 1 is clearly seen from Proposition 10, Proposition 16, and Proposition 11. ■