SPARQL Update for Materialised Triple Stores under *DL-Lite*$_{\text{RDFS}}$ Entailment

Albin Ahmeti[1], Diego Calvanese[2], and Axel Polleres[3]

[1] Vienna University of Technology, Favoritenstraße 9, 1040 Vienna, Austria
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
[3] Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Abstract. Updates in RDF stores have recently been standardised in the SPARQL 1.1 Update specification. However, computing answers entailed by ontologies in triple stores is usually treated orthogonally to updates. Even W3C's SPARQL 1.1 Update language and SPARQL 1.1 Entailment Regimes specifications explicitly exclude a standard behaviour for entailment regimes other than simple entailment in the context of updates. In this paper, we take a first step to close this gap. We define a fragment of SPARQL basic graph patterns corresponding to (the RDFS fragment of) *DL-Lite* and the corresponding SPARQL update language, dealing with updates both of ABox and of TBox statements. We discuss possible semantics along with potential strategies for implementing them. Particularly, we treat materialised RDF stores, which store all entailed triples explicitly, and preservation of materialisation upon ABox and TBox updates.

1 Introduction

The availability of SPARQL as a standard for accessing structured Data on the Web may well be called one of the key factors to the success and increasing adoption of RDF and the Semantic Web. Still, in its first iteration the SPARQL [23] specification has neither defined how to treat ontological entailments with respect to RDF Schema (RDFS) and OWL ontologies, nor provided means how to update dynamic RDF data. Both these gaps have been addressed within the recent SPARQL 1.1 specification, which provides both means to define query answers under ontological entailments (SPARQL 1.1 Entailment Regimes [9]), and an update language to update RDF data stored in a triple store (SPARQL 1.1 Update [8]). Nonetheless, these specifications leave it open how SPARQL endpoints should treat entailment regimes other than simple entailment in the context of updates; the main issue here is how updates shall deal with implied statements:

- What does it mean if an implied triple is explicitly (re-)inserted (or deleted)?
- Which (if any) additional triples should be inserted, (or, resp., deleted) upon updates?

For the sake of this paper, we address such questions with the focus on a deliberately minimal ontology language, namely the minimal RDFS fragment of [19].[4] As it turns out, even in this confined setting, updates as defined in the SPARQL 1.1 Update specification

[4] We ignore issues like axiomatic triples [13], blank nodes [17], or, in the context of OWL, inconsistencies arising through updates [5]. Neither do we consider named graphs in SPARQL, which is why we talk about "triple stores" as opposed to "graph stores" [8].

Table 1. *DL-Lite*$_{\text{RDFS}}$ assertions vs. RDF(S), where A, A' denote concept (or, class) names, P, P' denote role (or, property) names, Γ is a set of constants, and $x, y \in \Gamma$. For the RDF(S) vocabulary, we make use of similar abbreviations (sc, sp, dom, rng, a) introduced in [19].

	TBox	RDFS		TBox	RDFS		ABox	RDFS
1	$A' \sqsubseteq A$	A' sc A.	3	$\exists P \sqsubseteq A$	P dom A.	5	$A(x)$	x a A.
2	$P' \sqsubseteq P$	P' sp P.	4	$\exists P^{-} \sqsubseteq A$	P rng A.	6	$P(x,y)$	x P y.

impose non-trivial challenges; in particular, specific issues arise through the interplay of INSERT, DELETE, and WHERE clauses within a single SPARQL update operation, which —to the best of our knowledge— have not yet been considered in this combination in previous literature on updates under entailment (such as for instance [5, 11]).

Example 1. As a running example, we assume a triple store G with RDF (ABox) data and an RDFS ontology (TBox) O_{fam} about family relationships (in Turtle syntax [2]), where `:hasP`, `:hasM`, and `:hasF` denote the parent, mother, and father relations.

```
ABox:  :joe :hasP :jack.      :joe :hasM :jane.
TBox:  :Father sc :Parent.    :Mother sc :Parent.
 :hasF sp :hasP.  :hasM sp :hasP.  :hasP rng :Parent; dom :Child.
 :hasF rng :Father; dom :Child.    :hasM rng :Mother; dom :Child.
```

The following query should return `:jack` and `:jane` as (RDFS entailed) answers:

```
SELECT ?Y WHERE { :joe :hasP ?Y. }
```

SPARQL engines supporting simple entailment would only return `:jack`, though. ■

The intended behaviour for the query in Ex. 1 is typically achieved either *(i)* by query rewriting techniques computing entailed answers at query run-time, or *(ii)* by materialising all implied triples in the store, normally at loading time. That is, on the one hand, borrowing from query rewriting techniques for *DL-Lite* (such as, e.g., $\mathit{PerfectRef}$ [4]) one can reformulate such a query to return also implied answers.

Example 2 (cont'd). The rewriting according to $\mathit{PerfectRef}$ [4] of the query in Ex. 1 with respect to O_{fam} as a DL TBox in SPARQL yields

```
SELECT ?Y WHERE { {:joe :hasP ?Y}
             UNION {:joe :hasF ?Y}  UNION {:joe :hasM ?Y}}
```

Indeed, this query returns both `:jane` and `:jack`. ■

On the other hand, an alternative[5] is to materialise all inferences in the triple store, such that the original query can be used 'as is', for instance using the minimalistic inference rules for RDFS from [19][6] shown in Fig. 1.

Example 3 (cont'd). The materialised version of G would contain the following triples— for conciseness we only show assertional implied triples here, that is triples from the four leftmost rules in Fig. 1.

[5] This alternative is viable for RDFS, but not necessarily for more expressive DLs.
[6] These rules correspond to rules 2), 3), 4) of [19]; they suffice since we ignore blank nodes.

$$\frac{?C \text{ sc } ?D. \quad ?S \text{ a } ?C.}{?S \text{ a } ?D.} \qquad \frac{?P \text{ dom } ?C. \quad ?S\ ?P\ ?O.}{?S \text{ a } ?C.} \qquad \frac{?C \text{ sc } ?D. \quad ?D \text{ sc } ?E.}{?C \text{ sc } ?E.}$$

$$\frac{?P \text{ sp } ?Q. \quad ?S\ ?P\ ?O.}{?S\ ?Q\ ?O.} \qquad \frac{?P \text{ rng } ?C. \quad ?S\ ?P\ ?O.}{?O \text{ a } ?C.} \qquad \frac{?P \text{ sp } ?Q. \quad ?Q \text{ sp } ?R.}{?P \text{ sp } ?R.}$$

Fig. 1. Minimal RDFS inference rules

```
:joe a :Child;  :hasP :jack;  :hasM :jane;  :hasP :jane.
:jack a :Parent.    :jane a :Mother, :Parent.
```

On a materialised triple store, the query from Ex. 1 would return the expected results. ■

However, in the context of SPARQL 1.1 Update, things become less clear.

Example 4 (cont'd). The following operation tries to delete an implied triple and at the same time to (re-)insert another implied triple.

```
DELETE {?X a :Child} INSERT {?Y a :Mother} WHERE {?X :hasM ?Y}
```
■

Existing triple stores offer different solutions to these problems, ranging from ignoring entailments in updates altogether, to keeping explicit and implicit (materialised) triples separate and re-materialising upon updates. In the former case (ignoring entailments) updates only refer to explicitly asserted triples, which may result in non-intuitive behaviours, whereas the latter case (re-materialisation) may be very costly, while still not eliminating all non-intuitive cases, as we will see. The problem is aggravated by no systematic approach to which implied triples to store explicitly in a triple store and which not. In this paper we try to argue for a more systematic approach for dealing with updates in the context of RDFS entailments; we will focus on *materialised RDF stores*, which store all entailed ABox triples explicitly. We propose alternative update semantics and discuss possible implementation strategies, partially inspired by query rewriting techniques from ontology-based data access (OBDA) [15] and *DL-Lite* [4].

In previous work [1], we have also discussed *reduced RDF Stores*, i.e., redundancy-free RDF stores that do not store any assertional (ABox) triples already entailed by others, which we omit here for space limits. Here we have added a more in-depth discussion on the relation to DRed [6, 10, 16, 25], and we also discuss TBox updates.

2 Preliminaries

We introduce some basic notions about RDF graphs, RDFS ontologies, and SPARQL queries. Since we will draw from ideas coming from OBDA and *DL-Lite*, we introduce these notions in a way that is compatible with DLs.

Definition 1 (RDFS ontology, ABox, TBox, triple store). *We call a set $\mathcal{T}$ of inclusion assertions of the forms 1–4 in Table 1 an* (RDFS) TBox *(or* ontology*), a set $\mathcal{A}$ of assertions of the forms 5–6 in Table 1 an* (RDF) ABox*, and the union $G = \mathcal{T} \cup \mathcal{A}$ an* (RDFS) triple store*.*

In the context of RDF(S), the set Γ of constants coincides with the set I of IRIs. We assume the IRIs used for concepts, roles, and constants to be disjoint from IRIs of the RDFS and OWL vocabularies.[7] In the following, we view RDF and DL notation interchangeably, i.e., we treat any RDF graph consisting of triples without non-standard RDFS vocabulary as a set of TBox and ABox assertions. To define the semantics of RDFS, we rely on the standard notions of (first-order logic) *interpretation*, satisfaction of assertions, and *model*.

As for queries, we again treat the cases of SPARQL and DLs interchangeably. Let $\mathcal{V}$ be a countably infinite set of variables (written as '?' -prefixed alphanumeric strings).

Definition 2 (BGP, CQ, UCQ). *A* conjunctive query *(CQ) q, or* basic graph pattern (BGP)*, is a set of atoms of the forms 5–6 in Table 1, where now $x, y \in \Gamma \cup \mathcal{V}$. A* union of conjunctive queries *(UCQ) Q, or* UNION *pattern, is a set of CQs. We denote with $\mathcal{V}(q)$ (or $\mathcal{V}(Q)$) the set of variables from $\mathcal{V}$ occurring in q (resp., Q).*

In this definition we are considering only CQs in which all variables are *distinguished* (i.e., are answer variables), and such queries correspond to SPARQL basic graph patterns (BGPs). Also, we allow only for restricted forms of general SPARQL BGPs that correspond to standard CQs as formulated over a DL ontology; that is, we rule out more complex patterns in SPARQL 1.1 [12] (such as OPTIONAL, NOT EXISTS, FILTER), queries with variables in predicate positions, as well as "terminological" queries, e.g., $\{?x \; \mathsf{sc} \; ?y.\}$. We will relax the latter restriction later (see Sec. 4). Also, we do not consider here blank nodes separately[8]. By these restrictions, we can treat query answering and BGP matching in SPARQL analogously and define it in terms of interpretations and models (as usual in DLs). Specifically, an *answer* (under RDFS Entailment) to a CQ q over a triple store G is a substitution θ of the variables in $\mathcal{V}(q)$ with constants in Γ such that every model of G satisfies all facts in $q\theta$. We denote the set of all such answers with $ans_{\mathsf{rdfs}}(q, G)$ (or simply $ans(q, G)$). The set of answers to a UCQ Q is $\bigcup_{q \in Q} ans(q, G)$.

From now on, let $rewrite(q, \mathcal{T})$ be the UCQ resulting from applying $PerfectRef$ [4] to a CQ q and a triple store $G = \mathcal{T} \cup \mathcal{A}$, and let $mat(G)$ be the triple store obtained from exhaustive application on G of the inference rules in Fig. 1.

The next result follow immediately from, e.g., [4, 11, 19] and shows that query answering in RDFS can be done by either query rewriting or materialisation.

Proposition 1. *Let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store, q a CQ, and $\mathcal{A}'$ the set of ABox assertions in $mat(G)$. Then, $ans(q, G) = ans(rewrite(q, \mathcal{T}), \mathcal{A}) = ans(q, \mathcal{A}')$.*

Various triple stores (e.g., BigOWLIM [3]) perform ABox materialisation directly upon loading data. However, such triple stores do not necessarily materialise the TBox: in order to correctly answer UCQs as defined above, a triple store actually does not need to consider the two rightmost rules in Fig. 1. Accordingly, we will call a triple store or (ABox) *materialised* if in each state it always guarantees $G \setminus \mathcal{T} = mat(G) \setminus mat(\mathcal{T})$. We observe that, trivially, a triple store containing no ABox statements is materialised.

Finally, we introduce the notion of a SPARQL update operation.

[7] That is, we assume no "non-standard use" [22] of these vocabularies. While we could assume concept names, role names, and individual constants mutually disjoint, we rather distinguish implicitly between them "per use" (in the sense of "punning" [18]) based on their position.

[8] Blank nodes in a triple store may be considered as constants and we do not allow blank nodes in queries, which does not affect the expressivity of SPARQL.

Definition 3 (SPARQL update operation). *Let P_d and P_i be BGPs, and P_w a BGP or UNION pattern. Then an* update operation $u(P_d, P_i, P_w)$ *has the form*

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

Intuitively, the semantics of executing $u(P_d, P_i, P_w)$ on G, denoted as $G_{u(P_d,P_i,P_w)}$ is defined by interpreting both P_d and P_i as "templates" to be instantiated with the solutions of $ans(P_w, G)$, resulting in sets of ABox statements $\mathcal{A}_d$ to be deleted from G, and $\mathcal{A}_i$ to be inserted into G. A naïve update semantics follows straightforwardly.

Definition 4 (Naïve update semantics). *Let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store, and $u(P_d, P_i, P_w)$ an update operation. Then,* naive update of G with $u(P_d, P_i, P_w)$, *denoted $G_{u(P_d,P_i,P_w)}$, is defined as $(G \setminus \mathcal{A}_d) \cup \mathcal{A}_i$, where $\mathcal{A}_d = \bigcup_{\theta \in ans(P_w,G)} gr(P_d\theta)$, $\mathcal{A}_i = \bigcup_{\theta \in ans(P_w,G)} gr(P_i\theta)$, and $gr(P)$ denotes the set of ground triples in pattern P.*

As easily seen, this naïve semantics does not preserve materialisation.[9]

3 Alternative Mat-Preserving Semantics for ABox Updates

We investigate now alternative semantics for updates that preserve a materialised ABox (or simply, *mat-preserving* semantics) and in how far these semantics can—similar to query answering—be implemented on top of off-the-shelf SPARQL 1.1 triple stores.

Definition 5 (Mat-preserving semantics). *Let G and $u(P_d, P_i, P_w)$ be as in Def. 4. An update semantics Sem is called* mat-preserving, *if $G^{Sem}_{u(P_d,P_i,P_w)} = mat(G^{Sem}_{u(P_d,P_i,P_w)})$.*

Specifically, we consider the following variants, given an update $u(P_d, P_i, P_w)$:

Sem$_0^{mat}$: as a baseline for a mat-preserving semantics, we apply the naïve semantics, followed by (re-)materialisation of the whole triple store.

Sem$_1^{mat}$: an alternative approach for a mat-preserving semantics is to follow the so-called "delete and rederive" algorithm [10] for deletions, that is: *(i)* delete the instantiations of P_d *plus "dangling" effects*, i.e., effects of deleted triples that after deletion are not implied any longer *by any non-deleted triples*; *(ii)* insert the instantiations of P_i *plus all their effects*.

Sem$_2^{mat}$: Another mat-preserving semantics could take a different viewpoint with respect to deletions, following the intention to: *(i)* delete the instantiations of P_d *plus all their causes*; *(ii)* insert the instantiations of P_i *plus all their effects*.

Sem$_3^{mat}$: Finally, a mat-preserving semantics could combine **Sem**$_1^{mat}$ and **Sem**$_2^{mat}$ deleting both causes of instantiations of P_d and (recursively) "dangling" effects.[10]

The definition of semantics **Sem**$_0^{mat}$ is straightforward.

Definition 6 (Baseline mat-preserving update semantics). *Let G and $u(P_d, P_i, P_w)$ be as in Def. 4. Then, $G^{\mathbf{Sem}_0^{mat}}_{u(P_d,P_i,P_w)} = mat(G_{u(P_d,P_i,P_w)})$.*

[9] Consider, e.g., the update from Ex. 4 on the materialised store from Ex. 3.

[10] Note the difference to the basic "delete and rederive" approach. **Sem**$_1^{mat}$ in combination with the intention of **Sem**$_2^{mat}$ would also mean to recursively delete effects of causes, and so forth.

Let us proceed with a "reality-check" on this semantics using our running example.

Example 5. Consider the update from Ex. 4. It is easy to see that under $\mathbf{Sem}_0^{mat}$ executed on the materialised triple store of Ex. 3, it would not have *any* effect. ∎

As this behaviour is quite arguable, let us proceed with discussing the proposed alternative mat-preserving update semantics, and how they could be implemented.

As for $\mathbf{Sem}_1^{mat}$, we rely on a well-known technique in the area of updates for deductive databases called "delete and rederive" (DRed) [6, 10, 16, 25]. Informally translated to our setting, when given a logic program Π and its materialisation T_Π^ω, plus set of facts A_d to be deleted and A_i to be inserted, DRed *(i)* first deletes A_d and all its effects (computed via semi-naive evaluation [24]) from T_Π^ω, resulting in $(T_\Pi^\omega)'$, *(ii)* then, starting from $(T_\Pi^\omega)'$, re-materialises $(\Pi \setminus A_d) \cup A_i$ (again using semi-naive evaluation).

The basic intuition behind DRed of deleting effects of deleted triples and then re-materialising can be expressed in our notation as follows; as we will consider a variant of this semantics later on, we refer to this semantics as $\mathbf{Sem}_{1a}^{mat}$.

Definition 7. *Let $G = \mathcal{T} \cup \mathcal{A}$, $u(P_d, P_i, P_w)$, $\mathcal{A}_d$, and $\mathcal{A}_i$ be defined as in Def. 4. Then $G_{u(P_d,P_i,P_w)}^{\mathbf{Sem}_{1a}^{mat}} = mat(\mathcal{T} \cup (\mathcal{A} \setminus mat(\mathcal{T} \cup \mathcal{A}_d)) \cup \mathcal{A}_i)$.*

As opposed to the classic DRed algorithm, where Datalog distinguishes between view predicates (IDB) and extensional knowledge in the Database (EDB), in our setting we do not make this distinction, i.e., we do not distinguish between implicitly and explicitly inserted triples. This means that $\mathbf{Sem}_{1a}^{mat}$ would delete also those effects that had been inserted explicitly before.

We introduce now a different variant of this semantics, denoted $\mathbf{Sem}_{1b}^{mat}$, that makes a distinction between explicitly and implicitly inserted triples.

Definition 8. *Let $u(P_d, P_i, P_w)$ be an update operation, and $G = \mathcal{T} \cup \mathcal{A}_{expl} \cup \mathcal{A}_{impl}$ a triple store, where $\mathcal{A}_{expl}$ and $\mathcal{A}_{impl}$ respectively denote the explicit and implicit ABox triples. Then $G_{u(P_d,P_i,P_w)}^{\mathbf{Sem}_{1b}^{mat}} = \mathcal{T} \cup \mathcal{A}'_{expl} \cup \mathcal{A}'_{impl}$, where $\mathcal{A}_d$ and $\mathcal{A}_i$ are defined as in Def. 4, $\mathcal{A}'_{expl} = (\mathcal{A}_{expl} \setminus \mathcal{A}_d) \cup \mathcal{A}_i$, and $\mathcal{A}'_{impl} = mat(\mathcal{A}'_{expl} \cup \mathcal{T}) \setminus \mathcal{T}$.*

Note that in $\mathbf{Sem}_{1b}^{mat}$, as opposed to $\mathbf{Sem}_{1a}^{mat}$, we do not explicitly delete effects of $\mathcal{A}_d$ from the materialisation, since the definition just relies on re-materialisation from scratch from the explicit ABox $\mathcal{A}'_{expl}$. Nonetheless, the original DRed algorithm can still be used for computing $\mathbf{Sem}_{1b}^{mat}$ as shown by the following proposition.

Proposition 2. *Let us interpret the inference rules in Fig. 1 and triples in G respectively as rules and facts of a logic program Π; accordingly, we interpret $\mathcal{A}_d$ and $\mathcal{A}_i$ from Def. 8 as facts to be deleted from and inserted into Π, respectively. Then, the materialisation computed by DRed, as defined in [16], computes exactly $\mathcal{A}'_{impl}$.*

None of $\mathbf{Sem}_0^{mat}$, $\mathbf{Sem}_{1a}^{mat}$, and $\mathbf{Sem}_{1b}^{mat}$ are equivalent, as shown in Ex. 6.

Example 6. Given the triple store $G = \{$`:C sc :D . :D sc :E`$\}$, on which we perform the operation `INSERT{:x a :C, :D, :E.}`, *explicitly* adding three

triples, and subsequently perform `DELETE{:x a :C, :E.}`, we obtain, according to the three semantics discussed so far, the following ABoxes:
$\mathbf{Sem}_0^{mat}$: `{:x a :D. :x a :E.}`, $\mathbf{Sem}_{1a}^{mat}$: `{}`,
$\mathbf{Sem}_{1b}^{mat}$: `{:x a :D. :x a :E.}`.

While after this update $\mathbf{Sem}_0^{mat}$ and $\mathbf{Sem}_{1b}^{mat}$ deliver the same result, the difference between these two is shown by the subsequent update `DELETE{:x a :D.}`:
$\mathbf{Sem}_0^{mat}$: `{:x a :E.}`, $\mathbf{Sem}_{1a}^{mat}$: `{}`, $\mathbf{Sem}_{1b}^{mat}$: `{}`. ■

As for the subtle difference between $\mathbf{Sem}_{1a}^{mat}$ and $\mathbf{Sem}_{1b}^{mat}$, we point out that none of [16, 25], who both refer to using DRed in the course of RDF updates, make it clear whether explicit and implicit ABox triples are to be treated differently.

Further, continuing with Ex. 5, the update from Ex. 4 still would not have *any* effect, neither using $\mathbf{Sem}_{1a}^{mat}$, nor $\mathbf{Sem}_{1b}^{mat}$. That is, it is not possible in any of these update semantics to remove implicit information (without explicitly removing all its causes).

$\mathbf{Sem}_2^{mat}$ aims at addressing this problem concerning the deletion of implicit information. As it turns out, while the intention of $\mathbf{Sem}_2^{mat}$ to delete causes of deletions cannot be captured just with the mat operator, it can be achieved fairly straightforwardly, building upon ideas similar to those used in query rewriting.

As we have seen, in the setting of RDFS we can use algorithm $PerfectRef$ to expand a CQ to a UCQ that incorporates all its "causes". A slight variation can be used to compute the set of all causes, that is, in the most naïve fashion by just "flattening" the set of sets returned by $PerfectRef$ to a simple set; we denote this flattening on a set S of sets as $flatten(S)$. Likewise, we can easily define a modified version of $mat(G)$, applied to a BGP P using a TBox $\mathcal{T}$[11]. Let us call the resulting algorithm $mat_{eff}(P, \mathcal{T})$[12]. Using these considerations, we can thus define both rewritings that consider all causes, and rewritings that consider all effects of a given (insert or delete) pattern P:

Definition 9 (Cause/Effect rewriting). *Given a BGP insert or delete template P for an update operation over the triple store $G = \mathcal{T} \cup \mathcal{A}$, we define the* all-causes-rewriting *of P as $P^{caus} = flatten(rewrite(P, \mathcal{T}))$; likewise, we define the* all-effects-rewriting *of P as $P^{eff} = mat_{eff}(P, \mathcal{T})$.*

This leads (almost) straightforwardly to a rewriting-based definition of $\mathbf{Sem}_2^{mat}$.

Definition 10. *Let $u(P_d, P_i, P_w)$ be an update operation. Then*

$$G^{\mathbf{Sem}_2^{mat}}_{u(P_d,P_i,P_w)} = G_{u(P_d^{caus}, P_i^{eff}, \{P_w\}\{P_d^{fvars}\})}$$

where $P_d^{fvars} = \{?x \text{ a rdfs:Resource.} \mid \textit{for each } ?x \in Var(P_d^{caus}) \setminus Var(P_d)\}$.

The only tricky part in this definition is the rewriting of the WHERE clause, where P_w is joined[13] with a new pattern P_d^{fvars} that binds "free" variables (i.e., the fresh variables

[11] This could be viewed as simply applying the first four inference rules in Fig. 1 exhaustively to $P \cup \mathcal{T}$, and then removing $\mathcal{T}$.

[12] Note that it is not our intention to provide optimised algorithms here, but just to convey the feasibility of this rewriting.

[13] A sequence of '{}'-delimited patterns in SPARQL corresponds to a join, where such joins can again be nested with UNIONs, with the obvious semantics, for details cf. [12].

denoted by '_' in *PerfectRef*) in the rewritten DELETE clause, P_d^{caus}. Here, `?x a rdfs:Resource.` is a shortcut for `{{?x ?x`$_p$` ?x`$_o$`} UNION {?x`$_s$` ?x ?x`$_o$`} UNION {?x`$_s$` ?x`$_p$` ?x}}`, which binds `?x` to any term occurring in G.

Example 7. Getting back to the materialised version of the triple store G from Ex. 3, the update u from Ex. 4 would, according to $\mathbf{Sem}_2^{mat}$, be rewritten to

```
DELETE {?X a :Child. ?X :hasF ?x1. ?X :hasM ?x2. ?X :hasP ?x3.}
INSERT {?Y a :Mother. ?Y a :Parent. }
WHERE  {{?X :hasM ?Y.}    {?x1 a rdfs:Resource.
        ?x2 a rdfs:Resource.  ?x3 a rdfs:Resource.}}
```

with $G_u^{\mathbf{Sem}_2^{mat}}$ containing `:jane a :Mother, :Parent. :jack a :Parent.` ■

It is easy to argue that $\mathbf{Sem}_2^{mat}$ is mat-preserving. However, this semantics might still result in potentially non-intuitive behaviours. For instance, subsequent calls of INSERTs and DELETEs might leave "traces", as shown by the following example.

Example 8. Assume $G = O_{fam}$ from Ex. 1 with an empty ABox. Under $\mathbf{Sem}_2^{mat}$, the following sequence of updates would leave as a trace —among others— the resulting triples as in Ex. 7, which would not be the case under the naïve semantics.

```
DELETE{} INSERT {:joe :hasM :jane; :hasF :jack} WHERE{};
DELETE {:joe :hasM :jane; :hasF :jack} INSERT{} WHERE{}
```
■

$\mathbf{Sem}_3^{mat}$ tries to address the issue of such "traces", but can no longer be formulated by a relatively straightforward rewriting. For the present, preliminary paper we leave out a detailed definition/implementation capturing the intention of $\mathbf{Sem}_3^{mat}$; there are two possible starting points, namely combining $\mathbf{Sem}_{1a}^{mat}+\mathbf{Sem}_2^{mat}$, or $\mathbf{Sem}_{1b}^{mat}+\mathbf{Sem}_2^{mat}$, respectively. We emphasise though, that independently of this choice, a semantics that achieves the intention of $\mathbf{Sem}_3^{mat}$ would still potentially run into arguable cases, since it might run into removing seemingly "disconnected" implicit assertions, whenever removed assertions cause these, as shown by the following example.

Example 9. Assume a materialised triple store G consisting only of the TBox triples `:Father sc :Person, :Male .` The behaviour of the following update sequence under a semantics implementing the intention of $\mathbf{Sem}_3^{mat}$ is arguable:

```
DELETE {} INSERT {:x a :Father.} WHERE {};
DELETE {:x a :Male.} INSERT {} WHERE {};
```

We leave it open for now whether "recursive deletion of dangling effects" is intuitive: in this case, should upon deletion of x being Male, also be deleted that x is a Person? ■

In a strict reading of $\mathbf{Sem}_3^{mat}$'s intention, `:x a :Person.` would count as a dangling effect of the cause for `:x a :Male.`, since it is an effect of the inserted triple with no other causes in the store, and thus should be removed upon the delete operation.

Lastly, we point out that while implementations of (materialised) triple stores may make a distinction between implicit and explicitly inserted triples (e.g., by storing explicit and implicit triples separately, as sketched in $\mathbf{Sem}_{1b}^{mat}$ already), we consider the distinction between implicit triples and explicitly inserted ones non-trivial in the context of SPARQL 1.1 Update: for instance, is a triple inserted based upon implicit bindings in the WHERE clause of an INSERT statement to be considered "explicitly inserted" or not? We tend towards avoiding such distinction, but we have more in-depth discussions of such philosophical aspects of possible SPARQL update semantics on our agenda.

4 TBox Updates

So far, we have considered the TBox as static. As already noted in [11], additionally allowing TBox updates considerably complicates issues and opens additional degrees of freedom for possible semantics. While it is out of the scope of this paper to explore all of these, we limit ourselves to sketch these different degrees of freedom and suggest one pragmatic approach to extend updates expressed in SPARQL to RDFS TBoxes.

In order to allow for TBox updates, we have to extend the update language: in the following, we will assume *general BGPs*, extending Def. 2.

Definition 11 (general BGP). *A* general BGP *is a set of triples of any of the forms from Table 1, where $x, y, A, A', P, P' \in \Gamma \cup \mathcal{V}$.*

We observe that with this relaxation for BGPs, updates as per Def. 3 can query TBox data, since they admit TBox triples in P_w. In order to address this issue we need to also generalise the definition of query answers.[14]

Definition 12. *Let Q be a union of general BGPs and $[\![Q]\!]_G$ the simple SPARQL semantics as per [21], i.e., essentially the set of query answers obtained as the union of answers from simple pattern matching of the general BGPs in Q over the graph G. Then we define $ans_{\mathsf{RDFS}}(Q, G) = [\![Q]\!]_{mat(G)}$*

In fact, Def. 12 does not affect ABox inferences, that is, the following corollary follows immediately from Prop. 1 for non-general UCQs as per Def. 2.

Corollary 1. *Let Q be a UCQ as per Def. 2. Then $ans_{\mathsf{RDFS}}(Q, G) = ans_{\mathsf{rdfs}}(Q, G)$*

As opposed to the setting discussed so far, where the last two rules in Fig. 1 used for TBox materialisation were ignored, we now focus on the discussion of terminological updates under the standard "intensional" semantics (essentially defined by the inference rules in Fig. 1) and attempt to define a reasonable (that means computable) semantics under this setting. Note that upon terminological queries, the RDFS semantics and DL semantics differ, since this"intensional" semantics does not cover all terminological inferences derivable in DL, cf. [7]; we leave the details of this aspect to future work.

Observation 1. TBox updates potentially affects materialisation of the ABox, that is, *(i)* upon TBox *insertions* a materialised ABox might need to be re-materialised in order to preserve materialisation. *(ii)* upon TBox *deletions* in a materialised setting, we have a similar issue to what we called "dangling" effects earlier.

Observation 2. Whereas deletions of implicit ABox triples can be achieved deterministically by deleting all single causes, TBox deletions involving `sc` and `sp` chains can be achieved in several distinct ways, as already observed by [11].

Example 10. Consider the graph $G = \{$`:A sc :B. :B sc :C. :B sc :D. :C sc :E. :D sc :E. :E sc :F.`$\}$ with the update `DELETE{:A sc :F.}`

Independent of whether we assume a materialised TBox, we would have various choices here to remove triples, to delete all the causes for `:A sc :F`. ■

[14] As mentioned in Fn. 7, elements of Γ may act as individuals, concept, or roles names in parallel.

In order to define a deterministic semantics for TBox updates, we need a canonical way to delete implicit and explicit TBox triples. Minimal cuts are suggested in [11] in the `sc` (or `sp`, resp.) graphs as candidates for deletions of `sc` (or `sp`, resp.) triples. However, as easily verified by Ex. 10, minimal multicuts are still ambiguous.

Here, we suggest two update semantics using rewritings to SPARQL 1.1 property path patterns [12] that yield canonical cuts.

Definition 13. *Let $u(P_d, P_i, P_w)$ be an update operation where P_d, P_i, P_w are general BGPs. Then $G^{\mathbf{Sem}^{mat}_{outcut}}_{u(P_d,P_i,P_w)} = mat(G_{u(P'_d,P_i,P'_w)})$, where each triple $\{A_1\ scp\ A_2\} \in P_d$ such that $scp \in \{\mathsf{sc}, \mathsf{sp}\}$ is replaced within P'_d by $\{A_1\ scp\ ?x.\}$, and we add to P'_w the property path pattern $\{A_1\ scp\ ?x.\ ?x\ scp^*\ A_2\}$. Analogously, $\mathbf{Sem}^{mat}_{incut}$ is defined by replacing $\{?x\ scp\ A_2\}$ within P'_d, and adding $\{A_1\ scp^*\ ?x.\ ?x\ scp\ A_2\}$ within P'_w instead.*

Both $\mathbf{Sem}^{mat}_{outcut}$ and $\mathbf{Sem}^{mat}_{incut}$ may be viewed as straightforward extensions of $\mathbf{Sem}^{mat}_0$, i.e., both are mat-preserving and equivalent to the baseline semantics for non-general BGPs (i.e., on ABox updates):

Proposition 3. *Let $u(P_d, P_i, P_w)$ be an update operation, where P_d, P_i, P_w are (non-general) BGPs. Then $G^{\mathbf{Sem}^{mat}_{outcut}}_{u(P_d,P_i,P_w)} = G^{\mathbf{Sem}^{mat}_{incut}}_{u(P_d,P_i,P_w)} = G^{\mathbf{Sem}^{mat}_0}_{u(P_d,P_i,P_w)}$*

The intuition behind $\mathbf{Sem}^{mat}_{outcut}$ is to delete for every deleted $A\ scp\ B.$ triple, all directly outgoing *scp* edges from A that lead into paths to B, or, resp., in $\mathbf{Sem}^{mat}_{incut}$ all directly incoming edges to B. This choice is motivated by the following proposition.

Proposition 4. *Let $u = \texttt{DELETE}\ \{A\ scp\ B\}$, and let G be a triple store with materialised TBox $\mathcal{T}$. Then, the TBox statements deleted by $G^{\mathbf{Sem}^{mat}_{outcut}}_{u(P_d,P_i,P_w)}$ (or, $G^{\mathbf{Sem}^{mat}_{incut}}_{u(P_d,P_i,P_w)}$, resp.) form a minimal cut [11] of $\mathcal{T}$ disconnecting A and B.*

The following example illustrates that the generalisation of Prop. 4 to updates involving the deletion of several TBox statements at once does not hold.

Example 11. Assume the materialised triple store $G = \{$`:A` *scp* `:B,:C,:D. :B` *scp* `:C, :D.`$\}$ and $u =$ `DELETE{:A` *scp* `:C. :A` *scp* `:D.`$\}$. Here, $\mathbf{Sem}^{mat}_{incut}$ does not yield a minimal *multicut* in G wrt disconnecting (`:A, :C`) and (`:A, :D`).[15] ■

As the example shows, the extension of the baseline ABox update semantics to TBox updates already yields new degrees of freedom. We leave a more in-depth discussion of TBox updates also extending the other semantics from Sec. 3 for future work.

5 Further Related Work and Possible Future Directions

Previous work on updates in the context of tractable ontology languages such as RDFS [11] and *DL-Lite* [5] typically has treated DELETEs and INSERTs in isolation, but

[15] An orthogonal example, where $\mathbf{Sem}^{mat}_{outcut}$ would not yield a minimal multicut can be constructed symmetrically.

not both at the same time nor in combination with templates filled by WHERE clauses, as in SPARQL 1.1; that is, these approaches are not based on BGP matching but rather on a set of ABox assertions to be updated known a priori. Pairing both DELETE and INSERT, as in our case, poses new challenges, which we tried to address here in the practically relevant context of materialised triple stores. In the future, we plan to extend our work in the context of *DL-Lite*, where we could build upon thoroughly studied query rewriting techniques (not necessarily relying on materialisation), and at the same time benefiting from a more expressive ontology language. Expanding beyond our simple minimal RDFS language towards more features of *DL-Lite* or coverage of unrestricted RDF graphs would impose new challenges: for instance, consistency checking and consistency-preserving updates (as those treated in [5]) do not yet play a role in the setting of RDFS; extensions in these directions as well as practically evaluating the proposed semantics on existing triple stores is on our agenda.

In the area of database theory, there has been a lot of work on updating logical databases: Winslett [26] distinguishes between model-based and formula-based updates; our approach clearly falls in the latter category, more concretely, ABox updates could be viewed as sets of propositional knowledge base updates [14] generated by SPARQL instantiating DELETE/INSERT templates. Let us further note that in the more applied area of databases, there are obvious parallels between some of our considerations and CASCADE DELETEs in SQL (that is, deletions under foreign key constraints), in the sense that we trigger additional deletions of causes/effects in some of the proposed update semantics discussed herein.

6 Conclusions

We have discussed the semantics of SPARQL 1.1 Update in the context of RDFS. To the best of our knowledge, this is the first work to discuss how to combine RDFS with the new SPARQL 1.1 Update language. While in this paper we have been operating on a very restricted setting (only capturing minimal RDFS entailments, restricting BGPs to disallow non-standard-use of the RDFS vocabulary), we could demonstrate that even in this setting the definition of a SPARQL 1.1 Update semantics under entailments is a non-trivial task. We proposed several possible semantics, neither of which might seem intuitive for all possible use cases; this might well suggest that there is no "one-size-fits-all" update semantics. Further, while ontologies should be "ready for evolution" [20], we believe that more work into semantics for updates of ontologies alongside with data (TBox & ABox) is still needed to ground research in *Ontology Evolution* into standards (SPARQL, RDF, RDFS, OWL), particularly in the light of the increased importance that RDF and SPARQL are experiencing in dynamic domains where also data is continuously updated (dealing with dynamics in Linked Data, querying sensor data, or stream reasoning). We have taken a first step in the present paper.

Acknowledgements. This work has been funded by the Vienna Science and Technology Fund (WWTF, project ICT12-015), by the Vienna PhD School of Informatics, and by EU Project Optique (grant n. FP7-318338).

References

1. Ahmeti, A., Polleres, A.: SPARQL update under RDFS entailment in fully materialized and redundancy-free triple stores. In: Proc. of the 2nd Int. Workshop on Ordering and Reasoning (OrdRing) (2013)
2. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle – Terse RDF Triple Language. W3C Recommendation, World Wide Web Consortium (Feb 2014), available at `http://www.w3.org/TR/turtle/`
3. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: A family of scalable semantic repositories. Semantic Web J. 2(1), 33–42 (2011)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
5. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of *DL-Lite* knowledge bases. In: Proc. of the 9th Int. Semantic Web Conf. (ISWC). pp. 112–128 (2010)
6. Ceri, S., Widom, J.: Deriving incremental production rules for deductive data. Information Systems 19(6), 467–490 (1994)
7. Franconi, E., Gutierrez, C., Mosca, A., Pirrò, G., Rosati, R.: The logic of extensional RDFS. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 8218, pp. 101–116. Springer (2013)
8. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 update. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at `http://www.w3.org/TR/sparql11-update/`
9. Glimm, B., Ogbuji, C.: SPARQL 1.1 entailment regimes. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at `http://www.w3.org/TR/sparql11-entailment/`
10. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data. pp. 157–166 (1993)
11. Gutierrez, C., Hurtado, C., Vaisman, A.: Updating RDFS: from theory to practice. In: Proc. of the 8th Extended Semantic Web Conf. (ESWC) (2011)
12. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at `http://www.w3.org/TR/sparql11-query`
13. Hayes, P.: RDF semantics. W3C Recommendation, World Wide Web Consortium (Feb 2004), available at `http://www.w3.org/TR/rdf-mt/`
14. Katsuno, H., Mendelzon, A.O.: A unified view of propositional knowledge base updates. In: Proc. of the 11th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 1413–1419 (1989)
15. Kontchakov, R., Rodriguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: A short course. In: Reasoning Web. Semantic Technologies for Intelligent Data Access – 9th Int. Summer School Tutorial Lectures (RW), LNCS, vol. 8067, pp. 194–229. Springer (2013)
16. Kotowski, J., Bry, F., Brodt, S.: Reasoning as axioms change - Incremental view maintenance reconsidered. In: Proc. of the 5th Int. Conf. on Web Reasoning and Rule Systems (RR). LNCS, vol. 6902, pp. 139–154. Springer (2011)
17. Mallea, A., Arenas, M., Hogan, A., Polleres, A.: On blank nodes. In: Proc. of the 10th Int. Semantic Web Conf. (ISWC). LNCS, vol. 7031, pp. 421–437. Springer (2011)
18. Motik, B.: On the properties of metamodeling in OWL. J. of Logic and Computation 17(4), 617–637 (2007)
19. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. In: Proc. of the 4th European Semantic Web Conf. (ESWC). pp. 53–67 (2007)

20. Noy, N.F., Klein, M.C.A.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems 6(4), 428–440 (2004)
21. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. on Database Systems 34(3), 16:1–16:45 (2009)
22. Polleres, A., Hogan, A., Delbru, R., Umbrich, J.: RDFS & OWL reasoning for linked data. In: Reasoning Web. Semantic Technologies for Intelligent Data Access – 9th Int. Summer School Tutorial Lectures (RW), Lecture Notes in Computer Science, vol. 8067, pp. 91–149. Springer (2013)
23. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium (Jan 2008), available at `http://www.w3.org/TR/rdf-sparql-query`
24. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. 1. Computer Science Press (1988)
25. Urbani, J., Margara, A., Jacobs, C.J.H., van Harmelen, F., Bal, H.E.: DynamiTE: Parallel materialization of dynamic RDF data. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 8218, pp. 657–672. Springer (2013)
26. Winslett, M.: Updating Logical Databases. Cambridge University Press (2005)