Forgetting Concepts in DL-Lite

Zhe Wang[1], Kewen Wang[1], Rodney Topor[1], and Jeff Z. Pan[2]

[1] Griffith University, Australia
[2] The University of Aberdeen, UK

Abstract. To support the reuse and combination of ontologies in Semantic Web applications, it is often necessary to obtain smaller ontologies from existing larger ontologies. In particular, applications may require the omission of many terms, e.g., concept names and role names, from an ontology. However, the task of omitting terms from an ontology is challenging because the omission of some terms may affect the relationships between the remaining terms in complex ways. We present the first solution to this problem by adapting the technique of forgetting, previously used in other domains. Specifically, we present a semantic definition of forgetting for description logics in general, which generalizes the standard definition for classical logic. We then introduce algorithms that implement forgetting in both DL-Lite TBoxes and ABoxes, and in DL-Lite knowledge bases. We prove that the algorithms are correct with respect to the semantic definition of forgetting, and that they run in polynomial time.

1 Introduction

Ontologies are required for Semantic Web applications as they provide a shared representation of the terms and the relationships between terms in particular application domains. Such Semantic Web applications will normally require access to only some of the terms, i.e., concept and role names, in available ontologies. However the tasks of restricting attention to relevant terms in, or omitting irrelevant terms from, a given ontology is challenging because the omission of some terms generally affects the relationships between the remaining terms. Accordingly, the ontological engineering tasks of combining and transforming large ontologies have received extensive attention recently.

Current technologies, however, provide only limited support for such operations. The web ontology language OWL allows users to *import* one ontology into another using the $\langle owl : imports \rangle$ statement. Most ontology editors allow the reuse of another ontology by including it in the model that is being designed. For example, Protégé allows user to include other projects. However, these approaches to ontology reuse have at least two limitations: (1) Some ontologies are very large and ontology engineers might need to import only a part of the available ontologies. For instance, the medical ontology UMLS [3] contains around 300,000 terms and only some of these will be relevant to any particular application. (2) The ability to deal with inconsistency and contradiction caused by merging ontologies is very limited in these approaches. Hence, researchers

[3] `http://www.nlm.nih.gov/research/umls`

have proposed alternative approaches to address these problems in reusing and merging ontologies. In the literature, the problem of reusing portions of large ontologies is referred to as *partial use of ontologies*.

Consider the following scenario. Suppose we need an ontology about $Cancer$ but only have a large ontology $Medicine$, which describes many diseases and treatments. It would not be efficient to adopt and use the whole ontology $Medicine$, since we only need a small part of its contents. A better strategy is to discard those terms that are not required (such as $Odontia$) and use the resulting restriction of the ontology.

Another scenario occurs when ontology engineers are constructing a complex ontology. The engineers may wish to conveniently delete terms which are unnecessary or poorly defined. This task is relatively easy in traditional database systems, but in an ontology, where concepts are more closely related to each other, simply removing a concept or role name may destroy the consistency of the ontology and may cause problems with subsequent reasoning. More sophisticated methods for deleting (omitting/hiding) information in ontologies are required.

In this paper we address the issue of partial ontology reuse by employing the technique of *forgetting*, which has been previously been thoroughly studied in classical logic [13, 12] and logic programming [6, 14]. Informally, forgetting is a particular form of reasoning that allows a piece of information (say, p) in a knowledge base to be discarded or hidden in such a way that future reasoning on information irrelevant to p will not be affected. Forgetting has proved to be a very useful tool in many tasks such as query answering, planning, decision-making, reasoning about actions, knowledge update and revision in classical logic [13, 12] and logic programming [6, 14]. However, to the best of our knowledge, forgetting has not previously been applied to description logic ontology reuse.

In particular, we study new techniques for forgetting from knowledge bases in the DL-Lite family of description logics. This family was recently proposed by Calvanese *et al.* [2–4]. Logics in this family are particularly attractive because they are expressive enough for many purposes and have polynomial time reasoning algorithms in the worst case, in contrast to more common description logics which have exponential time reasoning algorithms. Indeed, logics in the DL-Lite family have been proved to be the maximal logics allowing efficient conjunctive query answering using standard database technology.

The main contributions of the paper include the following.

1. We provide a semantic definition of forgetting from DL-Lite TBoxes, which also applies to other description logics.
2. We prove that the result of forgetting from TBoxes in languages of the DL-Lite family can always be expressed in the same languages, and present algorithms to compute TBox forgetting for such languages.
3. We introduce a definition of forgetting for DL-Lite ABoxes, which preserves conjunctive query answering, and use this to define forgetting for arbitrary DL-Lite knowledge bases.
4. Finally, we prove our algorithms are correct and that they run in polynomial time.

The rest of the paper is organized as follows. Some basics of DL-Lite are briefly recalled in Section 2. We present the semantic definition of forgetting in arbitrary de-

scription logic (DL) TBoxes in Section 3 and show the DL forgetting has the same desirable properties that classical forgetting has. In Section 4, we introduce our algorithms for computing the result of forgetting in DL-Lite TBoxes, and show the algorithms are correct with respect to the semantic definition. We note that the forgetting algorithms are simple, run in polynomial time, and do not make other reasoning processes more complex. The forgetting technique and the results are then extended to DL-Lite ABoxes and knowledge bases in Section 5, and detailed examples are presented to show how forgetting algorithms work. Finally, Section 6 concludes the paper and discusses future work.

2 Preliminaries

Description logics (DLs) are a family of concept-based knowledge representation formalisms, equipped with well-defined model-theoretic semantics [1]. The DL-Lite family is a family of lightweight ontology languages that can express most features in UML class diagrams but still have low reasoning overheads [2]. Besides standard reasoning tasks like subsumption between concepts and satisfiability of knowledge bases, the issue of answering complex queries is especially considered. The DL-Lite family consists of a core language DL-Lite$_{core}$ and some extensions, among which two main extensions are DL-Lite$_{\mathcal{F},\sqcap}$ and DL-Lite$_{\mathcal{R},\sqcap}$.

The DL-Lite$_{core}$ language has the following syntax:

$$\begin{aligned} B &\longrightarrow A \mid \exists R \\ C &\longrightarrow B \mid \neg B \\ R &\longrightarrow P \mid P^- \end{aligned}$$

where A is an atomic concept and P is an atomic role (with P^- as its inverse). B is called a *basic concept*, R a *basic role* and C is called a *general concept*.

A DL-Lite$_{core}$ TBox is a set of inclusion axioms of the form

$$B \sqsubseteq C$$

A DL-Lite ABox is a set of membership assertions on atomic concepts and atomic roles:

$$A(a), P(a, b)$$

where a and b are constants.

A DL-Lite knowledge base (KB) is a tuple KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a DL-Lite TBox and $\mathcal{A}$ is a DL-Lite ABox.

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

$$\begin{aligned} (P^-)^{\mathcal{I}} &= \{(a, b) \mid (b, a) \in P^{\mathcal{I}}\} \\ (\exists R)^{\mathcal{I}} &= \{a \mid \exists b.(a, b) \in R^{\mathcal{I}}\} \\ (\neg B)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} \\ (B_1 \sqcap B_2)^{\mathcal{I}} &= B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}} \end{aligned}$$

An interpretation $\mathcal{I}$ is a model of $B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, *that is*, all instances of concept B are also instances of concept C.

An interpretation $\mathcal{I}$ is a model of $A(a)$ (resp., $P(a,b)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$).

An interpretation $\mathcal{I}$ is a model of a KB $\langle \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{I}$ is a model of all axioms of $\mathcal{T}$ and assertions of $\mathcal{A}$. A KB $\mathcal{K}$ is consistent if it has at least one model. Two KBs that have the same models are said to be equivalent. A KB $\mathcal{K}$ logically implies an axiom (or assertion) α, denoted $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

Although DL-Lite is a simple language, it is useful because it is sufficiently expressive and because conjunctive query evaluation over DL-Lite KBs is extremely efficient. A conjunctive query (CQ) $q(\boldsymbol{x})$ over a KB $\mathcal{K}$ is an expression of the form

$$\{\boldsymbol{x} \mid \exists \boldsymbol{y}.conj(\boldsymbol{x}, \boldsymbol{y})\}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ are lists of variables, $conj(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms, and atoms have the form $C(s)$ or $R(s,t)$, where C is a concept, R is a role, and s and t are either individual names or variables. Given an interpretation $\mathcal{I}$, $q^{\mathcal{I}}$ is the set of tuples of domain elements such that, when assigned to $\boldsymbol{x}$, $\exists \boldsymbol{y}.conj(\boldsymbol{x}, \boldsymbol{y})$ is true in $\mathcal{I}$. Given a CQ $q(\boldsymbol{x})$ and a KB $\mathcal{K}$, the answer to q over $\mathcal{K}$ is the set $\mathsf{ans}(q, \mathcal{K})$ of tuples $\boldsymbol{a}$ of constants in $\mathcal{K}$ such that $\boldsymbol{a}^{\mathcal{I}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

DL-Lite$_{\mathcal{F},\sqcap}$ extends DL-Lite$_{core}$ by allowing for conjunction of concepts in the left-hand side of inclusions and role functionality axioms in TBoxes. The extended syntax is:

$$\begin{array}{l} B \longrightarrow A \mid \exists R \\ D \longrightarrow B \mid D_1 \sqcap D_2 \\ C \longrightarrow B \mid \neg B \\ R \longrightarrow P \mid P^- \end{array}$$

and TBoxes contain axioms of the form:

$$D \sqsubseteq C \quad \text{and} \quad (\mathit{funct}\ R).$$

Given an interpretation $\mathcal{I}$, we define $(D_1 \sqcap D_2)^{\mathcal{I}} = {D_1}^{\mathcal{I}} \cap {D_2}^{\mathcal{I}}$. Then $\mathcal{I}$ is a model of $(\mathit{funct}\ R)$ iff $(a, b_1) \in R^{\mathcal{I}}$ and $(a, b_2) \in R^{\mathcal{I}}$ implies $b_1 = b_2$.

DL-Lite$_{\mathcal{R},\sqcap}$ extends DL-Lite$_{core}$ by allowing for conjunctions of concepts in the left-hand side of inclusions axioms, role complements, and role inclusion axioms in TBoxes. In this case the extended syntax is:

$$\begin{array}{l} B \longrightarrow A \mid \exists R \\ D \longrightarrow B \mid D_1 \sqcap D_2 \\ C \longrightarrow B \mid \neg B \\ R \longrightarrow P \mid P^- \\ S \longrightarrow R \mid \neg R \end{array}$$

and TBoxes contain axioms of the form:

$$D \sqsubseteq C \quad \text{and} \quad R \sqsubseteq S.$$

Given an interpretation $\mathcal{I}$, we define $(\neg R)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$. Then $\mathcal{I}$ is a model of $R \sqsubseteq E$ iff $R^{\mathcal{I}} \subseteq E^{\mathcal{I}}$.

We note that the data complexity of DL-Lite$_{core}$ is log-space whereas the data complexity of the two extensions (as defined here) is polynomial-time.

Example 2.1. The following is a simple DL-Lite$_{core}$ knowledge base "Library", which describes the resources, users and lending policies of a library. This knowledge base has a TBox $\mathcal{T}$ consisting of the following axioms:

$\exists onLoanTo \sqsubseteq LibItem$, $\exists onLoanTo^{-} \sqsubseteq Member$,
$Member \sqsubseteq Person$, $Visitor \sqsubseteq Person$, $Visitor \sqsubseteq \neg Member$,
$LibItem \sqsubseteq \exists hasCatNum$, $\exists hasCatNum^{-} \sqsubseteq CatNum$,
$\exists hasCatNum \sqcap Missing \sqsubseteq \neg LibItem$

and an ABox $\mathcal{A}$ consisting of the following assertions:

$LibItem(SWPrimer)$, $onLoanTo(DLHandBook, Jack)$.

Here, $LibItem$ denotes library items, $onLoanTo$ denotes the loan relationship between library items and people, $CatNum$ denotes catalogue numbers, and $hasCatNum$ denotes the relationship between library items and their catalogue numbers. Note that not every person is a member or visitor.

If we regarded this example as a knowledge base of DL-Lite$_{\mathcal{F},\sqcap}$, we could add the TBox axioms $(funct\ onLoanTo)$ and $(funct\ hasCatNum)$.

3 Forgetting concepts from DL-Lite TBoxes

In this section, we define the operation of forgetting about a concept A in a TBox $\mathcal{T}$. Informally, the TBox that results from forgetting about A in $\mathcal{T}$ should: (1) not contain any occurrence of A, (2) be weaker than $\mathcal{T}$, and (3) give the same answer to any query that is irrelevant to A. We will first give a semantic definition of forgetting in DL-Lite, investigate its properties and in the next section introduce algorithms for computing the result of forgetting in different languages of the family DL-Lite.

Let $\mathcal{L}$ be a DL-Lite language. The signature $\mathcal{S}_{\mathcal{L}}$ of $\mathcal{L}$ is the set of concept and role names in $\mathcal{L}$. We will omit the subscript if there is no confusion caused. Our semantic definition of forgetting in DL-Lite is an adaption of the corresponding definition for classical logic.

Let A be an atomic concept name in $\mathcal{L}$ and $\mathcal{I}_1$ and $\mathcal{I}_2$ interpretations of $\mathcal{L}$. We define $\mathcal{I}_1 \sim_A \mathcal{I}_2$ iff $\mathcal{I}_1$ and $\mathcal{I}_2$ agree on all atomic and concept role names except possibly A:

1. $\mathcal{I}_1$ and $\mathcal{I}_2$ have the same domain ($\Delta^{\mathcal{I}_1} = \Delta^{\mathcal{I}_2}$), and interpret every individual name the same ($a^{\mathcal{I}_1} = a^{\mathcal{I}_2}$ for every individual name a).
2. For every concept name A_1 distinct from A, $A_1^{\mathcal{I}_1} = A_1^{\mathcal{I}_2}$.
3. For every role name P, $P^{\mathcal{I}_1} = P^{\mathcal{I}_2}$.

Clearly, $\sim_A$ is an equivalence relation, and we say $\mathcal{I}_1$ is A-equivalent to $\mathcal{I}_2$.

Definition 3.1. *Let $\mathcal{T}$ be a TBox in $\mathcal{L}$ and A an atomic concept in $\mathcal{T}$. A TBox $\mathcal{T}'$ on the signature $\mathcal{S} \setminus \{A\}$ is a* result of forgetting *about A in $\mathcal{T}$ if any interpretation $\mathcal{I}'$ is a model of $\mathcal{T}'$ if and only if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $\mathcal{I} \sim_A \mathcal{I}'$.*

It follows from the above definition that the result of forgetting about an atomic concept A in a TBox $\mathcal{T}$ is unique in the sense that, if both $\mathcal{T}'$ and $\mathcal{T}''$ are results of forgetting about A in $\mathcal{T}$, then they are equivalent. So we will use $\mathsf{forget}(\mathcal{T}, A)$ to denote the result of forgetting about A in $\mathcal{T}$ throughout the paper.

Obviously, $\mathsf{forget}(\mathcal{T}, A)$ does not contain any occurrence of A and is weaker than $\mathcal{T}$. However, the definition of forgetting guarantees that $\mathsf{forget}(\mathcal{T}, A)$ and $\mathcal{T}$ are equivalent under query answering on $\mathcal{S} \setminus \{A\}$.

Note that the above definition of forgetting can be applied to other description logics.

Example 3.1. Consider the TBox $\mathcal{T}$ in Example 2.1. Suppose the library now wishes to allow nonmembers to borrow library items but still wishes to prevent visitors from borrowing library items, *i.e*, suppose the library wishes to forget about atomic concept $Member$ in $\mathcal{T}$. From the definition, $\mathsf{forget}(\mathcal{T}, Member)$ now consists of the following axioms:

$\exists onLoanTo \sqsubseteq LibItem$, $\exists onLoanTo^- \sqsubseteq Person$,
$Visitor \sqsubseteq Person$, $\exists onLoanTo^- \sqsubseteq \neg Visitor$,
$LibItem \sqsubseteq \exists hasCatNum$, $\exists hasCatNum^- \sqsubseteq CatNum$,
$\exists hasCatNum \sqcap Missing \sqsubseteq \neg LibItem$.

We believe this definition correctly captures the informal operation of forgetting a concept from a TBox.

Definition 3.1 clearly captures our informal understanding of forgetting. However, we have not yet shown that the result of forgetting about a concept always exists or how to compute the result of forgetting. In the next section, we introduce algorithms for computing the result of forgetting in different DL-Lite languages. From the soundness and completeness of these algorithms, we can immediately conclude that the result of forgetting about concepts exists for every TBox in DL-Lite.

Theorem 3.1. *Let $\mathcal{T}$ be a TBox in a DL-Lite language $\mathcal{L}$ and A an atomic concept. Then the result of forgetting about A in $\mathcal{T}$ always exists and is in $\mathcal{L}$.*

Forgetting in DL-Lite has other important properties. In particular, it preserves reasoning relative to TBoxes.

Proposition 3.1. *Let $\mathcal{T}$ be a TBox in $\mathcal{L}$ and A an atomic concept. Let $\mathcal{T}' = \mathsf{forget}(\mathcal{T}, A)$. Then, for any ABox $\mathcal{A}$ on $\mathcal{S} \setminus \{A\}$, we have:*

- *The knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent iff $\langle \mathcal{T}', \mathcal{A} \rangle$ is consistent.*
- *For any inclusion axiom α not containing A, $\mathcal{T} \models \alpha$ iff $\mathcal{T}' \models \alpha$.*
- *For any membership assertion β not containing A, $\langle \mathcal{T}, \mathcal{A} \rangle \models \beta$ iff $\langle \mathcal{T}', \mathcal{A} \rangle \models \beta$.*
- *For any conjunctive query q not containing A, $\mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathsf{ans}(q, \langle \mathcal{T}', \mathcal{A} \rangle)$.*

It is straightforward to generalize Definition 3.1 to the operation of simultaneously forgetting about a *set* of concept names.

We can forget about a set of concept names by forgetting one by one since, if A_1 and A_2 are concept names, it is easy to show that

$$\mathsf{forget}(\mathsf{forget}(\mathcal{T}, A_1), A_2) \equiv \mathsf{forget}(\mathsf{forget}(\mathcal{T}, A_2), A_1).$$

This property allows us to define the result of forgetting a *set* $S = \{A_1, \ldots, A_n\}$ of concept names by

$$\mathsf{forget}(\mathcal{T}, S) \equiv \mathsf{forget}(\ldots(\mathsf{forget}(\mathcal{T}, A_1), \ldots), A_n).$$

4 Computing the Result of Forgetting in DL-Lite

In Example 3.1, the result of forgetting about $Member$ in the TBox $\mathcal{T}$ was obtained from $\mathcal{T}$ by simple syntax transformations. In this section, we introduce algorithms for computing the result of forgetting in different languages of DL-Lite. We prove that our algorithms are sound and complete with respect to the semantic definition of forgetting in the previous section. Our algorithms show that the result of forgetting in a DL-Lite TBox can always be obtained using simple syntax-based transformations.

A language of DL-Lite$_{core}$ (resp. DL-Lite$_{\mathcal{F},\sqcap}$, DL-Lite$_{\mathcal{R},\sqcap}$) is denoted $\mathcal{L}_{core}$ (resp. $\mathcal{L}_{\mathcal{F},\sqcap}$, $\mathcal{L}_{\mathcal{R},\sqcap}$). We first introduce the forgetting algorithm for DL-Lite$_{core}$ as Algorithm 1, then extend it to algorithms for DL-Lite$_{\mathcal{F},\sqcap}$ and DL-Lite$_{\mathcal{R},\sqcap}$. The basic idea of Algorithm 1 is to first transform the given TBox into a standard form and then remove all occurrence of A.

Algorithm 1 (Computing the result of forgetting in DL-Lite$_{core}$)
Input: A TBox $\mathcal{T}$ in $\mathcal{L}_{core}$ and an atomic concept A.
Output: $\mathsf{forget}(\mathcal{T}, A)$
Method:
Step 1. Remove axiom $A \sqsubseteq A$ from $\mathcal{T}$ if it is present.
Step 2. If axiom $A \sqsubseteq \neg A$ is in $\mathcal{T}$, remove each axiom $A \sqsubseteq C$ or $B \sqsubseteq \neg A$ from $\mathcal{T}$, and replace each axiom $B \sqsubseteq A$ in $\mathcal{T}$ by $B \sqsubseteq \neg B$.
Step 3. Replace each axiom $B \sqsubseteq \neg A$ in $\mathcal{T}$ by $A \sqsubseteq \neg B$.
Step 4. For each axiom $B_i \sqsubseteq A$ ($1 \leq i \leq m$) in $\mathcal{T}$ and each axiom $A \sqsubseteq C_j$ ($1 \leq j \leq n$) in $\mathcal{T}$, where each B_i is a basic concept and each C_j is a general concept, if $B_i \sqsubseteq C_j$ is not in $\mathcal{T}$ already, add $B_i \sqsubseteq C_j$ to $\mathcal{T}$.
Step 5. Return the result of removing every axiom containing A in $\mathcal{T}$.

Fig. 1. Forgetting in a DL-Lite$_{core}$ TBox.

Example 4.1. Consider the TBox $\mathcal{T}$ in Example 2.1 again. Algorithm 1 replaces the axioms

$$\exists onLoanTo^{-} \sqsubseteq Member, Member \sqsubseteq Person \text{ and } Visitor \sqsubseteq \neg Member$$

by the axioms

$$\exists onLoanTo^- \sqsubseteq Person \text{ and } \exists onLoanTo^- \sqsubseteq \neg Visitor,$$

which gives the same result as Example 3.1.

If the library wants to completely eliminate lending restrictions, then the result of forgetting about the concept $Visitor$ can be obtained by removing the following (lending restriction) axioms:

$$Visitor \sqsubseteq Person \text{ and } \exists onLoanTo^- \sqsubseteq \neg Visitor.$$

We can now show that Algorithm 1 is sound and complete with respect to the semantic definition of forgetting in DL-Lite$_{core}$ TBoxes.

Theorem 4.1. *Let $\mathcal{T}$ be a TBox in $\mathcal{L}_{core}$ and A an atomic concept appearing in $\mathcal{T}$. Then Algorithm 1 always returns* $\mathsf{forget}(\mathcal{T}, A)$.

Given the forgetting algorithm for a DL-Lite$_{core}$ TBox, we can extend it to compute the results of forgetting in DL-Lite$_{\mathcal{F},\sqcap}$ and DL-Lite$_{\mathcal{R},\sqcap}$ TBoxes.

Recall that both DL-Lite$_{\mathcal{F},\sqcap}$ and DL-Lite$_{\mathcal{R},\sqcap}$ extend DL-Lite$_{core}$ by allowing conjunctions of basic concepts in the left-hand side of inclusion axioms. Therefore, to extend Algorithm 1 to these two extensions, we need a method of handling such conjunctions.

For inclusion axioms in which the concept A occurs negatively, *i.e.*, axioms of the form of $D \sqsubseteq \neg A$, where D is a conjunction, we cannot transform it into an equivalent axiom $A \sqsubseteq \neg D$, as we did previously, since this is not an axiom in the DL-Lite language. However, the following two lemmas give useful properties of conjunctions that can be applied in the extended algorithm.

Lemma 4.1. *Let B, B' be basic concepts and D a conjunction of basic concepts, then the two axioms $B \sqcap D \sqsubseteq \neg B'$ and $B' \sqcap D \sqsubseteq \neg B$ are equivalent.*

In order to eliminate A in conjunctions in the left-hand side of inclusions, we can see that the proof to Theorem 4.1 can be safely extended by allowing conjunctions in left-hand side of inclusions, *i.e.*, the following assertion is true.

Lemma 4.2. *Suppose $\mathcal{T}$ is a TBox in $\mathcal{L}$ and $\mathcal{T} = \{ D_i \sqsubseteq A \mid i = 1, \ldots, m \} \cup \{ A \sqcap D'_j \sqsubseteq C_j \mid j = 1, \ldots, n \} \cup \mathcal{T}_A$, where $m, n \geq 0$, A is an atomic concept, each D_i is a nonempty conjunction, each D'_j is a (possibly empty) conjunction, each D_i, D'_j and C_j does not contain A, and $\mathcal{T}_A$ is a set of axioms that do not contain A. Then $\mathcal{T}' = \{D_i \sqcap D'_j \sqsubseteq C_j \mid i = 1, \ldots, m, j = 1, \ldots, n\} \cup \mathcal{T}_A$ = $\mathsf{forget}(\mathcal{T}, A)$. (If m or n is 0, then $\mathcal{T}'$ is the empty set).*

Because we are currently concerned only with forgetting concepts, the extensions to functional role axioms and role inclusion axioms are not relevant to our algorithm. Hence, the same algorithm implements concept forgetting for both DL-Lite$_{\mathcal{F},\sqcap}$ and DL-Lite$_{\mathcal{R},\sqcap}$ TBoxes. This algorithm is now given as Algorithm 2. Throughout Algorithm 2, D denotes a possibly empty conjunction and D' denotes a nonempty conjunction.

From the definition of Algorithm 2 it is easy to see that the following result holds.

Algorithm 2 (Computing the result of forgetting in DL-Lite$_{core}$ extensions)
Input: A TBox $\mathcal{T}$ in $\mathcal{L}_{\mathcal{F},\sqcap}$ or $\mathcal{L}_{\mathcal{R},\sqcap}$ and an atomic concept name A.
Output: $\mathsf{forget}(\mathcal{T}, A)$
Method:
Step 1. Remove each axiom $A \sqcap D \sqsubseteq A$ in $\mathcal{T}$.
Step 2. If axiom $A \sqsubseteq \neg A$ is in $\mathcal{T}$, remove each axiom $A \sqcap D \sqsubseteq C$ or $B \sqcap D \sqsubseteq \neg A$, and replace each axiom $B \sqcap D' \sqsubseteq A$ by $D' \sqsubseteq \neg B$ and $B \sqsubseteq A$ by $B \sqsubseteq \neg B$.
Step 3. Replace each axiom $A \sqcap D \sqsubseteq \neg A$ in $\mathcal{T}$ by $D \sqsubseteq \neg A$.
Step 4. Replace each axiom $B \sqcap D \sqsubseteq \neg A$ in $\mathcal{T}$ by $A \sqcap D \sqsubseteq \neg B$.
Step 5. For each axiom $D'_i \sqsubseteq A$ $(1 \leq i \leq m)$ in $\mathcal{T}$ and each axiom $A \sqcap D_j \sqsubseteq C_j$ $(1 \leq j \leq n)$ in $\mathcal{T}$ if $D'_i \sqcap D_j \sqsubseteq C_j$ is not in $\mathcal{T}$, simplify $D'_i \sqcap D_j$ and add $D'_i \sqcap D_j \sqsubseteq C_j$ to $\mathcal{T}$.
Step 6. Return the result of removing every axiom containing A in $\mathcal{T}$.

Fig. 2. Forgetting in DL-Lite$_{core}$ extension TBoxes.

Proposition 4.1. *Let $\mathcal{T}$ be a TBox $\mathcal{T}$ in $\mathcal{L}_{\mathcal{F},\sqcap}$ or $\mathcal{L}_{\mathcal{R},\sqcap}$ and A an atomic concept name appearing in $\mathcal{T}$. Then Algorithm 2 always terminates and takes polynomial time in the size of $\mathcal{T}$.*

It is an immediate corollary that Algorithm 1 has the same properties.

From Lemma 4.1 and Lemma 4.2, it can easily be shown that Algorithm 2 is sound and complete with respect to the semantic definition of forgetting for DL-Lite. In other words, we have following theorem.

Theorem 4.2. *Let $\mathcal{T}$ be a TBox $\mathcal{T}$ in $\mathcal{L}_{\mathcal{F},\sqcap}$ or $\mathcal{L}_{\mathcal{R},\sqcap}$ and A an atomic concept name appearing in $\mathcal{T}$. Then Algorithm 2 always returns* $\mathsf{forget}(\mathcal{T}, A)$.

Example 4.2. Recall the TBox $\mathcal{T}$ in Example 2.1. If we want to forget about concept name $LibItem$ in $\mathcal{T}$, then Algorithm 2 returns the following axioms, which comprise $\mathsf{forget}(\mathcal{T}, LibItem)$:

$\exists onLoanTo \sqsubseteq \exists hasCatNum$, $\exists onLoanTo^- \sqsubseteq Member$,
$Member \sqsubseteq Person$, $Visitor \sqsubseteq Person$, $Visitor \sqsubseteq \neg Member$,
$\exists hasCatNum^- \sqsubseteq CatNum$,
$\exists hasCatNum \sqcap \exists onLoanTo \sqsubseteq \neg Missing$.

5 Forgetting Concepts from DL-Lite Knowledge Bases

Given the semantic definition of forgetting in TBoxes, we want to extend the notion of forgetting to ABoxes and DL-Lite Knowledge Bases (KB). However, DL-Lite languages are not closed under ABox forgetting, i.e., the classical forgetting result in an DL-Lite ABox is not expressible in the same language. A simple example would be, a DL-Lite KB with one axiom $A \sqsubseteq \neg B$ in TBox and one assertion $A(a)$ in ABox. To forget about concept name A in ABox, we have to remove the assertion $A(a)$. However, the original KB has a logical consequence $\neg B(a)$, that is, a cannot be interpreted as a member of concept B, which can not be expressed by standard DL-Lite ABoxes. Once concept A is forgotten, we also lose the information about a and B.

Adopting classical forgetting into DL-Lite ABoxes might lead outside of the original language. However, we argue that, for ABoxes, the classical forgetting is not necessary. Since ABoxes are used for maintain membership information and for query-answering, an ABox forgetting operation, as long as it preserves membership relation (rather than non-membership relation) and query-answering, should be good enough for ontology uses.

Given a Knowledge Base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ in DL-Lite language $\mathcal{L}$, the result of forgetting about a concept name A in $\mathcal{K}$ should be a KB $\mathcal{K}'$ in $\mathcal{L}$ such that (1) $\mathcal{K}'$ does not contain any occurrence of A and (2) $\mathcal{K}$ and $\mathcal{K}'$ are equivalent w.r.t. any conjunctive query on the signature $\mathcal{L} \setminus \{A\}$.

Definition 5.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base in a DL-Lite language $\mathcal{L}$ and A an atomic concept in $\mathcal{A}$. An ABox $\mathcal{A}'$ on the signature $\mathcal{S} \setminus \{A\}$ is a* result of forgetting *about A in $\mathcal{A}$ with respect to $\mathcal{K}$ if, for any conjunctive query q on the signature of $\mathcal{S} \setminus \{A\}$, $\mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A}' \rangle)$ holds.*

Example 5.1. Consider the following ABox from Example 2.1:

$$\mathcal{A} = \{LibItem(SWPrimer), onLoanTo(DLHandBook, Jack)\}$$

To forget about atomic concept $LibItem$, we can't just remove the membership assertion $LibItem(SWPrimer)$, because we would lose information about $SWPrimer$. According to the TBox axiom, we know that $SWPrimer$ must have a catalogue number, so we can replace $LibItem(SWPrimer)$ with $hasCatNum(SWPrimer, z)$ where z denotes the catalogue number of $SWPrimer$. This is the result of forgetting about $LibItem$ in $\mathcal{A}$.

From the above example we can see that a major issue in forgetting about an atomic concept A in an ABox $\mathcal{A}$ is how to preserve subsumption relations between different concepts. For example, if $A \sqsubseteq B$ is in TBox $\mathcal{T}$ and $A(a)$ is in $\mathcal{A}$, then $B(a)$ is derivable from $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. However, if we simply delete $A(a)$ from $\mathcal{A}$, we will be unable to derive $B(a)$ from the resulting knowledge base. Hence, we have to propagate selected information before a concept name can be forgotten. For convenience, we slightly extend our definition of ABoxto allow variables in assertions. An assertion with variables represents a scheme (in particular, we use the symbol '_' to represent non-distinguished variables) in ABox [8].

To forget about $A(a)$ from DL-Lite$_{core}$ ABox $\mathcal{A}$, an intuitive way is to remove $A(a)$ and add all $B(a)$ to $\mathcal{A}$ such that B subsumes A. However, as we will show later, it is sufficient to add each $\Phi(a)$, where Φ is explicitly asserted to subsume A.

New membership assertions generated from $A(a)$ form a set $f(A, a) = \{\, \Phi(a) \mid A \sqsubseteq \Phi \text{ in } \mathcal{T} \,\}$, where Φ is a concept description in DL-Lite, and

$$\Phi(a) = \begin{cases} B(a) & \text{if } \Phi = B \text{ is an atomic concept,} \\ P(\_, a) & \text{if } \Phi = \exists P^-, \\ P(a, \_) & \text{if } \Phi = \exists P. \end{cases}$$

The result of forgetting about a concept name A in $\mathcal{A}$, denoted $\mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A)$, is a new ABox obtained by replacing every assertion $A(a) \in \mathcal{A}$ with the corresponding set

$f(A, a)$ of assertions. When the DL-Lite KB $\mathcal{K}$ is clear from the context, we can omit it from $\mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A)$ and write $\mathsf{forget}(\mathcal{A}, A)$ instead.

In a DL-Lite$_{\mathcal{F},\sqcap}$ or DL-Lite$_{\mathcal{R},\sqcap}$ ABox, if we want to forget about an atomic concept A, then axioms of the form $A \sqcap D \sqsubseteq \Phi$ must also be considered, *i.e.*, if $A(a)$ needs to be removed and $\mathcal{K} \models D(a)$, then $\Phi(a)$ must be added into the new knowledge base.

Let $\mathcal{A}$ be an ABox in $\mathcal{L}_{\mathcal{F},\sqcap}$ or $\mathcal{L}_{\mathcal{R},\sqcap}$ and A an atomic concept name. Then, as in the case of DL-Lite$_{core}$, we can define $\Phi(a)$ and thus $f(A, a) = \{\, \Phi(a) \mid A \sqcap D \sqsubseteq \Phi$ in $\mathcal{T}$ and $\mathcal{K} \models D(a) \,\}$, where D is a concept conjunction.

An important observation is that ABox forgetting does not coincides with classical forgetting, *i.e.*, for some model $\mathcal{I}'$ of the result of forgetting $\mathcal{A}'$, there may not exist a model $\mathcal{I}$ of $\mathcal{A}$ such that $\mathcal{I} \sim_A \mathcal{I}'$.

For simplicity, we will only consider consistent knowledge bases in the following theorem.

Theorem 5.1. *Let A be an atomic concept name in a consistent knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ in DL-Lite language $\mathcal{L}$. For any conjunctive query q on the signature $\mathcal{S}_{\mathcal{L}} \setminus \{A\}$, we have*

$$\mathsf{ans}(q, \mathcal{K}) = \mathsf{ans}(q, \langle \mathcal{T}, \mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A) \rangle).$$

Having defined forgetting for both ABoxes and TBoxes, we are now able to define forgetting in a DL-Lite knowledge base.

Definition 5.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite knowledge base and A a concept in DL-Lite language $\mathcal{L}$. The result of forgetting about A in $\mathcal{K}$ is defined to be*

$$\mathsf{forget}(\mathcal{K}, A) = \langle \mathsf{forget}(\mathcal{T}, A), \mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A) \rangle,$$

where $\mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A) = \mathcal{A} \setminus \{A(a) \mid a$ *is a constant in* $\mathcal{L}\} \cup \{\, \Phi(a) \mid A \sqsubseteq \Phi$ *in* $\mathcal{T} \,\}$.

It is important to note that the forgetting operation in ABoxes is with respect to the original knowledge base (especially, the original TBox). However, forgetting in the TBox does not depend on the ABox. The computation process of forgetting about a concept in a DL-Lite knowledge base is shown in Algorithm 3. In this algorithm, D denotes a possibly empty conjunction.

From Proposition 3.1 and Theorem 5.1, it is not hard to see following theorem holds.

Theorem 5.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent knowledge base in DL-Lite language $\mathcal{L}$ and A an atomic concept in $\mathcal{K}$. For any conjunctive query q on $\mathcal{S}_{\mathcal{L}} \setminus \{A\}$, we have*

$$\mathsf{ans}(q, \mathcal{K}) = \mathsf{ans}(q, \mathsf{forget}(\mathcal{K}, A))$$

.

6 Related Work

An alternative approach to partial reuse of ontologies is based on the notion of *conservative extensions* [7], a notion that has been well investigated in classical logic. An ontology $\mathcal{T}$ is a conservative extension of its subontology $\mathcal{T}' \subseteq \mathcal{T}$ w.r.t a signature $\mathcal{S}$

Algorithm 3 (Computing the result of forgetting in an DL-Lite knowledge base)
Input: A knowledge base $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$ in $\mathcal{L}$ and a concept name A.
Output: $\mathsf{forget}(\mathcal{K}, A)$.
Method:
Step 1. For each assertion in $\mathcal{A}$ of the form $A(a)$,
- for every axiom $A \sqcap D \sqsubseteq B$ in $\mathcal{T}$, if $\mathcal{K} \models D(a)$ (true when D is empty), add $B(a)$ to $\mathcal{A}$;
- for every axiom $A \sqcap D \sqsubseteq \exists P$ in $\mathcal{T}$, if $\mathcal{K} \models D(a)$, add $P(a, \_)$ to $\mathcal{A}$;
- for every axiom $A \sqcap D \sqsubseteq \exists P^-$ in $\mathcal{T}$, if $\mathcal{K} \models D(a)$, add $P(\_, a)$ to $\mathcal{A}$;
Step 2. Remove all assertions containing A in $\mathcal{A}$, giving $\mathcal{A}'$. (As shown before, $\mathcal{A}' = \mathsf{forget}_{\mathcal{K}}(\mathcal{A}, A)$)
Step 3. Let $\mathcal{T}'$ be the result of forgetting about A in $\mathcal{T}$, ignoring $\mathcal{A}'$.
Step 4. Return the knowledge base $\mathcal{K}' = \{\mathcal{T}', \mathcal{A}'\}$.

Fig. 3. Forgetting in a DL-Lite knowledge base.

if and only if $\mathcal{T}$ and $\mathcal{T}'$ are equivalent under query answering on $\mathcal{S}$. The theory of conservative extensions can be applied in ontology refinement and ontology merging [7]. Since the goal of partial use of ontology is to obtain a smaller ontology from a larger ontology, a dual theory of conservative extensions, called *modularity of ontology* is also explored in [10, 11]. Unfortunately, given an ontology $\mathcal{T}$, there does not exist a conservative extension in many cases. Moreover, it is exponential to decide if there is a conservative extension/module for a given ontology in common description logics such as $\mathcal{ALC}$. A more recent approach is introduced in [9]. Unlike modularity approach, this approach focuses on ABoxes update. A shortcoming of this approach is that DL-Lite languages are not closed under their ABox update operation. Eiter *et al* [5] have also attempted to define forgetting for OWL ontologies by transforming an OWL ontology into a logic program. However, this approach only works for some OWL ontologies.

7 Conclusion

We have presented the first complete account of forgetting from DL-Lite description logics. This process is a key operation in reusing and combining ontologies, which are based on such description logics. We have presented a semantic definition of forgetting from DL-Lite TBoxes and ABoxes and correct, efficient implementations of the semantics.

Natural next steps are to extend these approaches to other description logic families, to investigate forgetting roles as well as concepts, to find more efficient algorithms for forgetting a set of concepts (or roles) simultaneously, and to implement and apply the methods to practical problems.

Acknowledgments

The authors would like to thank three anonymous referees for their helpful comments. This work was partially supported by the Australia Research Council (ARC) Discovery Projects DP0666107.

References

1. F. Baader, D. Calvanese, D.McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2002.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 602–607, 2005.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, pages 260–270, 2006.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
5. T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, and K. Wang. Forgetting in managing rules and ontologies. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI-06)*, pages 411–419, 2006.
6. T. Eiter and K. Wang. Forgetting and conflict resolving in disjunctive logic programming. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pages 238–243, 2006.
7. S. Ghilardi, C. Lutz, and F. Wolter. Did i damage my ontology? a case for conservative extensions in description logics. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 187–197, 2006.
8. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-06)*, 2006.
9. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the approximation of instance level update and erasure in description logics. In *Proceedings of the 22th National Conference on Artificial Intelligence (AAAI-07)*, pages 403–408, 2007.
10. B. Grau, Y. Kazakov, I. Horrocks, and U. Sattler. A logical framework for modular integration of ontologies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 298–303, 2007.
11. B. Cuenca Grau, Y. Kazakov, I. Horrocks, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proceedings of the 16th International World Wide Web Conference (WWW-07)*, pages 717–726, 2007.
12. J. Lang, P. Liberatore, and P. Marquis. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res. (JAIR)*, 18:391–443, 2003.
13. F. Lin and R. Reiter. Forget it. In *Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159. New Orleans (LA), 1994.
14. K. Wang, A. Sattar, and K. Su. A theory of forgetting in logic programming. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 682–687. AAAI Press, 2005.