Updating DL-Lite Ontologies through First-Order Queries

Giuseppe De Giacomo[1], Xavier Oriol[2],
Riccardo Rosati[1], Domenico Fabio Savo[1]

[1] Sapienza Università di Roma, Rome, Italy
lastname@dis.uniroma1.it
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
xoriol@essi.upc.edu

Abstract. In this paper we study instance-level update in *DL-Lite*$_A$, the description logic underlying the OWL 2 QL standard. In particular we focus on formula-based approaches to ABox insertion and deletion. We show that *DL-Lite*$_A$, which is well-known for enjoying first-order rewritability of query answering, enjoys a first-order rewritability property also for updates. That is, every update can be reformulated into a set of insertion and deletion instructions computable through a non-recursive datalog program. Such a program is readily translatable into a first-order query over the ABox considered as a database, and hence into SQL. By exploiting this result, we implement an update component for *DL-Lite*$_A$-based systems and perform some experiments showing that the approach works in practice.

1 Introduction

In this paper we study effective techniques to perform updates over *DL-Lite* ontologies. In particular, we focus on *DL-Lite*$_A$, which is the most expressive member of the *DL-Lite* family of Description Logics (DLs) [4,5]. *DL-Lite*$_A$ includes virtually all constructs of the OWL 2 QL profile of the W3C OWL 2 standard. In addition, it includes the most typical cardinality restrictions on the participation in roles of UML class diagrams, i.e., any combination of mandatory participation and functional participation.

The crucial characteristic of *DL-Lite*$_A$ ontologies is that they enable the so-called *ontology-based data access* by virtue of first-order rewritability of query answering, that is, every (union of) conjunctive query over a *DL-Lite*$_A$ ontology can be rewritten into a first-order query to be evaluated over the ABox only (i.e., the individual data) considered as a database. This property, on the one hand, gives us a very low worst-case computational complexity bound w.r.t. data, namely AC^0 data complexity. On the other hand, it gives us a very effective practical technique to deal with ontologies that include very large ABoxes (i.e., a lot of individual data): perform the rewriting; transform the first-order query into SQL, or SPARQL, depending on how data are stored; and perform the

resulting query exploiting a data management engine to take advantage of all optimizations available for these standard languages.

When we come to updates over ontologies, several approaches are available in the literature [10,20,7,17]. In particular, in this paper we are interested in the so-called *instance-level* update: we add and delete (or erase) facts about individuals only. Namely, we change the ABox, while we keep the TBox unchanged. This is the most common form of update in practice, since it is essentially concerned with keeping the intensional part of the ontology fixed, while changing freely the individual data (indeed, the ABox changes are typically frequent whereas the TBox typically evolves slowly). Even in this specific kind of updates, there are sophisticated semantic issues to consider in general. One crucial issue is that, in practice, we need the result of the update to be still in the same language as the original ontology, in order to keep using the same system [20]. The most promising approaches that enjoy this property are the so-called *formula-based* approaches [9,13,14,23], in which the update is seen as a change of the ontology axioms. Again, several forms of *formula-based* instance-level updates have been considered [22,6,18,19]. Interestingly, however, for the DLs in the *DL-Lite* family, virtually all proposals in the literature reduce to two main approaches: the one in which we simply act on the ABox assertions explicitly stated in the ontology, and another one in which we act also on the ABox assertions that are not present but logically entailed through the use of the TBox. Notice that, while the first approach is syntax-dependent (i.e., updating logically equivalent ontologies that are stated through different assertions may give rise to logically different resulting ABoxes), the second one is not. In both cases, the semantics have been clarified, their computational tractability established, and ad-hoc algorithms are available. Though, for both approaches, there are essentially no implemented tools yet.

In this paper we look again at the problem of instance-level formula-based update in *DL-Lite*$_A$, and we establish a result that may turn out to be crucial to generate efficient implementations: like query answering, updating an ontology is *first-order rewritable*. That is, given an update specification, we can rewrite it into a set of addition and deletion instructions over the ABox which can be characterized as the result of a first-order query. This means that (i) updating a *DL-Lite*$_A$ ontology is AC^0 in data complexity, and, (ii) updates can be processed by widely used data management engines, e.g., based on SQL or SPARQL. We proof this by showing that every update can be reformulated into a datalog program that generates the set of insertion and deletion instructions to change the ABox while preserving its consistency w.r.t. the TBox. Since the obtained datalog program is non-recursive, it can be further translated as first-order queries over the ABox considered as a database. Exploiting this result, we implement an update component for *DL-Lite*$_A$-based systems and perform some experiments over (a *DL-Lite*$_A$ version of) the LUBM ontology [15] with increasing ABox sizes, showing that the approach works in practice.

As far as we know, this is the first time that the first-order rewritability property for *DL-Lite*$_A$ ontology updating is defined, proved, and empirically evaluated. It is important to mention here that some previous work has been

done in the context of RDF triplestores [2,3], but only for the more restricted case of RDFS (with class disjunctions), which is a proper subset of the expressiveness of *DL-Lite*$_A$, the language we deal with in this paper.

2 Preliminaries

In this section, we first present the notion of Description Logic (DL) ontology, then we provide the definition of the specific DL considered in this work, and finally we summarize some datalog basic concepts and notation.

2.1 Description Logic Ontologies

Let $\mathcal{S}$ be a signature of symbols for individual (object and value) constants, and atomic elements, i.e., concepts, value-domains, attributes, and roles. If $\mathcal{L}$ is a DL, then an $\mathcal{L}$-ontology $\mathcal{O}$ over $\mathcal{S}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, called *TBox*, is a finite set of intensional assertions over $\mathcal{S}$ expressed in $\mathcal{L}$, and $\mathcal{A}$, called *ABox*, is a finite set of instance assertions, i.e., assertions on individuals, over $\mathcal{S}$ expressed in $\mathcal{L}$. Different DLs allow for different kinds of concept, attribute, and role expressions, and different kinds of TBox and ABox assertions over such expressions. In this paper we assume that ABox assertions are always *atomic*, i.e., they correspond to ground atoms, and therefore we omit to refer to $\mathcal{L}$ when we talk about ABox assertions.

The semantics of a DL ontology is given in terms of interpretations. An interpretation is a *model* of an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions in $\mathcal{T} \cup \mathcal{A}$, where the notion of satisfaction depends on the constructs allowed by the specific DL in which $\mathcal{O}$ is expressed. We denote the set of models of $\mathcal{O}$ with $Mod(\mathcal{O})$.

Let $\mathcal{T}$ be a TBox in $\mathcal{L}$, and let $\mathcal{A}$ be an ABox. We say that $\mathcal{A}$ is $\mathcal{T}$-*consistent* if $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, i.e., if $Mod(\langle \mathcal{T}, \mathcal{A} \rangle) \neq \emptyset$, $\mathcal{T}$-inconsistent otherwise. The $\mathcal{T}$-*closure* of $\mathcal{A}$ with respect to $\mathcal{T}$, denoted $\mathsf{cl}_{\mathcal{T}}(\mathcal{A})$, is the set of all atomic ABox assertions that are formed with individuals in $\mathcal{A}$, and are logically implied by $\langle \mathcal{T}, \mathcal{A} \rangle$. Note that if $\langle \mathcal{T}, \mathcal{A} \rangle$ is an $\mathcal{L}$-ontology, then $\langle \mathcal{T}, \mathsf{cl}_{\mathcal{T}}(\mathcal{A}) \rangle$ is an $\mathcal{L}$-ontology as well, and is logically equivalent to $\langle \mathcal{T}, \mathcal{A} \rangle$, i.e., $Mod(\langle \mathcal{T}, \mathcal{A} \rangle) = Mod(\langle \mathcal{T}, \mathsf{cl}_{\mathcal{T}}(\mathcal{A}) \rangle)$. $\mathcal{A}$ is said to be $\mathcal{T}$-*closed* if $\mathsf{cl}_{\mathcal{T}}(\mathcal{A}) = \mathcal{A}$.

2.2 The Description Logic *DL-Lite*$_A$

The *DL-Lite* family [4] is a family of low-complexity DLs particularly suited for dealing with ontologies with very large ABoxes. It constitutes the basis of OWL 2 QL, a tractable profile of OWL 2, the official ontology specification language of the World Wide Web Consortium (W3C)[1].

We now present the DL *DL-Lite*$_A$, which is one of the most expressive logics in the family. *DL-Lite*$_A$ distinguishes concepts from *value-domains*, which denote

[1] http://www.w3.org/TR/2008/WD-owl2-profiles-20081008/

sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. Concepts, roles, attributes, and value-domains in this DL are formed according to the following syntax:

$$\begin{array}{lll} B \longrightarrow A \mid \exists Q \mid \delta(U) & & E \longrightarrow \rho(U) \\ C \longrightarrow B \mid \neg B & & T \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n \\ Q \longrightarrow P \mid P^- & & R \longrightarrow Q \mid \neg Q \\ V \longrightarrow U \mid \neg U & & \end{array}$$

where A, P, and U are symbols in $\mathcal{S}$ denoting respectively an atomic concept name, an atomic role name and an attribute name, $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, $\top_D$ denotes the union of all domain values. Furthermore, P^- denotes the inverse of P, $\exists Q$ denotes the objects related to by the role Q, $\neg$ denotes negation, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values, and $\rho(U)$ denotes the *range* of U, i.e., the set of values related to objects by U.

A *DL-Lite*$_A$ TBox $\mathcal{T}$ contains intensional assertions of the form:

$$\begin{array}{lll} B \sqsubseteq C & (\textit{concept inclusion}) & E \sqsubseteq T \quad (\textit{value-domain inclusion}) \\ Q \sqsubseteq R & (\textit{role inclusion}) & U \sqsubseteq V \quad (\textit{attribute inclusion}) \\ (\mathsf{funct}\ Q) & (\textit{role functionality}) & (\mathsf{funct}\ U) \quad (\textit{attribute functionality}) \end{array}$$

A concept inclusion assertion expresses that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions. Inclusion assertions that do not contain (resp. contain) the symbols '$\neg$' in the right-hand side are called *positive inclusions* (resp. *negative inclusions*). Role and attribute functionality assertions are used to impose that roles and attributes are actually functions respectively from objects to objects and from objects to domain values.

Finally, a *DL-Lite* TBox $\mathcal{T}$ satisfies the following condition: each role (resp., attribute) that occurs (in either direct or inverse direction) in a functional assertion, is not specialized in $\mathcal{T}$, i.e., it does not appear in the right-hand side of assertions of the form $Q \sqsubseteq Q'$ (resp., $U \sqsubseteq U'$).

A *DL-Lite*$_A$ ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$, $P(a, b)$, and $U(a, v)$, where A, P, and U are as above, a and b are object constants in $\mathcal{S}$, and v is a value constant in $\mathcal{S}$.

We refer to [21] for the semantics of a *DL-Lite*$_A$ ontology. Here, we present an example of one such ontology.

Example 1. We consider a slightly modified version of the LUBM ontology [15] about the university domain. We know that a Person can be either a Professor or a Student, where every Student takes (takesCourse role) at least one Course, and every Professor can be either a FullProfessor or an AssociateProfessor. Finally, we know that john is a FullProfessor and that bob is a Student. The corresponding ontology $\mathcal{O}$ is:

$$\mathcal{T} = \{\ \mathsf{Student} \sqsubseteq \mathsf{Person} \qquad \mathsf{Professor} \sqsubseteq \mathsf{Person}$$
$$\mathsf{FullProfessor} \sqsubseteq \mathsf{Professor} \qquad \mathsf{AssociateProfessor} \sqsubseteq \mathsf{Professor}$$
$$\mathsf{Student} \sqsubseteq \neg\mathsf{Professor} \qquad \mathsf{FullProfessor} \sqsubseteq \neg\mathsf{AssociateProfessor}$$
$$\mathsf{Student} \sqsubseteq \exists\mathsf{takesCourse} \qquad \exists\mathsf{takesCourse}^{-} \sqsubseteq \mathsf{Course}\ \}$$
$$\mathcal{A} = \{\ \mathsf{FullProfessor}(\mathsf{john}),\ \mathsf{Student}(\mathsf{bob})\ \}$$

□

A notable characteristic of *DL-Lite*$_A$ is that both satisfiability checking and conjunctive query answering are First-Order (FO) rewritable. Intuitively, FO-rewritability of satisfiability (resp., query answering) captures the property that we can reduce satisfiability checking (resp., query answering) to evaluating a FO query over the ABox $\mathcal{A}$ considered as a relational database. We remark that FO-rewritability of a reasoning problem that involves the ABox of an ontology (such as satisfiability or query answering) is tightly related to low data complexity of the problem. Indeed, since the evaluation of a First-Order Logic query (i.e., an SQL query without aggregation) over an ABox is in AC^0 in data complexity [1], the FO-rewritability of a problem has as the immediate consequence that the problem is in AC^0 in data complexity.

2.3 Datalog Concepts and Notation

A *term* T is either a *variable* or a *constant*. An *atom* is formed by a n-ary *predicate* p together with n terms, i.e., $p(T_1, ..., T_n)$. We may write $p(\overline{T})$ for short. If all the terms $\overline{T}$ of an atom are constants, we call the atom to be *ground*. A *literal* is either an atom $p(\overline{T})$, a negated atom $\neg p(\overline{T})$, or an inequality $T_i \neq T_j$.

A predicate p is said to be *derived* (or *intensional*) if the evaluation of an atom $p(\overline{T})$ depends on some derivation rules, otherwise, it is said to be *base* (or *extensional*). A *derivation rule* is a rule of the form $p(\overline{T_p}) \leftarrow \phi(\overline{T})$, where $p(\overline{T_p})$ is an atom called the *head* of the rule, and $\phi(\overline{T})$ is a conjunction of literals called the *body*. All derivation rules must be *safe*, i.e., every variable appearing in the head or in a negated or inequality literal of the body should also appear in a positive literal of the body. Additionally, all the predicates must be stratified, i.e., it should be possible to partition the set of predicates P into several pairwise disjoint *strata* $P_1 \cup ... \cup P_m$ s.t. for each predicate $p \in P_i$, each predicate appearing in the derivation rules of p should belong to a stratum P_j with $j < i$, if it appears in a negated literal, or, $j \leq i$, if it only appears in positive literals.

Finally, a *datalog program* is a set of derivation rules together with a set of *facts*, where a fact is a ground atom of a non-derived predicate.

3 Formula-Based Approach for Updating DL Ontologies

In the following, we first present the intuitions on ontology update, then we define two distinct formula-based update semantics, and we argue that, for the case of *DL-Lite*$_A$, these two semantics capture virtually all other formula-based update semantics proposed so far. Then, we show that the *careful semantics*, a different formula-based update semantics proposed in the literature, is not

uniquely defined in the case of *DL-Lite*$_A$, contradicting a result stated in [6], which makes this update semantics inappropriate in our approach due to its inherent nondeterminism.

3.1 Update Semantics for *DL-Lite*$_A$

In the formula-based approaches to the update, the objects of change are sets of formulae. That is, the result of the change is explicitly defined in terms of a formula, by resorting to some minimality criterion with respect to the formula expressing the original ontology.

Thus, an *update* is a set $\mathcal{U}$ of operations of two types: insertion operations, denoted by $\mathsf{i}(\alpha)$, and deletion operations denoted by $\mathsf{d}(\alpha)$, where α is an ABox assertion. Intuitively, updating a consistent ontology with an insertion operation $\mathsf{i}(A(o))$, where $A(o)$ is a concept ABox assertion, means changing the extensional level of the ontology in such a way that the ontology resulting from the update is still consistent and entails the fact $A(o)$. Conversely, updating a consistent ontology with a deletion operation $\mathsf{d}(A(o))$, means changing the extensional level of the ontology in such a way that the ontology resulting from the update is still consistent and does not entail the fact $A(o)$.

After adding new facts into an ontology, one may find that the revised ontology becomes inconsistent. A strategy to overcome such a situation is to remove part of the original ABox to the aim of preserving consistency. Similarly, if the goal is to update the ontology by deleting a fact, we might need to retract several facts from the original ABox that entailed it. When applying these modifications to the original ABox, one should respect the *minimal change principle*, a widely accepted principle of the knowledge base evolution literature [8,11,16]. This principle states that the ontology resulting from the update should be as *close* as possible to the original one. In updating an ontology at the instance level following the formula-based approach, the goal becomes the preservation of the facts contained in the original ABox. In what follows we formalize this idea.

Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an update $\mathcal{U}$, and an ABox $\mathcal{A}'$, we say that $\mathcal{A}'$ *accomplishes the update* of $\mathcal{O}$ with $\mathcal{U}$ if it satisfies all the insertions/deletions in $\mathcal{U}$ minimally. To formalize this notion, we first need to introduce the set $\mathcal{A}^+_\mathcal{U}$, which denotes the set of ABox assertions appearing in $\mathcal{U}$ in insertion operations, and the set $\mathcal{A}^-_\mathcal{U}$, which denotes the set of ABox assertions appearing in $\mathcal{U}$ in deletion operations.

Definition 1. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, $\mathcal{U}$ an update, and $\mathcal{A}'$ be an ABox.* $\mathcal{A}'$ accomplishes the update *of $\mathcal{O}$ with $\mathcal{U}$ if $\mathcal{A}' = \mathcal{A}'' \cup \mathcal{A}^+_\mathcal{U}$ for some maximal subset $\mathcal{A}''$ of $\mathcal{A}$ s.t. $\mathcal{A}'' \cup \mathcal{A}^+_\mathcal{U}$ is $\mathcal{T}$-consistent and $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$.*

It easy to see that, by definition, if such ABox $\mathcal{A}'$ exists, it also satisfies $\langle \mathcal{T}, \mathcal{A}' \rangle \models \alpha$ for each $\alpha \in \mathcal{A}^+_\mathcal{U}$ since $\mathcal{A}^+_\mathcal{U} \subseteq \mathcal{A}'$. In order to ensure its existence, note that $\mathcal{U}$ has to respect both of the following conditions:

i) $Mod(\langle \mathcal{T}, \mathcal{A}^+_\mathcal{U} \rangle) \neq \emptyset$, which means that the set of facts we are adding is consistent with the TBox of the ontology.

ii) $\mathcal{A}_{\mathcal{U}}^{-} \cap \mathsf{cl}_{\mathcal{T}}(\mathcal{A}_{\mathcal{U}}^{+}) = \emptyset$, which means that the update is not asking for deleting and inserting the same knowledge at the same time.

Given a TBox $\mathcal{T}$ and an update $\mathcal{U}$, we say that $\mathcal{U}$ is *coherent* with $\mathcal{T}$ if $\mathcal{U}$ respects both the above conditions with respect to a TBox $\mathcal{T}$.

Given a consistent ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and an update $\mathcal{U}$ coherent with $\mathcal{T}$, there might be more than one ABox accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$. This fact leads to different update semantics, each one addressing this issue by means of a different criterium, like the *Cross Product Approach* [9], the *When In Doubt Throw It Out* principle [14,23,18,19], allowing the user to choose the update [22], or even nondeterminism [6]. Fortunately, when the TBox of the ontology is expressed in *DL-Lite*$_A$, the ABox accomplishing the update is uniquely defined [6]. Hence, the application of all the above approaches leads to the same result, which can be defined as follows:

Definition 2. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and $\mathcal{U}$ be an update coherent with $\mathcal{T}$. The result of updating $\mathcal{O}$ with $\mathcal{U}$, denoted by $\mathcal{O} \circ \mathcal{U}$, is the ontology $\langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{A}'$ is the ABox accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$.*

When dealing with ontology updating, there is a fundamental philosophical aspect that has to be considered: one has to decide if the formulae explicitly given in our ontology provide a justification for our knowledge (foundational semantics) or if they are just used as a finite representation of our knowledge (coherence semantics) [11,12]. Depending on this point of view, one may or may not need to preserve a fact that is entailed in the ontology despite not being explicitly asserted. The choice depends on the particular application and personal preferences (we refer to [12] for more details).

Clearly, the update semantics given in Definition 2 embraces the foundational theory. Depending on the specific scenario, and the particular application at hand, this semantics might be considered inappropriate. This motivates the definition of the following update semantics [6,18] for *DL-Lite*$_A$ ontologies based on the coherence theory, in which the objects of the update is not the original ABox, but its deductive closure with respect to the TBox.

Definition 3. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and let $\mathcal{U}$ be an update coherent with $\mathcal{T}$. The result of updating $\mathcal{O}$ with $\mathcal{U}$ according to the coherence semantics, denoted by $\mathcal{O} \bullet \mathcal{U}$, is the ontology $\langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{A}'$ is the ABox accomplishing the update of $\langle \mathcal{T}, \mathsf{cl}_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$.*

3.2 Careful Semantics in *DL-Lite*$_A$

An alternative formula-based update semantics based on the coherence theory is the *Careful semantics* [6] which was proposed with the aim of preventing *unexpected information*. Formally, an ontology updated according to the careful semantics should not entail a role constraint ϕ (i.e., a rule of the form $\exists x(R(o, x)) \wedge (x \neq c_1) \wedge \cdots \wedge (x \neq c_n)$), unless ϕ is entailed by the original

ABox, or the update itself. In practice, the careful update semantics encompasses deleting more ABox assertions so that the final ontology does not entail any new role constraint ϕ. However, although the careful update semantics was thought to be uniquely defined [6, Theorem 16], it can bring to several solutions as we show in the following example.

Example 2. Consider the *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where:

$$\begin{aligned}
\mathcal{T} = \{\ & A \sqsubseteq \exists R_A, \quad R_A \sqsubseteq R, \quad \exists R_A^- \sqsubseteq \neg \exists R_B^-, \\
& B \sqsubseteq \exists R_B, \quad R_B \sqsubseteq R, \quad \exists R_A^- \sqsubseteq \neg \exists R_C^-, \\
& C \sqsubseteq \exists R_C, \quad R_C \sqsubseteq R, \quad \exists R_B^- \sqsubseteq \neg \exists R_C^-, \\
& D \sqsubseteq \exists R_D, \quad R_D \sqsubseteq R, \quad \exists R_C^- \sqsubseteq \neg \exists R_D^- \} \\
\mathcal{A} = \{\ & A(o), B(o)\ \}
\end{aligned}$$

and the update $\mathcal{U} = \{\mathrm{i}(C(o)), \mathrm{i}(D(o))\}$. It is easy to see that the ABox $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_{\mathcal{U}}^+$ is $\mathcal{T}$-consistent and that it accomplishes the update of $\mathcal{O}$ with $\mathcal{U}$. Moreover, $\langle \mathcal{T}, \mathcal{A}' \rangle \models \varphi$, where $\varphi = \exists x(R(o,x)) \wedge (x \neq c_1 \wedge (x \neq c_2)))$ (since the negative inclusions in $\mathcal{T}$ imply that in every model $\mathcal{I}$ of $\langle \mathcal{T}, \mathcal{A}' \rangle$ there are three distinct individuals d_a, d_b, d_c such that $\langle o, d_a \rangle \in R_A^{\mathcal{I}}$, $\langle o, d_b \rangle \in R_B^{\mathcal{I}}$, $\langle o, d_c \rangle \in R_C^{\mathcal{I}}$). However, since neither $\langle \mathcal{T}, \mathcal{A} \rangle \models \varphi$ nor $\langle \mathcal{T}, \mathcal{A}_{\mathcal{U}}^+ \rangle \models \varphi$, we have that $\mathcal{A}'$ does not accomplish the update of $\mathcal{O}$ with $\mathcal{U}$ carefully. Conversely, both the ABoxes $\{A(o)\} \cup \mathcal{A}_{\mathcal{U}}^+$ and $\{B(o)\} \cup \mathcal{A}_{\mathcal{U}}^+$ accomplish the update of $\mathcal{O}$ with $\mathcal{U}$ carefully. This is because the only role-constraining formula $\exists x(R(o,x)) \wedge (x \neq c_1))$ that both entail with $\mathcal{T}$, is also entailed by $\langle \mathcal{T}, \mathcal{A}_{\mathcal{U}}^+ \rangle$. Hence, we have more than one ABox that accomplishes the update of $\mathcal{O}$ with $\mathcal{U}$ carefully. □

4 Foundational-Semantic Updates through Datalog

Now, our intention is, given a *DL-Lite*$_A$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, and some update $\mathcal{U}$, to define a datalog program $\mathcal{D}$ that permits querying whether $\mathcal{U}$ is coherent with $\mathcal{T}$ and, in such a case, allows for generating a set of insertion/deletion instructions that should be applied to $\mathcal{A}$ to accomplish $\mathcal{U}$ according to Definition 2 (foundational-semantic updates).

For ease of presentation, from now on we assume that the TBox $\mathcal{T}$ does not contain inclusions involving attributes and value-domains. However, all the results presented in the next two sections can be easily extended to TBoxes containing such kinds of axioms.

Formally, the datalog program $\mathcal{D}$ contains a derived predicate *incoherent_update*, together with a pair of derived predicates *ins_a/del_a* for each concept/role A such that:

- *incoherent_update()* is true iff $\mathcal{U}$ is not coherent with $\mathcal{T}$.

and, in case *incoherent_update()* is false,

- *ins_a*$(\overline{o})$ is true iff the assertion $A(\overline{o})$ was not in $\mathcal{A}$, but $A(\overline{o}) \in \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$. That is, *ins_a* captures the assertions of $\mathcal{A}$ that should be inserted into $\mathcal{A}$ to accomplish the (foundational-semantic) update $\mathcal{U}$.

– *del_a(ō)* is true iff the assertion $A(\bar{o})$ was in $\mathcal{A}$, but $A(\bar{o}) \notin \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$. That is, *del_a* captures the assertions of $\mathcal{A}$ that should be deleted from $\mathcal{A}$ to accomplish the (foundational-semantic) update $\mathcal{U}$.

Briefly, the main idea of the translation is to map each ABox assertion in $\mathcal{A}$, and each operation in $\mathcal{U}$ into different datalog facts. Then, we map each assertion in the closure of $\mathcal{T}$ into several datalog derivation rules that define the *incoherent_update*, *ins_a(*$\overline{X}$*)*, *del_a(*$\overline{X}$*)* predicates. In the following, we formally describe how to obtain such a datalog program $\mathcal{D}$. Then, we prove that the set of instructions generated in $\mathcal{D}$ are sound and complete to obtain $\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$.

4.1 Translation Rules

Translation of $\mathcal{A}$ and $\mathcal{U}$ All the assertions in $\mathcal{A}$ and operations in $\mathcal{U}$ are translated as different facts in $\mathcal{D}$. In particular:

Each assertion $A(\bar{o}) \in \mathcal{A}$ is translated as the fact *a(ō)*.
Each operation $\mathsf{i}(A(\bar{o})) \in \mathcal{U}$ is translated as the fact *ins_a_request(ō)*.
Each operation $\mathsf{d}(A(\bar{o})) \in \mathcal{U}$ is translated as the fact *del_a_request(ō)*.

Intuitively, *ins_a_request(ō)*/*del_a_request(ō)* means that the ontology has received the request to insert/delete the ABox assertion $A(\bar{o})$. Since according to the Definition 2 all the insertions/deletions requested should be applied, we define the datalog rules:

```
ins_a(X) :- ins_a_request(X), not a(X).
del_a(X) :- del_a_request(X), a(X).
incoherent_update() :- ins_a_request(X), del_a_request(X).
```

for each atomic concept A. Note that *incoherent_update* becomes true in case we request for the insertion and deletion of the same axiom. Similarly, we define the rules *ins_p(X, Y)*/*del_p(X,Y)* for each atomic role P.

Translation of $cl(\mathcal{T})$ We translate positive and negative/functional axioms in the closure of $\mathcal{T}$ differently. In particular, for each positive inclusion axiom $B \sqsubseteq A$ in the closure of $\mathcal{T}$, where A is an atomic concept, we define the rules:

```
del_b(X) :- b(X), del_a_request(X).
incoherent_update() :- ins_b_request(X), del_a_request(X).
```

Intuitively, when we request for deleting $A(o)$, we have to delete any other ABox assertion $B(o)$ that entails $A(o)$. Note that it cannot be accomplished if there is a request for inserting $B(o)$, so, this case makes *incoherent_update* true. We define similar rules when the left-hand side of the axiom is of the form $\exists P$, and also for role inclusion axioms.

Note that we translate the closure of $\mathcal{T}$, instead of $\mathcal{T}$ itself, to be able to capture deletions that are propagated along the concept/role hierarchy. E.g. if in our example we have $\mathcal{U} = \mathsf{d}(\mathsf{Person}(\mathsf{john}))$, the translated datalog program $\mathcal{D}$ generates the deletion of $\mathsf{FullProfessor}(\mathsf{john})$ because of the translation of the assertion $\mathsf{FullProfessor} \sqsubseteq \mathsf{Person}$ appearing in $cl(\mathcal{T})$:

```
del_fullprof(X) :- fullprof(X), del_person_request(X).
```

Differently, for each negative inclusion axiom $B \sqsubseteq \neg A$ in $cl(\mathcal{T})$, we define the rules:

```
del_b(X) :- b(X), ins_a_request(X).
del_a(X) :- ins_b_request(X), a(X).
incoherent_update() :- ins_a_request(X), ins_b_request(X).
```

Intuitively, if we insert $A(o)$ when we have $B(o)$ in the ABox, we have to delete $B(o)$. In the case where the requested update tries to insert both things, we reach a contradiction and thus, *incoherent_update* becomes true. We define similar rules for role negative inclusions, negative inclusions involving the $\exists$ constructor, and functional axioms. In this last case, we require using the inequality built-in predicate to check whether the requested role assertion insertion is going to violate the functional axiom. E.g., given a functional axiom defined over R, we define:

```
del_r(X,Y) :- r(X,Y), ins_r_request(X,Z), Y<>Z.
incoherent_update() :- ins_r_request(X,Y),ins_r_request(X,Z),
    Y<>Z.
```

Again, note that since we translate the closure of $\mathcal{T}$, the rules are able to capture deletions due to inconsistencies generated by propagation. E.g. if in our previous example we have the update $\mathcal{U} = \mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{bob}))$, $\mathcal{D}$ generates the deletion of $\mathsf{Student}(\mathsf{bob})$ because of the first rule obtained when translating the assertion $\mathsf{Student} \sqsubseteq \neg\mathsf{AssociateProfessor}$ appearing in $cl(\mathcal{T})$:

```
del_student(X) :- student(X), ins_assocprof_request(X).
del_assocprof(X) :- assocprof(X), ins_student_request(X).
```

4.2 Datalog Program Soundness and Completeness

The update generated by the datalog program $\mathcal{D}$ is sound in the sense that, for every axiom $A(\overline{o})$ that should be inserted/deleted according to $\mathcal{D}$, $A(\overline{o})$ should be truly inserted/deleted according to the foundational-semantic update. Formally:

Theorem 1. *Given a consistent ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, and an update $\mathcal{U}$, the datalog program $\mathcal{D}$ obtained through the translation defined in Section 4.1, satisfies that: if* incoherent_update() *is true in $\mathcal{D}$, $\mathcal{U}$ is incoherent with $\mathcal{T}$, otherwise, for each concept/role A, if* ins_a($\overline{o}$) *is true in $\mathcal{D}$, then, $A(\overline{o}) \in \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U} \setminus \mathcal{A}$, and if* del_a($\overline{o}$) *is true in $\mathcal{D}$, then, $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$.*

Proof. (Sketch) If *incoherent_update()* is true, it can only be because of a rule generated when translating the update $\mathcal{U}$, the positive axioms of $cl(\mathcal{T})$, or the negative/functional axioms of $cl(\mathcal{T})$. The rules generated in the first two cases are true only if $\mathcal{A}^-_{\mathcal{U}} \cap \mathcal{A}^+_{\mathcal{U}} \neq \emptyset$ and $\mathcal{A}^-_{\mathcal{U}} \cap cl_{\mathcal{T}}(\mathcal{A}^+_{\mathcal{U}}) \neq \emptyset$, respectively. The rules of the third case are true only if $Mod(\langle \mathcal{T}, \mathcal{A}^+_{\mathcal{U}} \rangle) = \emptyset$. Thus, if *incoherent_update()* is true, $\mathcal{U}$ is incoherent with $\mathcal{T}$.

If *ins_a($\overline{o}$)* is true, it is because of a rule generated when translating $\mathcal{U}$, which can only be true if $A(\overline{o}) \notin \mathcal{A}$, and $A(\overline{o}) \in \mathcal{A}^+_{\mathcal{U}}$, thus $A(\overline{o}) \in \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U} \setminus \mathcal{A}$.

If *del_a($\overline{o}$)* is true, it can only be because of (1) a rule generated when translating $\mathcal{U}$, where in such case we have $A(\overline{o}) \in \mathcal{A}$, and $A(\overline{o}) \in \mathcal{A}^-_{\mathcal{U}}$, thus $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$; or (2) a rule generated when translating a positive axiom in $\mathcal{T}$, where in such case we have that $A(\overline{o}) \in \mathcal{A}$ and that for some $B(\overline{o}) \in \mathcal{A}^-_{\mathcal{U}}$, $A(\overline{o}) \models_{\mathcal{T}} B(\overline{o})$, thus, $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$; or (3) a rule generated when translating a negative/functional axiom in $cl(\mathcal{T})$ where in such case we have $A(\overline{o}) \in \mathcal{A}$ and $Mod(\langle \mathcal{T}, \mathcal{A}^+_{\mathcal{U}} \cup \{A(\overline{o})\} \rangle) = \emptyset$, and thus, $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$. □

Conversely, $\mathcal{D}$ is also complete in the sense that any axiom insertion/deletion of $A(\overline{o})$ that should be applied according to the foundational-semantic update is also generated in $\mathcal{D}$. Formally:

Theorem 2. *Given a consistent ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, and an update $\mathcal{U}$, the datalog program $\mathcal{D}$ obtained through the translation defined in Section 4.1, satisfies that: if $\mathcal{U}$ is incoherent with $\mathcal{T}$, then,* incoherent_update() *is true in $\mathcal{D}$, otherwise, for each concept/role $\mathcal{A}$, if $A(\overline{o}) \in \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U} \setminus \mathcal{A}$, then,* ins_a($\overline{o}$) *is true in $\mathcal{D}$, and if $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$, then,* del_a($\overline{o}$) *is true in $\mathcal{D}$.*

Proof. (Sketch) First, if $\mathcal{U}$ is incoherent with $\mathcal{T}$, it is immediate to verify that then, *incoherent_update()* is true in $\mathcal{D}$. So, from now on we assume that $\mathcal{U}$ is coherent with $\mathcal{T}$. Moreover, since $\mathcal{U}$ is coherent with $\mathcal{T}$, $\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U} \setminus \mathcal{A} = \mathcal{A}^+_{\mathcal{U}} \setminus \mathcal{A}$, and by definition of $\mathcal{D}$, it easily follows that, for each concept/role A, if $A(\overline{o}) \in \mathcal{A}^+_{\mathcal{U}} \setminus \mathcal{A}$, *ins_a($\overline{o}$)* is true in $\mathcal{D}$. Finally, we prove that for every assertion deleted from $\mathcal{A}$ there is a corresponding deletion instruction in $\mathcal{D}$. To this aim, we define the following algorithm:

Algorithm ComputeDeletedAssertions($\mathcal{T}, \mathcal{A}, \mathcal{U}$)
Input: *DL-Lite*$_A$ TBox $\mathcal{T}$, ABox $\mathcal{A}$, update $\mathcal{U}$ coherent with $\mathcal{T}$
Output: ABox $\mathcal{A}_d = \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$
begin
  $\mathcal{A}_d = \emptyset$;
  for each $C(a) \in \mathcal{A}^+_{\mathcal{U}}$ **do begin**
    for each $D(a) \in \mathcal{A}$ such that $\mathcal{T} \models C \sqsubseteq \neg D$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{D(a)\}$;
    for each $R(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models C \sqsubseteq \neg \exists R$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{R(a, x)\}$;
    for each $R(x, a) \in \mathcal{A}$ such that $\mathcal{T} \models C \sqsubseteq \neg \exists R^-$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{R(x, a)\}$
  end;
  for each $R(a, b) \in \mathcal{A}^+_{\mathcal{U}}$ **do begin**
    for each $S(a, b) \in \mathcal{A}$ such that $\mathcal{T} \models R \sqsubseteq \neg S$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(a, b)\}$;
    for each $S(b, a) \in \mathcal{A}$ such that $\mathcal{T} \models R \sqsubseteq \neg S^-$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(b, a)\}$;
    for each $C(a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg C$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{C(a)\}$;
    for each $C(b) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^- \sqsubseteq \neg C$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{C(b)\}$;
    for each $S(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg \exists S$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(a, x)\}$;
    for each $S(x, a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg \exists S^-$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(x, a)\}$;
    for each $S(b, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^- \sqsubseteq \neg \exists S$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(b, x)\}$;
    for each $S(x, b) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^- \sqsubseteq \neg \exists S^-$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(x, b)\}$
  end;
  for each $C(a) \in \mathcal{A}^-_{\mathcal{U}}$ **do begin**
    for each $D(a) \in \mathcal{A}$ such that $\mathcal{T} \models D \sqsubseteq C$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{D(a)\}$;
    for each $R(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq C$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{R(a, x)\}$;

for each $R(x,a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^- \sqsubseteq C$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{R(x,a)\}$
end;
for each $R(a,b) \in \mathcal{A}_\mathcal{U}^-$ **do begin**
for each $S(a,b) \in \mathcal{A}$ such that $\mathcal{T} \models S \sqsubseteq R$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(a,b)\}$;
for each $S(b,a) \in \mathcal{A}$ such that $\mathcal{T} \models S \sqsubseteq R^-$ do $\mathcal{A}_d = \mathcal{A}_d \cup \{S(b,a)\}$
end;
return $\mathcal{A}_d$
end

It can easily be shown that the ABox returned by such an algorithm is equal to $\mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}$. Moreover, it is easy to see that, for each concept/role $\mathcal{A}$, if $A(\overline{o})$ belongs to the ABox returned by ComputeDeletedAssertions($\mathcal{T}, \mathcal{A}, \mathcal{U}$), then *del_a($\overline{o}$)* is true in $\mathcal{D}$. □

5 Coherent-Semantic Updates through Datalog

The previous datalog program $\mathcal{D}$ generates the set of insertions/deletions that should be applied to an ABox $\mathcal{A}$ to accomplish an update $\mathcal{U}$ according to the foundational-semantics. Now, our purpose is to modify this datalog program to deal with the coherent-semantics as described in Definition 3.

Briefly, to accomplish the coherent-semantics, we need to generate more insertion instructions in $\mathcal{D}$. This is because in the coherent-semantics we need to keep the updated ABox as *close* as possible to the $\mathcal{T}$*-closure* of the original ABox, instead of the ABox itself. For instance, if in our previous example we apply the update $\mathcal{U} = \{\mathsf{d}(\mathsf{Student}(\mathsf{bob}))\}$ with coherent-semantics, besides deleting the assertion $\mathsf{Student}(\mathsf{bob})$, we also need to apply the insertion $\mathsf{Person}(\mathsf{bob})$ since $\mathsf{Person}(\mathsf{bob})$ appears in $cl_\mathcal{T}(\mathcal{A})$.

Thus, in practice, we only need to extend our datalog program $\mathcal{D}$ to (1) additionally capture those assertions $A(\overline{o})$ entailed by assertions $B(\overline{o})$ that are requested for deletion, and (2) derive their insertion in case they do not get in conflict with the assertions in $\mathcal{A}_\mathcal{U}^+$. Intuitively, we do (1) by considering an additional derived predicate *ins_a_closure* for each concept/role A; then, we use this new predicate to define new derivation rules for *ins_a* in case they do not get in conflict with any axiom in $\mathcal{A}_\mathcal{U}^+$, thus accomplishing (2).

In the following, we first define how we obtain these new derivation rules, and then we prove that the insertion/deletion instructions generated by this extended datalog program $\mathcal{D}$ are sound and complete with respect to the coherent-semantics.

5.1 Translation Rules

Capturing Closure Insertions due to Deletions For each positive inclusion axiom $B \sqsubseteq A$ in the closure of $\mathcal{T}$, where A is an atomic concept, let A_1, ..., A_m be all the atomic concepts having a positive inclusion axiom of the form $A \sqsubseteq A_i$ in the TBox closure of $\mathcal{T}$, then we define the rules:

```
ins_a_closure(X) :- del_b(X), not a(X), not ins_a_request(X),
    not del_a_request(X), not del_a1_request(X), ..., not
    del_am_request(X).
```

For example, for the assertion $\mathsf{FullProfessor} \sqsubseteq \mathsf{Professor}$, we define the rules:

```
ins_prof_closure(X) :- del_fullprof(X), not prof(X), not
    ins_prof_request(X), not del_prof_request(X), not
    del_person_request(X).
```

Intuitively, when we delete a $\mathsf{FullProfessor}(o)$, we might need to insert $\mathsf{Professor}(o)$ because of the closure of the semantics. However, such *closure insertion* is not necessary if $\mathsf{Professor}(o)$ is already in the ABox, or if there is a request for its insertion, or if it is requested for deletion (either $\mathsf{Professor}(o)$ itself or its parent concepts $\mathsf{Person}(o)$). We define similar rules for role positive inclusion axioms and positive inclusion axioms in which the left-hand side uses the $\exists$ constructor.

Defining New Insertions due to Closure Insertions Once we have defined the predicates *ins_a_closure*, we use them for defining new insertions in case they do not get in conflict with the assertions in $\mathcal{A}_{\mathcal{U}}^{+}$. To do so, for each atomic concept A, let $B_1, \ldots, B_n$ be all the concepts having a negative inclusion axiom with A in the TBox closure of $\mathcal{T}$, then we define the rules:

```
ins_a(X) :- ins_a_closure(X), not ins_b1_request(X) ... not
    ins_bn_request(X).
```

Following the previous example, we would define:

```
ins_prof(X):-ins_prof_closure(X), not ins_student_request(X).
```

Intuitively, any derived *closure insertion* of $\mathsf{Professor}(o)$ should be applied only if it does not get in conflict with any negative inclusion axiom. Such a conflict might arise if there is a request to insert some $\mathsf{Student}(o)$ because of the negative inclusion assertion $\mathsf{Student} \sqsubseteq \neg\mathsf{Professor}$. Similarly, we define the rules for roles.

5.2 Datalog Program Soundness and Completeness

We finally state that the generated insertion/deletions instructions generated by the datalog program $\mathcal{D}$ is sound and complete with respect to the coherent-semantics (the proof of the following theorem can be obtained by easily extending the proofs of Theorem 1 and Theorem 2).

Theorem 3. *Given a consistent ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, and an update $\mathcal{U}$, the datalog program $\mathcal{D}$ obtained through the translation defined in Sections 4.1 and 5.1, satisfies that: (i)* incoherent_update() *is true in $\mathcal{D}$ iff $\mathcal{U}$ is incoherent with $\mathcal{T}$; (ii) if $\mathcal{U}$ is coherent with $\mathcal{T}$, then for each concept/role $\mathcal{A}$,* ins_a($\overline{o}$) *is true in $\mathcal{D}$ iff $A(\overline{o}) \in \langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \setminus \mathcal{A}$, and* del_a($\overline{o}$) *is true in $\mathcal{D}$ iff $A(\overline{o}) \in \mathcal{A} \setminus \langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U}$.*

6 Implementation and Experiments

To show the feasibility and scalability of our technique, we have developed a Java program that, given a closed *DL-Lite*$_A$ TBox, builds the datalog program that generates the insertion/deletion instructions for applying a coherent-semantic update. Furthermore, the program translates this datalog into standard SQL queries. Since these queries depend only on the TBox, but not on the ABox nor the requested update, all of them are created in compilation time and stored in the database as SQL views. Thus, on runtime, the user can generate the instructions by means of inserting the operations s/he wants to perform in the *ins_a_request/del_a_request* tables of the database and querying these views.

We have run the experiments using a *DL-Lite*$_A$ approximation of the LUBM benchmark, an ontology describing university concepts (e.g., teachers, departments, etc) with 75 basic concept/roles and 243 assertions. For our purposes, we have removed those axioms not expressible in *DL-Lite*$_A$, and added 20 disjointness/functional assertions to increase the complexity of the updates. Thus, our final ontology consisted of 195 axioms.

Regarding the data, we have created different ABoxes of increasing size (from 10^5 to $3.5 * 10^7$ assertions). To do so, we have modified the UBA Data Generator to create a single university, but with an increasing number of connected departments, teachers, etc. Due to this increasing number of connected objects, the updates became more complex when increasing the data size. Then, we have defined an update request by means of selecting 3 tuples to delete, and 3 tuples to insert. Such tuples were selected in a way to ensure several interactions with the TBox assertions, thus, generating several insertions/deletions.

In Figure 1 we summarize the results we have obtained using the MySQL 5.7 DBMS, running on a Windows 8.1 over an Intel Core i7-4710HQ, with 8GB of RAM [2]. In particular, we show the times to generate the instructions (x points in the first diagram), the time to generate and execute the instructions (+ points in the first diagram), and the number of instructions generated (x points in the second diagram). We also depict the different trend lines in the diagrams.

As it can be seen, our method has generated from 139 insertion/deletion instructions in 12s for the smallest ABox, to 479 instructions in 16s for the largest. Thus, although there is a constant time penalty of about 12s to generate the instructions, the time increment in function of the ABox size is small. Adding this time to the time to execute the instructions, we got a total cost near to 20s. We argue that this low time increment behavior is due to the fact that, in *DL-Lite*$_A$, an update request only causes updates *locally*, i.e., the unique tuples to insert/delete are a subset of those that are *connected* to the requested insertions/deletions. Thus, since ABoxes tends to increase its size by considering more objects, rather than infinitely augmenting the connectivity between them, increasing the ABox size barely increases the generated instructions, as can be seen in the second diagram. Hence, we argue that our approach can be effectively used in practice with large ABoxes.

[2] More experiment details and results at www.essi.upc.edu/~xoriol/dllitea/

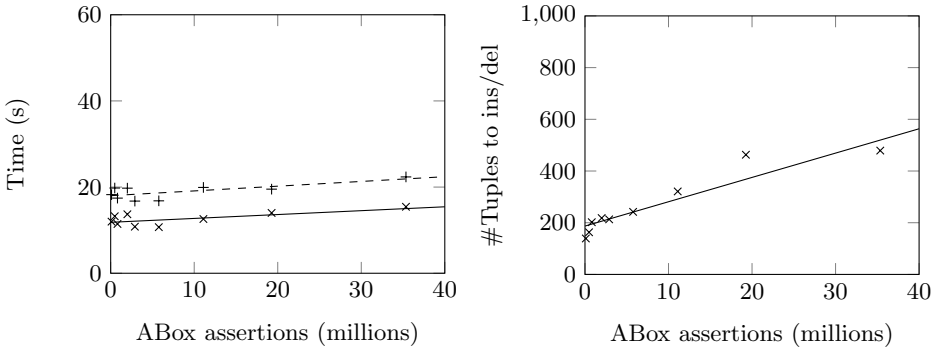

Fig. 1: Experimental Results

7 Conclusions

In this paper we have shown that the *DL-Lite* family, in particular *DL-Lite*$_A$, enjoys the first-order rewritability of instance level updates. Apart from the theoretical interest, this result gives us a practical and effective technique to perform updates over *DL-Lite* ontologies.

Although we have not considered any specific syntax to express the update, what we proposed here is fully compatible with SPARQL update operators studied in [2]. There, the set of insertions and deletions are defined through unions of conjunctive queries over the current ontology. We can immediately extend our approach in the same way, producing update operators that are equivalent to the ones defined in [2] in the case of RDFS, but that deal with the more expressive *DL-Lite*$_A$ and OWL 2 QL languages.

There are several directions for future work, but maybe the most compelling one, encouraged by the practical applicability of our results, is to extend our datalog-based approach blurring the distinction between TBox and ABox assertions, in line with the use of SPARQL over OWL 2 QL ontologies.

Acknowledgments. This research has been partially supported by the EU under FP7 project Optique (grant n. FP7-318338), by the Ministerio de Economía y Competitividad (project TIN2014-52938-C2-2-R), and by the Sapienza project "Immersive Cognitive Environments".

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.
2. A. Ahmeti, D. Calvanese, and A. Polleres. Updating RDFS aboxes and tboxes in SPARQL. In *Proc. of ISWC 2014*, pages 441–456, 2014.
3. A. Ahmeti, D. Calvanese, A. Polleres, and V. Savenkov. Dealing with inconsistencies due to class disjointness in SPARQL update. In *Proc. of DL 2015*, volume 1350 of *CEUR*, 2015.

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. of Automated Reasoning*, 39(3):385–429, 2007.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.
6. D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Evolution of *DL-Lite* knowledge bases. In *Proc. of ISWC 2010*, volume 6496 of *LNCS*, pages 112–128. Springer, 2010.
7. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On instance-level update and erasure in description logic ontologies. *J. of Logic and Computation, Special Issue on Ontology Dynamics*, 19(5):745–770, 2009.
8. T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.
9. R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc. of PODS 1983*, pages 352–365, 1983.
10. G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: Classification and survey. *Knowledge Engineering Review*, 23(2):117–152, 2008.
11. G. Flouris and D. Plexousakis. Handling ontology change: Survey and proposal for a future research direction. Technical report TR-362 FORTH-ICS, Institute of Computer Science, Forth. Greece, 2005.
12. P. Gärdenfors. Propositional logic based on the dynamics of belief. *J. Symb. Log.*, 50(2):390–394, 1985.
13. M. L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30(1):35–79, 1986.
14. M. L. Ginsberg and D. E. Smith. Reasoning about action I: A possible worlds approach. Technical Report KSL-86-65, Knowledge Systems, AI Laboratory, 1987.
15. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. of Web Semantics*, 3(2–3):158–182, 2005.
16. H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. of KR 1991*, pages 387–394, 1991.
17. E. Kharlamov, D. Zheleznyakov, and D. Calvanese. Capturing model-based ontology evolution at the instance level: The case of dl-lite. *J. of Computer and System Sciences*, 79(6):835–872, 2013.
18. M. Lenzerini and D. F. Savo. On the evolution of the instance level of *DL-Lite* knowledge bases. In *Proc. of DL 2011*, volume 745 of *CEUR,* ceur-ws.org, 2011.
19. M. Lenzerini and D. F. Savo. Updating inconsistent Description Logic knowledge bases. In *Proc. of ECAI 2012*, 2012.
20. H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic ABoxes. In *Proc. of KR 2006*, pages 46–56, 2006.
21. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
22. L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *In Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management*, pages 133–140, 2002.
23. M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.