Towards SPARQL instance-level Update in the Presence of OWL-DL TBoxes

Claudia Schon [a,1], Steffen Staab [a,b]
[a] *Institute for Web Science and Technologies, University of Koblenz-Landau, Koblenz, Germany*
[b] *Web and Internet Science Research Group, University of Southampton, Southampton, UK*

Abstract. Represented knowledge is subject to frequent changes. To meet these requirements for dynamics, SPARQL update, an update language for RDF graphs, was developed. Even though there is some research on semantics of SPARQL ABox update for RDFS ontologies and approaches addressing updates in interplay with rather restricted TBoxes languages, up till now there is no definition of semantics for SPARQL updates for OWL knowledge bases. In this paper, we define a SPARQL update semantics for OWL-DL ABoxes and show how existing methods like justification-based explanation can be used to conduct SPARQL updates in the presence of OWL-DL TBoxes. We argue that the interplay of the deletion and the insertion task of SPARQL update queries renders it not advisable to consider these tasks separately. This is why we introduce query-driven semantics aiming at optimizing the effect of the whole update operation.

Keywords. SPARQL update, OWL, Justification

1. Introduction

OWL [16] is a family of languages which are broadly used for knowledge representation purposes. It is based on description logics (DL) [4] and offers versatile means to model terminological knowledge as well as knowledge about data. For knowledge represented in RDF [21], SPARQL [22] provides possibilities to query this knowledge. Similarly, the SPARQL-DL [23] query language, a distinct subset of the SPARQL query language, renders it possible to query OWL knowledge bases. A SPARQL-DL interface is included in every OWL API 3 reasoner which allows these reasoners to answer SPARQL-DL queries.

Besides querying represented knowledge, the management of changes constitutes an interesting challenge. Nearly all represented knowledge is subject to frequent changes and even the construction of a knowledge base can be seen as an evolutionary development. To meet these requirements for dynamics, SPARQL update [9], an update language for RDF graphs, was developed. Even though

[1]Work supported by DFG EVOWIPE.

there is some research on semantics of SPARQL update for RDFS ontologies [1,13] and approaches addressing updates in interplay with rather restricted TBoxes languages [2], up till now there is no definition of semantics for SPARQL updates for OWL knowledge bases. We define SPARQL update semantics for OWL-DL ABoxes and show how existing methods like justification-based explanation can be used to conduct SPARQL updates in the presence of OWL-DL TBoxes.

2. Running Example

We consider the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, given in description logic syntax,

$$\begin{aligned}
\mathcal{T} = \{&\mathit{SoftwareEngineer} \sqsubseteq \mathit{Employee}, && (1)\\
&\mathit{Secretary} \sqsubseteq \mathit{Employee}, && (2)\\
&\mathit{Student} \sqcap \mathit{Employee} \sqsubseteq \mathit{StudentTrainee}, && (3)\\
&\exists \mathit{attendsCourse}.\top \sqsubseteq \mathit{Student}\} && (4)\\
\mathcal{A} = \{&\mathit{SoftwareEngineer}(\mathit{john}), && (5)\\
&\mathit{Employee}(\mathit{john}), && (6)\\
&\mathit{StudentTrainee}(\mathit{john}), && (7)\\
&\mathit{Student}(\mathit{john})\} && (8)
\end{aligned}$$

with a SPARQL update which removes the status of being a *StudentTrainee* from all students who are employees and assigns them to some special course $c1$.

```
INSERT {?X :attendsCourse :c1}
DELETE {?X a :StudentTrainee}
WHERE {?X a :Student, ?X a :Employee}
```

The update is supposed to be incorporated into the ABox by removing and adding as few assertions as possible. According to [9], the overall processing model is to first evaluate the pattern in the `WHERE` clause, then to perform the deletion and after that the insertion. In order to use a uniform notation, we stick to description logic syntax and represent the triples which are supposed to be deleted or inserted as ABox assertions. W.r.t. the query presented above, the task is to delete $\mathit{StudentTrainee}(\mathit{john})$ and to insert $\mathit{attendsCourse}(\mathit{john}, c1)$. To accomplish the deletion, we have to find a minimal set of ABox assertions whose deletion prevents $\mathit{StudentTrainee}(\mathit{john})$ from being entailed by the knowledge base. This can be done by removing one of the following two sets

$$\{\mathit{StudentTrainee}(\mathit{john}), \mathit{Student}(\mathit{john})\}$$

$$\{\mathit{StudentTrainee}(\mathit{john}), \mathit{Employee}(\mathit{john}), \mathit{SoftwareEngineer}(\mathit{john})\}$$

from the ABox. Note that both these possibilities constitute a minimal way to delete $StudentTrainee(john)$ since no proper subset implements the deletion. For example the knowledge base resulting from removing only $StudentTrainee(john)$ from the ABox would still entail $StudentTrainee(john)$ since $Student(john)$, $Employee(john)$ together with the TBox axiom (3) imply $StudentTrainee(john)$. In general, when considering knowledge bases with expressive TBoxes, there are many possibilities to accomplish a deletion. Determining which assertions to delete leads to different semantics for SPARQL update delete. Sec. 6 introduces some possible choices. Considering the insertion part of the example SPARQL update query, the task is to insert $attendCourse(john, c1)$ into the ABox. Taking a closer look at the knowledge base, the last axiom in the TBox reveals that inserting $attendCourse(john, c1)$ implies $Student(john)$ which directly contradicts the first possibility to delete $StudentTrainee(john)$ mentioned above. This illustrates, that it is not advisable to consider insertion and deletion separately as this might have undesired effects. To remedy this situation, in Sec. 6 we introduce *query-driven* semantics taking into account the whole query.

Inserting new assertions into an ABox might introduce inconsistencies. In this case, debugging techniques can be used to remove the inconsistency. Choosing one of the possibly many ways to resolve an inconsistency can be done by choosing one of the semantics we introduce in Sec. 6.2.

3. Background and Related Work

3.1. Related Work

The problem of belief revision and updating knowledge bases has received much attention in research [3,12]. [11] introduces a kernel semi-revision operator for belief bases in OWL-DL. The basic idea of this operator is to add the new information to the knowledge base, then compute all justifications for possible inconsistencies and remove one element from each of these justifications. Since in the last step the previously introduced information might be removed, it is not guaranteed that the inserted information is contained in the result. In contrast to this, some of the semantics we introduce guarantee that the result contains the inserted information whenever this is possible.

In [5], the prioritized assertional-based revision of DL-Lite knowledge bases is addressed. In their setting, it is assumed that the data is associated with priority or credibility information, possibly originating from the combination of data from different sources where each source has a certain reliability level.

Instance-level insertion and deletion for DL-Lite$_{\mathcal{F}}$, a tractable extension of DL-Lite which is oriented towards data intensive applications, is investigated in [6]. Since all reasoning tasks in DL-Lite$_{\mathcal{F}}$ are tractable, approaches developed for this logic cannot be expected to be suitable for OWL-DL.

With respect to SPARQL update, [1] discusses possible semantics for SPARQL update in the presence of DL-Lite TBoxes. Due to the low expressivity of DL-Lite, inconsistencies are not an issue. [2] addresses the problem of handling inconsistencies due to class disjointness introduced by SPARQL updates. The approach is restricted to a DL-Lite fragment called $DL-Lite_{RDFS_\neg}$ covering RDFS as well as concept disjointness axioms. Different semantics of SPARQL ABox updates are defined similar to the notion of the inconsistency tolerant semantics introduced in [15]. [2] use skillful query-rewriting to determine and resolve inconsistencies. These rewritings exploit the fact that in $DL-Lite_{RDFS_\neg}$ inconsistencies are caused by at most two ABox assertions. In contrast to this, inconsistencies in OWL-DL can be caused by an arbitrary number of ABox assertions.

[18] presents an approach for interactive ontology revision. User interaction is used to decide if an axiom should be accepted or rejected. This is combined with automated reasoning to ensure consistency.

3.2. Further Background

We introduce syntax and semantics of the description logic $\mathcal{SHOIN}$ which corresponds to OWL-DL. Given a set of *atomic roles* N_R, the set of *roles* is defined as $N_R \cup \{R^- \mid R \in N_R\}$, where R^- denotes the *inverse role* corresponding to the atomic role R. A *role inclusion axiom* is an expression of the form $R \sqsubseteq S$, where R and S are atomic or inverse roles. A *transitivity axiom* is of the form $\mathit{Trans}(S)$ for S an atomic or inverse role. An RBox $\mathcal{R}$ is a finite set of role inclusion axioms and transitivity axioms. $\sqsubseteq^*$ denotes the reflexive, transitive closure of $\sqsubseteq$ over $\{R \sqsubseteq S, \mathit{Inv}(R) \sqsubseteq \mathit{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is *transitive* in $\mathcal{R}$ if there exists a role S such that $S \sqsubseteq^* R$, $R \sqsubseteq^* S$, and either $\mathit{Trans}(S) \in \mathcal{R}$ or $\mathit{Trans}(\mathit{Inv}(S)) \in \mathcal{R}$. If no transitive role S with $S \sqsubseteq^* R$ exists, R is called *simple*.

Let N_C be the set of *atomic concepts* and N_I a set of individuals. The set of *concepts* is inductively defined using the following grammar:

$$C \rightarrow \top \mid \bot \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq nS \mid \leq nS \mid \{a\}$$

where $A \in N_C$, C_i $\mathcal{SHOIN}$ concepts, $R \in N_R$, $S \in N_R$ a simple role and $a \in N_I$.

A *general concept inclusion* (GCI) is of the form $C \sqsubseteq D$ with C and D $\mathcal{SHOIN}$ concepts. A TBox $\mathcal{T}$ is a finite set of GCIs also called axioms. An ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$ and $R(a,b)$, with A an atomic concept, R an atomic role and a, b are individuals from N_I. Note that in our setting, the ABox is only allowed to contain assertions about the belonging of individuals to atomic concepts and roles.

A knowledge base $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$ with signature $\Sigma = (N_C, N_R, N_I)$. The tuple $\mathcal{I} = (\cdot^\mathcal{I}, \Delta^\mathcal{I})$ is an *interpretation* for $\mathcal{K}$ iff $\Delta^\mathcal{I} \neq \emptyset$ and $\cdot^\mathcal{I}$ assigns an element $a^\mathcal{I} \in \Delta^\mathcal{I}$ to each individual a, a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ to each atomic concept A, and a relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ to each atomic role R. $\cdot^\mathcal{I}$ then assigns values to more complex concepts and roles as described in Tab. 1. $\mathcal{I}$ is a *model* of $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) if it satisfies all axioms and assertions in $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$ as shown in Tab. 1. A TBox $\mathcal{T}$ is called consistent, if there is an interpretation satisfying all axioms in $\mathcal{T}$. A concept C is called *satisfiable w.r.t.* $\mathcal{R}$ and $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^\mathcal{I} \neq \emptyset$. An assertion A of the form $B(a)$ or $R(a,b)$ is entailed by a knowledge base $\mathcal{K}$, denoted by $\mathcal{K} \models A$, iff $\mathcal{I} \models A$ for all models $\mathcal{I}$ of $\mathcal{K}$.

Since this paper only considers updates concerning the ABox, we restrict the following definition to ABox assertions. For the task of deleting assertions from

Concepts and Roles					
$\top^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}}$	$\{a\}^{\mathcal{I}}$	$=$	$\{a^{\mathcal{I}}\}$
$\bot^{\mathcal{I}}$	$=$	$\emptyset$	$(\forall R.C)^{\mathcal{I}}$	$=$	$\{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$(\exists R.C)^{\mathcal{I}}$	$=$	$\{x \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$(C \sqcup D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$(\geq n\ S)^{\mathcal{I}}$	$=$	$\{x \mid \|\{y \mid (x,y) \in S^{\mathcal{I}}\}\| \geq n\}$
$(C \sqcap D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$(\leq n\ S)^{\mathcal{I}}$	$=$	$\{x \mid \|\{y \mid (x,y) \in S^{\mathcal{I}}\}\| \leq n\}$
$(R^{-})^{\mathcal{I}}$	$=$	$\{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$			
TBox & RBox axioms			ABox assertion		
$C \sqsubseteq D$	$\Rightarrow$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	$A(a)$	$\Rightarrow$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
$R \sqsubseteq S$	$\Rightarrow$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$R(a,b)$	$\Rightarrow$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
$\mathit{Trans}(R)$	$\Rightarrow$	$(R^{\mathcal{I}})^{+} \subseteq R^{\mathcal{I}}$			

Table 1. Model-theoretic semantics of $\mathcal{SHOIN}$. R^{+} is the transitive closure of R, R^{-} is the inverse of role R.

ABox	RDF
$A(x)$	x `a` A.
$P(x,y)$	x `P` y.

Table 2. $\mathcal{SHOIN}$ ABox assertions vs. RDF as presented in [2], with A a concept name, P a role (or property) name, Γ a set of IRIs and x, $y \in \Gamma$.

a knowledge base or the task of debugging an ABox which is unsatisfiable w.r.t. the TBox and RBox, the notion of justifications is very helpful. Given an ABox assertion α and a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, a justification of α in $\mathcal{K}$ is a minimal subset of $\mathcal{A}$ which together with $\mathcal{T}$ and $\mathcal{R}$ implies α.

Definition 1 (Justification [14]) *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base and α be an ABox assertion. $\mathcal{J} \subseteq \mathcal{A}$ is a* justification *for α in $\mathcal{K}$ if $(\mathcal{R}, \mathcal{T}, \mathcal{J}) \models \alpha$ and for all $\mathcal{J}' \subset \mathcal{J}$, $(\mathcal{R}, \mathcal{T}, \mathcal{J}') \not\models \alpha$. The set of all justifications for α in $\mathcal{K}$ is denoted by $\mathit{Just}(\alpha, \mathcal{K})$.*

An *Internationalized Resource Identifier (IRI)* is a character string used to identify a resource. *Blank nodes* identify anonymous resources which are not directly identifiable from an RDF statement. A *literal* is a character string which possibly can be interpreted by means of a certain datatype.

Definition 2 (RDF Triple, RDF Graph) *Let I, B and L be disjoints sets of IRIs, blank nodes and literals. An* RDF triple *is of the form $(s,p,o) \in (I \cup B) \times I \times (I \cup B \cup L)$. A set of RDF triples is called* RDF graph.

A triple without blank node is called *ground triple*. Every $\mathcal{SHOIN}$ knowledge base can be mapped to an RDF graph and vice versa. See Tab. 2 for the mapping of ABox assertions and [19] for further details on the mapping. Therefore, we consider a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ to be equivalent to the RDF graph produced by this mapping. Furthermore, for the sake of uniform notation, in most cases we will stick to DL syntax.

Definition 3 (BGP, CQ, UCQ, Query Answer [2]) *A* conjunctive query (CQ) *q, or* basic graph pattern (BGP)*, is a set of atoms of the form given in Tab. 2 where $x, y \in \Gamma \cup \mathcal{V}$ with $\mathcal{V}$ a countable infinite set of variables (written as* `'?-'` *prefixed alphanumeric strings) and Γ a set of IRIs. A* union of conjunctive queries (UCQ) *or* UNION *pattern Q, is a set of CGs. $\mathcal{V}(q)$ ($\mathcal{V}(Q)$) denotes the set of variables, occurring in q (Q). An* answer *to a CQ q over a knowledge base $\mathcal{K}$ is a substitution θ of the variables in $\mathcal{V}(q)$ with constants in Γ such that every model of $\mathcal{K}$ satisfies all facts in $q\theta$. The set of all such answers is denoted with $ans(q, \mathcal{K})$. The set of answers to a UCQ Q is $\cup_{q \in Q} ans(q, \mathcal{K})$.*

Please note, that our definition of a BGP disallows blank nodes. Next, we introduce the notion of a SPARQL update operation and simple update semantics.

Definition 4 (SPARQL Update Operation[2]) *Let P_d and P_i be BGPs and P_w a BGP or UNION pattern. An* update operation *$u(P_d, P_i, P_w)$ has the form*

$$\texttt{DELETE } P_d \texttt{ INSERT } P_i \texttt{ WHERE } P_w$$

Definition 5 (Simple Update Semantics [2]) *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base then the* simple update *of $\mathcal{K}$ w.r.t. an SPARQL update operation $u(P_d, P_i, P_w)$ is defined as $\mathcal{K}_{u(P_d,P_i,P_w)} = (\mathcal{R}, \mathcal{T}, ((\mathcal{A} \setminus \mathcal{A}_d) \cup \mathcal{A}_i)$ where $\mathcal{A}_d = \cup_{\theta \in ans(P_w,\mathcal{K})} gr(P_d\theta)$ and $\mathcal{A}_i = \cup_{\theta \in ans(P_w,\mathcal{K})} gr(P_i\theta)$ with gr a selection function which selects all ground triples from a BGP.*

The simple update of a knowledge base w.r.t. $u(P_d, P_i, P_w)$ results in removing the set of ABox assertions corresponding to P_d and adding the set of ABox assertions corresponding to P_i. Note that in case of using simple update semantics, it is not guaranteed that the assertions in P_d are not entailed anymore. Furthermore, it is possible that the resulting knowledge base is inconsistent. Since this outcome is not always acceptable for an update, in Sec. 6 we suggest different semantics for SPARQL update operations which behave differently.

4. Overall Architecture

According to [9], the overall processing model for performing a SPARQL update query is to first evaluate the pattern in the `WHERE` clause, then perform the deletion and after that the insertion. This is why we suggest the workflow given in Algorithm 1.

In Line 1 a `CONSTRUCT` query for the `DELETE` and `WHERE` clause is used to create a graph G_D consisting of the triples which are supposed to be deleted. These triples correspond to a set of ABox assertions denoted by $\mathcal{A}_d$. Similar to this, Line 2 constructs the set $\mathcal{A}_i$ of ABox assertions which are supposed to be inserted. Next, the task of deleting the set of assertions given in $\mathcal{A}_d$ is addressed. For this, in Line 3 function *compDeletion* is called which, given a knowledge base, a set of ABox assertions which should be deleted and a keyword indicating the desired semantics, determines a subset of $\mathcal{A}$ which accomplishes the deletion. The pseudo code for this procedure is given in Algorithm 2. Line 4 checks the satisfiability of

Algorithm 1 Perform a SPARQL update query to a knowledge base.

Input: $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, SPARQL update query $u(P_d, P_i, P_w)$, *DeleteSem* the desired sematics for deletion and *InsertSem* the desired semantics for insertion.
Output: Revised knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A}_{Revised})$

```
1: $\mathcal{A}_d$ = set of ABox ass. for the result of asking CONSTRUCT $P_d$ WHERE $P_w$ to $\mathcal{K}$
2: $\mathcal{A}_i$ = set of ABox ass. for the result asking CONSTRUCT $P_i$ WHERE $P_w$ to $\mathcal{K}$
3: $Deletion = compDeletion(\mathcal{K}, \mathcal{A}_d, DeleteSem)$
4: if $\mathcal{K} = (\mathcal{R}, \mathcal{T}, (\mathcal{A} \setminus Deletion) \cup \mathcal{A}_i)$ consistent then
5:     return $\mathcal{K} = (\mathcal{R}, \mathcal{T}, (\mathcal{A} \setminus Deletion) \cup \mathcal{A}_i)$
6: else
7:     $Debug = compDeletion(\mathcal{K} = (\mathcal{R}, \mathcal{T}, (\mathcal{A} \setminus Deletion) \cup \mathcal{A}_i), \{\bot\}, InsertSem)$
8:     return $\mathcal{K} = (\mathcal{R}, \mathcal{T}, ((\mathcal{A} \setminus Deletion) \cup \mathcal{A}_i) \setminus Debug)$
9: end if
```

the knowledge base resulting from first deleting the ABox assertions determined by the *compDeletion* function and then adding the assertions in $\mathcal{A}_i$. If the resulting knowledge base is satisfiable, then Line 5 returns the resulting knowledge base. If however the resulting knowledge base is unsatisfiable, the *compDeletion* procedure is used to debug the knowledge base. For this purpose, the *compDeletion* procedure is called with $\{\bot\}$ as the set of assertions which are supposed to be deleted together with the desired insert semantics *InsertSem*. *compDelete* computes a set of ABox assertions which should be deleted in order to debug the knowledge base. This approach exploits the fact that debugging a knowledge base can be accomplished by deleting *false* from the knowledge base. Line 8 returns the knowledge base resulting from removing the computed set *Debug*, resulting in the knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, ((\mathcal{A} \setminus Deletion) \cup \mathcal{A}_i) \setminus Debug)$.

5. Deletion

For a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ and $\mathcal{A}_d$ a set of ABox assertions which are supposed to be deleted, we now describe how to determine a minimal subset of $\mathcal{A}$ whose deletion prevents the assertions in $\mathcal{A}_d$ from being entailed by $\mathcal{K}$.

Definition 6 (Deletion) *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base and $\mathcal{A}_d$ a set of ABox assertions. A* deletion *D of $\mathcal{A}_d$ in $\mathcal{K}$ is a minimal subset of the ABox $\mathcal{A}$ such that no element in $\mathcal{A}_d$ is entailed by $(\mathcal{R}, \mathcal{T}, \mathcal{A} \setminus D)$.*

As demonstrated by the example provided in Sec. 2, it is not sufficient to only remove all elements contained in $\mathcal{A}_d$ from the ABox since even after their removal they might still be entailed by the knowledge base. Furthermore, there might be more than one possible way to accomplish the deletion of $\mathcal{A}_d$. The latter aspects will be addressed in Sec. 6. To determine a deletion of $\mathcal{A}_d$ in a knowledge base $\mathcal{K}$, we suggest to use justifications.

5.1. Justification based Deletion

Justifications can be used to compute a deletion for an assertion α from a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$. Please note that according to Def. 1, we restrict justifications to be subsets of the ABox. Since the set of all justifications for α in $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ corresponds to the set of all minimal subsets of $\mathcal{A}$ which together with $\mathcal{R}$ and $\mathcal{T}$ imply α, it is sufficient to delete exactly one element from each justification in order to prevent α from being entailed. This corresponds to the construction of a minimal hitting set of the set of all justifications for α in $\mathcal{K}$.

Definition 7 (Hitting Set) *Let $S = \{S_1, \dots S_n\}$ be a set of sets. A hitting set for S is a set $H \subseteq \cup_{i=1}^{n} S_i$ with $H \cap S_i \neq \emptyset, \forall 1 \leq i \leq n$. If further no proper subset of H is a hitting set for S, H is called a minimal hitting set. The set of all minimal hitting sets for S is denoted by $HS(S)$.*

Each minimal hitting set in $HS(Just(\alpha, \mathcal{K}))$ constitutes a deletion of $\{\alpha\}$ in $\mathcal{K}$.

Example 1 *The set of all justifications for StudentTrainee(john) in the knowledge base $\mathcal{K}$ given in Sec. 2 is:*

$$\begin{aligned} Just(StudentTrainee(john), \mathcal{K}) = \{&\{StudentTrainee(john)\},\\ &\{Student(john), Employee(john)\},\\ &\{Student(john), SoftwareEngineer(john)\}\} \end{aligned}$$

The two minimal hitting sets for $Just(StudentTrainee(john), \mathcal{K})$, which both provide a deletion for StudentTrainee(john) in $\mathcal{K}$, are:

$$\begin{aligned} H_1 =&\{StudentTrainee(john), Employee(john), SoftwareEngineer(john)\}\\ H_2 =&\{StudentTrainee(john), Student(john)\} \end{aligned}$$

Potentially every subset of the ABox can be a justification for α, resulting in justifications with arbitrary cardinalities. Hence, there can be more than one hitting set for the set of all justifications of α and different ways to perform a deletion. When dealing with SPARQL update queries, it is reasonable to assume that more than one ABox assertion is supposed to be deleted. The first idea to compute all possible deletions for a set of assertions $\mathcal{A}_d$ would be to compute all justifications for all assertions in $\mathcal{A}_d$ separately and then compute all minimal hitting sets for these justifications. However this approach considers an unnecessarily high number of justifications because the union of all justifications might contain non-minimal sets.

Example 2 *Consider the knowledge base presented in the Sec. 2 together with $\mathcal{A}_d = \{StudentTrainee(john), Student(john)\}$ the set of assertions which are supposed to be deleted. The task is to find a minimal set $\mathcal{A}' \subseteq \mathcal{A}$ such that $(\mathcal{T}, \mathcal{A} \setminus \mathcal{A}') \not\models StudentTrainee(john)$ and $(\mathcal{T}, \mathcal{A} \setminus \mathcal{A}') \not\models Student(a)$. We construct the set of justifications for both assertions:*

$$\begin{aligned} Just(StudentTrainee(john), \mathcal{K}) = \{&\{StudentTrainee(john)\},\\ &\{Employee(john), Student(john)\},\\ &\{SoftwareEngineer(john), Student(john)\}\}\\ Just(Student(john), \mathcal{K}) = \{&\{Student(john)\}\}. \end{aligned}$$

The only hitting for $Just(StudentTrainee(john), \mathcal{K}) \cup Just(Student(john), \mathcal{K})$ *is*

$$\begin{aligned}&HS(Just(StudentTrainee(john), \mathcal{K}) \cup Just(Student(john), \mathcal{K}))\\&= \{\{StudentTrainee(john), Student(john)\}\}.\end{aligned}$$

Only the justifications $\{StudentTrainee(john)\}$ *and* $\{Student(john)\}$ *have to be considered for the construction of the hitting set, since the other justifications are supersets of* $\{Student(john)\}$*. This leads to the notion of root justifications.*

Definition 8 (Root Justification [17]) *Let* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ *be a knowledge base,* $U = \{\alpha_1, \ldots, \alpha_n\}$ *a set of ABox assertions and* $Just(\alpha_i, \mathcal{K})$ *the set of all justifications of* α_i *in* $\mathcal{K}$*. A set* $J \in \cup_{i=1}^{n} Just(\alpha_i, \mathcal{K})$ *is a* root justification *for* U *in* $\mathcal{K}$ *iff there is no* $J' \in \cup_{i=1}^{n} Just(\alpha_i, \mathcal{K})$ *with* $J' \subset J$*. All justifications in* $\cup_{i=1}^{n} Just(\alpha_i, \mathcal{K})$ *which are not a root justification are called* derived justification*.*

Example 3 *Only* $\{StudentTrainee(john)\}$ *and* $\{Student(john)\}$ *are root justifications in Ex. 2.*

As shown in [17], for a set of assertions $\mathcal{A}_d$ which are supposed to be deleted from a knowledge base $\mathcal{K}$ it is sufficient to consider only root justifications in order to determine what to delete.

Next we present Algorithm 2 which uses the notion of root justifications to compute a deletion for a set of ABox assertions from a knowledge base. We use a black box for the computation of root justifications. One way to compute all root justifications for a given set of assertions $\mathcal{A}_d$ is to use an off the shelf reasoner like Pellet [24] to compute first all justifications for the different elements of $\mathcal{A}_d$ and then to remove all derived justifications. Another way is to use the algorithm introduced in [17] which directly computes the root justifications. For a knowledge

Algorithm 2 (compDeletion)
Given $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ and a set of ABox assertions $\mathcal{A}_d$ which should be deleted, construct a minimal set $\mathcal{A}'$ with $\mathcal{A}' \subseteq \mathcal{A}$ and $(\mathcal{R}, \mathcal{T}, \mathcal{A} \setminus \mathcal{A}') \not\models A$ for all $A \in \mathcal{A}_d$.

Input: $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, a set of ABox assertions $\mathcal{A}_d$ which should be deleted and the desired semantics.
Output: A deletion for $\mathcal{A}_d$ in $\mathcal{K}$.

```
1: $RJust$ = set of all root justifications for $\mathcal{A}_d$ in $\mathcal{K}$
2: $HS$ = set of all minimal hitting sets for $RJust$
3: $Deletion = chooseDeletion(\mathcal{K}, u(P_d, P_i, P_w), HS, Semantics)$
4: return $Deletion$
```

base $\mathcal{K}$ and a set of ABox assertions $\mathcal{A}_d$ which are supposed to be deleted, line 1 computes all root justifications for $\mathcal{A}_d$ in $\mathcal{K}$. Next, line 2 constructs all minimal hitting sets for the set of root justifications. Line 3 uses these minimal hitting sets together with a specified semantic to choose a subset of the ABox which should be deleted. The next Section presents possible choices for these semantics.

6. Semantics

In the previous section, we learnt that there can be more than one possible way to delete a set of assertions from a knowledge base. Furthermore, there might be more than one possible way to debug a knowledge base resulting from an insertion. To handle this problem, different semantics are suggested to choose among the possibilities.

6.1. Semantics for Deletion

Maxichoice Semantics [8]: In maxichoice semantics, the deletion with minimal cardinality is chosen. This corresponds to maximizing the number of ABox assertions which are preserved. Thus the resulting ABox implements the deletion while possessing maximal cardinality.

In cases, where there is more than one deletion with minimal cardinality, the maxichoice semantics does not specify which of these deletions to choose. We refrain from choosing an arbitrary one just for the sake of determinism. One way to solve this issue would be to offer the different possibilities to the user and let the user choose. Another possibility would be to use additional information associated with the ABox assertions like suggested in the priority/credibility based semantics introduced at the end of this section.

Meet Semantics [8]: This semantics is rather pessimistic since it removes all assertions occurring in a deletion. Thus the resulting ABox implements the deletion while possessing minimal cardinality.

Priority/Credibility based Semantics [8]: In some cases, additional information associated to the assertions in the ABox is available. Examples for this kind of information are credibility or time stamps and are summarized by the term provenance information. This provenance information can be used to decide which assertions to delete by for example preferably deleting those assertions associated with a low credibility or an old time stamp [20].

6.2. Semantics for Insertion

Insertion of a set of ABox assertions $\mathcal{A}_i$ can be accomplished by adding it to the knowledge base and then checking if the resulting knowledge base is satisfiable. If it is satisfiable, the resulting knowledge base corresponds to the result of the update operation. If it is unsatisfiable, the methods for deletion can be used to remove the inconsistencies from the knowledge base by deleting $\{\perp\}$ from the knowledge base. As soon as a debugging step is used during an insertion, there might be several possibilities to perform the insertion. Different semantics can be used to pick one of these possibilities. Please note that it is possible that some (or even all) possibilities to debug the knowledge base remove the inserted assertions.

In addition to the semantics introduced in the previous section, semantics for insertion can take into account the fact that one can distinguish between old information, which was present in the ABox before the insertion and new information which was introduced by the insertion. The semantics presented below build on this distinction and were first introduced in [2] for $DL - Lite_{RDFS_{\neg}}$.

Brave Semantics: This semantics gives priority to new assertions which are inserted by the query meaning that the possibility to debug is selected which contains the highest number of assertions in $\mathcal{A}_i$.

Note that there are cases, where there is is more than one possible way for debugging which contains the highest number of assertions in $\mathcal{A}_i$. Similar to the maxichoice semantics we are facing non-determinism here and refrain from choosing an arbitrary possibility to debug just for the sake of determinism. One way to solve this issue would be to leave the decision to the user by offering the different debugging possibilities to the user. Another possibility would be to use additional provenance information associated with the ABox assertions.

Cautious Semantics: In contrast to brave semantics, the cautious semantics gives priority to assertions which were already present before the insertion. This means that the possibility to debug is selected, which preserves as many of the already present assertions as possible. In the worst case, the debugging step removes the inserted assertion, making the insertion a *semi-revision* operator as corresponding to the one introduced in [11].

Fainthearted Semantics: This semantics is rather overcautious since it totally discards an insertion as soon as there is a conflict between the inserted assertions and the already present assertions.

6.3. Update Semantics taking into account the Interplay of Deletion and Insertion

Next, we introduce another semantics, which aims at optimizing the overall result of an update. When considering the deletion and the insertion part of an update separately, it is possible that the insertion cancels the deletion. More precisely, it is possible that inserting assertions causes other assertions to be entailed which were deleted by the very same update.

Example 4 *We consider an update operation deleting $StudentTrainee(john)$ and inserting $Secretary(john)$ into the knowledge base introduced in Sect 2. As shown in Ex. 1, there are two possibilities to delete $StudentTrainee(john)$ from the knowledge base, which we here denote by Del_1 and Del_2:*

$$Del_1 = \{StudentTrainee(john), Employee(john), SoftwareEngineer(john)\}$$
$$Del_2 = \{StudentTrainee(john), Student(john)\}$$

We choose Del_1 to perform the deletion of $StudentTrainee(john)$ leaving only the assertion $Student(john)$ in the ABox. Next we address the insertion: Adding $Secretary(john)$ to the ABox together with the TBox assertions (2) and (3) from Sec. 2 causes the deleted assertion $StudentTrainee(john)$ to be entailed which clearly is not intended.

Ex. 4 illustrates that, in order to optimize the overall result of an update operation, it might be worthwhile not to consider the deletion and the insertion separately but to take their interplay into account. Before presenting semantics implementing this idea, we introduce the notion of an *implementation of an update operation to a knowledge base*, which corresponds to one possibility to perform an update operation to this knowledge base.

Definition 9 (Implementation of an Update Operation) *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base, $u(P_d, P_i, P_w)$ an update operation, $\mathcal{A}_d$ the set of ABox assertions for the result of asking* `CONSTRUCT` P_d `WHERE` P_w*, $\mathcal{A}_i$ set of ABox assertions for the result of asking* `CONSTRUCT` P_i `WHERE` P_w *to $\mathcal{K}$. Then (Del, Dbg) is called an* implementation of update operation $u(P_d, P_i, P_w)$ *w.r.t. $\mathcal{K}$, if Del is a deletion for $\mathcal{A}_d$ in $\mathcal{K}$ and Dbg is a deletion for $\{\bot\}$ in $(\mathcal{R}, \mathcal{T}, (\mathcal{A} \setminus Del) \cup \mathcal{A}_i)$. The knowlege base resulting from (Del, Dbg) is defined as $\mathcal{K}^{(Del,Dbg)}_{u(P_d,P_i,P_w)} = (\mathcal{R}, \mathcal{T}, ((\mathcal{A} \setminus Del) \cup \mathcal{A}_i) \setminus Dbg)$.*

We omit the knowledge base if it is clear from the context and simply say that (Del, Dbg) is an implementation of update operation $u(P_d, P_i, P_w)$. Next we present query-driven semantics which, can be used to choose one of the possible implementations of an update operation $u(P_d, P_i, P_w)$.

Query-driven Semantics: Given knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, an update operation $u(P_d, P_i, P_w)$, $\mathcal{A}_d$, $\mathcal{A}_i$ defined as in Def. 9 and an implementation (Del, Dbg) of $u(P_d, P_i, P_w)$. All assertions in $\mathcal{A}_d$ which are not entailed by $\mathcal{K}^{(Del,Dbg)}_{u(P_d,P_i,P_w)}$ can be seen as successful deletions (w.r.t. this implementation (Del, Dbg)). Similarly, all assertions in $\mathcal{A}_i$ which are entailed by $\mathcal{K}^{(Del,Dbg)}_{u(P_d,P_i,P_w)}$ can be seen as successful insertions (w.r.t. this implementation (Del, Dbg)). The aim of query-driven semantics is to maximize the number of successful deletions and insertions.

Definition 10 (Query-driven Update Semantics) *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base, $u(P_d, P_i, P_w)$ an update operation, $\mathcal{A}_d$ the set of ABox assertions for the result of asking* `CONSTRUCT` P_d `WHERE` P_w *and $\mathcal{A}_i$ the set of ABox assertions for the result of asking* `CONSTRUCT` P_i `WHERE` P_w *to $\mathcal{K}$. The* query-driven update *of $\mathcal{K}$ w.r.t. $u(P_d, P_i, P_w)$ is $\mathcal{K}^{\mathbf{Sem}^{qd}}_{u(P_d,P_i,P_w)} = (\mathcal{R}, \mathcal{T}, ((\mathcal{A} \setminus Del) \cup \mathcal{A}_i) \setminus Dbg)$ where (Del, Dbg) is the implementation of $u(P_d, P_i, P_w)$ maximizing*

$$|\{\alpha \in \mathcal{A}_d \mid \mathcal{K}^{(Del,Dbg)}_{u(P_d,P_i,P_w)} \not\models \alpha\}| + |\{\beta \in \mathcal{A}_i \mid \mathcal{K}^{(Del,Dbg)}_{u(P_d,P_i,P_w)} \models \beta\}| \tag{9}$$

If the SPARQL update query does not contain a `DELETE` clause, *Del* is empty and the query-driven update semantics behaves like the brave semantics. If the SPARQL update query does not contain an `INSERT` clause, Equation 9 does not choose a deletion. For these cases, query-driven semantics should be combined with one of the semantics for deletion presented in Sec. 6.1.

Note that there are other cases where query-driven semantics still leaves some choices: there could be more than one combination of deletion and debugging maximizing Eq. (9). This non-determinism could be avoided by presenting the set of successful deletions and insertions for these combinations to the user and let the user decide which of the possibilities is closest to the desired result. Another possibility would be to enhance the query mechanism such that the user is able to give priorities to certain parts of the query. Considering for example the query presented in Sec. 2 the user could specify that the deletion of the *StudentTrainee* status is more important than the assignment of those students to course $c1$.

Example 5 *Let us reconsider the update operation introduced in Ex. 4. Please recall that* $\mathcal{A}_d = \{StudentTrainee(john)\}$ *and* $\mathcal{A}_i = \{Secretary(john)\}$ *and that two different deletions* Del_1 *and* Del_2 *were presented.*

$$\begin{aligned}(\mathcal{A} \setminus Del_1) \cup \mathcal{A}_i &= \{Student(john), Secretary(john)\}\\ (\mathcal{A} \setminus Del_2) \cup \mathcal{A}_i &= \{SoftwareEngineer(john), Employee(john),\\ &\qquad Secretary(john)\}\end{aligned}$$

Both $(\mathcal{A} \setminus Del_1) \cup \mathcal{A}_i$ *and* $(\mathcal{A} \setminus Del_2) \cup \mathcal{A}_i$ *are satisfiable w.r.t. the TBox, hence no debugging step is necessary. The two possible results of the update operation are:*

$$\begin{aligned}\mathcal{K}_1 &= (\mathcal{T}, (\mathcal{A} \setminus Del_1) \cup \mathcal{A}_i)\\ \mathcal{K}_2 &= (\mathcal{T}, (\mathcal{A} \setminus Del_2) \cup \mathcal{A}_i)\end{aligned}$$

Secretary(john) constitutes a successful insertion in both $\mathcal{K}_1$ *and* $\mathcal{K}_2$*. Since* $\mathcal{K}_1 \models StudentTrainee(john)$ *and* $\mathcal{K}_2 \not\models StudentTrainee(john)$*, StudentTrainee(john) is a successful deletion in* $\mathcal{K}_2$ *but not in* $\mathcal{K}_1$*. Query-driven semantics results in* $\mathcal{K}_2$ *since it maximizes the number of successful insertions and deletions.*

Query-driven semantics aims at the optimization of the overall result of the query. In extreme cases, where the sizes of $\mathcal{A}_d$ and $\mathcal{A}_i$ differ dramatically, it could happen that the smaller set is neglected. For example if $|\mathcal{A}_d| = 5$ and $|\mathcal{A}_i| = 1000$, Eq. (9) is dominated by the set of successful insertions. This could be remedied by introducing a weight indicating the importance of the desired deletion or insertion.

7. Conclusion and Future Work

In this paper, we presented an approach to compute SPARQL ABox update in the presence of OWL-DL TBoxes. We address the fact that there might be more than one way to perform a deletion with different semantics according to which one of the possibilities is chosen. To master the interplay of the deletion and the insertion task of a SPARQL update query, we introduce query-driven semantics which aim at maximizing the overall effect of an update operation.

An implementation of our approach is on the way. We are using SPARQL-DL to evaluate the `CONSTRUCT` query and Pellet to compute all possible justifications.

Besides evaluating the approach, future work addresses the use of summarization techniques to efficiently compute justifications. Since the set of ABox assertions which are supposed to be deleted are created by a basic graph pattern, this set is likely to contain many similar assertions. We want to exploit these similarities by first aggregating similar individuals into one individual and then computing the justification for a bunch of assertions in one single step. Possible summarization techniques we are looking into are [7] and [10].

Acknowledgements We would like to thank Albin Ahmeti, Diego Calvanese, Axel Polleres and Vadim Savenkov for being so kind to share the implementation of their approach for SPARQL instance-level update in the presence of $DL-Lite_{RDFS_\neg}$ TBoxes which will serve as a basis for our implementation.

References

[1] A. Ahmeti, D. Calvanese, and A. Polleres. Updating RDFS aboxes and tboxes in SPARQL. In *Semantic Web Conference (1)*, volume 8796 of *LNCS*. Springer, 2014.

[2] A. Ahmeti, D. Calvanese, A. Polleres, and V. Savenkov. Handling inconsistencies due to class disjointness in SPARQL updates. In *ESWC*, volume 9678 of *LNCS*. Springer, 2016.

[3] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.

[4] F. Baader and W. Nutt. Basic description logics. In *Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.

[5] S. Benferhat, Z. Bouraoui, O. Papini, and E. Würbel. A prioritized assertional-based revision for dl-lite knowledge bases. In *JELIA*, volume 8761 of *LNCS*. Springer, 2014.

[6] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On instance-level update and erasure in description logic ontologies. *J. Log. Comput.*, 19(5):745–770, 2009.

[7] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In *ISWC*, volume 4273 of *LNCS*. Springer, 2006.

[8] P. Gärdenfors and H. Rott. Belief revision. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 4)*. Oxford University Press, Oxford, UK, 1995.

[9] P. Gearon, A. Polleres, and A. Passant. SPARQL 1.1 update. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

[10] B. Glimm, Y. Kazakov, and T. Tran. Ontology materialization by abstraction refinement in horn SHOIF. In *AAAI*, pages 1114–1120. AAAI Press, 2017.

[11] C. Halaschek-Wiener and Y. Katz. Belief base revision for expressive description logics. In *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[12] S. O. Hansson. *A Textbook of Belief Dynamics*, volume 11 of *Applied Logic Series*. Kluwer Academic Publishers, 1999.

[13] R. Horne, V. Sassone, and N. Gibbins. Operational semantics for SPARQL update. In *JIST*, volume 7185 of *LNCS*. Springer, 2011.

[14] M. Horridge. *Justification based explanation in ontologies*. PhD thesis, University of Manchester, UK, 2011.

[15] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *RR*, volume 6333 of *LNCS*. Springer, 2010.

[16] D. McGuinness, E. Kendall, J. Bao, and P. Patel-Schneider. OWL 2 web ontology language quick reference guide (second edition). Technical report, W3C, Dec. 2012. http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/.

[17] K. Moodley. Debugging and repair of description logic ontologies. Master's thesis, University of KwaZulo-Natal, Durban, South Africa, 2010.

[18] N. Nikitina, S. Rudolph, and B. Glimm. Interactive ontology revision. *J. Web Sem.*, 12:118–130, 2012.

[19] P. Patel-Schneider and B. Motik. OWL 2 web ontology language mapping to RDF graphs (second edition). W3C recommendation, W3C, Dec. 2012. http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/.

[20] S. Schenk, R. Q. Dividino, and S. Staab. Using provenance to debug changing ontologies. *J. Web Sem.*, 9(3):284–298, 2011.

[21] G. Schreiber and Y. Raimond. RDF 1.1 primer. W3C note, W3C, June 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[22] A. Seaborne and S. Harris. SPARQL 1.1 query language. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[23] E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL. In *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[24] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.