Handling Updates in Ontology-based Data Access

Giuseppe De Giacomo[1], Domenico Lembo[1], Xavier Oriol[2],
Domenico Fabio Savo[1], Ernest Teniente[2]

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

Abstract. Ontology-based Data Access (OBDA) is gaining importance both scientifically and practically. However, little attention has been paid so far to the problem of updating OBDA systems. This is an essential issue if we want to be able to cope with modifications of data both at the ontology and at the source level, while maintaining the independence of the data sources. In this paper, we propose mechanisms to properly handle updates in this context. We show that updating data both at the ontology and source level is first-order rewritable. We also provide a practical implementation of such updating mechanisms based on non-recursive Datalog.

1 Introduction

Ontology Based Data Access (OBDA) is a data integration approach that allows for querying data sources through a unified conceptual view of the application domain, expressed as an ontology [17]. In this way, users may ask queries without being aware of the underlying structure of the data, while considering additional knowledge provided by the ontology. One interesting feature of OBDA is that data sources remain independent and only loosely coupled with the ontology through the use of declarative mappings.

In OBDA, the ontology is usually specified in a lightweight language, like a Description Logic (DL) of the *DL-Lite* family [4]. *DL-Lite* logics have the ability of essentially capturing conceptual models such as UML class diagrams, while being characterized by nice computational properties with respect to query answering. Indeed, this task in *DL-Lite* based OBDA systems is *first-order (FO) rewritable*, which means that any conjunctive query over the ontology (or TBox) can be answered by rewriting it first into a FO-query over a *virtual* set of facts (or ABox), and then into FO-queries over the data sources, by suitably unfolding (traversing backward) the mappings [17].

Little attention has been paid so far in OBDA to the problem of updating, which is the main target of this paper. Namely, we consider "write-also OBDA systems", where a user may change the *extensional level* of the system, in contrast with "read-only OBDA systems", where this service is not provided. We recall that updating a logical theory means changing the old beliefs with new ones, through both addition and removal of pieces of information. This is usually accomplished according to the principle of minimal change, i.e., old information contradicting the new one should be removed in a way that the new theory is as close as possible to the previous one [8, 9, 16, 18, 21].

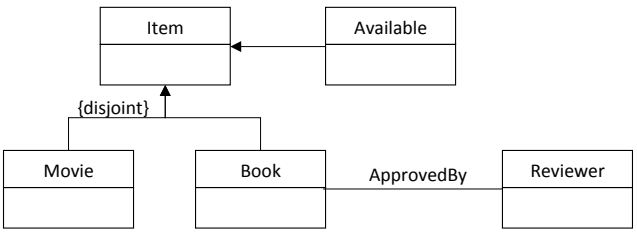

Fig. 1. UML Ontology of a Library

Besides guaranteeing the above behaviour, our goal is to allow users to update the data at the ontology level while maintaining the independence of the data sources. This is in contrast with the traditional way to handle updates in databases, since we should not force the update to propagate to the sources, as done in view updating [10, 11, 19]. Indeed, sources are not under the exclusive control of the ontology, and changing them has a high risk of deeply impacting the contents used by other source clients.

For example, consider the ontology of a library specified as a UML class diagram in Figure 1, where books are approved by reviewers, movies and books are items, some of which are available. Obviously, a movie is not a book, and an item is not a reviewer. Such an ontology can be encoded through the following *DL-Lite* axioms:

$$\texttt{Movie} \sqsubseteq \texttt{Item} \quad \exists\texttt{ApprovedBy} \sqsubseteq \texttt{Book} \quad \exists\texttt{ApprovedBy}^- \sqsubseteq \texttt{Reviewer}$$
$$\texttt{Available} \sqsubseteq \texttt{Item} \quad \texttt{Book} \sqsubseteq \texttt{Item} \quad \texttt{Book} \sqsubseteq \neg\texttt{Movie} \quad \texttt{Item} \sqsubseteq \neg\texttt{Reviewer}$$

Then, consider an external source whose schema contains the relational tables `T_Movie`, `T_Book`, `T_Copy`, `T_Borrow`, `T_RevAuthor`, and `T_Rev`, and link it to the ontology through the mapping below, which we write as Datalog rules, whose heads (resp. bodies) contain only ontology (resp. database) predicates.

```
Movie(x)     :- T_Movie(x)
Book(x)      :- T_Book(x)
Available(x):- T_Copy(x,y), ¬T_Borrow(y,z)
Reviewer(y) :- T_RevAuthor(r,y)
ApprovedBy(x,y):- T_Rev(x,r,z), z>=5, T_RevAuthor(r,y)
```

Let the following set of facts be a database instance at the sources:

```
T_Movie(Alien), T_Book(Ubik), T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)
```

It is not difficult to see that the above mapping and database imply the (virtual) ABox $\{$ `Movie(Alien),Book(Ubik),Available(Ubik)` $\}$. Assume now that we want to insert `Item(Matrix)` and to delete `Available(Ubik)`. Notice that this update does not correspond to any source database update. Indeed, to insert in the database the item 'Matrix', we have to classify it either as a movie or as a book, thus, entailing an unintended fact. The problem is even worse for the case of deleting the availability of 'Ubik', for which we have to either delete the copy 'C2' (and thus deleting an existing copy of the book), or mark it as borrowed by some unknown user of the library (when

no borrowing might exist). Moreover, these (unintended) changes in the database affect the contents used by other database clients, whereas we only want to change some ABox assertions for the users of the OBDA system.

To avoid these situations, we materialize the ABox facts that the user of the OBDA system inserts (resp. deletes) and that are not derived (resp. derived) from the data sources. In this way, the requested updates can always be accomplished without affecting the contents of the sources. This is achieved by materializing the *differences* between the current (virtual) ABox (as generated by the data sources through the mappings) and the one desired by the user. To handle properly these materialized facts, we have to use some special auxiliary *ins/del* relational tables and to suitably extend the mappings. As an example, consider the following new mappings for `Item` and `Available` (which replaces the previous one):

```
Item(x)      :- ins_Item(x)
Available(x):- T_Copy(x,y), ¬T_Borrow(y,z), ¬del_Available(x)
```

Now, we can achieve the previous ontology update by materializing the facts `ins_Item(Matrix)` and `del_Available(Ubik)`.

Let us now consider an update that contradicts previous data. Assume that we want to insert `Book(Alien)`. This contrasts the fact that 'Alien' is already known to be a movie. We manage situations like this through the materialization of additional insertions/deletions that allow us to keep the system consistent, according to a specific minimal change criterion introduced in [7]. In our example, to fully accomplish the update we materialize both `ins_Book(Alien)` and `del_Movie(Alien)`.

There is a further update scenario of interest in write-also OBDA systems. Since the data sources are autonomous, they in turn can be freely changed by their users. Thus we need to deal with two kinds of updates: *ontology-level* and *source-level*. An ontology-level update is posed over the ontology, and is the update we discussed so far. Instead, a source-level update occurs when a data source is modified.

For the source-level case, our framework detects how the update at the sources is reflected, through the mapping, in ABox insertions/deletions, and based on them it computes the additional insertions/deletions that will maintain the system consistent. As we will show, only ABox insertions induced by a source-level update may cause inconsistency, and to repair it we essentially treat them as they were ontology-level updates. Note however that, whereas we can expect ontology-level updates directly specified by users to be coherent with the ontology, i.e., they alone do not violate TBox axioms, which is a classical assumption in update theory, this does not necessarily hold for ABox insertions induced by a source-level update. Consider an update at the sources that inserts the facts `T_Movie(Inferno)` and `T_Book(Inferno)`. This is a legal source-level update, since no constraints are specified on the source database (it can even be possible that tables `T_Movie` and `T_Book` belong to different databases). This source-level update induces two ABox insertions, i.e., `Movie(Inferno)` and `Book(Inferno)`, which together violate the disjointness $\texttt{Book} \sqsubseteq \neg\texttt{Movie}$. To cope with this problem our framework repairs the induced ontology-level update according to a minimality criterion which allows to filter away the conflicting insertions but to maintain their common consistent logical consequences. In our example, this means that both `Movie(Inferno)` and `Book(Inferno)` will be invalidated at the ontology level (i.e., the OBDA system

will not infer them), but their common consequence `Item(Inferno)` will be considered as an ABox insertion induced by the source-level update. We remark that the last form of inconsistency, which we call incoherence, is due to mutually conflicting insertions in the update itself, and has not to be confused with the case when the update is inconsistent with the previous state of the OBDA system, which we discussed before. Dealing with incoherent updates, to the best of our knowledge, has not been studied before. Indeed, as a side contribution, we formalize and study different solutions to this problem.

Finally, we show that both ontology-level and source-level update mechanisms are first-order rewritable. That is, the new contents of the materialized *differences* when an update occurs can be computed by means of first-order queries. This entails that ontology-level and source-level updates are in AC^0 (i.e., sub-polynomial) in data complexity, which is the usual desired complexity for OBDA tasks. We prove this result by computing these updates by means of non-recursive Datalog programs, which can be straightforwardly translated into other (relational-algebra equivalent) languages, such as SQL or SPARQL. Thus, we argue that our framework is not only computationally feasible, but also practically embeddable in current OBDA solutions with existing technology, and without affecting the clients working on the source databases.

The rest of the paper is organized as follows. In Section 2 we provide some preliminaries on ontologies and read-only OBDA systems. In Section 3 we describe how to transform read-only OBDA systems into write-also ones and provide an overview of our techniques to manage both ontology-level and source-level updates. Then, in Section 4 and Section 5 we provide the algorithms to accomplish the two kinds of updates, respectively, and show that both are first-order rewritable. Finally, we conclude the paper in Section 6.

2 Preliminaries

We assume to have three pairwise disjoint, countably infinite alphabets: $\mathsf{N_O}$ for ontology predicates, $\mathsf{N_S}$ for relational predicates, and $\mathsf{N_I}$ for constants. Moreover, we use standard notions for relational databases [1].

Ontologies. A DL ontology $\mathcal{O}$ is pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ is the ABox, providing intensional and extensional knowledge, respectively [2]. Roughly, DL ontologies represent knowledge in terms of concepts, denoting sets of objects, and roles, denoting binary relationships between objects. In this paper we focus on ontologies expressed in *DL-Lite*$_\mathcal{A}$ [17]. A *DL-Lite*$_\mathcal{A}$ TBox is a finite set of axioms of the form $B_1 \sqsubseteq B_2$, $B_1 \sqsubseteq \neg B_2$, $R_1 \sqsubseteq R_2$, $R_1 \sqsubseteq \neg R_2$, and $(\mathsf{funct}\ R)$, where: R, possibly with subscript, is an *atomic role* P, i.e., a binary predicate in $\mathsf{N_O}$, or its inverse P^-; B_i, called *basic concept*, is an *atomic concept* A, i.e., a unary predicate in $\mathsf{N_O}$, or a concept of the form $\exists R$, which denotes the set of objects occurring as first argument of R; $(\mathsf{funct}\ R)$ denotes the functionality of R, which states that its first argument is a key. Suitable restrictions are imposed on the combination of inclusions among roles and functionalities. A *DL-Lite*$_\mathcal{A}$ *ABox* is a finite set of facts of the form $A(c)$ or $P(c, c')$, where $c, c' \in \mathsf{N_I}$.

As for the semantics, we denote with $Mod(\mathcal{O})$ the set of models of $\mathcal{O}$. We say that $\mathcal{O}$ is consistent if $Mod(\mathcal{O}) \neq \emptyset$, inconsistent otherwise, and that an ABox $\mathcal{A}$ is $\mathcal{T}$-consistent if $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent. Moreover, we denote with $\mathcal{O} \models \alpha$ the entailment of a fact or

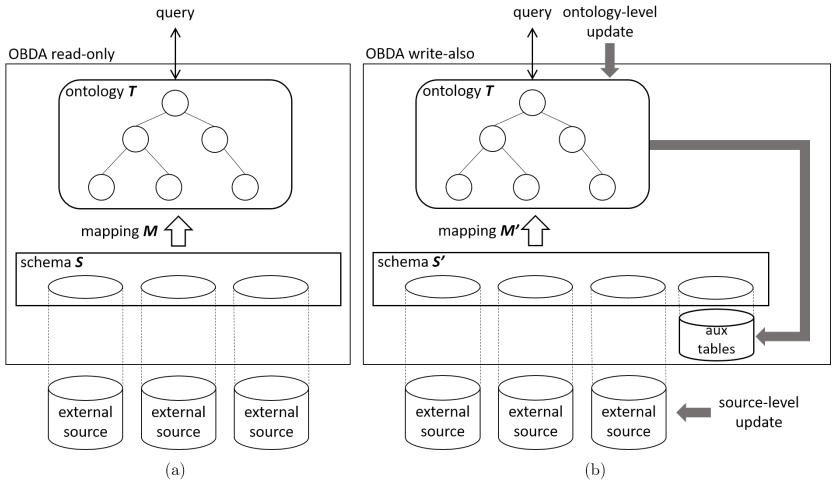

Fig. 2. (a) read-only OBDA architecture (b) write-also OBDA architecture.

axiom α by $\mathcal{O}$, and with $\mathsf{cl}_{\mathcal{T}}(\mathcal{A})$ the ground closure of $\mathcal{A}$, i.e., set of ABox facts α such that $\langle \mathcal{T}, \mathcal{A} \rangle \models \alpha$. We assume that, for each atomic concept or role N, $\mathcal{T} \not\models N \sqsubseteq \neg N$.

Read-only OBDA systems. An *OBDA specification* is a triple $\mathcal{J} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, where $\mathcal{T}$ is a DL TBox, $\mathcal{S}$ is a relational schema, called *source schema*, and $\mathcal{M}$ is a *mapping* between $\mathcal{S}$ and $\mathcal{T}$. As usual in OBDA, we assume $\mathcal{M}$ to be a GAV mapping [14], which we represent as Datalog rules, whose head predicates are from $\mathsf{N_O}$ and body predicates are from $\mathsf{N_S}$. As usual in Datalog we require such rules to be safe [1]. It is easy to see that $\mathcal{M}$, seen as a program, is non-recursive. Note that OBDA specifications of the above form can be considered *read-only*, since they are not specifically thought to be updated, but are usually only queried by users.

An *OBDA system* is a pair $(\mathcal{J}, D)$, where $\mathcal{J} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ is an OBDA specification, and D is a *source database*, i.e., a set of facts for $\mathcal{S}$. A representation of a read-only OBDA system is given in Figure 2(a). The semantics of $(\mathcal{J}, D)$ is given in terms of interpretations of $\mathcal{T}$. To define it, we make use of the *retrieved ABox*, i.e., the set

$$\mathit{ret}(\mathcal{M}, D) = \{N(\boldsymbol{t}) \mid \boldsymbol{t} \in \mathit{eval}(\varphi(\boldsymbol{x}), D) \text{ and } N(\boldsymbol{x})\text{:-}\varphi(\boldsymbol{x}) \in \mathcal{M}\}$$

where N is a concept or role in $\mathsf{N_O}$ and $\mathit{eval}(\varphi(\boldsymbol{x}), D)$ denotes the evaluation of $\varphi(\boldsymbol{x})$, seen as a query, over D. Then, a model of $(\mathcal{J}, D)$ is a model of the ontology $\langle \mathcal{T}, \mathit{ret}(\mathcal{M}, D) \rangle$, and the notions of consistency and entailment introduced before naturally extend to an OBDA system. We point out that in OBDA systems the retrieved ABox is usually not really computed. To emphasize this, we often refer to the retrieved ABox as the *virtual ABox* of an OBDA system.

3 Write-also OBDA Systems

Given a "read-only" OBDA specification $\mathcal{J} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, our framework extends the source schema $\mathcal{S}$ to be able to materialize some ABox insertions/deletions without

affecting the original source database. More in detail, the framework extends the database schema $\mathcal{S}$ to a new schema $\mathcal{S}'$ by considering, for each ontology atomic concept/role N, two additional tables *ins_N* and *del_N*, used to trace insertions/deletions of ABox facts for N[1]. Then, the framework systematically changes the mapping $\mathcal{M}$ into a mapping $\mathcal{M}'$ in the following way:

1. For each atomic concept/role N, add the new mapping assertion $N(\boldsymbol{x})$:- $ins\_N(\boldsymbol{x})$. This guarantees that the instances in ins_N belong to the retrieved ABox as instances of N (i.e., as N facts);
2. Replace each mapping assertion of the form $N(\boldsymbol{x})$:- $\phi(\boldsymbol{x})$, with the mapping assertion $N(\boldsymbol{x})$:- $\phi(\boldsymbol{x}) \wedge \neg del\_N(\boldsymbol{x})$. This avoids the entailment of N facts that are stored as deleted through instances of del_N.

We call $\mathcal{J}' = \langle \mathcal{T}, \mathcal{M}', \mathcal{S}' \rangle$ a *write-also OBDA specification*. It is not difficult to realize that the OBDA specifications $\mathcal{J}$ and $\mathcal{J}'$ are equivalent, in the sense that, when the contents of the new tables *ins_N*/*del_N* are empty, both OBDA specifications have the same retrieved ABox. Thus, this mapping extension preserves the semantics of the original one, but permits modifying the retrieved ABox through the *ins_N*/*del_N* tables without collateral effects. In the following, given a write-also mapping $\mathcal{M}'$, we denote by $\pi(\mathcal{M}')$ the original read-only mapping $\mathcal{M}$.

We now intuitively illustrate how does the framework modifies the contents of the *ins_N*/*del_N* tables for accomplishing ontology-level and source-level updates.

Ontology-level update. An ontology-level update refers to the situation where the update is posed over the ontology. It is intended to change the extensional level of the write-also OBDA system, but without modifying the data at the sources. Thus, it does not change the content of source predicates in the original source schema $\mathcal{S}$. It is accomplished by (1) computing the full set of ontological insertions/deletions that are required to satisfy it in a consistent manner, and (2) realizing the previous set of ontological insertions/deletions. The first step is done through a Datalog program computed at compile time (that is, the Datalog rules are fully determined by the OBDA specification, whereas Datalog facts comes from the user requested update and the current database state of the source schema $\mathcal{S}'$). Such program encodes the update semantics presented in [7], which allows for solving possible inconsistencies between the new beliefs implied by the update and the old ones. Such semantics also allow to preserve logical consequences of the old beliefs that are still consistent with the update. Then, the second step manipulates the ins/del tables accordingly, in order to satisfy the previously computed insertions/deletions. Since such tables are not accessible to data source clients, such update is transparent to them.

Source-level update. A source-level update refers to the situation in which the update is posed over the source database. Such kind of update is always applied to the sources as requested. However, it may have effect at the ontological level, since it is propagated by the mapping. To handle source-level updates, the framework: (1) computes which insertions/deletions of ABox facts are caused by the database update (we call such facts

[1] These tables are typically stored in a different database from those containing actual data, but conceptually are part of $\mathcal{S}'$.

retrieved ABox changes); (2) computes the set of ontological insertions/deletions that are required to accomplish the changes computed previously in a consistent manner; (3) realizes the previous ontological updates. Step (1) is performed through the adaptation of a technique from the literature on view change computation [20]. Step (2), even though similar in principle to Step (1) for ontology-level updates, presents some further complications. Indeed, even though the modification is coherent at the level of the sources, there are no guarantees that it corresponds to a coherent update at the level of the ontology. For instance, a source-level update might cause the insertion of both the facts $C(o)$ and $D(o)$ in the retrieved ABox, whereas the ontology entails that C and D are disjoint. In this situation, our framework adopts a new update semantics suited for dealing with incoherent updates and, according to it, modifies the content of the ins/del tables in order to reflect the proper changes upon the retrieved ABox. Similarly as before, the first two steps are computed through Datalog programs built at compile time.

4 Ontology-level Update

We start with some notions on update over ontologies. Following [5, 7, 15], an *ontology update* $\mathcal{U}$ is a pair of sets of ABox facts $(\mathcal{A}^+_\mathcal{U}, \mathcal{A}^-_\mathcal{U})$, where $\mathcal{A}^+_\mathcal{U}$ are *insertions* and $\mathcal{A}^-_\mathcal{U}$ are *deletions*. We say that an update $\mathcal{U} = (\mathcal{A}^+_\mathcal{U}, \mathcal{A}^-_\mathcal{U})$ is *coherent* with a TBox $\mathcal{T}$ if: (i) $Mod(\langle \mathcal{T}, \mathcal{A}^+_\mathcal{U} \rangle) \neq \emptyset$, i.e., the set of facts we are adding is consistent with $\mathcal{T}$; (ii) $\mathcal{A}^-_\mathcal{U} \cap \mathsf{cl}_\mathcal{T}(\mathcal{A}^+_\mathcal{U}) = \emptyset$, i.e., the update is not asking for deleting and inserting the same knowledge at the same time. Specifically, we define the result of updating an ontology as follows.

Definition 1. *[7] Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_\mathcal{A}$ ontology and let $\mathcal{U} = (\mathcal{A}^+_\mathcal{U}, \mathcal{A}^-_\mathcal{U})$ be an update coherent with $\mathcal{T}$. The result of updating $\mathcal{O}$ with $\mathcal{U}$, denoted by $\mathcal{O} \bullet \mathcal{U}$, is the ABox $\mathcal{A}^\mathcal{U} = \mathcal{A}' \cup \mathcal{A}^+_\mathcal{U}$, where $\mathcal{A}'$ is a maximal subset of the closure $\mathsf{cl}_\mathcal{T}(\mathcal{A})$ such that $\mathcal{A}' \cup \mathcal{A}^+_\mathcal{U}$ is $\mathcal{T}$-consistent, and $\langle \mathcal{T}, \mathcal{A}^\mathcal{U} \rangle \not\models \beta$ for each $\beta \in \mathcal{A}^-_\mathcal{U}$.*

The above update semantics is syntax-independent, consequence conservative, and the ABox resulting from the update operation is, up to logical equivalence, unique [7].

An ontology-level update over a write-also OBDA system $(\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle, D)$ is an update over the ontology $\langle \mathcal{T}, \mathit{ret}(\mathcal{M}, D) \rangle$. To realize the update, we first compute the ABox facts that should be inserted-to/deleted-from the retrieved ABox $\mathit{ret}(\mathcal{M}, D)$, according to Definition 1. Then, we specify the changes to be performed on the ins/del tables from these ABox facts.

For the first task, we make use of a non-recursive Datalog program able to manage updates over *DL-Lite*$_\mathcal{A}$ ontologies, which has been presented in [7]. This program derives the insertions/deletions for a concept/role N as derived literals of the form *ins_N'*$(\boldsymbol{x})$ and *del_N'*$(\boldsymbol{x})$. To do so, the program uses as base facts the current contents of the database D, together with the requested ontology update. That is, the program has a fact *ins_N_ol*$(\boldsymbol{t})$ for each $N(\boldsymbol{t}) \in \mathcal{A}^+_\mathcal{U}$, and *del_N_ol*$(\boldsymbol{t})$ for each $N(\boldsymbol{t}) \in \mathcal{A}^-_\mathcal{U}$. Since the Datalog derivation rules are fully determined by $\mathcal{T}$ and $\mathcal{M}$, we refer to it as *Datalog*$(\mathcal{T}, \mathcal{M})$, and denote the base facts as D+$\mathcal{U}$.

Basically, *Datalog*$(\mathcal{T}, \mathcal{M})$ derives insertions/deletions from the requested update, and computes some extra deletions to avoid violating disjoint/functionality axioms in $\mathcal{T}$,

Algorithm 1: ontology-level-Update($\mathcal{T}$, $\mathcal{M}$, $\mathcal{U}$, D)

```
1 D' ← D
2 foreach fact ins_N'(t) derived by Datalog(T, M) from D+U do
3     if del_N(t) ∈ D then remove del_N(t) from D' else insert ins_N(t) into D'
4 foreach fact del_N'(t) derived by Datalog(T, M) from D+U do
5     if ins_N(t) ∈ D then remove ins_N(t) from D' else insert del_N(t) into D'
6 return D'
```

and some extra insertions to preserve information, according to the update semantics of Definition 1. We illustrate these ideas by showing some of the rules for our example:

```
del_Movie'(x) :- T_Movie(x), del_Item_ol(x)
del_Movie'(x) :- T_Movie(x), ins_Book_ol(x)
ins_Item'(x)  :- del_Movie'(x), ¬del_Item_ol(x)
```

The first rule states that a movie should be deleted if it is deleted as an item. This is required to fully accomplish the deletion since, otherwise, the item would still be implied because of $\texttt{Movie} \sqsubseteq \texttt{Item}$. The second rule implies the deletion of a movie because of the insertion of a book when the movie is in the database, to avoid violating $\texttt{Book} \sqsubseteq \neg\texttt{Movie}$. This reflects the principle that information in the update has to be preferred to the old one, in case of contradiction. The third one entails the insertion of an item when it is deleted as a movie for preserving this entailed belief. This reflects the consequence conservative nature of our update semantics (cf. Definition 1).

Datalog($\mathcal{T}$, $\mathcal{M}$) is sound and complete to compute the ABox modifications required to accomplish an update [7].

Then, we realize these derived insertions/deletions using the ins/del database tables by means of Algorithm 1. Intuitively, the algorithm tries to insert a fact by first removing its deletion from D' (if any). Indeed, this means that the fact is implied by $\pi(\mathcal{M})$ (i.e., the read-only version of the mapping) and D. If there is no deletion of this fact in D, then, it is recorded as an insertion. The case of deletions is analogous. The following result is a consequence of the correctness of Datalog($\mathcal{T}$, $\mathcal{M}$) and Algorithm 1.

Theorem 1. *Let $(\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle, D)$ be a consistent write-also OBDA system, and $\mathcal{U}$ be an update coherent with $\mathcal{T}$. Algorithm 1 computes D' s.t. $\langle \mathcal{T}, ret(\mathcal{M}, D) \rangle \bullet \mathcal{U} = ret(\mathcal{M}, D')$.*

The above theorem says that Algorithm 1 correctly realizes an ontology-level update. Considering the data complexity of non-recursive Datalog, Theorem 1 immediately implies that computing ontology-level updates is in AC^0 in data complexity, i.e., in the size of D+$\mathcal{U}$.

5 Source-level Update

A source level update is a set of update operations, both insertions and deletions, over the source database. We denote it by $\mathcal{U}_{sl}$. The basic idea is to first use the event rules in [20] to compute the changes over the ABox that are induced by $\mathcal{U}_{sl}$.

ABox changes induced by $\mathcal{U}_{sl}$ are of two kinds: insertion and deletion. More formally, let $(\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle, D)$ be a write-also OBDA system, $\mathcal{U}_{sl}$ a source-level update, and D' the database obtained by applying $\mathcal{U}_{sl}$ to D. The *retrieved ABox changes* derived by D, $\mathcal{M}$ and $\mathcal{U}_{sl}$ are represented as a pair $(\mathcal{A}^+, \mathcal{A}^-)$, where $\mathcal{A}^+ = \mathit{ret}(\pi(\mathcal{M}), D') \setminus \mathit{ret}(\pi(\mathcal{M}), D)$, and $\mathcal{A}^- = \mathit{ret}(\pi(\mathcal{M}), D) \setminus \mathit{ret}(\pi(\mathcal{M}), D')$. $\mathcal{A}^+$ and $\mathcal{A}^-$ are called the retrieved ABox insertions and deletions, respectively.

The deletion of ABox facts cannot make the ontology inconsistent. So, when a new ABox deletion is retrieved, we simply check if such deletion was present in the corresponding del table, and if so, we remove it. In this way, we ensure that del tables only contains deletions of facts currently retrieved by $\pi(\mathcal{M})$. The case of retrieved ABox insertions is more complicated, since adding new ABox facts might make the ontology inconsistent. Hence, besides removing from the ins tables the facts corresponding to the new retrieved insertions (if any), we need to deal with possible inconsistencies. This is similar to what happens for ontology-level updates. However, in this case, retrieved ABox insertions might not be coherent with the TBox (i.e. the newly inserted ABox facts alone might directly contradict the TBox). Thus, we need some further machinery to deal with incoherency.

For ease of exposition, in the following we first discuss the simplified setting in which we assume that the retrieved ABox insertions are coherent with the TBox (although not necessarily consistent with the TBox and the virtual retrieved ABox). Then we tackle the full setting, providing a solution for the case in which retrieved ABox insertions may be incoherent (and inconsistent).

5.1 Coherent Source-level Updates

Let $\mathcal{J} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be a write-also OBDA specification, D a database for $\mathcal{S}$, and $\mathcal{U}_{sl}$ a source-level update (thus, involving source predicates but no auxiliary ins/del predicates in $\mathcal{S}$). We proceed as follows: (1) obtain the retrieved ABox changes $(\mathcal{A}^+, \mathcal{A}^-)$ derived by D, $\mathcal{M}$, and $\mathcal{U}_{sl}$; (2) for that part of $(\mathcal{A}^+, \mathcal{A}^-)$ that is already realized through facts in the ins/del tables (due to previous updates) remove the corresponding ins/del facts that become redundant, (3) for the non-redundant part of $\mathcal{A}^+$ proceed as for ontology-level updates to compute the necessary deletions from the current retrieved ABox for preserving the ontology consistency.

The first step can be performed by exploiting a *view change* computation technique. Indeed, each mapping rule can be seen as a *relational view* by considering the head of the rule as a *relational query*. Specifically, we use the technique described in [20], which has been shown to be sound and complete for computing insertions and deletions of *view contents* in the *view change* computation problem for general first-order queries.

The idea of this technique is to materialize the insertion/deletion operations in an update $\mathcal{U}_{sl}$ over the source database in some ad-hoc *ins_T_Table*/*del_T_Table*, and compute the resulting retrieved ABox change $(\mathcal{A}^+, \mathcal{A}^-)$ through a Datalog program: for each $N(\boldsymbol{t})$ fact in $\mathcal{A}^+/\mathcal{A}^-$ the program generates a *ins_N_sl*$(\boldsymbol{t})$/*del_N_sl*$(\boldsymbol{t})$ fact.

For instance, in our running example, we can detect that an item is inserted as available through the following rules:[2]

[2] Unsafe rules in the example can be made easily safe using auxiliary predicates.

```
ins_Avail_sl(x):- ins_T_Copy(x,y), del_T_Borrow(y,z),
                  ¬ins_T_Borrow(y,w), ¬T_Borrowed_pre(y), ¬T_Avail(x)
ins_Avail_sl(x):- ins_T_Copy(x,y), ¬T_Borrow(y,w),
                    ¬ins_T_Borrow(y,z), ¬T_Avail(x)
ins_Avail_sl(x):- T_Copy(x,y), del_T_Borrow(y,z),
                  ¬ins_T_Borrow(y,w), ¬T_Borrowed_pre(y), ¬T_Avail(x)
T_Borrowed_pre(y):- T_Borrow(y,z),¬del_T_Borrow(y,z)
T_Avail(x):- T_Copy(x,y),¬T_Borrow(y,z)
```

The first two rules detect that x is newly available when we insert a new copy of it which is not borrowed anymore, or has never been borrowed, respectively (provided that x was not available according to the original mapping $\mathcal{M}$ before the update). The third rule corresponds to the case that a preexisting copy of the item is no longer borrowed. Deletions are computed using similar rules:

```
del_Avail_sl(x):- del_T_Copy(x,y), ¬T_Borrowed_aux(y),
                  ¬T_Avail_pre(x), ¬ins_Avail_sl(x)
del_Avail_sl(x):- T_Copy(x,y), ¬T_Borrow(y), ins_T_Borrow(y, z),
                  ¬T_Avail_pre(x), ¬ins_Avail_sl(x)
T_Borrowed_aux(y):- T_Borrow(y,z)
T_Avail_pre(x):- T_Copy(x,y), ¬del_T_Copy(x,y), ¬T_Borrowed_pre(y)
```

The first detects that x is no longer available because we have deleted a copy of it that was not borrowed, being this copy the unique one still available, and without adding any other copy nor deleting a borrowing from another one. Similarly, the second detects that x is no longer available because of borrowing the last available copy without inserting new copies nor deleting previous borrowings.

The computed *ins_N_sl*/*del_N_sl* facts are directly derived from the update over the source database and the mapping $\mathcal{M}$. Therefore, if the corresponding *ins_N*/*del_N* facts were already present in the OBDA system due to some previous updates, now there is no need to still keep them. Hence, for the sake of non-redundancy, they must be deleted from D if they were part of it. We notice that in this case, we do not have to take care of inconsistencies that may arise due to the update. Indeed, inconsistencies, if any, have been already solved by the accomplishment of previous updates, which required the insertions of the same facts that now are entailed by the source-level update.

However, *ins_N_sl* facts that do not already have a corresponding *ins_N* (due to previous updates), may lead to inconsistencies when combined with the current retrieved ABox. Indeed, consider the case that our current retrieved ABox contains *Book(Eat)*, and because of a source-level update we have *ins_Movie_sl(Eat)*. Note that *Book(Eat)* is not violating any TBox constraint, neither applying *ins_Movie_sl(Eat)* violates any TBox constraint per se, but the combination of both violates the TBox disjunction assertion between *Book* and *Movie*.

To solve this situation, we have to delete some ABox facts. This deletion is exactly the same we do in the case of ontology-level insertions. Thus, we can compute these extra deletions by directly invoking the ontology-level update algorithm given in Section 4 (Algorithm 1: ontology-level-Update). Note that *del_N_sl* updates cannot lead to inconsistencies, therefore, they can be omitted when invoking the ontology-level-Update.

Algorithm 2: source-level-Update($\mathcal{T}$, $\mathcal{M}$, $\mathcal{U}_{sl}$, D)

$\mathcal{A}^+ \leftarrow \emptyset$
foreach *fact ins_N_sl(**t**) derived* by *Datalog*^{sl}*($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
    if *ins_N(**t**)* $\in D$ **then** remove *ins_N(**t**)* from D **else** include $N(\boldsymbol{t})$ in $\mathcal{A}^+$
foreach *fact del_N_sl(**t**) derived* by *Datalog*^{sl}*($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
    if *del_N(**t**)* $\in D$ **then** remove *del_N(**t**)* from D
$D' =$ apply($\mathcal{U}_{sl}$,D)
return ontology-level-Update($\mathcal{T}$, $\mathcal{M}$, ($\mathcal{A}^+$, {}), D')

All this behavior is formally shown in Algorithm 2. Given a write-also OBDA system $(\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle, D)$, the algorithm takes as input $\mathcal{T}$, $\mathcal{M}$, the requested source-level update $\mathcal{U}_{sl}$ (expressed as ins_T_Table/del_T_Table facts[3]) and D. Also, it makes use of *Datalog*^{sl}, the Datalog program encoding the rules discussed above. In the algorithm, apply($\mathcal{U}_{sl}$,D) indicates the application $\mathcal{U}_{sl}$ to the source database D.

5.2 Incoherent Source-level Update

When the retrieved ABox insertions are not necessarily coherent with the ontology (i.e., they might violate, by themselves, the TBox), we can no longer proceed as done in Section 5.1. In particular, we cannot simply invoke, as in Algorithm 2, the algorithm ontology-level-Update, since this algorithm requires the input update to be coherent.

To cope with the above problem, in the following we consider a new kind of ontology-level update, which we call *weakly-coherent*, and study it. Intuitively, a weakly-coherent update is an ABox update whose insertions might directly contradict the TBox, but that cannot contradict its own deletions. More formally, given a consistent ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and an update $\mathcal{U} = (\mathcal{A}^+_{\mathcal{U}}, \mathcal{A}^-_{\mathcal{U}})$, we say that $\mathcal{U}$ is *weakly-coherent* with $\mathcal{T}$ if $\mathcal{A}^-_{\mathcal{U}} \cap \mathsf{cl}_{\mathcal{T}}(\mathcal{A}^+_{\mathcal{U}}) = \emptyset$. In other terms, differently from coherent updates, in weakly-coherent ones we do not require that $Mod(\langle \mathcal{T}, \mathcal{A}^+_{\mathcal{U}} \rangle) \neq \emptyset$. Note that all updates of the form $(\mathcal{A}^+, \emptyset)$, like the ontology-level updates inferred by source-level ones, which we are analyzing in this section, are always trivially weakly-coherent.

Then, our idea is to introduce a new operator for ontology-level weakly-coherent updates, and show that the result of applying such operator can be easily computed by adapting the previous algorithms and Datalog programs for coherent updates.

To this aim, in the following we in fact present and discuss two new semantics for updating a consistent ontology with a weakly-coherent update. Similar to the update semantics given in Definition 1, these new semantics are consequence conservative, that is, they allow to preserve both coherent consequences of incoherent updates, as well as consistent knowledge inferred by the ontology before an inconsistent update is performed. We will show that the result of the update obtained according to the first semantics that we present always contains the result that we obtain with the second semantics, that is, the former is more conservative than the latter. Thus, we will base our algorithmic solution for incoherent source-level updates on the second semantics.

[3] These rules can be transparently captured through database triggers.

Before proceeding further we need to give some notions. Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ we denote with $HB(\mathcal{O})$ the Herbrand Base of $\mathcal{O}$, i.e. the set of ABox facts that can be built over the ontology alphabet $\mathsf{N_O}$. Moreover, we introduce the notion of *consistent logical consequences* [12] of $\mathcal{A}$ with respect to $\mathcal{T}$ as the set $\mathsf{clc}_\mathcal{T}(\mathcal{A}) = \{\alpha \mid \alpha \in HB(\mathcal{O})$ and there exists $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ is $\mathcal{T}$-consistent, and $\langle \mathcal{T}, \mathcal{A}' \rangle \models \alpha\}$. Note that, if the ontology $\mathcal{A}$ is $\mathcal{T}$-consistent, then $\mathsf{clc}_\mathcal{T}(\mathcal{A}) = \mathsf{cl}_\mathcal{T}(\mathcal{A})$.

The new update semantics we are presenting refer to the notion of *closed ABox repair* [12] of an inconsistent ontology.

Definition 2. *Let $\mathcal{T}$ be a TBox and $\mathcal{A}$ be an ABox. A* closed ABox repair (*CA*-repair) *of $\mathcal{A}$ with respect to $\mathcal{T}$ is a $\mathcal{T}$-consistent ABox $\mathcal{A}'$ such that $\mathsf{cl}_\mathcal{T}(\mathcal{A}')$ is a maximal subset of $\mathsf{clc}_\mathcal{T}(\mathcal{A})$ that is $\mathcal{T}$-consistent.*

The set of all *CA*-repairs of an ABox $\mathcal{A}$ with respect to $\mathcal{T}$ is denoted by $\mathit{carSet}_\mathcal{T}(\mathcal{A})$.

Example 1. Consider the TBox $\mathcal{T}$ of our running example and the following ABox:

$$\mathcal{A}_{inc} = \{\texttt{Movie(Moon)}, \texttt{ApprovedBy(Moon,Pit)}\}.$$

It is easy to see that the ABox $\mathcal{A}_{inc}$ is not $\mathcal{T}$-consistent, since both `Movie(Moon)` and `Book(Moon)` follows from $\mathcal{T}$ and $\mathcal{A}_{inc}$. The set $\mathit{carSet}_\mathcal{T}(\mathcal{A}_{inc})$ contains the following $\mathcal{T}$-consistent ABoxes:

$\mathcal{A}_{r1} = \{\texttt{Movie(Moon)}, \texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$;
$\mathcal{A}_{r2} = \{\texttt{Book(Moon)}, \texttt{ApprovedBy(Moon,Pit)}, \texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$. □

Intuitively, our first solution for updating an ontology with a weakly-coherent update consists in first restoring the consistency of the update with respect to the TBox, and then proceeding as in the case of coherent update. Since, given an update $\mathcal{U}$ and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, there may exist more then one repair of $\mathcal{A}^+_\mathcal{U}$ with respect to $\mathcal{T}$, we compute a single update by taking the intersection of all the *CA*-repairs of $\mathcal{A}^+_\mathcal{U}$ with respect to $\mathcal{T}$, thus following the *When In Doubt Throw It Out* (WIDTIO) principle [21].

Definition 3. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_\mathcal{A}$ ontology, and let $\mathcal{U}$ be a weakly-coherent update. The operator $\bullet_1$ is the update operator such that $\mathcal{O} \bullet_1 \mathcal{U} = \mathcal{O} \bullet \mathcal{U}_{rep}$, where $\mathcal{U}_{rep} = (\bigcap_{\mathcal{A}^r_i \in \mathit{carSet}_\mathcal{T}(\mathcal{A}^+_\mathcal{U})} \mathsf{cl}_\mathcal{T}(\mathcal{A}^r_i), \mathcal{A}^-_\mathcal{U})$.*

We note that $\mathcal{U}_{rep}$ actually coincides with the repair of $\mathcal{A}^+_\mathcal{U}$ with respect to $\mathcal{T}$ under the *ICAR semantics* presented in [12].

Example 2. Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite$_\mathcal{A}$* ontology where $\mathcal{T}$ is the TBox of our running example and $\mathcal{A}$ is the ABox $\{\texttt{Movie(Moon)}\}$. Moreover, let $\mathcal{U}$ be the weakly-coherent update $(\mathcal{A}_{inc}, \{\})$, where $\mathcal{A}_{inc}$ is as in Example 1. It is easy to see that $\mathcal{U}_{rep} = \mathsf{cl}_\mathcal{T}(\mathcal{A}_{r1} \cap \mathcal{A}_{r2}) = \{\texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$. Consequently, $\mathcal{O} \bullet_1 \mathcal{U} = \mathcal{O} \bullet \mathcal{U}_{rep} = \{\texttt{Movie(Moon)}, \texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$. □

The second update semantics follows a different approach. Instead of computing a coherent update by performing the intersection of all the repairs of the original weakly-coherent update and then using it for updating the ontology as described in Section 4, we first update the ontology with each repair separately, and then we apply the WIDTIO principle in order to have a single ABox as result.

Definition 4. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_\mathcal{A}$ ontology, and let $\mathcal{U}$ be a weakly-coherent update. The operator $\bullet_2$ is the update operator such that $\mathcal{O} \bullet_2 \mathcal{U} = \langle \mathcal{T}, \mathcal{A}_\cap \rangle$ where $\mathcal{A}_\cap = \bigcap_{\mathcal{A}_i^r \in carSet_\mathcal{T}(\mathcal{A}_\mathcal{U}^+)} \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet (\mathcal{A}_i^r, \mathcal{A}_\mathcal{U}^-))$.*

Example 3. Consider the ontology $\mathcal{O}$ and the update $\mathcal{U}$ of Example 2. The update semantics given in Definition 4 requires, for each repair $\mathcal{A}_{ri}$ of $\mathcal{A}_{inc}$ with respect to $\mathcal{T}$, to compute $\mathcal{O} \bullet \mathcal{A}_{ri}$. Easily, one can see that:

$\mathcal{O} \bullet \mathcal{A}_{r1} = \{\texttt{Movie(Moon)}, \texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$
$\mathcal{O} \bullet \mathcal{A}_{r2} = \{\texttt{Book(Moon)}, \texttt{ApprovedBy(Moon,Pit)}, \texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$.

Hence, we have that $\langle \mathcal{T}, \mathcal{A} \rangle \bullet_2 \mathcal{U} = \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet \mathcal{A}_{r1}) \cap \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet \mathcal{A}_{r2}) = \{\texttt{Reviewer(Pit)}, \texttt{Item(Moon)}\}$. □

The following result determines the relation between the above update semantics.

Theorem 2. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent DL-Lite$_\mathcal{A}$ ontology, and $\mathcal{U}$ be an update possibly inconsistent with $\mathcal{T}$. Then $\mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_2 \mathcal{U}) \subseteq \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_1 \mathcal{U})$.*

Proof. Let $\mathcal{A}^\cap = \bigcap_{\mathcal{A}_i^r \in carSet_\mathcal{T}(\mathcal{A}_\mathcal{U}^+)} \mathsf{cl}_\mathcal{T}(\mathcal{A}_i^r)$. Toward a contradiction, assume that $\mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_2 \mathcal{U}) \not\subseteq \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_1 \mathcal{U})$. This means that there is at least one ABox assertion $\alpha \in \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_2 \mathcal{U})$ such that $\alpha \notin \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_1 \mathcal{U})$. Only two cases are conceivable.

First case: $\mathcal{O} \models \alpha$. Since $\alpha \notin \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_1 \mathcal{U})$, then there is an assertion $\beta \in \mathsf{cl}_\mathcal{T}(\mathcal{A}^\cap)$ such that $\langle \mathcal{T}, \{\beta\} \rangle \models \neg\alpha$. Since for each $\mathcal{A}_i^r \in carSet_\mathcal{T}(\mathcal{A}_\mathcal{U}^+)$ we have that $\mathcal{A}^\cap \subseteq \mathcal{A}_i^r$, then $\beta \in \mathsf{cl}_\mathcal{T}(\mathcal{A}_i^r)$. This means that for each ABox $\mathcal{A}_i^{new} = \mathcal{O} \bullet (\mathcal{A}_i^r, \mathcal{A}^-)$, $\beta \in \mathsf{cl}_\mathcal{T}(\mathcal{A}_i^{new})$. Therefore $\beta \in \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_2 \mathcal{U})$, and $\langle \mathcal{T}, \mathcal{O} \bullet_2 \mathcal{U} \rangle \models \neg\alpha$ which is a contradiction.

Second case: $\mathcal{O} \not\models \alpha$. Since $\alpha \in \mathsf{cl}_\mathcal{T}(\mathcal{O} \bullet_2 \mathcal{U})$, then for each $\mathcal{A}_i^r \in carSet_\mathcal{T}(\mathcal{A}_\mathcal{U}^+)$, and for each $\mathcal{A}_i^{new} = \mathcal{O} \bullet (\mathcal{A}_i^r, \mathcal{A}^-)$, $\alpha \in \mathsf{cl}_\mathcal{T}(\mathcal{A}_i^{new})$. Since $\mathcal{O} \not\models \alpha$, then for each $\mathcal{A}_i^r \in carSet_\mathcal{T}(\mathcal{A}_\mathcal{U}^+)$, $\alpha \in \mathcal{A}_i^r$. Hence, $\alpha \in \mathsf{cl}_\mathcal{T}(\mathcal{A}^\cap)$ and so $\langle \mathcal{T}, \mathcal{O} \bullet_1 \mathcal{U} \rangle \models \alpha$ which is a contradiction. □

Interestingly, the converse is not true (cf. Examples 2 and 3). As a consequence, we see that the first semantics is more *conservative* then the second. For this reason (and for lack of space), in the rest of this paper we focus on the first semantics and leave the study of the second for future work.

We now turn back to the management of the case in which the ontology update implied by a source-level update is incoherent. To this aim, we modify step (2) described in Section 5.1. In particular, in step (2) we now identify the part of the update that is coherent with the TBox, which has to be realized as before. Also, we repair the remaining part (i.e., the incoherent one) according to Definition 3, that is, by deriving the deletion of all incoherent inserted facts and the insertion of all their coherent consequences. Again, all these computations can be done with a non-recursive Datalog program.

We note that retrieved ABox deletions are always coherent since they cannot contradict the TBox, but an insertion is coherent only if it is not paired to an insertion in a disjoint predicate, or if there is no other insertion that together with it violates a functional role. To compute this we make use of suitable Datalog rules. Namely, for each atomic concept A we pose:

```
ins_A_coherent(x) :- ins_A_sl(x), ¬ins_A1_sl(x), .., ¬ins_An_sl(x),
                                  ¬ins_P1_sl(x, y1), .., ¬ins_Pm_sl(x, ym),
                                  ¬ins_Q1_sl(z1, x), .., ¬ins_Qk_sl(zk, x)
```

where each A_i is an atomic concept such that $\mathcal{T} \models A \sqsubseteq \neg A_i$, each P_i is an atomic role such that $\mathcal{T} \models A \sqsubseteq \neg\exists P_i$, and each Q_i is an atomic role such that $\mathcal{T} \models A \sqsubseteq \neg\exists Q_i^-$. We proceed similarly for roles. In this case however, besides disjointnesses, we have also to consider that a role R can be involved in functionality axioms or can be asymmetric, i.e., R is such that $\mathcal{T} \models R \sqsubseteq \neg R^-$. Assuming R functional and not involved in any disjointness (both between concepts and relations), we write the following rules to deal with insertions in R:

```
ins_R_coherent(x, y) :- ins_R_sl(x, y), ¬clash_R(x)
clash_R(x) :- ins_R_sl(x, y), ins_R_sl(x, z), y ≠ z
```

Note that the above rules are similar in spirit to those used in [13] for query rewriting.

Next, we deal with the rest of *ins_N_sl*, i.e., those that directly contradict a TBox axiom. For each one of them, we obtain the additional insertions/deletions that must be effectively performed, according to Definition 3, for both solving incoherency and preserving consistent consequences. In explaining this step we consider only inclusions and disjointnesses between atomic concepts. Other forms of axioms are dealt with in a similar way.

We consider two kinds of Datalog rules. The first kind computes the insertions (coherent or not) entailed by insertions clashing with the TBox. That is, for each pair of TBox axioms of the form $A_1 \sqsubseteq A_2$, $A_1 \sqsubseteq \neg A_3$ entailed by $\mathcal{T}$ we have the rule:

```
ins_A2_closure(x) :- ins_A1_sl(x), ins_A3_sl(x)
```

The second kind of rules filters these new insertions to apply only those not contradicting the TBox. Concretely, for each atomic concept A, we consider a Datalog rule with the form:

```
ins_A_ol(x) :- ins_A_closure(x), ¬ins_A1_sl(x), .., ¬ins_An_sl(x)
```

where each A_i is an atomic concept such that $\mathcal{T} \models A \sqsubseteq \neg A_i$.

Note that we derive a new ontology-level insertion. Indeed, we use such new insertions to invoke the ontology-level-Update algorithm, which will insert these new facts while deleting those currently retrieved ABox facts that clashes with it, so, ensuring the consistency of the ontology. This ontology-level update invocation is performed after applying the source-level update in D, that is, after inserting/deleting each tuple in the *ins_T_Table*/*del_T_Table* tables in/from the corresponding *T_Table*.

Finally, we must avoid entailing a clash because of the insertions in the database. Thus, for each $A_1 \sqsubseteq \neg A_2$ assertion entailed by the TBox, where each A_i is a basic concept/role, we consider the rules:

```
del_A1'(x̄) :- ins_A1_sl(x̄), ins_A2_sl(x̄)
del_A2'(x̄) :- ins_A1_sl(x̄), ins_A2_sl(x̄)
```

Intuitively, these rules are only meant to *cancel* the insertions that cause the clash. The entire general procedure is described by Algorithm 3. Notice that by removing rows

Algorithm 3: source-level-Update($\mathcal{T}$, $\mathcal{M}$, $\mathcal{U}_{sl}$, D)

1 $\mathcal{A}^+ \leftarrow \emptyset$
2 **foreach** *fact ins_N_coherent($\boldsymbol{t}$) derived* by *Datalog^{sl}($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
3 **if** *ins_N($\boldsymbol{t}$)* $\in D$ **then** remove *ins_N($\boldsymbol{t}$)* from D **else** include $N(\boldsymbol{t})$ in $\mathcal{A}^+$
4 **foreach** *fact del_N_sl($\boldsymbol{t}$) derived* by *Datalog^{sl}($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
5 **if** *del_N($\boldsymbol{t}$)* $\in D$ **then** remove *del_N($\boldsymbol{t}$)* from D
6 // Dealing with incoherent insertions
7 **foreach** *fact ins_N_ol derived* by *Datalog^{sl}($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
8 include $N(\boldsymbol{t})$ in $\mathcal{A}^+$
9 **foreach** *fact del_N'($\boldsymbol{t}$) derived* by *Datalog^{sl}($\mathcal{T}$, $\mathcal{M}$)* from $\mathcal{U}_{sl} + D$ **do**
10 **if** *ins_N($\boldsymbol{t}$)* $\in D$ **then** remove *ins_N($\boldsymbol{t}$)* from D
11 **else** insert *del_N($\boldsymbol{t}$)* into D
12 $D' =$ apply($\mathcal{U}_{sl}$, D)
13 **return** ontology-level-Update($\mathcal{T}$, $\mathcal{M}$, ($\mathcal{A}^+$, {}), D')

6-11, this algorithm is exactly as Algorithm 2, with the proviso that in line 2 we are using in_N_coherent in place of ins_N_sl. Indeed, in the general setting we have to add the treatment of facts ins_N_ol, and del_N', has produced by the new version of the program *Datalog^{sl}($\mathcal{T}$, $\mathcal{M}$)*. We conclude by stating the correctness of the algorithm.

Theorem 3. *Let $(\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle, D)$ be a consistent write-also OBDA system, $\mathcal{U}_{sl}$ an update over D, and $\mathcal{A}^{ret} = (\mathcal{A}^+, \mathcal{A}^-)$ be the retrieved ABox change derived by D, $\pi(\mathcal{M})$, and $\mathcal{U}_{sl}$. Algorithm 3 returns a D' such that $\langle \mathcal{T}, ret(\mathcal{M}, D) \setminus \mathcal{A}^- \rangle \bullet_1 (\mathcal{A}^+, \emptyset) = ret(\mathcal{M}, D')$.*

Intuitively, the retrieved ABox computed from D', in turn obtained by Algorithm 3, is equivalent to realizing the ontology-level update $(\mathcal{A}^+, \emptyset)$ over the ontology $\langle \mathcal{T}, ret(\mathcal{M}, D) \setminus \mathcal{A}^- \rangle$, i.e., over the original retrieved ABox after deleting $\mathcal{A}^-$.

From this theorem we get that computing the result of a source-level update is in AC^0 in data complexity as for ontology-level update.

6 Conclusion

In this paper we have studied write-also OBDA Systems under *ontology-level* and *source-level* updates. We have shown how to handle both updates through non-recursive Datalog programs. Such programs can be easily translated into first-order query languages, and thus we have shown that update computation in our framework is first-order rewritable. We stress that the techniques proposed in this paper are ready-implementable and can be adopted by state-of-the-art tools for OBDA, such as Mastro [6] and Ontop [3]. This will be the subject of our future work.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.
3. D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web J.*, 8(3):471–487, 2017.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
5. D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Evolution of *DL-Lite* knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC)*, volume 6496 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2010.
6. C. Civili, M. Console, G. De Giacomo, D. Lembo, M. Lenzerini, L. Lepore, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, V. Santarelli, and D. F. Savo. MASTRO STUDIO: managing ontology-based data access applications. *Proc. of the VLDB Endowment*, 6(12):1314–1317, 2013.
7. G. De Giacomo, X. Oriol, R. Rosati, and D. F. Savo. Updating DL-Lite ontologies through first-order queries. In *Proc. of the 15th Int. Semantic Web Conf. (ISWC)*, volume 9981, pages 167–183, 2016.
8. T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.
9. G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: Classification and survey. *Knowledge Engineering Review*, 23(2):117–152, 2008.
10. A. Guessoum and J. W. Lloyd. Updating knowledge bases. *New Generation Comput.*, 8(1):71–89, 1990.
11. A. C. Kakas and P. Mancarella. Database updates through abduction. In *Proc. of the 16th Int. Conf. on Very Large Data Bases (VLDB)*, pages 650–661, 1990.
12. D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR)*, 2010.
13. D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Query rewriting for inconsistent *DL-Lite* ontologies. In *Proc. of the 5th Int. Conf. on Web Reasoning and Rule Systems (RR)*, 2011.
14. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.
15. M. Lenzerini and D. F. Savo. Updating inconsistent Description Logic knowledge bases. In *Proc. of the 20th Eur. Conf. on Artificial Intelligence (ECAI)*, 2012.
16. A. Olivé. Integrity constraints checking in deductive databases. In *Proc. of the 17th Int. Conf. on Very Large Data Bases (VLDB)*, pages 513–523, 1991.
17. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
18. R. Reiter. On specifying database updates. *J. of Logic Programming*, 25(1):53–91, 1995.
19. E. Teniente and A. Olivé. Updating knowledge bases while maintaining their consistency. *Very Large Database J.*, 4(2):193–241, 1995.
20. T. Urpí and A. Olivé. A method for change computation in deductive databases. In *Proc. of the 18th Int. Conf. on Very Large Data Bases (VLDB)*, pages 225–237, 1992.
21. M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.