Journal of Artificial Intelligence Research 70 (2021) 1335-1371 Submitted 09/2020; published 04/2021

Instance-Level Update in DL-Lite Ontologies through First-Order Rewriting

Giuseppe De Giacomo DEGIACOMO@DIAG.UNIROMA1.IT
Sapienza Università di Roma, Rome, Italy

Xavier Oriol XORIOL@ESSI.UPC.EDU
Universitat Politècnica de Catalunya, Barcelona, Spain

Riccardo Rosati ROSATI@DIAG.UNIROMA1.IT
Sapienza Università di Roma, Rome, Italy

Domenico Fabio Savo DOMENICOFABIO.SAVO@UNIBG.IT
Università degli Studi di Bergamo, Bergamo, Italy

Abstract

In this paper we study instance-level update in *DL-Lite*$_A$, a well-known description logic that influenced the OWL 2 QL standard. Instance-level update regards insertions and deletions in the ABox of an ontology. In particular, we focus on formula-based approaches to instance-level update. 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 instance-level update. That is, every update can be reformulated into a set of insertion and deletion instructions computable through a non-recursive Datalog program with negation. 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) (Calvanese et al., 2007, 2013). *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* (Poggi et al., 2008). Indeed, *DL-Lite*$_A$ enjoys *first-order rewritability* of query answering, cf. Figure 1. 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 of the query answering problem, 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

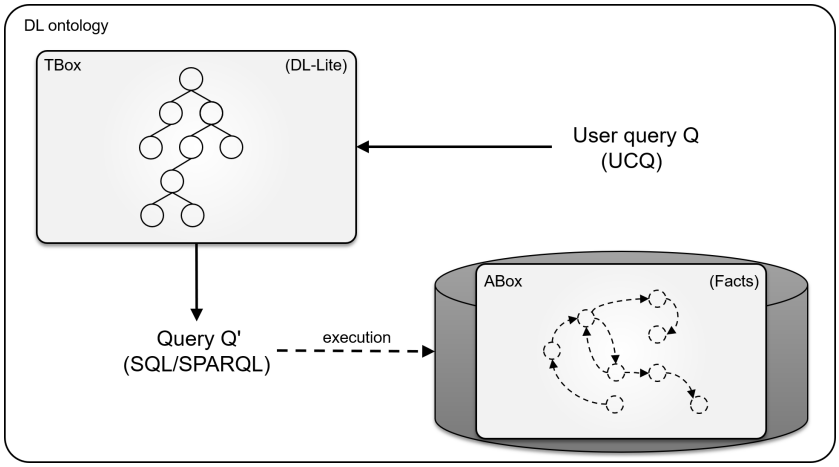

Figure 1: *DL-Lite* Query Rewriting

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 (Flouris et al., 2008; Liu et al., 2006; De Giacomo et al., 2009; Zhuang et al., 2016; Kharlamov et al., 2013; Zheleznyakov et al., 2019). 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 (i.e., the extensional part of the ontology), while we keep the TBox (i.e., the intensional part of the ontology) 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 (Liu et al., 2006). The most well-known approaches that enjoy this property are the so-called *formula-based* approaches (Fagin, Ullman, & Vardi, 1983; Ginsberg, 1986; Ginsberg & Smith, 1987; Winslett, 1990), 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 (Stojanovic, Maedche, Motik, & Stojanovic, 2002; Calvanese, Kharlamov, Nutt, & Zheleznyakov, 2010; Lenzerini & Savo, 2011, 2012). For the DLs in the *DL-Lite* family, virtually all formula-based proposals in the literature reduce to two main approaches: 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, and ad-hoc tractable algorithms are available, although no software tools have been developed yet.

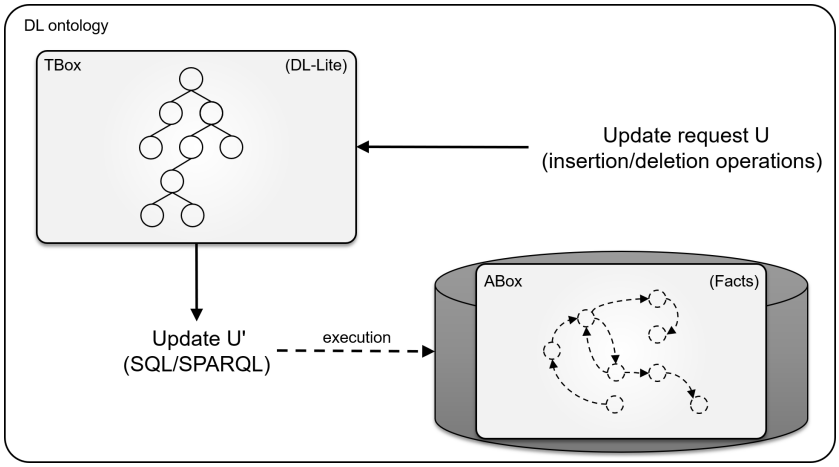

Figure 2: *DL-Lite* Update Rewriting

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. Thus, we can devise a technique based on rewriting for updates, which is analogous to the one for query answering, cf. Figure 2. This means that (*i*) updating a *DL-Lite*$_A$ ontology is in AC^0 in data complexity,[1] and, (*ii*) updates can be processed by widely used data management engines, e.g., based on SQL or SPARQL. We prove this result constructively, by showing that every update can be reformulated into a non-recursive Datalog program with negation that generates the set of insertion and deletion instructions to change the ABox while preserving its consistency w.r.t. the TBox. Such a Datalog program can be further translated into a first-order query over the ABox considered as a database. Exploiting this result, we implement an update tool for *DL-Lite*$_A$-based systems and perform some experiments over (a *DL-Lite*$_A$ version of) the LUBM ontology (Guo, Pan, & Heflin, 2005) with increasing ABox sizes, showing that the approach works in practice. While in the present paper the first-order rewritability property for *DL-Lite*$_A$ ontology updating is defined, proved, and empirically evaluated for the first time, our work is related to research done in the context of RDF triple-stores (Ahmeti et al., 2014, 2015), which focuses on RDFS (with class disjunctions), a proper subset of *DL-Lite*$_A$.

The rest of the paper is organized as follows: after some preliminaries in Section 2, in Section 3 we review formula-based approaches to instance-level ontology updates, in particular in the case of *DL-Lite*$_A$. Then, in Section 4 and 5 we show how to express updates in non-recursive Datalog, and hence show the first-order rewritability of instance-level updates for *DL-Lite*$_A$. In Section 6, we demonstrate the practical applicability of the

1. When we say that update, which is not a decision problem, is in AC^0, we actually refer to the associated recognition problem (Abiteboul, Hull, & Vianu, 1995) of checking whether an ABox assertion belongs to the result of the update.

approach through a series of experiments over the LUBM ontology. Finally, we draw some conclusions in Section 7.[2]

2. Preliminaries

In this section, we first briefly introduce Description Logic (DL) ontologies (Baader et al., 2003), we provide the definition of *DL-Lite*$_A$, the specific DL considered in this work, and finally we summarize some basic concepts and notation for Datalog.

2.1 Description Logic Ontologies

Description Logics allow one to represent the domain of interest in terms of *concepts*, denoting sets of objects, *value-domains*, denoting sets of values, *attributes*, denoting binary relations between objects and values, and *roles*, denoting binary relations over objects. DL expressions are built starting from an alphabet Γ of symbols for atomic concepts, atomic value-domains, atomic attributes, atomic roles, and object and value constants. We denote by Γ_O the set of object constants, and by Γ_V the set of value constants. Complex expressions are constructed starting from atomic elements, and applying suitable constructs.

If $\mathcal{L}$ is a DL, then an $\mathcal{L}$-ontology $\mathcal{O}$ (over Γ) is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, called *TBox*, is a finite set of intensional assertions expressed in $\mathcal{L}$, and $\mathcal{A}$, called *ABox*, is a finite set of instance assertions, also called *facts*, (i.e., assertions on single individuals) 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 First-Order (FO) interpretations, i.e., the interpretations structures of First-Order Logic (cf. (Baader et al., 2003)). We denote with $Mod(\mathcal{O})$ the set of models of $\mathcal{O}$. A FO 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 definition of satisfaction depends on the kind of expressions and assertions in the specific DL language in which $\mathcal{O}$ is specified. As usual, an ontology $\mathcal{O}$ is said to be *satisfiable* if it admits at least one model, i.e., if $Mod(\mathcal{O}) \neq \emptyset$, and $\mathcal{O}$ is said to *entail* a FO sentence ϕ, denoted $\mathcal{O} \models \phi$, if $\phi^{\mathcal{I}} = \texttt{true}$ for all $\mathcal{I} \in Mod(\mathcal{O})$.

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, $\mathcal{T}$-inconsistent otherwise. The $\mathcal{T}$-*closure* of $\mathcal{A}$ with respect to $\mathcal{T}$, denoted $cl_{\mathcal{T}}(\mathcal{A})$, is the set of all atomic ABox assertions that are formed using individuals in $\mathcal{A}$, and are entailed by $\langle \mathcal{T}, \mathcal{A} \rangle$. Given two ABoxes $\mathcal{A}$ and $\mathcal{A}'$, and a TBox $\mathcal{T}$, we say that $\mathcal{A}$ and $\mathcal{A}'$ are logically equivalent with respect to $\mathcal{T}$ if $cl_{\mathcal{T}}(\mathcal{A}) = cl_{\mathcal{T}}(\mathcal{A}')$. Note that if $\langle \mathcal{T}, \mathcal{A} \rangle$ is an $\mathcal{L}$-ontology, then $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ is an $\mathcal{L}$-ontology as well, and is logically equivalent to $\langle \mathcal{T}, \mathcal{A} \rangle$, i.e., $Mod(\langle \mathcal{T}, \mathcal{A} \rangle) = Mod(\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle)$. Similarly, given a TBox $\mathcal{T}$ in $\mathcal{L}$, we denote by $cl(\mathcal{T})$ the deductive closure of $\mathcal{T}$, i.e., the set of all the TBox assertions in $\mathcal{L}$ that follow from $\mathcal{T}$.

2. This paper is an extended version of the conference paper (De Giacomo, Oriol, Rosati, & Savo, 2016). The main differences are the following: (i) we spell out the technique defined for obtaining the Datalog program that computes the actual insertion and deletion instructions for the update; (ii) we add the proofs of theorems; (iii) we illustrate the technique in details, using examples. As a result, the contents of the whole paper have been revised and expanded.

Sometimes, with a little abuse of notation, we write $\alpha \in \langle \mathcal{T}, \mathcal{A} \rangle$ to denote that the assertion α belongs to $\mathcal{T} \cup \mathcal{A}$.

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

The *DL-Lite* family (Calvanese et al., 2007) is a family of low-complexity DLs particularly suited for dealing with ontologies with very large ABoxes. Such DLs constitute the basis of OWL 2 QL, a tractable profile of OWL 2, the official ontology specification language of the World Wide Web Consortium (W3C)[3].

Here we focus on *DL-Lite*$_A$ (Poggi et al., 2008), which is one of the most expressive logics in the family. *DL-Lite*$_A$ distinguishes concepts from *value-domains*, which denote 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 denote 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 (i.e. countably infinite sets of values), and $\top_D$ denotes the union of all domain values. Furthermore, P^- denotes the inverse of P, $\exists Q$ denotes the domain of Q (to denote the range of Q, we use $\exists Q^-$), '$\neg$' 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:

$B \sqsubseteq C$	(*concept inclusion*)	$E \sqsubseteq T$	(*value-domain inclusion*)
$Q \sqsubseteq R$	(*role inclusion*)	$U \sqsubseteq V$	(*attribute inclusion*)
(funct Q)	(*role functionality*)	(funct U)	(*attribute functionality*)

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 the symbol '$\neg$' in the right-hand side are called *positive inclusions*, while those that do are called *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. A TBox $\mathcal{T}$ in *DL-Lite*$_A$ 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 Γ_O, and v is a value constant in Γ_V.

3. https://www.w3.org/TR/owl2-profiles/

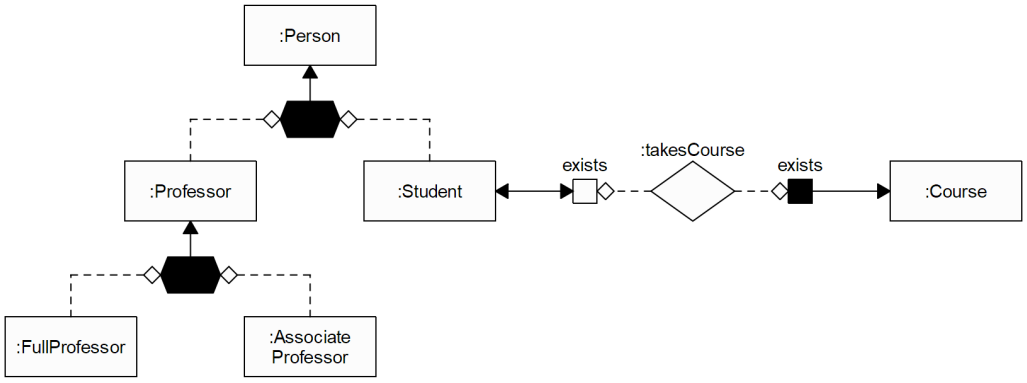

Figure 3: Graphical representation of the running example ontology (in Graphol).

The semantics of a *DL-Lite*$_A$ KB is given in terms of FO interpretations $\mathcal{I} = (\Delta^I, \cdot^I)$ where Δ^I is the interpretation domain and $\cdot^I$ is the interpretation function. In particular, Δ^I is a non-empty set partitioned into $\Delta_V^{\mathcal{I}}$ and Δ_O^I, where Δ_O^I is the subset of Δ^I used to interpret object constants in Γ_O, and $\Delta_V^{\mathcal{I}}$ is the subset of Δ^I used to interpret data values. In other words, for every $c \in \Gamma_O$, $c^I \in \Delta_O^I$ and for every $f \in \Gamma_V$, $f^I \in \Delta_V^{\mathcal{I}}$. The interpretation function $\cdot^I$ is defined as follows.

- For every $c \in \Gamma_O$, $c^I \in \Delta_O^I$, and for every $f \in \Gamma_V$, $f^I \in \Delta_V^{\mathcal{I}}$.
- For all $d_1, d_2 \in \Gamma_O \cup \Gamma_V$, $d_1^{\mathcal{I}} \neq d_2^{\mathcal{I}}$, i.e., interpretations for *DL-Lite*$_A$ follow the unique name assumption (UNA).
- For all $1 \leq i \leq n$, T_i is a countably infinite set of values.
- The following equations are satisfied by $\cdot^I$:

$$
\begin{array}{lcl@{\qquad}lcl}
A^I & \subseteq & \Delta_O^I & P^I & \subseteq & \Delta_O^I \times \Delta_O^I \\
(\delta(U))^I & = & \{\ o \mid \exists v.\, (o,v) \in U^I\ \} & (P^-)^I & = & \{\ (o,o') \mid (o',o) \in P^I\ \} \\
(\exists Q)^I & = & \{\ o \mid \exists o'.\, (o,o') \in Q^I\ \} & (\neg Q)^I & = & (\Delta_O^I \times \Delta_O^I) \setminus Q^I \\
(\neg B)^I & = & \Delta_O^I \setminus B^I & U^I & \subseteq & \Delta_O^I \times \Delta_V^{\mathcal{I}} \\
(\rho(U))^I & = & \{\ v \mid \exists o.\, (o,v) \in U^I\ \} & (\neg U)^I & = & (\Delta_O^I \times \Delta_V^{\mathcal{I}}) \setminus U^I \\
T_i^I & \subseteq & \top_D^I = \Delta_V^{\mathcal{I}}\ (1 \leq i \leq n) & T_i \cap T_j & = & \emptyset \quad (1 \leq i,j \leq n,\ i \neq j)
\end{array}
$$

An interpretation $\mathcal{I}$ satisfies a concept (resp., role) inclusion assertion $B \sqsubseteq C$ (resp., $Q \sqsubseteq R$) if $B^I \subseteq C^I$ (resp., $Q^I \subseteq R^I$), and satisfies a role functionality assertion (funct Q) if, for each $o, o', o'' \in \Delta_O^I$, we have that $(o,o') \in Q^I$ and $(o,o'') \in Q^I$ implies $o' = o''$. The semantics for attribute and value-domain inclusion assertions, and for functionality assertions over attributes can be defined analogously. Finally, $\mathcal{I}$ satisfies the ABox assertions $A(a)$, $P(a,b)$ and $U(a,v)$ if $a^I \in A^I$, $(a^I, b^I) \in P^I$ and $(a^I, v^I) \in U^I$ respectively.

Example 1 We consider a slightly modified version of the LUBM ontology (Guo et al., 2005) about the university domain. We use this ontology as running example throughout the paper.

We state 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 state that john is a FullProfessor and that bob is taking the algebra course. The corresponding ontology $\mathcal{O}$ is:

$$
\begin{aligned}
\mathcal{T} = \{ \quad & \mathsf{Student} \sqsubseteq \mathsf{Person} && \mathsf{Professor} \sqsubseteq \mathsf{Person} \\
& \mathsf{FullProfessor} \sqsubseteq \mathsf{Professor} && \mathsf{AssociateProfessor} \sqsubseteq \mathsf{Professor} \\
& \mathsf{Student} \sqsubseteq \neg \mathsf{Professor} && \mathsf{FullProfessor} \sqsubseteq \neg \mathsf{AssociateProfessor} \\
& \mathsf{Student} \sqsubseteq \exists \mathsf{takesCourse} && \exists \mathsf{takesCourse}^{-} \sqsubseteq \mathsf{Course} \\
& \exists \mathsf{takesCourse} \sqsubseteq \mathsf{Student} && \}
\end{aligned}
$$

$$
\mathcal{A} = \{ \quad \mathsf{FullProfessor}(\mathsf{john}),\ \mathsf{takesCourse}(\mathsf{bob}, \mathsf{algebra}) \ \}
$$

Note that the ontology described above is in *DL-Lite*$_A$, and that we have enriched the LUBM ontology with some concept disjointnesses, since this kind of assertions is missing in the original version. Figure 3 depicts the TBox graphically, by using the Graphol ontology visual language (Console et al., 2014). □

DL-Lite$_A$ ontologies support a very effective way to perform reasoning and query answering. Indeed, a notable characteristic of *DL-Lite*$_A$ is that both satisfiability checking and query answering for union of conjunctive queries are first-order rewritable (FO-rewritable), i.e., rewritable in FOL. Intuitively, FO-rewritability of query answering of union of conjunctive queries, as well as of satisfiability, captures the property that we can reduce both query answering and satisfiability checking 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 FO query (i.e., an SQL query without aggregation) over an ABox is in AC^0 in data complexity (Abiteboul et al., 1995), the FO-rewritability of a problem has as the immediate consequence that the problem is in AC^0 in data complexity.

2.3 Non-Recursive Datalog and FO Queries

We close this preliminary section by briefly summarizing the basic notions regarding syntax and semantics of non-recursive Datalog (with negation).

A *term* τ is either a *variable* or a *constant*. An *atom* is formed by a n-ary *predicate* p together with n terms, i.e., $p(\tau_1, ..., \tau_n)$. We may write $p(\overline{\tau})$ for short. If all the terms $\overline{\tau}$ of an atom are constants, we say that the atom is *ground*. A *literal* is either an atom $p(\overline{\tau})$, a negated atom *not* $p(\overline{\tau})$, or an inequality $\tau_i \neq \tau_j$.

A predicate p is said to be *derived* (or *intensional*) if the evaluation of an atom $p(\overline{\tau})$ depends on some derivation rules, otherwise, it is said to be *base* (or *extensional*). A *derivation rule* is a rule of the form $p(\overline{\tau_p}) \leftarrow \phi(\overline{\tau})$, where $p(\overline{\tau_p})$ is an atom called the *head* of the rule, and $\phi(\overline{\tau})$ is a conjunction of literals called the *body*. Given several derivation rules with predicate p in its head, $p(\overline{\tau})$ is evaluated to true if and only if one of the bodies of such derivation rules is evaluated to true.

Example 2 We consider some Datalog derivation rules, regarding the university domain, to exemplify the previous notions:

```
EmptyCourse(X) :- Course(X), not CourseWithStudents(X).
CourseWithStudents(X) :- Student(Y), takesCourse(Y, X).
```

Course, *Student* and *takesCourse* are base predicates whereas *EmptyCourse* and *CourseWithStudents* are derived predicates. Intuitively, X is a *CourseWithStudents* if some *Student* Y takes X. On the contrary, X is an *EmptyCourse* if X is a *Course* without students.

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. Following our previous example, the derivation rule of *EmptyCourse* is safe because the variable X, appearing both in the head of the rule and also inside a negated literal, appears also in the positive literal *Course*. Similarly, *CourseWithStudents* is also safe.

In order to remain in the realm of FOL, we require all the predicates to be non-recursive, 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 $j < i$. Following our example, the predicates are non-recursive since we can define the stratification: $P_1 = \{Course, Student, takesCourse\}$, $P_2 = \{CourseWithStudents\}$, $P_3 = \{EmptyCourse\}$.

In Datalog, queries are defined by means of derived predicates. Non-recursive Datalog queries, as the ones we consider in this paper, are known to be FO queries (i.e., equivalent to relational algebra, and thus, expressible in SQL or SPARQL).

Example 3 The *CourseWithStudents* predicate can be written as the following SQL query:

```
SELECT tC.course
FROM Student AS S
    JOIN takesCourse AS tC ON (tC.student = S.student)
```

The *EmptyCourse* predicate can be written as an SQL query using the previous one as a subquery:

```
SELECT C.course
FROM Course AS C
WHERE NOT EXISTS (
    SELECT tC.course
    FROM Student AS S
    JOIN takesCourse AS tC ON (tC.student = S.student)
    WHERE tC.course = C.course
)
```

In order to evaluate a Datalog query, we need some underlying data. This gives rise to the concept of *Datalog program*. A Datalog program is a set of derivation rules (a.k.a. queries) together with a set of *facts*, where a fact is a ground atom of a base predicate. The set of facts represents the database contents, which determine the result of the queries (i.e., the extension of the derived predicates).

Example 4 Consider the Datalog program formed by the Datalog derivation rules from Example 2 together with the following facts:

```
Student(john).
Course(algebra).
takesCourse(john, algebra).
Course(chemistry).
```

Given this set of facts, running the *CourseWithStudents* query retrieves *algebra*, and running the *EmptyCourse* query retrieves *chemistry*.

3. Review of Instance-Level Update in *DL-Lite*$_A$ Ontologies

In this section, we review the main concepts and results on instance-level ontology update, focusing on *DL-Lite*$_A$. Besides reasoning and querying, it is of interest to update the ontology itself, and in particular to update the extensional data in it. This form of update is called instance-level update and is the focus of the present paper.

In instance-level update we enforce the condition that the ontology resulting from the application of the update operation has the same TBox as the original ontology, hence we allow changes only in the ABox. Instance-level update is well motivated in information and knowledge management systems where the extensional level tends to change frequently, while the intensional level, which provides the representation of the domain of interest, typically remains unchanged for longer periods of time. Indeed, in these systems, changes at the extensional level correspond to changes in the data, while changes at the intensional level corresponds to changes in the schema and are usually the result of an accurate manual revision process.

In this paper, we concentrate on formula-based approaches which are computationally more manageable and hence more appropriate when developing actual ontology-based systems (Calvanese et al., 2010; Lenzerini & Savo, 2011, 2012). We review two distinct formula-based update semantics: the foundational semantics and coherence semantics (Gärdenfors, 1990; Flouris & Plexousakis, 2005). Both these semantics generate an unique update result when applied to *DL-Lite*$_A$ ontologies. We also illustrate that a third variant, the *careful semantics*, proposed in the literature (Calvanese et al., 2010), is inappropriate in our setting due to its inherent non-uniqueness of the update result in the general case.

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. An *update* is a set $\mathcal{U}$ of basic update operations of two types: insertion operations, denoted by $\mathsf{i}(A(o))$ (resp., $\mathsf{i}(P(o,o'))$), and deletion operations denoted by $\mathsf{d}(A(\vec{o}))$ (resp., $\mathsf{d}(P(o,o'))$), where both $A(o)$ and $P(o,o')$ are ABox assertions. Intuitively, updating a consistent ontology with an insertion operation $\mathsf{i}(\alpha)$, where α is an 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 α. Conversely, updating a consistent ontology with a deletion operation $\mathsf{d}(\alpha)$, 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 α.

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 (Eiter & Gottlob, 1992; Flouris & Plexousakis, 2005; Katsuno & Mendelzon, 1991). This principle states that the ontology resulting from the update should be as *close* as possible to the original one. The term *close* has no single interpretation in literature about ontology evolution. 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.

Following (Fagin et al., 1983; Lenzerini & Savo, 2011, 2012), we say that an ABox $\mathcal{A}'$ *accomplishes the update* of an ontology $\mathcal{O}$ with an update $\mathcal{U}$ if it satisfies all the insertions/deletions in $\mathcal{U}$ while remaining as close as possible to the original ABox $\mathcal{A}$. To formalize this notion it is convenient to introduce two sets: 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 (*i*) $\mathcal{A}'$ is $\mathcal{T}$-consistent, (*ii*) $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$, and (*iii*) $\mathcal{A}' = \mathcal{A}'' \cup \mathcal{A}^+_\mathcal{U}$ for some maximal subset $\mathcal{A}''$ of $\mathcal{A}$.

It easy to see that, by definition, if such ABox $\mathcal{A}'$ exists, not only satisfies $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \beta$, but it also satisfies $\langle \mathcal{T}, \mathcal{A}' \rangle \models \alpha$ for each $\alpha \in \mathcal{A}^+_\mathcal{U}$ since $\mathcal{A}^+_\mathcal{U} \subseteq \mathcal{A}'$.

The following proposition, which we prove formally below, provides necessary and sufficient conditions over the update $\mathcal{U}$ to ensure the existence of an ABox that accomplishes the update of an ontology $\mathcal{O}$ with $\mathcal{U}$.

Proposition 1 *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology and $\mathcal{U}$ be an update. An ABox $\mathcal{A}'$ that accomplishes the update of $\mathcal{O}$ with $\mathcal{U}$ exists if and only if $\mathcal{U}$ satisfies both 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) $\mathcal{A}^-_\mathcal{U} \cap cl_\mathcal{T}(\mathcal{A}^+_\mathcal{U}) = \emptyset$*, which means that the update is not asking for deleting and inserting the same knowledge at the same time.*

Proof. (If part). We start showing that if $\mathcal{U}$ satisfies both condition (*i*) and (*ii*), then there exists an ABox $\mathcal{A}'$ accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$. Since $Mod(\langle \mathcal{T}, \mathcal{A}^+_\mathcal{U} \rangle) \neq \emptyset$ then there always exists a maximal subset $\mathcal{A}''$ of $\mathcal{A}$ such that $Mod(\langle \mathcal{T}, \mathcal{A}'' \cup \mathcal{A}^+_\mathcal{U} \rangle) \neq \emptyset$. Moreover, since $\mathcal{A}^-_\mathcal{U} \cap cl_\mathcal{T}(\mathcal{A}^+_\mathcal{U}) = \emptyset$, there always exists a maximal subset $\mathcal{A}'''$ of $\mathcal{A}''$ such that $\langle \mathcal{T}, \mathcal{A}''' \cup \mathcal{A}^+_\mathcal{U} \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$. Clearly, the ABox $\mathcal{A}''' \cup \mathcal{A}^+_\mathcal{U}$ accomplishes the update of $\mathcal{O}$ with $\mathcal{U}$ according to Definition 1 since it satisfies all the three conditions specified in the definition.

(Only if part). Suppose there exists an ABox $\mathcal{A}'$ that accomplishes the update of $\mathcal{O}$ with $\mathcal{U}$. We show that $\mathcal{U}$ satisfies both conditions (i) and (ii). We have that $\mathcal{A}' = \mathcal{A}'' \cup \mathcal{A}^+_\mathcal{U}$ for some subset $\mathcal{A}''$ of $\mathcal{A}$. Since $\mathcal{A}'$ is $\mathcal{T}$-consistent, the $\mathcal{A}^+_\mathcal{U}$ is $\mathcal{T}$-consistent too, and so $\mathcal{U}$ satisfies condition (i). Moreover, since $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$, then $\langle \mathcal{T}, \mathcal{A}^+_\mathcal{U} \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$, and so also condition (ii) is satisfied by $\mathcal{U}$. ∎

Definition 2 Given a TBox $\mathcal{T}$ and an update $\mathcal{U}$, we say that $\mathcal{U}$ is *compatible* with $\mathcal{T}$ if $\mathcal{U}$ respects both the above conditions with respect to a TBox $\mathcal{T}$.

When dealing with instance-level update, one has to decide on a key aspect: whether the assertion explicitly given in the ABox have a special role which reflects a designer's choice (foundational approach) or if they are just used as a finite representation of the available extensional knowledge (coherence approach) (Flouris & Plexousakis, 2005; Gärdenfors, 1990). If we follow the "foundational approach" we do not need to preserve the facts that are entailed by the ontology but explicitly asserted in the ABox. On the other hand, if we follow the "coherence approach" the assertions in the ABox do not have a special role with respect to the ABox assertions entailed by the ontology, and hence it is natural to preserve all such entailed facts. We refer to (Gärdenfors, 1990) for a detailed discussion.

The two approaches give rise to two distinct semantics of the update: the *foundational semantics* and *coherence semantics*. As shown in the following, technically the coherence semantics can be seen as an extension of the foundational semantics. For this reason, we start our analysis by reviewing the foundational update semantics in *DL-Lite*$_A$.

3.1 Foundational Update Semantics in *DL-Lite*$_A$

We start by observing that, in formula-based approaches to the update, the formula constituting the result of the application of an update is not unique in general (Eiter & Gottlob, 1992).

Example 5 Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be the following ontology (not in *DL-Lite*$_A$ since it includes a qualified existential restriction of the left-hand side of the TBox assertion):

$$\mathcal{T} = \{ \quad \exists \mathsf{headOf.Department} \sqsubseteq \mathsf{Chair} \quad \}$$
$$\mathcal{A} = \{ \quad \mathsf{AssociateProfessor(john)},\ \mathsf{headOf(john, dep1)},\ \mathsf{Department(dep1)} \ \}$$

In words, the ontology $\mathcal{O}$ specifies that whoever is the head of a department is a chair, and that john is an associate professor that is the head of the dep1 department. Note that from the TBox assertion $\exists \mathsf{headOf.Department} \sqsubseteq \mathsf{Chair}$ in $\mathcal{T}$ and the two ABox assertions $\mathsf{headOf(john, dep1)}$ and $\mathsf{Department(dep1)}$ in $\mathcal{A}$, it follows that john is a chair, i.e., $\mathcal{O} \models \mathsf{Chair(john)}$. Suppose we want to change the ontology's ABox by performing the update $\mathcal{U} = \{\mathsf{d(Chair(john))}\}$, that is we want to change the ontology in such a way that the fact that john is a chair is no longer entailed. To achieve this, we need to modify $\mathcal{A}$ by deleting either $\mathsf{headOf(john, dep1)}$ or $\mathsf{Department(dep1)}$. Indeed, it is easy to verify that both the following ABoxes accomplish the update of $\mathcal{O}$ with $\mathcal{U}$.

$$\mathcal{A}_1 = \{ \quad \mathsf{AssociateProfessor(john)},\ \mathsf{Department(dep1)} \quad \}$$
$$\mathcal{A}_2 = \{ \quad \mathsf{AssociateProfessor(john)},\ \mathsf{headOf(john, dep1)} \quad \}$$

Note that the ABox in which we delete both $\mathsf{headOf}(\mathsf{john}, \mathsf{dep1})$ and $\mathsf{Department}(\mathsf{dep1})$ from $\mathcal{A}$ does not accomplish the update of $\mathcal{O}$ with $\mathcal{U}$ since it does not respect the minimal change principle. □

In literature, several ways to address this problem have been proposed. The most straightforward one is inspired by the *Set-Of-Theories* (SOT) principle proposed by Ginsberg (Ginsberg, 1986) and, independently, by Fagin, Ullman, and Vardi in (Fagin et al., 1983). In such an principle the idea is considering, as result of the update, the set of ontologies built by using the ABoxes which accomplish the update. In order to obtain a single ontology after the update, (Fagin et al., 1983) refines the SOT principle into the *Cross-Product approach*. Roughly speaking, by adopting the *Cross-Product approach* the ABox resulting from the update is formed by the disjunction of all the ABoxes accomplishing the update, viewing each such ABox as the conjunction of its membership assertions. In the *Cross-Product approach* we have no loss of information but the resulting ontology needs to allow for disjunctions of ABoxes, which is typically not expressible in DL ontologies; moreover, such a result of the update can be exponentially larger than the original one (Cadoli et al., 1999).

A radical approach to the ontology update proposed to deal with the problem of the multiplicity of the results is the one suggested by the *WIDTIO* (*When In Doubt Throw It Out*) *principle* (Ginsberg & Smith, 1987; Winslett, 1990), recently adopted in (Lenzerini & Savo, 2011, 2012). The idea at the base of the *WIDTIO principle* consists in combining the ontologies resulting from the SOT principle into a single one, by considering their intersection. Following the *WIDTIO principle*, the result of the update is the ontology formed by the TBox of the original ontology, and by the ABox computed as the intersection of all the ABoxes accomplishing the update.

Another strategy, proposed in (Stojanovic et al., 2002), is to let the user choose which ABox among the ones that accomplish the update should be considered as the result of the update. (Calvanese et al., 2010) instead propose a pragmatic alternative that consists in choosing non-deterministically such an ABox. Clearly, in both these approaches the result of the update cannot be uniquely defined.

When it comes to choosing among multiple ABoxes accomplishing the update, apart from the options described above, one could also resort to a preference-based approach (also called *entrenchment of beliefs*) where the system computes the result of the update on the basis of some preference criteria specified over the ontology. These preferences could be predefined at design-time, or provided as guidelines at run-time by the user to the system. Preferences could lead to a single result, or to multiple ones. See, e.g.,(Flouris et al., 2013) for an application of this approach in a relevant setting.

Fortunately, when the TBox of the ontology is expressed in *DL-Lite*$_A$, the ABox accomplishing the update is *unique*, so all the approaches above collapse into the same one: return the unique ABox. This uniqueness property was originally shown (for insertion only) in (Calvanese et al., 2010). Here, we give a full proof for sake of completeness.

Proposition 2 *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and $\mathcal{U}$ be an update compatible with $\mathcal{T}$. The ABox $\mathcal{A}'$ accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$ is unique.*

Proof. Let us first consider the case of insertion operations. We exploit the result given in (Calvanese et al., 2010, Lemma 12) which states that if a *DL-Lite*$_A$ ontology $\langle \mathcal{T}_0, \mathcal{A}_0 \rangle$ is unsatisfiable, then there exists a subset $\mathcal{A}_1 \subseteq \mathcal{A}_0$ made of at most two facts such that $\langle \mathcal{T}_0, \mathcal{A}_1 \rangle$ is unsatisfiable. As a consequence, considering our *DL-Lite*$_A$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ and the update $\mathcal{U}$, if $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_\mathcal{U}^+$ is $\mathcal{T}$-inconsistent then there are two facts $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{A}_\mathcal{U}^+$ such that $\{\alpha, \beta\}$ is $\mathcal{T}$-inconsistent. In order to preserve consistency, the fact α cannot belong to any ABox $\mathcal{A}'$ accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$, since, according to Definition 1, we have that $\mathcal{A}_\mathcal{U}^+ \subseteq \mathcal{A}'$. This means that, for each fact $\alpha \in \mathcal{A}$, deciding if α belongs to the ABox $\mathcal{A}'$ accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$ can be done by considering α alone with $\mathcal{T}$ and $\mathcal{A}_\mathcal{U}^+$, i.e. without considering any other fact in $\mathcal{A}$. Therefore, the computation of $\mathcal{A}'$ is deterministic.

As for the case of deletion operations in $\mathcal{U}$, we exploit again a property of *DL-Lite*$_A$ which directly follows from its syntactic structure: given a consistent *DL-Lite*$_A$ ontology $\langle \mathcal{T}_0, \mathcal{A}_0 \rangle$ and a fact α, if $\langle \mathcal{T}_0, \mathcal{A}_0 \rangle \models \alpha$, then there exists a single fact $\beta \in \mathcal{A}_0$ such that $\langle \mathcal{T}, \{\beta\} \rangle \models \alpha$. As a consequence, in computing the ABox $\mathcal{A}'$ accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$, we can consider each fact $\beta \in \mathcal{A}$ individually and verify if there exists a fact $\alpha \in \mathcal{A}_\mathcal{U}^-$ such that $\langle \mathcal{T}, \{\beta\} \rangle \models \alpha$. If this is the case, β cannot belong to $\mathcal{A}'$. So, also in this case the computation of $\mathcal{A}'$ is deterministic, which proves the proposition. ∎

As observed, all the approaches mentioned above differ in handling multiple update results. However, when applied to a *DL-Lite*$_A$ ontology, they all coincide, since there is at most one ABox accomplishing the update (cf. Proposition 2). Hence we can simply define the foundational semantics for instance-level update in *DL-Lite*$_A$ ontologies as follows:

Definition 3 [*Foundational Update Semantics*] Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent *DL-Lite*$_A$ ontology and $\mathcal{U}$ be an update compatible 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 (unique) ABox accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$.

Example 6 Consider the *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}\ A \rangle$ of Example 1 and the following update request $\mathcal{U} = \{\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john})), \mathsf{d}(\mathsf{Course}(\mathsf{algebra}))\}$. Since the update asks to insert the assertion $\mathsf{AssociateProfessor}(\mathsf{john})$ stating that john is an associate professor, it follows that, to preserve the consistency, in the updated ontology john can no longer be a full professor. Moreover, since the update asks for retracting the fact that $\mathsf{algebra}$ is a course, it follows from the TBox that no student can take it. Hence, the assertion $\mathsf{takesCourse}(\mathsf{bob}, \mathsf{algebra})$ has to be deleted. On the basis of the above considerations, it is easy to verify that the ABox accomplishing the update of $\mathcal{O}$ with $\mathcal{U}$ is:

$$\mathcal{A}_{new} = \{ \quad \mathsf{AssociateProfessor}(\mathsf{john}) \quad \}$$

So, according to Definition 3, we have that $\mathcal{O} \circ \mathcal{U} = \langle \mathcal{T}, \mathcal{A}_{new} \rangle$. □

3.2 Coherence Update Semantics in *DL-Lite*$_A$

Observe that, in the foundational semantics, the update result is *syntax dependent*, which means that it depends on the actual assertions in the original ABox, as the following example shows.

Example 7 Consider the ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{T}$ is the TBox given in Example 1 and $\mathcal{A}'$ is as follows:

$$\mathcal{A}' = \{ \quad \mathsf{FullProfessor(john)},\ \mathsf{takesCourse(bob, algebra)},\ \mathsf{Student(bob)} \quad \}$$

Note that, since $\mathcal{T} \models \exists \mathsf{takesCourse} \sqsubseteq \mathsf{Student}$, the ontology $\mathcal{O}'$ is logically equivalent to the ontology $\mathcal{O}$ of Example 1. Moreover, let $\mathcal{U} = \{\mathsf{i}(\mathsf{AssociateProfessor(john)}), \mathsf{d}(\mathsf{Course(algebra)})\}$ be the update request given in Example 6. We have that $\mathcal{O}' \circ \mathcal{U} = \langle \mathcal{T}, \mathcal{A}'_{new} \rangle$ where

$$\mathcal{A}'_{new} = \{ \quad \mathsf{AssociateProfessor(john)},\ \mathsf{Student}(\mathit{bob}) \quad \}$$

which is clearly different from the ABox $\mathcal{A}_{new}$ of Example 6. □

Depending on the specific scenario, and the particular application at hand, the syntax dependency of foundational semantics might be considered inappropriate. Hence a *syntax independent* semantics for update, following the "coherence approach", is of interest. This motivates the definition of the following update semantics for *DL-Lite*$_A$ ontologies, in which the object of the update is not the original ABox, but its deductive closure with respect to the TBox (Calvanese et al., 2010; Lenzerini & Savo, 2011).

Definition 4 [*Coherence Update Semantics*] Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent *DL-Lite*$_A$ ontology and let $\mathcal{U}$ be an update compatible with $\mathcal{T}$. An ontology $\langle \mathcal{T}, \mathcal{A}'' \rangle$ results from updating $\mathcal{O}$ with $\mathcal{U}$ according to the coherence semantics, denoted by $\mathcal{O} \bullet \mathcal{U}$, if $\langle \mathcal{T}, \mathcal{A}'' \rangle$ is logically equivalent to $\langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{A}'$ is the (unique) ABox accomplishing the update of $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$.

The following proposition states that the Coherence Update Semantics is indeed syntax independent.

Proposition 3 *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ be two consistent and logically equivalent DL-Lite$_A$ ontologies and let $\mathcal{U}$ be an update compatible with $\mathcal{T}$. Then $\mathcal{O} \bullet \mathcal{U}$ and $\mathcal{O}' \bullet \mathcal{U}$ are logically equivalent.*

Proof. The proof trivially follows from the fact that $cl_{\mathcal{T}}(\mathcal{A}) = cl_{\mathcal{T}}(\mathcal{A}')$. ∎

Example 8 Consider again the *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ of Example 1 and the update request $\mathcal{U} = \{\mathsf{i}(\mathsf{AssociateProfessor(john)}), \mathsf{d}(\mathsf{Course(algebra)})\}$ given in Example 6. According to Definition 4, the result of updating of $\mathcal{O}$ with $\mathcal{U}$ is the ontology $\langle \mathcal{T}, \mathcal{A}^*_{new} \rangle$ where:

$$\mathcal{A}^*_{new} = \{ \quad \mathsf{AssociateProfessor(john)},\ \mathsf{Professor(john)},\ \mathsf{Person(john)},\ \mathsf{Student}(\mathit{bob}),\ \mathsf{Person}(\mathit{bob}) \quad \}$$

Indeed, it is easy to see that $\mathcal{A}^*_{new}$ is the ABox computed from the deductive closure of $\mathcal{A}$ with respect to $\mathcal{T}$ by inserting the assertion $\mathsf{AssociateProfessor(john)}$ and deleting the assertions $\mathsf{Course(algebra)}$, $\mathsf{takesCourse(bob, algebra)}$, and $\mathsf{FullProfessor(john)}$. As in Example 6, this last assertion is removed as a consequence of the insertion request $\mathsf{i}(\mathsf{AssociateProfessor(john)})$.

Now, consider the $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ of Example 7 which is logically equivalent to $\mathcal{O}$. Since $cl_{\mathcal{T}}(\mathcal{A}') = cl_{\mathcal{T}}(\mathcal{A})$, it is easy to verify that the result of updating of $\mathcal{O}'$ with $\mathcal{U}$ is the same as the result of updating $\mathcal{O}$ with $\mathcal{U}$. □

Observe that the ontology resulting from the update according the coherence semantics is *not* unique (see Definition 4). Indeed, all updated ontologies $\langle \mathcal{T}, \mathcal{A}'' \rangle$, such that $cl_{\mathcal{T}}(\mathcal{A}'') = cl_{\mathcal{T}}(\mathcal{A}')$ satisfy the definition. However, all such ontologies are logically equivalent to each other. Therefore, up to logical equivalence, we can still consider *unique* the result of the update. For this reason, with a little abuse of terminology, in the rest of the paper, we continue to refer to *the* result of the update also in case of the coherence semantics.

3.3 Careful Update Semantics in *DL-Lite*$_A$

An alternative formula-based update semantics, based on a variant of the coherence semantics, is the *Careful semantics* (Calvanese et al., 2010). This 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 (Calvanese et al., 2010, Theorem 16), this is not the case in general, as the following example shows.

Example 9 Consider the *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where:

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

and the update $\mathcal{U} = \{\mathsf{i}(C(o)), \mathsf{i}(D(o))\}$. 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. □

Since the result of the update can be not unique in general, we consider the Careful Semantics as inappropriate for our tasks in this paper.

4. FO-Rewritability of Foundational Update Semantics

Now we are ready to present the main result of this paper: FO-rewritability of instance-level update in *DL-Lite*$_A$. In doing this we provide a technique that can be exploited in practice, which is based on non-recursive Datalog. We start by handling the foundational semantics in this section, and then extend the technique to handle the coherence semantics in the next section. 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 immediately extended to TBoxes containing such kinds of assertions.

From now on, given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote by $ABox(\mathcal{O})$ the ABox $\mathcal{A}$. Therefore, if $\mathcal{O} \circ \mathcal{U} = \langle \mathcal{T}, \mathcal{A}' \rangle$, then $ABox(\mathcal{O} \circ \mathcal{U}) = \mathcal{A}'$.

We aim at characterizing the complexity of the following basic problem for the update operation, under both the foundational and the coherence semantics: given a *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an update $\mathcal{U}$, and an ABox assertion α, decide whether α belongs to the ABox $\mathcal{A}'$, where $\mathcal{A}' = ABox(\mathcal{O} \circ \mathcal{U})$ under the foundational semantics, and $\mathcal{A}' = ABox(\mathcal{O} \bullet \mathcal{U})$ under the coherence semantics.[4] We call such a problem the *decision problem of the update*.

Given a *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, and an update $\mathcal{U}$, we define a Datalog program $\mathcal{D}$ that permits querying whether $\mathcal{U}$ is compatible 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 the foundational update semantics given in Definition 3.

The Datalog program $\mathcal{D}$ contains a derived predicate *incompatible_update()*, together with a pair of derived predicates *ins_a*/*del_a* for each concept/role A such that:

- *incompatible_update()* is true if and only if $\mathcal{U}$ is not compatible with $\mathcal{T}$.

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

- *ins_a*$(\overline{o})$ is true if and only if the assertion $A(\overline{o})$ was not in $\mathcal{A}$, but $A(\overline{o}) \in ABox(\mathcal{O} \circ \mathcal{U})$. That is, *ins_a* captures the assertions of $\mathcal{A}$ that should be inserted into $\mathcal{A}$ to accomplish the update $\mathcal{U}$ according to the foundational semantics.
- *del_a*$(\overline{o})$ is true if and only if the assertion $A(\overline{o})$ was in $\mathcal{A}$, but $A(\overline{o}) \notin ABox(\mathcal{O} \circ \mathcal{U})$. That is, *del_a* captures the assertions of $\mathcal{A}$ that should be deleted from $\mathcal{A}$ to accomplish the update $\mathcal{U}$ according to the foundational semantics.

Intuitively, the main idea of the translation is to see each ABox assertion in $\mathcal{A}$, and each update request in $\mathcal{U}$ as a Datalog fact. Then, we embed each assertion in the closure of $\mathcal{T}$ into several Datalog derivation rules that define the *incompatible_update()*, *ins_a*$(\overline{X})$, *del_a*$(\overline{X})$ predicates. In the following, we detail how to obtain such a Datalog program $\mathcal{D}$. Then, we prove that the set of instructions generated by $\mathcal{D}$ are sound and complete w.r.t. computing $ABox(\mathcal{O} \circ \mathcal{U})$.

4. As explained in the previous section, we can consider as unique the result of the update under the coherence semantics: more precisely, in the following we assume that $ABox(\mathcal{O} \bullet \mathcal{U})$ is the ABox accomplishing the update of $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$.

4.1 Encoding in Datalog

We want to compute *incompatible_update()*, *ins_a($\overline{X}$)*, *del_a($\overline{X}$)* predicates. As said before, if *incompatible_update()* is true, then the update is not compatible with the TBox and hence cannot be applied. If, instead, *incompatible_update()* is false, then the program $\mathcal{D}$ returns all the *ins_a($\overline{X}$)*, *del_a($\overline{X}$)* predicates indicating the facts to be inserted and deleted from the ABox respectively. Observe that, the set of *ins_a($\overline{X}$)* and the set of *del_a($\overline{X}$)* are guaranteed to be disjoint, being the update compatible with the TBox.

All the assertions in $\mathcal{A}$ and operations in $\mathcal{U}$ can be seen as different facts in Datalog. In particular:

- each assertion $A(\overline{o}) \in \mathcal{A}$ is seen as the fact *a($\overline{o}$)*;
- each operation $\mathsf{i}(A(\overline{o})) \in \mathcal{U}$ is seen as the fact *ins_a_request($\overline{o}$)*;
- each operation $\mathsf{d}(A(\overline{o})) \in \mathcal{U}$ is seen as the fact *del_a_request($\overline{o}$)*.

Intuitively, a fact *ins_a_request($\overline{o}$)*/*del_a_request($\overline{o}$)* indicate that the ontology has received the request to insert/delete the ABox assertion $A(\overline{o})$.

The facts corresponding to $\mathcal{A}$ and $\mathcal{U}$ can directly lead to compute the *ins_a($\overline{o}$)*/*del_a($\overline{o}$)* atoms as well as the atom *incompatible_update()*. In particular, we have the following rules:

- For each atomic concept A of the ontology $\mathcal{O}$:

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

- For each atomic role P of the ontology $\mathcal{O}$:

```
ins_p(X,Y) :- ins_p_request(X,Y), not p(X,Y).
del_p(X,Y) :- del_p_request(X,Y), p(X,Y).
incompatible_update() :- ins_p_request(X,Y), del_p_request(X,Y).
```

Note that *incompatible_update()* becomes true in case $\mathcal{U}$ requests for the insertion and deletion of the same fact.

We deal with positive, negative, and functional assertions in the closure of $\mathcal{T}$ separately. As for the positive inclusion assertions, intuitively, when an update requests for deleting an assertion $A(o)$, we have to delete any other assertion $B(o)$ in the ABox that with the TBox should entail $A(o)$. Note that this is cannot be accomplished if there is a request for inserting $B(o)$ in $\mathcal{U}$. So if this happens *incompatible_update()* is set to true. For instance, the positive inclusion assertion $\mathsf{Professor} \sqsubseteq \mathsf{Person}$ in our running example leads to have in $\mathcal{D}$ the following rules:

```
del_prof(X) :- prof(X), del_person_request(X).
incompatible_update() :- ins_prof_request(X), del_person_request(X).
```

Notice that we use the closure of $\mathcal{T}$, instead of $\mathcal{T}$ itself in order to capture deletions that are propagated along the concept and role hierarchies in the ontology. For instance, 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 encoding of the assertion $\mathsf{FullProfessor} \sqsubseteq \mathsf{Person}$ appearing in $cl(\mathcal{T})$:

```
del_full(X) :- full(X), del_person_request(X).
```

The Datalog program $\mathcal{D}$ is therefore specified as follows:

- For each positive inclusion assertion $B \sqsubseteq A$ in $cl(\mathcal{T})$, with A and B atomic concepts, we have in $\mathcal{D}$ the rules:

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

- For each positive inclusion assertion in $cl(\mathcal{T})$ of the form $\exists P \sqsubseteq B$, we have:

```
del_p(X,Y) :- p(X,Y), del_b_request(X).
incompatible_update() :- ins_p_request(X,Y), del_b_request(X).
```

- For each positive inclusion assertion in $cl(\mathcal{T})$ of the form $\exists P^- \sqsubseteq B$, we have:

```
del_p(X,Y) :- p(X,Y), del_b_request(Y).
incompatible_update() :- ins_p_request(X,Y), del_b_request(U).
```

- For each positive inclusion assertion in $cl(\mathcal{T})$ of the form $R \sqsubseteq P$, we have:

```
del_r(X,Y) :- r(X,Y), del_p_request(X,Y).
incompatible_update() :- ins_r_request(X,Y), del_p_request(X,Y).
```

- For each positive inclusion assertion in $cl(\mathcal{T})$ of the form $R \sqsubseteq P^-$ or $R^- \sqsubseteq P$, we have:

```
del_r(X,Y) :- r(X,Y), del_p_request(Y,X).
incompatible_update() :- ins_r_request(X,Y), del_p_request(Y,X).
```

We do not define rules for translating assertions of the form $A \sqsubseteq \exists P$, since the accomplishment of the request of deleting an assertion $P(o, o')$ does not require to delete the assertion $A(o)$ from the ABox.

As for the negative inclusion assertions, consider the disjointness $\mathsf{FullProfessor} \sqsubseteq \neg\mathsf{AssociateProfessor}$ in our ontology. if the update contains the insertion request $\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john}))$ when $\mathsf{FullProfessor}(\mathsf{john})$ is in the ABox, we have to delete $\mathsf{FullProfessor}(\mathsf{john})$, otherwise the ABox resulting from the update will be inconsistent with respect to the TBox.

```
del_full(X) :- b(X), ins_associate_request(X).
del_associate(X) :- a(X),  ins_full_request(X).
incompatible_update():-ins_full_request(X),ins_associate_request(X).
```

The last rule is used for managing the case where the update requests to insert both $\mathsf{FullProfessor}(\mathsf{john})$ and $\mathsf{AssociateProfessor}(\mathsf{john})$. Indeed, in this case we have a contradiction and thus, *incompatible_update()* becomes true.

By following the above intuition, each negative inclusion assertion in the closure of $\mathcal{T}$ is encoded in the Datalog program $\mathcal{D}$ as follows:

- For each negative inclusion assertion $B \sqsubseteq \neg A$ in $cl(\mathcal{T})$, we have:

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

– For each negative inclusion assertion $B \sqsubseteq \neg\exists P$ and $\exists P \sqsubseteq \neg B$ in $cl(\mathcal{T})$, we have:

```
del_b(X) :- b(X), ins_p_request(X,Y).
del_p(X,Y) :- p(X,Y), ins_b_request(X).
incompatible_update() :- ins_b_request(X), ins_p_request(X,Y).
```

– For each negative inclusion assertion $B \sqsubseteq \neg\exists P^-$ and $\exists P^- \sqsubseteq \neg B$ in $cl(\mathcal{T})$, we have:

```
del_b(X) :- b(X), ins_p_request(Y,X).
del_p(X,Y) :- p(X,Y), ins_b_request(Y).
incompatible_update() :- ins_b_request(X), ins_p_request(Y,X).
```

– For each negative inclusion assertion $R \sqsubseteq \neg P$ in $cl(\mathcal{T})$, we have:

```
del_r(X,Y) :- r(X,Y), ins_p_request(X,Y).
del_p(X,Y) :- p(X,Y), ins_r_request(X,Y).
incompatible_update() :- ins_p_request(X,Y), ins_r_request(X,Y).
```

– For each negative inclusion assertion $R \sqsubseteq \neg P^-$ in $cl(\mathcal{T})$, we have:

```
del_r(X,Y) :- r(X,Y), ins_p_request(Y,X).
del_p(X,Y) :- p(X,Y), ins_r_request(Y,X).
incompatible_update() :- ins_p_request(X,Y), ins_r_request(Y,X).
```

Again, note that since we are considering the closure of $\mathcal{T}$, the rules are able to capture deletions due to negative inclusion assertions generated by entailment. For instance, if we try to update the ontology of Example 1 with the request $\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{bob}))$, $\mathcal{D}$ generates the deletion of $\mathsf{takesCourse}(\mathsf{bob}, \mathsf{algebra})$ because of the rule `del_takes(X,Y) :- takes(X,Y), ins_associate_request(X)` obtained when translating the negative inclusion assertion $\exists\mathsf{takesCourse} \sqsubseteq \neg\mathsf{AssociateProfessor}$ in $cl(\mathcal{T})$.

Finally, in order to deal with role functionality assertions, we use the inequality built-in predicate to check whether a requested role assertion insertion is going to violate the functional assertion,[5] and proceed as follows:

– For each role functionality assertion $(\mathsf{funct}\ P)$ in $\mathcal{T}$, we have:

```
del_p(X,Y) :- p(X,Y), ins_p_request(X,Z), Y<>Z.
incompatible_update() :- ins_p_request(X,Y),ins_p_request(X,Z),
                         Y<>Z.
```

– For each role functionality assertion $(\mathsf{funct}\ P^-)$ in $\mathcal{T}$, we have:

```
del_p(X,Y) :- p(X,Y), ins_p_request(Z,Y), X<>Z.
incompatible_update() :- ins_p_request(X,Y),ins_p_request(Z,Y),
                         X<>Z.
```

5. We remind the reader that *DL-Lite*$_A$ enforces the unique name assumption (UNA), cf. Section 2.

Note that since in *DL-Lite*$_A$ no new functionality assertions can be derived by reasoning over the TBox (Calvanese et al., 2006), we can consider $\mathcal{T}$ instead of its closure $cl(\mathcal{T})$.

Let us illustrate the construction above with an example.

Example 10 Consider again the *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}\, A \rangle$ given in Example 1 and the update request $\mathcal{U} = \{\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john})), \mathsf{d}(\mathsf{Course}(\mathsf{algebra}))\}$ of Example 6.

The Datalog program $\mathcal{D}$ contains the following rules:

```
%ABox assertions
full(john).
takes(bob, algebra).
%Update requests
ins_associate_request(john).
del_course_request(algebra).
 ...
ins_associate(X) :- ins_associate_request(X), not associate(X).          %r1
del_associate(X) :- del_associate_request(X), associate(X).
incompatible_update() :- ins_associate_request(X), del_associate_request(X).
 ...
ins_course(X) :- ins_course_request(X), not course(X).
del_course(X) :- del_course_request(X), course(X).
incompatible_update() :- ins_course_request(X), del_course_request(X).
 ...
del_takes(X,Y) :- takes(X,Y), del_student_request(X).
del_takes(X,Y) :- takes(X,Y), del_course_request(Y).                      %r2
incompatible_update() :- ins_takes_request(X,Y), del_student_request(X).
incompatible_update() :- ins_takes_request(X,Y), del_course_request(Y).
 ...
del_full(X) :- full(X), ins_associate_request(X).                         %r3
del_associate(X) :- associate(X), ins_full_request(X).
 ...
```

Since in $\mathcal{U}$ we have the update request $\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john}))$, the rule `r1` in $\mathcal{D}$ generates `ins_associate(john)`, and, since we have that $\mathsf{FullProfessor} \sqsubseteq \neg\mathsf{AssociateProfessor} \in cl(\mathcal{T})$, which leads to have rule `r3` in $\mathcal{D}$, we compute `del_full(john)`. For accomplishing the update request $\mathsf{d}(\mathsf{Course}(\mathsf{algebra}))$ in $\mathcal{U}$, we have `del_takes(bob,algebra)` from the rule `r2`. The latter is in $\mathcal{D}$ because of the assertion $\exists\mathsf{takesCourse}^- \sqsubseteq \mathsf{Course}$ in $cl(\mathcal{T})$. At the end, $\mathcal{D}$ will compute to the following result:

```
ins_associate(john).
del_takes(bob,algebra).
del_full(john).
```

which leads to compute the ABox $\mathcal{A}_{new} = \{\ \mathsf{AssociateProfessor}(\mathsf{john})\ \}$ of Example 6. □

4.2 Results

In the Datalog program the atom *incompatible_update()* is used to detect if an update is compatible with the TBox or not. The following result shows that such check is correctly performed by our procedure.

Theorem 1 *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and $\mathcal{U}$ be an update, and let $\mathcal{D}$ be the corresponding Datalog program defined as above (Section 4.1). $\mathcal{U}$ is not compatible with $\mathcal{T}$ if and only if* incompatible_update() *is true in $\mathcal{D}$.*

Proof. (If part.) We start by showing that if *incompatible_update()* is true in $\mathcal{D}$, then $\mathcal{U}$ is not compatible with $\mathcal{T}$. `incompatible_update()` can be true only because of the facts generated in $\mathcal{D}$ from the translation of the update $\mathcal{U}$, together with: (i) one of the rules `incompatible_update():- ins_a_request(X), del_a_request(X)` (resp., `incompatible_update():-ins_r_request(X,Y), del_r_request(X,Y)`) defined in $\mathcal{D}$ for each concept A (resp., role R); (ii) a rule `incompatible_update() :- ins_b_request(X), del_a_request(X)` (resp., `incompatible_update():-ins_p_request(X,Y), del_r_request(X,Y)`) defined in $\mathcal{D}$ for each concept inclusion assertion $B \sqsubseteq A$ (resp., role inclusion assertion $P \sqsubseteq R$) in $cl(\mathcal{T})$; (iii) one of the rules generated by translating the negative inclusion assertions and the functionality assertions in $cl(\mathcal{T})$. 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 `incompatible_update()` is true, $\mathcal{U}$ is not compatible with $\mathcal{T}$.

(Only if part.) Now, we show that if $\mathcal{U}$ is not compatible with $\mathcal{T}$, then *incompatible_update()* is true in $\mathcal{D}$. $\mathcal{U}$ can be not compatible with $\mathcal{T}$ because (i) $Mod(\langle \mathcal{T}, \mathcal{A}_\mathcal{U}^+ \rangle) \neq \emptyset$; or (ii) $\mathcal{A}_\mathcal{U}^- \cap cl_\mathcal{T}(\mathcal{A}_\mathcal{U}^+) = \emptyset$. About (i), the following two cases are possible. There are two facts $r(o, o')$ and $r(o, o'')$ in $\mathcal{A}_\mathcal{U}^+$ which violate the functionality of the role r specified $\mathcal{T}$. In this case, since `incompatible_update() :- ins_r_request(X,Y),ins_r_request(X,Z),Y<>Z` is in $\mathcal{D}$, then `incompatible_update()` will be true. In the second case, we have two facts $A(\overline{o})$ and $B(\overline{o})$ such that $A \sqsubseteq \neg B \in cl(\mathcal{T})$. Also in this case, since we have that `incompatible_update() :- ins_a_request(X), ins_b_request(X)` is in $\mathcal{D}$, then `incompatible_update()` will be true. About (ii), since for each assertion $B \sqsubseteq A$ in $cl(\mathcal{T})$ we have that `incompatible_update() :- ins_b_request(X), del_a_request(X)` is in $\mathcal{D}$, then it follows that if $\mathcal{A}_\mathcal{U}^- \cap cl_\mathcal{T}(\mathcal{A}_\mathcal{U}^+) \neq \emptyset$, then `incompatible_update()` will be true in $\mathcal{D}$. ∎

If *incompatible_update()* is false in $\mathcal{D}$, then we can compute all the insertion and deletion required to perform the update using the Datalog program $\mathcal{D}$. We now show that this way to proceed is sound and complete. We start by showing soundness, i.e., that for every ABox assertion $A(\overline{o})$ that should be inserted/deleted according to $\mathcal{D}$, $A(\overline{o})$ should be inserted/deleted according to the foundational update semantics.

Theorem 2 (Soundness) *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and $\mathcal{U}$ be an update compatible with $\mathcal{T}$, and let $\mathcal{D}$ be the corresponding Datalog program defined as above (Section 4.1). For each concept/role A, if* ins_a$(\overline{o})$ *is true in $\mathcal{D}$, then, $A(\overline{o}) \in ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}) \setminus \mathcal{A}$, and if* del_a$(\overline{o})$ *is true in $\mathcal{D}$, then, $A(\overline{o}) \in \mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$.*

Proof. If the fact `ins_a(o)` (resp., `ins_r(o,o')`) is true in $\mathcal{D}$, it is because of the rule `ins_a(X) :- ins_a_request(X), not a(X)` generated when translating $\mathcal{U}$ (resp., `ins_r(X,Y) :- ins_r_request(X,Y), not r(X,Y)`), which can only be true if $A(o) \notin \mathcal{A}$ (resp., $R(o, o') \notin \mathcal{A}$), and $A(o)$ (resp., $R(o, o')$) is in $\mathcal{A}_\mathcal{U}^+$, thus $A(o)$ (resp., $R(o, o')$) belongs to $ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}) \setminus \mathcal{A}$.

If `del_a(o)` (resp., `del_r(o,o')`) is true in $\mathcal{D}$, it can only be because of: (i) the rule `del_a(X) :- del_a_request(X), a(X)` (resp., `del_r(X,Y) :- del_r_request(X,Y),`

`r(X,Y)`) generated when translating $\mathcal{U}$, where in such case we have $A(o)$ (resp., R(o.o')) belongs to both $\mathcal{A}$ and $\mathcal{A}^-_{\mathcal{U}}$, thus $A(o)$ (resp., $R(o, o')$) belongs to $\mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$; or (ii) a rule generated when translating a positive inclusion assertion in $cl(\mathcal{T})$, where in such case we have that $A(\overline{o}) \in \mathcal{A}$ and that for some $B(\overline{o}) \in \mathcal{A}^-_{\mathcal{U}}$, $\langle \mathcal{T}, \{A(\overline{o})\} \rangle \models B(\overline{o})$, thus, $A(\overline{o}) \in \mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$ (where $A(\overline{o})$ and $B(\overline{o})$ are denoting concepts or roles); or (iii) a rule generated when translating a negative inclusion assertion or a functionality assertion 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, $A(\overline{o}) \in \mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$. ∎

Now, we turn to completeness, i.e., that any insertion/deletion assertion of $A(\overline{o})$ that should be applied according to the foundational update semantics is also generated in $\mathcal{D}$. We first need to better characterize which atoms are inserted and deleted according to the update semantics for an update $\mathcal{U}$. As for the atoms to be inserted, these are those in $\mathcal{A}^+_{\mathcal{U}}$ (i.e., the assertions requested to be inserted in $\mathcal{U}$) that are not already in $\mathcal{A}$. As for the atoms to be deleted, we need to suitably close $\mathcal{A}^-_{\mathcal{U}}$ (i.e., the assertions requested to be deleted in $\mathcal{U}$) so as to have $\mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$. To do so, we can use the saturation procedure in Algorithm 1.

We now prove that $ComputeDeletedAssertions(\mathcal{T}, \mathcal{A}, \mathcal{U})$ actually returns the assertions that must be deleted from $\mathcal{A}$ to compute the update. To this aim, we state the following lemmas, which involve the algorithms for satisfiability and query answering for *DL-Lite*$_A$ defined in (Calvanese et al., 2009). The proof of these lemmas is immediate.

Lemma 1 *Let $\langle \mathcal{T}, A \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update compatible with $\mathcal{T}$, $\mathcal{A}' = \mathcal{A}^+_{\mathcal{U}} \cup (\mathcal{A} \setminus \mathcal{A}_{sat})$, and let $Satisfiable$ be the algorithm defined in Sec. 4.3 of (Calvanese et al., 2009). Then, $Satisfiable(\langle \mathcal{T}, \mathcal{A}' \rangle)$ returns true.*

Lemma 2 *Let $\langle \mathcal{T}, A \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update compatible with $\mathcal{T}$, $\alpha \in \mathcal{A}^-_{\mathcal{U}}$, $\mathcal{A}' = \mathcal{A} \setminus \mathcal{A}_{ans}$, and let $Answer$ be the algorithm defined in Sec. 5.3 of (Calvanese et al., 2009). Then, $Answer(\alpha, \langle \mathcal{T}, \mathcal{A}' \rangle)$ returns false.*

The above lemmas, together with the correctness of the algorithms for satisfiability and query answering in (Calvanese et al., 2009), allow us to prove the correctness of Algorithm 1.

Lemma 3 (Correctness of Algorithm 1) *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology and $\mathcal{U}$ be an update compatible with $\mathcal{T}$. Then $ComputeDeletedAssertions(\mathcal{T}, \mathcal{A}, \mathcal{U}) = \mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$.*

Proof. The soundness of the algorithm is trivial. As regards the completeness, from Lemma 1 and from Lemma 4.13 of (Calvanese et al., 2009) it follows that the set $\mathcal{A}_{sat}$ computed by $ComputeDeletedAssertions(\mathcal{T}, \mathcal{A}, \mathcal{U})$ contains all the facts in $\mathcal{A}$ that are in conflict with $\mathcal{A}^+_{\mathcal{U}}$. Moreover, from Lemma 2 and from Theorem 5.14 of (Calvanese et al., 2009) it follows that the set $\mathcal{A}_{ans}$ computed by $ComputeDeletedAssertions(\mathcal{T}, \mathcal{A}, \mathcal{U})$ contains all the facts in $\mathcal{A}$ that, together with the TBox $\mathcal{T}$, entail any fact in $\mathcal{A}^-_{\mathcal{U}}$. ∎

By exploiting the above result we can show completeness of our Datalog-based procedure.

Algorithm 1 ComputeDeletedAssertions($\mathcal{T}, \mathcal{A}, \mathcal{U}$)

Input: *DL-Lite*$_A$ TBox $\mathcal{T}$, ABox $\mathcal{A}$, update $\mathcal{U}$ compatible with $\mathcal{T}$
Output: the ABox $\mathcal{A}_{sat} \cup \mathcal{A}_{ans}$
begin
  $\mathcal{A}_{sat} = \emptyset$;
  $\mathcal{A}_{ans} = \emptyset$;
  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 $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{D(a)\}$;
    for each $R(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models C \sqsubseteq \neg\exists R$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{R(a, x)\}$;
    for each $R(x, a) \in \mathcal{A}$ such that $\mathcal{T} \models C \sqsubseteq \neg\exists R^{-}$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{R(x, a)\}$
  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 $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(a, b)\}$;
    for each $S(b, a) \in \mathcal{A}$ such that $\mathcal{T} \models R \sqsubseteq \neg S^{-}$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(b, a)\}$;
    for each $C(a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg C$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{C(a)\}$;
    for each $C(b) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^{-} \sqsubseteq \neg C$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{C(b)\}$;
    for each $S(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg\exists S$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(a, x)\}$;
    for each $S(x, a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq \neg\exists S^{-}$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(x, a)\}$;
    for each $S(b, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^{-} \sqsubseteq \neg\exists S$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(b, x)\}$;
    for each $S(x, b) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^{-} \sqsubseteq \neg\exists S^{-}$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{S(x, b)\}$
    for each $R(a, c) \in \mathcal{A}$ such that $b \neq c$ and $\mathcal{T} \models (funct\ R)$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{R(a, c)\}$;
    for each $R(c, b) \in \mathcal{A}$ such that $a \neq c$ and $\mathcal{T} \models (funct\ R^{-})$ do $\mathcal{A}_{sat} = \mathcal{A}_{sat} \cup \{R(c, b)\}$;
  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 $\mathcal{A}_{ans} = \mathcal{A}_{ans} \cup \{D(a)\}$;
    for each $R(a, x) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R \sqsubseteq C$ do $\mathcal{A}_{ans} = \mathcal{A}_{ans} \cup \{R(a, x)\}$;
    for each $R(x, a) \in \mathcal{A}$ such that $\mathcal{T} \models \exists R^{-} \sqsubseteq C$ do $\mathcal{A}_{ans} = \mathcal{A}_{ans} \cup \{R(x, a)\}$
  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 $\mathcal{A}_{ans} = \mathcal{A}_{ans} \cup \{S(a, b)\}$;
    for each $S(b, a) \in \mathcal{A}$ such that $\mathcal{T} \models S \sqsubseteq R^{-}$ do $\mathcal{A}_{ans} = \mathcal{A}_{ans} \cup \{S(b, a)\}$
  end;
  return $\mathcal{A}_{sat} \cup \mathcal{A}_{ans}$
end

Theorem 3 (Completeness) *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update compatible with $\mathcal{T}$, and let $\mathcal{D}$ be the corresponding Datalog program defined as above (Section 4.1). Then, for each concept/role $\mathcal{A}$, if $A(\overline{o}) \in ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}) \setminus \mathcal{A}$, then,* ins_a($\overline{o}$) *is true in $\mathcal{D}$, and if $A(\overline{o}) \in \mathcal{A} \setminus ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U})$, then,* del_a($\overline{o}$) *is true in $\mathcal{D}$.*

Proof. Since $\mathcal{U}$ is compatible with $\mathcal{T}$, then $ABox(\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U}) \setminus \mathcal{A} = \mathcal{A}_{\mathcal{U}}^{+} \setminus \mathcal{A}$, and by definition of $\mathcal{D}$, for each fact $A(\overline{o})$ in $\mathcal{A}_{\mathcal{U}}^{+}$, we have that `ins_a(`$\overline{X}$`) :- ins_a_request(`$\overline{X}$`), not a(`$\overline{X}$`)` is in $\mathcal{D}$, so, 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, by Lemma 3 we have that for every assertion deleted from $\mathcal{A}$ there is a corresponding deletion instruction provided by $\mathcal{D}$. Indeed, 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 $\mathcal{D}$. ∎

The above results show that by means of the Datalog program $\mathcal{D}$, when paired with facts in $\mathcal{A}$ and $\mathcal{U}$, we are able to compute all the facts that must be inserted and deleted from the ABox $\mathcal{A}$ to perform the update. It is easy to see that $\mathcal{D}$ is polynomial in the size of the TBox $\mathcal{T}$ and independent of the data in the ABox $\mathcal{A}$ and in the update $\mathcal{U}$. More importantly, given the non-recursive form of $\mathcal{D}$, we have that each *ins_a($\overline{x}$)/del_a($\overline{x}$)* over $\mathcal{D}$ can be translated into a FO query. This means that, given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and update $\mathcal{U}$, from the TBox $\mathcal{T}$ only, we can build an SQL/SPARQL query that evaluated over the ABox $\mathcal{A}$ and the update $\mathcal{U}$ allows for instantiating all the insertion and deletion instructions needed to compute the ABox of the update $\mathcal{O} \circ \mathcal{U}$. Hence, we have the following result.

Theorem 4 (FO-Rewritability) *The decision problem of the update under the foundational semantics for DL-Lite$_A$ ontologies is FO-rewritable and in* AC^0 *in data complexity.*

Proof. The proof directly follows from Theorem 1, Theorem 6, Theorem 3, and from the fact that the evaluation of a FO query over an ABox is in AC^0 in data complexity (Abiteboul et al., 1995). ∎

5. FO-Rewritability of Coherence Update Semantics

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 update semantics. In this section we show how to modify such Datalog program to deal with the coherence update semantics given in Definition 4.

Briefly, to accomplish the coherence update semantics, we need to generate more insertion instructions in $\mathcal{D}$. This is because, in the coherence update semantics, for computing the update we need to consider the *$\mathcal{T}$-closure* of the original ABox, instead of the ABox itself. For instance, if in our running example we ask for deleting Course(algebra), then, according to the coherence semantics, besides deleting the assertion takesCourse(bob, algebra), we should also apply some insertions for preserving the facts Student(bob) and Person(bob), since both are entailed by the original ontology and do not contradict any update operations.

Thus, in practice, we need to extend our Datalog program $\mathcal{D}$ to:

(1) additionally capture those assertions $A(\overline{o})$ entailed by the TBox and assertions $B(\overline{o})$ that are requested for deletion, and

(2) derive their insertion in case they are not in conflict with respect to the TBox 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 assertion 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 coherence update semantics.

5.1 Encoding in Datalog

According to the coherence semantics we have to preserve all the facts that are originally entailed by the ontology and that do not conflict with the update. For instance, if our running example the update requests for deleting the fact FullProfessor(john), we need to insert Professor(john) because of the assertion FullProfessor $\sqsubseteq$ Professor in $\mathcal{T}$. Clearly, such *closure insertion* would be not necessary in case Professor(john) is already in the ABox, or if there is a request for its insertion, or if it is requested for deletion (either Professor(john) itself or its super-concept Person(john)). By following this intuition, for the positive inclusion assertion FullProfessor $\sqsubseteq$ Professor in $\mathcal{T}$, we need to define the following rules in $\mathcal{D}^*$:

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

We define similar rules for role positive inclusion assertions and concept positive inclusion assertions in which the left-hand side uses the $\exists$ constructor.

Given a *DL-Lite*$_A$ TBox $\mathcal{T}$ we extend the Datalog program $\mathcal{D}$ as follows:

- For each positive inclusion assertion $B \sqsubseteq A$ in $cl_{\mathcal{T}}(\mathcal{T})$, where A is an atomic concept, let A_1, ..., A_m be all the atomic concepts such that $A \sqsubseteq A_i \in cl_{\mathcal{T}}(\mathcal{T})$, then we have in $\mathcal{D}^*$ 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 each positive inclusion assertion $\exists P \sqsubseteq A$ in $cl_{\mathcal{T}}(\mathcal{T})$, where A is an atomic concept, let A_1, ..., A_m be all the atomic concepts such that $A \sqsubseteq A_i \in cl_{\mathcal{T}}(\mathcal{T})$, then we have:

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

- For each positive inclusion assertion $\exists P^- \sqsubseteq A$ in $cl_{\mathcal{T}}(\mathcal{T})$, where A is an atomic concept, let A_1, ..., A_m be all the atomic concepts such that $A \sqsubseteq A_i \in cl_{\mathcal{T}}(\mathcal{T})$, then we have:

```
ins_a_closure(X) :- del_P(Y,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 each positive inclusion assertion $P \sqsubseteq R$ in $cl_{\mathcal{T}}(\mathcal{T})$, where R is an atomic role, let R_1, ..., R_m be all the atomic roles such that $R \sqsubseteq R_i \in cl_{\mathcal{T}}(\mathcal{T})$, and let S_1, ..., S_n be all the atomic roles such that $R \sqsubseteq S_i^- \in cl_{\mathcal{T}}(\mathcal{T})$ then we have:

```
ins_r_closure(X,Y) :- del_p(X,Y), not r(X,Y),
           not ins_r_request(X,Y), not del_r_request(X,Y),
           not del_r1_request(X,Y),..., not del_rm_request(X,Y),
           not del_s1_request(Y,X),..., not del_sn_request(Y,X).
```

– For each positive inclusion assertion $P \sqsubseteq R^-$ in $cl_{\mathcal{T}}(\mathcal{T})$, where R is an atomic role, let R_1, ..., R_m be all the atomic roles such that $R \sqsubseteq R_i \in cl_{\mathcal{T}}(\mathcal{T})$, and let S_1, ..., S_n be all the atomic roles such that $R \sqsubseteq S_i^- \in cl_{\mathcal{T}}(\mathcal{T})$ then we have:

```
ins_r_closure(X,Y) :- del_p(Y,X), not r(X,Y),
           not ins_r_request(X,Y), not del_r_request(X,Y),
           not del_r1_request(X,Y),..., not del_rm_request(X,Y),
           not del_s1_request(Y,X),..., not del_sn_request(Y,X).
```

Once we have defined the predicates *ins_a_closure($\vec{x}$)*, 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, we define in $\mathcal{D}^*$ the following rules:

– For each atomic concept A, let $B_1, \ldots, B_n$ be all the atomic concepts such that $A \sqsubseteq \neg B_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $P_1, \ldots, P_m$ be all the atomic roles such that $A \sqsubseteq \neg\exists P_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, and let $R_1, \ldots, R_w$ be all the atomic roles such that $A \sqsubseteq \neg\exists R_i^-$ is in $cl_{\mathcal{T}}(\mathcal{T})$, then we have:

```
ins_a(X) :- ins_a_closure(X),
       not ins_b1_request(X),..., not ins_bn_request(X),
       not ins_p1_request(X,Y1),..., not ins_pm_request(X,Ym),
       not ins_r1_request(Y1,X),..., not ins_rw_request(Yw,X).
```

– For each atomic role P, let $R_1, \ldots, R_n$ be all the atomic roles such that $P \sqsubseteq \neg R_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $S_1, \ldots, S_m$ be all the atomic roles such that $P \sqsubseteq \neg\exists S_i^-$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $T_1, \ldots, T_v$ be all the atomic roles such that $\exists P \sqsubseteq \neg\exists T_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $Q_1, \ldots, Q_l$ be all the atomic roles such that $\exists P \sqsubseteq \neg\exists Q_i^-$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $W_1, \ldots, W_k$ be all the atomic roles such that $\exists P^- \sqsubseteq \neg\exists W_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $U_1, \ldots, U_h$ be all the atomic roles such that $\exists P^- \sqsubseteq \neg\exists U_i^-$ is in $cl_{\mathcal{T}}(\mathcal{T})$, let $A_1, \ldots, A_j$ be all the atomic concepts such that $\exists P \sqsubseteq \neg A_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, and let $B_1, \ldots, B_s$ be all the atomic concepts such that $\exists P^- \sqsubseteq \neg B_i$ is in $cl_{\mathcal{T}}(\mathcal{T})$, then we have:

```
ins_P(X,Y) :- ins_p_closure(X,Y),
         not ins_r1_request(X,Y),..., not ins_rn_request(X,Y),
         not ins_s1_request(Y,X),..., not ins_sm_request(X,Y),
         not ins_t1_request(X,Y1),..., not ins_tv_request(X,Yv),
         not ins_q1_request(Y1,X),..., not ins_ql_request(Yl,X),
         not ins_w1_request(Y,X1),..., not ins_wk_request(Y,Xk),
         not ins_u1_request(X1,Y),..., not ins_qh_request(Xh,Y),
         not ins_a1_request(X),..., not ins_aj_request(X),
         not ins_b1_request(Y),..., not ins_bs_request(Y),
```

Following the previous example, the derived *closure insertion* of Professor(john) should be applied only if it does not get in conflict with any negative inclusion assertion entailed by the TBox. In the example, such a conflict might arise if there is a request to insert Student(john) because of the negative inclusion assertion Student $\sqsubseteq \neg$Professor in $\mathcal{T}$. So, in $\mathcal{D}^*$ we have:

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

Let us illustrate the construction above with an example.

Example 11 Consider the ontology $\mathcal{O} = \langle \mathcal{T}\, A \rangle$ given in Example 1 and the update request $\mathcal{U} = \{\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john})), \mathsf{d}(\mathsf{Course}(\mathsf{algebra}))\}$ of Example 6. The Datalog program $\mathcal{D}^*$ for computing $\mathcal{O} \bullet \mathcal{U}$ contains the following rules:

```
%ABox assertions
full(john).
takes(bob, algebra).
%Update requests
ins_associate_request(john).
del_course_request(algebra).
...
ins_associate(X) :- ins_associate_request(X), not associate(X).          %r1
del_associate(X) :- del_associate_request(X), associate(X).
incompatible_update() :- ins_associate_request(X), del_associate_request(X).
...
ins_course(X) :- ins_course_request(X), not course(X).
del_course(X) :- del_course_request(X), course(X).
incompatible_update() :- ins_course_request(X), del_course_request(X).
...
del_takes(X,Y) :- takes(X,Y), del_student_request(X).
del_takes(X,Y) :- takes(X,Y), del_course_request(Y).                      %r2
incompatible_update() :- ins_takes_request(X,Y), del_student_request(X).
incompatible_update() :- ins_takes_request(X,Y), del_course_request(Y).
...
del_full(X) :- full(X), ins_associate_request(X).                         %r3
del_associate(X) :- associate(X), ins_full_request(X).
...
ins_prof_closure(X) :- del_full(X), not prof(X),
                       not ins_prof_request(X),
                       not del_prof_request(X),
                       not del_person_request(X).                         %r4
ins_person_closure(X) :- del_full(X), not person(X),
                         not ins_person_request(X),
                         not del_person_request(X).                       %r5
ins_student_closure(X) :- del_takes(X,Y), not student(X),
                          not ins_student_request(X),
                          not del_student_request(X),
                          not del_person_request(X).                      %r6
ins_person_closure(X) :- del_takes(X,Y), not person(X),
                         not ins_person_request(X),
                         not del_person_request(X).                       %r7
ins_prof(x) :- ins_prof_closure(X),  not ins_student_request(X).          %r8
ins_person(x) :- ins_person_closure(X).                                   %r9
ins_prof(x) :- ins_student_closure(X),  not ins_prof_request(X).         %r10
ins_person(x) :- ins_person_closure(X).                                  %r11
```

Similarly to Example 10, since $\mathsf{i}(\mathsf{AssociateProfessor}(\mathsf{john})) \in \mathcal{U}$, the rule `r1` in $\mathcal{D}^*$ generates `ins_associate(john)`, moreover, from rule `r3`, which belongs to $\mathcal{D}^*$ since $\mathsf{FullProfessor} \sqsubseteq \neg\mathsf{AssociateProfessor} \in cl(\mathcal{T})$, we compute `del_full(john)`. At this point, rules `r4` and `r5` lead to compute `ins_prof_closure(john)` and `ins_person_closure(john)`. Then, from rules `r8` and `r9` we compute `ins_prof(john)` and `ins_person(john)`.

Since $\exists\mathsf{takesCourse}^- \sqsubseteq \mathsf{Course}$ in $cl(\mathcal{T})$, the deletion request $\mathsf{d}(\mathsf{Course}(\mathsf{algebra}))$ in $\mathcal{U}$ leads to compute `del_takes(bob,algebra)` via rule `r2`. Then, rules `r6` and `r7` compute

`ins_student_closure(bob)` and `ins_person_closure(bob)`. Finally, from rules `r10` and `r11` we have `ins_student(bob)` and `ins_person(bob)`.

Summarizing, $\mathcal{D}^*$ computes:

```
ins_associate(john).          del_full(john).
ins_prof(john).               ins_person(john).
del_takes(bob,algebra).       ins_student(bob).
ins_person(bob).
```

from which we compute the same ABox of Example 8.

$$\mathcal{A}^*_{new} = \{ \quad \mathsf{AssociateProfessor}(\mathsf{john}),\ \mathsf{Professor}(\mathsf{john}),\ \mathsf{Person}(\mathsf{john}),\ \mathsf{Student}(\mathit{bob}),\ \mathsf{Person}(\mathit{bob}) \quad \}$$

□

5.2 Results

We prove that, given a consistent ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and an update $\mathcal{U}$ compatible with $\mathcal{O}$, the insertion/deletion instructions generated by the Datalog program $\mathcal{D}^*$ lead to compute $\mathcal{O} \bullet \mathcal{U}$. In what follows, we denote by $\mathcal{A}_{\mathcal{D}^*}$ the ABox computed from the ABox $\mathcal{A}$ by adding (resp. removing) an assertion $A(\overline{o})$ if *ins_a($\overline{o}$)* (resp. *del_a($\overline{o}$)*) is true in $\mathcal{D}^*$.

The following theorem shows that the Datalog program $\mathcal{D}^*$ can be used for detecting if an update $\mathcal{U}$ is compatible with a *DL-Lite*$_A$ TBox $\mathcal{T}$ or not.

Theorem 5 *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update, and let $\mathcal{D}^*$ be the corresponding Datalog program defined as above (Section 5.1). $\mathcal{U}$ is not compatible with $\mathcal{T}$ if and only if* incompatible_update() *is true in $\mathcal{D}^*$.*

Proof. We refer to the proof of Theorem 1. Indeed, all the rules specified in the program $\mathcal{D}^*$ for detecting if the update $\mathcal{U}$ is not compatible with the TBox $\mathcal{T}$ are the same as in the Datalog program $\mathcal{D}$ specified for the foundational update semantics. In particular, the new rules in $\mathcal{D}^*$ do not affect any preconditions of the rules involving *incompatible_update()*. ∎

Next we show that $\mathcal{D}^*$ is sound, i.e., that for every ABox assertion $A(\vec{o})$ that should be inserted/deleted according to $\mathcal{D}^*$, $A(\vec{o})$ should be inserted/deleted according to the coherence update semantics.

Theorem 6 (Soundness) *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update compatible with $\mathcal{T}$, and let $\mathcal{D}^*$ be the corresponding Datalog program defined as above (Section 5.1). Then, for each ABox assertion $A(\overline{o})$, if $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(\overline{o})$, then $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(\overline{o})$.*

Proof. We focus on concept assertions, leaving aside role assertions, since the proof of the role assertion cases follows similarly. Thus, on the following we show that for any concept assertion $A(o)$ s.t. $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(o)$, then $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(o)$.

If $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(o)$, then there exists in $\mathcal{A}_{\mathcal{D}^*}$ an assertion $B(o)$ (possibly equals to $A(o)$) such that $\langle \mathcal{T}, \{B(o)\} \rangle \models A(o)$. Given this assertion $B(o) \in \mathcal{A}_{\mathcal{D}^*}$, suppose by contradiction that $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \not\models B(o)$. Now, there are two cases to consider: the case where $B(o)$ belongs

to the original ABox $\mathcal{A}$ and the case where it does not. We are going to show that we reach a contradiction in both cases, and thus, $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models B(o)$, which implies $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(o)$.

First, consider the case where $B(o) \in \mathcal{A}$. Since, by hypothesis, $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \not\models B(o)$, we have one of the following cases: (*i*) $B(o) \in \mathcal{A}^-_{\mathcal{U}}$, therefore `del_b(o)` is true in $\mathcal{D}^*$ because of the rule `del_b(X) :- b(X), del_b_request(X)` in $\mathcal{D}^*$ obtained in translating $\mathcal{U}$; (*ii*) there is in $\mathcal{A}^-_{\mathcal{U}}$ an assertion $C(o)$ and $\mathcal{T} \models B \sqsubseteq C$, which means that because of the rule `del_b(X) :- b(X), del_c_request(X)` in $\mathcal{D}^*$, `del_b(o)` is true in $\mathcal{D}^*$; (*iii*) there exists an assertion $C(o)$ in $\mathcal{A}^+_{\mathcal{U}}$ such that $\langle \mathcal{T}, \{B(o), C(o)\} \rangle$ is inconsistent, that is $\mathcal{T} \models B \sqsubseteq \neg C$ and so the rule `del_b(X) :- b(X), ins_c_request(X)` is in $\mathcal{D}^*$ leads to compute `del_b(o)`. In all such cases `del_b(o)` is true in $\mathcal{D}^*$ and so $B(o)$ cannot belong to $\mathcal{A}_{\mathcal{D}^*}$, contradiction.

Lets now consider the case where $B(o) \notin \mathcal{A}$. Since $B(o) \in \mathcal{A}_{\mathcal{D}^*}$, we know that `ins_b(o)` is true in $\mathcal{D}^*$. This literal can be true because of two cases: *i*) it can be because of the rule `ins_b(X) :- ins_b_request(X), not b(X)` generated when translating $\mathcal{U}$, which can only be true if $B(o)$ is in $\mathcal{A}^+_{\mathcal{U}}$, thus $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models B(o)$, contradiction. *ii*) `ins_b(o)` can be true in $\mathcal{D}^*$ because of a rule of the form `ins_b(X) :- ins_b_closure(X), not ins_c1_request(X), ... ins_cn_request(X)`, where $\mathcal{T} \models B \sqsubseteq \neg C_i$ for each $C_1, ..., C_n$. So, `ins_b(o)` can be true only if the `ins_b_closure(o)` is true, and if there is in $\mathcal{A}^+_{\mathcal{U}}$ no insertion request that, with respect to $\mathcal{T}$, contradicts $B(o)$. If `ins_b_closure(o)` is true, it is because of a rule of the form `ins_b_closure(X) :- del_d(X), not b(X), not ins_b_request(X), not del_b_request(X), not del_b1_request(X),..., not del_bm_request(X)`, where $\mathcal{T} \models D \sqsubseteq B$, $\mathcal{T} \models B \sqsubseteq B_i$ for each $B_1, ..., B_n$, and because `del_d(o)` is true in $\mathcal{D}^*$ and there are no deletion request for facts that are entailed by B together with $\mathcal{T}$. Since `del_d(o)` can be true only if $D(o) \in \mathcal{A}$, it follows that $B(o) \in cl_{\mathcal{T}}(\mathcal{A})$. Thus, since it does not contradict any assertion in $\mathcal{A}^+_{\mathcal{U}}$, and no deletion request in $\mathcal{A}^-_{\mathcal{U}}$ requires its deletion from $cl_{\mathcal{T}}(\mathcal{A})$, it follows that it belongs to the ABox accomplishing the update of $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$, hence $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models B(o)$, contradiction. ∎

Next we show the completeness of $\mathcal{D}^*$ with respect to the coherence update semantics.

Theorem 7 (Completeness) *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_A$ ontology, $\mathcal{U}$ be an update compatible with $\mathcal{T}$, and let $\mathcal{D}^*$ be the corresponding Datalog program defined as above (Section 5.1). Then, for each ABox assertion $A(\overline{o})$, if $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(\overline{o})$ then $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(\overline{o})$.*

Proof. We show that each assertion $A(\overline{o})$ that is entailed by $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U}$ is also entailed by $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle$. There are two cases, the case in which $A(\overline{o})$ is new because of the update ($A(\overline{o})$ is entailed by $\mathcal{A}^+_{\mathcal{U}}$), and the case where $A(\overline{o})$ was already in the initial ontology, and still remains true despite the update.

Consider the case where $A(\overline{o})$ is entailed by $\mathcal{A}^+_{\mathcal{U}}$. Thus, for some $B(\overline{o}) \in \mathcal{A}^+_{\mathcal{U}}$ (possibly equal to $A(\overline{o})$), we have $\langle \mathcal{T}, \{B(\overline{o})\} \rangle \models A(\overline{o})$. By definition of $\mathcal{D}$, for each fact $B(\overline{o})$ in $\mathcal{A}^+_{\mathcal{U}}$, we have that `ins_b(`$\overline{X}$`) :- ins_b_request(`$\overline{X}$`), not b(`$\overline{X}$`)` is in $\mathcal{D}$, so, if $B(\overline{o}) \in \mathcal{A}^+_{\mathcal{U}}$, then $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models B(\overline{o})$, and hence, $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(\overline{o})$.

Now, we show that each assertion that was entailed by the original ABox $\mathcal{A}$ and that is still entailed by $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U}$ is also entailed by $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle$. To this aim we refer again to Algorithm 1. Indeed, since $\mathcal{A} \subseteq cl_{\mathcal{T}}(\mathcal{A})$, the ABox accomplishing the update of $\langle \mathcal{T}, \mathcal{A} \rangle$ with $\mathcal{U}$ and the one accomplishing the update of $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$ preserve the same portion

of $\mathcal{A}$. Hence, for each assertion $A(\overline{o}) \in \mathcal{A}$, $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \not\models A(\overline{o})$ iff $\langle \mathcal{T}, \mathcal{A} \rangle \circ \mathcal{U} \not\models A(\overline{o})$. It follows that if an assertion $A(\overline{o})$ was removed from $\mathcal{A}$, we derived `del_a(o)` in $\mathcal{D}^*$ from the Datalog rules already generated in $\mathcal{D}$ for the foundational update semantics, whose soundness and completeness are already proved. Hence, we have shown that for each $A(\overline{o}) \in \mathcal{A}$, if $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(\overline{o})$ then $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle \models A(\overline{o})$. Let $\mathcal{A}'$ be the ABox accomplishing the update of $\langle \mathcal{T}, cl_{\mathcal{T}}(\mathcal{A}) \rangle$ with $\mathcal{U}$. We only lack to prove that each assertion $A(\overline{o})$ entailed in the original ABox $\mathcal{A}$ but not explicitly contained in $\mathcal{A}'$ (i.e., $A(\overline{o}) \in cl_{\mathcal{T}}(\mathcal{A} \setminus \mathcal{A}')$), and entailed by $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U}$, is still entailed by $\langle \mathcal{T}, \mathcal{A}_{\mathcal{D}^*} \rangle$. If $A(\overline{o}) \in cl_{\mathcal{T}}(\mathcal{A} \setminus \mathcal{A}')$ and $\langle \mathcal{T}, \mathcal{A} \rangle \bullet \mathcal{U} \models A(\overline{o})$, then there is an assertion $B(\overline{o}) \in \mathcal{A} \setminus \mathcal{A}'$ such that $\mathcal{T} \models B \sqsubseteq A$, $Mod(\mathcal{T}, \mathcal{U}^+ \cup \{A(\overline{o})\}) \neq \emptyset$ and $A(\overline{o}) \not\models_{\mathcal{T}} u$ for any $u \in \mathcal{U}^-$. By construction of $\mathcal{D}^*$, we have the rule `ins_a_closure(`$\overline{X}$`) :- del_b(`$\overline{X}$`), not a(`$\overline{X}$`), not ins_a_request(`$\overline{X}$`), not del_a_request(`$\overline{X}$`), not del_a1_request(`$\overline{X}$`),..., not del_am_request(`$\overline{X}$`)`,
where $\mathcal{T} \models A \sqsubseteq A_i$ for each $A_1, ..., A_n$. Therefore, `ins_a_closure(`$\overline{o}$`)` is true in $\mathcal{D}^*$. Moreover, the rule `ins_a(`$\overline{X}$`) :- ins_a_closure(`$\overline{X}$`), not ins_c1_request(`$\overline{X}$`), ... ins_cn_request(`$\overline{X}$`)`, where $\mathcal{T} \models A \sqsubseteq \neg C_i$ for each $C_1, ..., C_n$, in $\mathcal{D}^*$ leads `ins_a(`$\overline{o}$`)` to be true. Thus, $A(\overline{o}) \in \mathcal{A}_{\mathcal{D}^*}$, and so the claim is proved. ∎

Finally, we are ready to show FO-rewritability of the update. It is easy to see that, also in this case, the Datalog program $\mathcal{D}^*$ built for the update is polynomial in the size of the TBox $\mathcal{T}$ and independent of the data in the update $\mathcal{U}$ and in the ABox $\mathcal{A}$. Hence we get the desired result.

Theorem 8 (FO-Rewritability) *The decision problem of the update under the coherence semantics for DL-Lite$_A$ ontologies is FO-rewritable and in* AC^0 *in data complexity.*

Proof. The proof directly follows from Theorem 5, Theorem 6, Theorem 7, and from the fact that evaluation of a FO query over an ABox is in AC^0 in data complexity (Abiteboul et al., 1995). ∎

6. Implementation and Experiments

In this section we illustrate the results of the experiments we carried out to demonstrate the feasibility and scalability of our technique. To this aim, we have implemented a prototype tool developed in Java that uses a MySQL database to store the ABox. Briefly, the Java program takes as input a closed *DL-Lite$_A$* TBox and computes the Datalog program that generates the insertion and deletion instructions necessary to perform the update according to the coherence semantics presented in the previous sections. Moreover, the program translates the Datalog derivation rules into standard SQL queries. Since this process depend on the TBox only, without taking into account the ABox or the update request, all of them are created in compilation time and stored in the MySQL database as views.

To generate the instructions for modifying the ABox that are used to perform the update, the user must (*i*) insert the update request instructions in the *ins_a_request/del_a_request* MySQL tables, and (*ii*) query the contents of the *ins_a/del_a* views.

We focus on the coherence update semantics only for sake of brevity. We chose the coherence semantics since it is computationally harder than the foundational one. Indeed, as shown, the set of insertion/deletion instructions that have to be applied for performing

an update according to the foundational update semantics is a subset of the ones needed to perform the update according to the coherence semantics. So, we have carried out our experiments in the most complicated scenario. On the other hand, we do not expect significant changes in adopting the simplest foundational semantics, being the results for the coherence semantics already quite good.

In the following, we first discuss the design of our experiments, then we present the experimental results, and finally, we close this section with a discussion.

6.1 Experiment Design

The experiment design is composed of three main input elements: a *DL-Lite*$_A$ TBox that we use to define the Datalog derivation rules (i.e., the SQL views computing the insertions/deletions), an ABox (i.e., the database content), and a set of update request instructions. These three elements determine the set of insertions/deletions to apply according to the coherence update semantics.

Given this input, we analyze (*i*) the number of instructions required for applying the update request, (*ii*) the time to generate them, and (*iii*) the time to apply them. We measure these three metrics with respect to the size of the ABox, i.e., the database data. Thus, we have fixed the input ontology, together with the update request, leaving the size of the data to be our variable under study.

In our experiments, we used the LUBM benchmark (Guo et al., 2005). The LUBM ontology describes the university domain and contains 75 ontology predicates and 243 assertions. Since its expressivity goes beyond *DL-Lite*$_A$ (due to the presence of positive inclusion assertions with qualified existential restrictions or concept conjunctions in their left-hand side), we first computed an approximation in *DL-Lite*$_A$ of it, and then, given that the LUBM ontology contains neither negative inclusion assertions nor functionality, we slightly modified the approximated ontology by adding 20 assertions (negative inclusions and functionalities). Our final ontology consists of 195 assertions.

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[6] (i.e., the default data generator provided within the LUBM benchmark) to create data about a single university with an increasing number of connected departments, teachers, etc. We did this because by increasing the number of connected objects in the ABox, the update procedure became more complex when increasing the data size. Indeed, by default, the UBA Data Generator increases the size of the data by creating facts about new universities. However, the data relating to distinct universities are not linked to each other (thus, an update operation on one university does not affect other universities) and, therefore, it is reasonable to expect that the measurement values would barely vary as the data size increased.

Finally, we have specified an update request containing three insertion operations and three deletion operations. As for the insertion operations, we selected three facts to add in a way to ensure several interactions with the TBox assertions and the facts in the ABox, thus, generating several insertions/deletions. Specifically, we choose three instances of the concept Professor and request to add them as instances of the concept Student (Professor

6. http://swat.cse.lehigh.edu/projects/lubm/

Table 1: Experiment Results Table. Time values are given in seconds, while the database size is given in number of tuples.

Database size	Generated instructions	Generation time	Execution time	Total time
$1*10^5$	139	11.98	6.265	18.25
$4.8*10^5$	163	13.23	6.546	19.78
$2*10^6$	218	13.69	6.06	19.75
$2.9*10^6$	213	10.81	5.94	16.75
$5.7*10^6$	242	10.70	6.11	16.81
$1.1*10^7$	321	12.59	7.33	19.92
$1.9*10^7$	463	14.00	5.52	19.52
$3.5*10^7$	479	15.43	6.93	22.36

and Student are specified as disjoint concepts in the ontology). Additionally, we requested to remove three instances of the concept Professor (not related to the inserted ones).

6.2 Experimental Results

We present the results of our experiment in Table 1. In particular, the table shows for each database size (i.e., ABox size), (*i*) the number of insertion/deletion instructions generated by the tool that have to be performed over the database to meet the update request, (*ii*) the time to compute them, (*iii*) the time to execute them, and (*iv*) the total time to compute and execute them. The insertion/deletion queries are run in parallel. All time values are given in seconds.[7]

First, we observe that, although the update requested consists of only 6 update operations, the numbers of insertion/deletion instructions computed by the tool is significantly higher. Indeed, one can see that the update requires 139 insertion/deletion instructions to be accomplished in the smallest database ($1*10^5$ ABox assertions), and 479 in the largest one ($3.5*10^7$ ABox assertions).

The times to generate such instructions range from almost 12 seconds to 15.43 seconds. In addition, we have to consider the execution times of such instructions which range from just under 6 seconds to just over 7 seconds. So, in total, we have times ranging from 18.25 seconds to 22.36 seconds.

We observe that all results seem to grow linearly with respect to the database size. To better illustrate such a trend, we show the same data in the diagrams in Figure 4. Specifically, in the left-hand side diagram, we depict both the time to generate the instructions ($\times$ points) and the total time (+ points), while, in the right-hand side diagram, we show the number of generated instructions ($\times$ points). In both diagrams, the trend lines hint at a linear growth with respect to the database size.

7. Experiments were executed on a MySQL DBMS running on a Windows machine with an Intel Core i7 processor and 8GB of RAM. More details on the experiments can be found at `www.essi.upc.edu/~xoriol/dllitea/`.

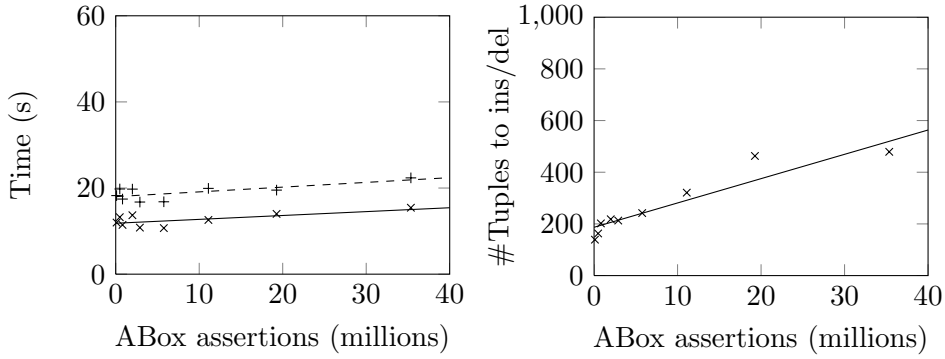

Figure 4: Experiment Results Graphics

6.3 Discussion of the Experimental Results

We now discuss the result of our experiments. The slow increment in the number of generated instructions can be traced back to the fact that, in *DL-Lite*$_A$, an update request only requires modification to those instances that are *connected* to the assertions in the update request. Thus, since ABoxes tend to increase their size by increasing the number of objects, rather than infinitely augmenting the connectivity between them, increasing the ABox size barely increases the number of generated update instructions, as it has been observed in the right-hand side diagram in Figure 4.

One can observe that the time for generating the insertion/deletion instructions take a constant time penalty of about 12 seconds (independently of the size of the database). Indeed, since the LUBM ontology is composed of 75 basic concepts/roles, our method needed to apply 150 queries to compute the insertions and the deletions to apply to every concept/role. Considering that each query requires time to be processed, this phenomenon naturally creates a constant execution time penalty to execute them all. To reduce this issue, we have run the queries in parallel, but since our machine can run a limited number of queries in parallel we still observe the constant penalty cost. These results might be further improved by applying some precompiled optimizations. For instance, we can compute, at compile-time, which are the concepts/roles which might need to be changed when inserting/deleting some facts into other concepts/roles. Thus, at runtime, given an update request, instead of running the 150 queries, we could run only the subset of queries effectively needed to implement the request.

Similarly, the time to execute the instructions, which was about six seconds independently of the size of the database, might be due to the high number of procedures that are executed to update all the 75 concepts/roles in the LUBM ontology. Again, these times can be reduced if, instead of running the procedures to update all the 75 concepts/roles, we just run those over the concepts/roles that need to change. This optimization, as in the previous case, requires precomputing those insertions/deletions in one concept/role that might produce insertions/deletions in another concept/role. Such a computation depends on the TBox only, and thus, is affordable at compilation time.

We would like to stress that we can observe a very low linear increase in the execution time with respect to the database size. In fact, the right-hand side diagram in Figure 4 shows that most of the time consumed is due to the above discussed constant time penalty, which, as shown, might be reduced by applying precompiled optimizations.

Concluding, even without the optimizations (which are per se an interesting subject for future work) the obtained running times are satisfactory. Hence, we believe that our approach can be effectively used in practice for updating ontologies with large ABoxes.

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: compute the rewritings of the specified update in terms of queries characterizing the facts (ABox assertions) to be added and removed, and apply the additions and deletions on the ABox, considered as a database.

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 (Ahmeti et al., 2014). 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 (Ahmeti et al., 2014) in the case of RDFS, but that deal with the more expressive *DL-Lite*$_A$ and OWL 2 QL languages.

Note that here we have dealt with ABoxes and not with external data sources that need to be suitably mapped to the ontology as in (Poggi et al., 2008). Extending our techniques to handling external data sources is indeed possible. Although in that case it becomes of interest not only updating through the ontology as done here, but also reflecting source updates as some form of ontology update, see (De Giacomo et al., 2017).

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 and to consider the TBox itself as (meta) data, in line with the use of SPARQL over OWL 2 QL ontologies (De Carvalho et al., 2017; Cima et al., 2017; Pinto et al., 2019).

Acknowledgments

This work was partly supported by the EU within the H2020 Programme under the grant agreement 834228 (ERC Advanced Grant WhiteMech), the grant agreement 952215 (TAILOR), and the grant agreement 825333 (MOSAICrOWN). Moreover, it is partly supported by Ministerio de Economia, Industria y Competitividad under the grant agreement TIN2017-87610-R (REMEDiAL), by the Generalitat de Catalunya under the grant agreement 2017-SGR-1749, and by Regione Lombardia within the Call Hub Ricerca e Innovazione under the grant agreement 1175328 (WATCHMAN).

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison Wesley Publ. Co.

Ahmeti, A., Calvanese, D., & Polleres, A. (2014). Updating RDFS aboxes and tboxes in SPARQL. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC)*, pp. 441–456.

Ahmeti, A., Calvanese, D., Polleres, A., & Savenkov, V. (2015). Dealing with inconsistencies due to class disjointness in SPARQL update. In *Proc. of DL 2015*, Vol. 1350 of *CEUR*.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Cadoli, M., Donini, F. M., Liberatore, P., & Schaerf, M. (1999). The size of a revised knowledge base. *Artif. Intell.*, *115*(1), 25–64.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., & Rosati, R. (2006). Linking data to ontologies: The description logic *DL-Lite*$_A$. In *Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED)*, Vol. 216 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, *39*(3), 385–429.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2013). Data complexity of query answering in description logics. *Artificial Intelligence*, *195*, 335–360.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., & Rosati, R. (2009). Ontologies and databases: The dl-lite approach. In *Reasoning Web*, Vol. 5689 of *Lecture Notes in Computer Science*, pp. 255–356. Springer.

Calvanese, D., Kharlamov, E., Nutt, W., & Zheleznyakov, D. (2010). Evolution of *DL-Lite* knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC)*, Vol. 6496 of *Lecture Notes in Computer Science*, pp. 112–128. Springer.

Cima, G., De Giacomo, G., Lenzerini, M., & Poggi, A. (2017). On the SPARQL metamodeling semantics entailment regime for OWL 2 QL ontologies. In *Proc. of 7th Int. Conf. on Web Intelligence, Mining and Semantics (WIMS'17)*, pp. 10:1–10:6. ACM.

Console, M., Lembo, D., Santarelli, V., & Savo, D. F. (2014). Graphol: Ontology representation through diagrams. In *Proc. of the 27th Int. Workshop on Description Logic (DL)*, Vol. 1193 of *CEUR Workshop Proceedings*, pp. 483–495. CEUR-WS.org.

De Carvalho, V. A., Almeida, J. P. A., Fonseca, C. M., & Guizzardi, G. (2017). Multi-level ontology-based conceptual modeling. *Data Knowl. Eng.*, *109*, 3–24.

De Giacomo, G., Lembo, D., Oriol, X., Savo, D. F., & Teniente, E. (2017). Practical update management in ontology-based data access. In *Proc. of the 16th Int. Semantic Web Conf. (ISWC'17)*, Vol. 10587 of *Lecture Notes in Computer Science*, pp. 225–242. Springer.

De Giacomo, G., Lenzerini, M., Poggi, A., & Rosati, R. (2009). On instance-level update and erasure in description logic ontologies. *J. of Logic and Computation, Special Issue on Ontology Dynamics*, *19*(5), 745–770.

De Giacomo, G., Oriol, X., Rosati, R., & Savo, D. F. (2016). Updating dl-lite ontologies through first-order queries. In *Proc. of the 15th Int. Semantic Web Conf. (ISWC)*, Vol. 9981 of *Lecture Notes in Computer Science*, pp. 167–183.

Eiter, T., & Gottlob, G. (1992). On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, *57*, 227–270.

Fagin, R., Ullman, J. D., & Vardi, M. Y. (1983). On the semantics of updates in databases. In *Proc. of the 2nd ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS)*, pp. 352–365.

Flouris, G., Konstantinidis, G., Antoniou, G., & Christophides, V. (2013). Formal foundations for RDF/S KB evolution. *Knowl. Inf. Syst.*, *35*(1), 153–191.

Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G. (2008). Ontology change: Classification and survey. *Knowledge Engineering Review*, *23*(2), 117–152.

Flouris, G., & Plexousakis, D. (2005). Handling ontology change: Survey and proposal for a future research direction. Technical report TR-362 FORTH-ICS, Institute of Computer Science, Forth. Greece.

Gärdenfors, P. (1990). The dynamics of belief systems: Foundations vs coherence theories. *Revue Internationale de Philosophie*, *44*(172 (1)), 24–46.

Ginsberg, M. L. (1986). Counterfactuals. *Artificial Intelligence*, *30*(1), 35–79.

Ginsberg, M. L., & Smith, D. E. (1987). Reasoning about action I: A possible worlds approach. Tech. rep. KSL-86-65, Knowledge Systems, AI Laboratory.

Guo, Y., Pan, Z., & Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *J. of Web Semantics*, *3*(2–3), 158–182.

Katsuno, H., & Mendelzon, A. (1991). On the difference between updating a knowledge base and revising it. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pp. 387–394.

Kharlamov, E., Zheleznyakov, D., & Calvanese, D. (2013). Capturing model-based ontology evolution at the instance level: The case of DL-Lite. *J. of Computer and System Sciences*, *79*(6), 835–872.

Lenzerini, M., & Savo, D. F. (2011). On the evolution of the instance level of *DL-Lite* knowledge bases. In *Proc. of the 24th Int. Workshop on Description Logic (DL)*, Vol. 745 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`.

Lenzerini, M., & Savo, D. F. (2012). Updating inconsistent Description Logic knowledge bases. In *Proc. of the 20th Eur. Conf. on Artificial Intelligence (ECAI)*.

Liu, H., Lutz, C., Milicic, M., & Wolter, F. (2006). Updating description logic ABoxes. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pp. 46–56.

Pinto, F. D., Giacomo, G. D., Lembo, D., Lenzerini, M., & Rosati, R. (2019). Acquiring ontology axioms through mappings to data sources. *Future Internet*, *11*(12), 260.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. *J. on Data Semantics*, *X*, 133–173.

Stojanovic, L., Maedche, A., Motik, B., & Stojanovic, N. (2002). User-driven ontology evolution management. In *In Proc. of the 13th Int.l Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pp. 133–140.

Winslett, M. (1990). *Updating Logical Databases*. Cambridge University Press.

Zheleznyakov, D., Kharlamov, E., Nutt, W., & Calvanese, D. (2019). On expansion and contraction of dl-lite knowledge bases. *Journal Web Semantics*, *57*.

Zhuang, Z., Wang, Z., Wang, K., & Qi, G. (2016). DL-Lite Contraction and Revision. *Journal of Artificial Intellicence Research*, *56*, 329–378.