SPARQL Update under RDFS Entailment in Fully Materialized and Redundancy-Free Triple Stores

Albin Ahmeti[1] and Axel Polleres[2]

[1] Vienna University of Technology, Favoritenstraße 9, 1040 Vienna, Austria
[2] Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Abstract. Processing the dynamic evolution of RDF stores has recently been standardized in the SPARQL 1.1 Update specification. However, computing answers entailed by ontologies in triple stores is usually treated orthogonal to updates. Even the W3C's recent SPARQL 1.1 Update language and SPARQL 1.1 Entailment Regimes specifications explicitly exclude a standard behavior how SPARQL endpoints should treat entailment regimes other than simple entailment in the context of updates. In this paper, we take a first step to close this gap, by drawing from query rewriting techniques explored in the context of DL-Lite. We define a fragment of SPARQL basic graph patterns corresponding to (the RDFS fragment of) DL-Lite and the corresponding SPARQL update language discussing possible semantics along with potential strategies for implementing them. We treat both (i) reduced RDF Stores, that is, redundancy-free RDF stores that do not store any RDF triples (corresponding to DL Lite ABox statements) entailed by others already, and (ii) materialized RDF stores, which store all entailed triples explicitly.

1 Introduction

SPARQL provides a standard for accessing structured Data on the Web and the availability of such a standard may well be called one of the key factors to the success and increasing adoption of RDF and semantic technologies. Still, in its first iteration the SPARQL specification has neither defined any standard way how to treat entailments with respect to RDFS and OWL ontologies, nor provided standard means how to update dynamic RDF data. Both these gaps have been addressed within the recent SPARQL 1.1 specification, which both provides means to define query answers under ontological entailments (SPARQL 1.1 Entailment Regimes [6]) as well as an update language to update RDF data stored in a triple store (SPARQL 1.1 Update [5]).[3] Nonetheless, these specifications do not provide a standard behavior how SPARQL endpoints should treat entailment regimes other than simple entailment in the context of updates. The main issue of updates under entailments is how updates shall deal with implied statements:

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

[3] For the sake of simplicity we do not take into account NAMED graphs in this paper, which is why we speak of "triple stores" as opposed to "graph stores".

In the context of RDFS one might further ask about how to deal with axiomatic triples such as `rdf:type a rdf:Property.`, issues around the treatment of blank nodes [12], or, in the context of OWL, the treatment of inconsistencies arising through updates; for the sake of this present paper, we do not treat these additional issues separately, but focus on a deliberately minimal treatment of RDFS inferences along the lines of [14]. As it turns out, even in this confined setting, updates as defined in the SPARQL 1.1 Update specification impose non-trivial challenges; particularly, 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 been considered in the literature in combination as of yet.

Example 1. As a running example, let us assume a RDF triple store G, containing data about family relationships along with a RDFS ontology O_{fam} comprising the following axioms (in Turtle syntax).

```
:hasFather rdfs:subPropertyOf :hasParent.
:hasMother rdfs:subPropertyOf :hasParent.
:Father rdfs:subClassOf :Parent. :Mother rdfs:subClassOf :Parent.
:hasFather rdfs:range :Father; rdfs:domain :Child.
:hasMother rdfs:range :Mother; rdfs:domain :Child.
:hasParent rdfs:range :Parent; rdfs:domain :Child.
```

Assuming now that the following data assertions are also in the RDF Store G

```
:joe :hasParent :jack.
:joe :hasMother :jane.
```

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

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

A SPARQL endpoint that only considers simple entailment though, would only return `:jack` as an answer.

The intended behavior for the query in Example 1 is typically achieved by either (i) computing entailed answers at query runtime, which can actually be achieved by query rewriting techniques, or (ii) by materializing all implied triples in the store, normally at loading time.

That is, on the one hand, borrowing from query rewriting techniques from DL-Lite (such as e.g. PerfectRef [3], several refinements of which have been proposed in the literature) one can reformulate such a query to return also the implied answers. A minimalist query reformulation algorithm – that is, a down-stripped version of PerfectRef [3], covering the essential entailments of RDFS – is shown in Alg. 1.

Example 2 (cont'd). If we interpret the `WHERE` clause in the query in Example 1 as a conjunctive query, and the RDFS axioms from O_{fam} as DL TBox, the resulting UCQ as from Alg. 1 can be just translated back into SPARQL as follows.

```
SELECT ?Y WHERE {{ :joe :hasParent ?Y. } UNION
          { :joe :hasFather ?Y. } UNION { :joe :hasMother ?Y. }}
```

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

While the rewritten query is exponential in the worst case wrt. the size of the ontology, for moderate size ontologies, such a rewriting approach is quite feasible.

On the other hand, an alternative is to materialize all inferences (for instance produced by the minimalist inference rules set for RDFS from [14]) in the triple store, such that the original query can be used 'as is'.

Example 3 (cont'd). The materialized version of G would contain the following triples – for conciseness we only show assertional implied triples here, that is triples from rules $2)b)$, $3)b)$, $4)a)$+$b)$ from [14].

```
:joe :hasParent :jack. :joe a :Child. :jack a :Parent.
:joe :hasMother :jane; :hasParent :jane. :jane a :Mother, :Parent.
```

Obviously, on such an endpoint, the original query from Example 1 would already return the expected results.

However, when it comes to *updating* the RDF store in terms of SPARQL 1.1 Update, things become less clear.

Example 4 (cont'd). The following operation tries to delete an implied class membership whereas it tries to (re-)insert another implied class membership.

```
DELETE{?X a :Child.} INSERT {?Y a :Mother.}
WHERE {?X :hasMother ?Y.}
```

Various existing triple stores offer different solutions to these problems, ranging from – probably most commonly – ignoring entailments in updates altogether to keeping explicit and implicit triples separate and recomputing all entailments upon updates. That is, in the former case (ignoring entailments) updates only refer to explicitly asserted information, which may result in non-intuitive behaviors, whereas the latter case (re-materialization) 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. As an example, take for instance

Algorithm 1: rewrite

Input: Conjunctive query q, TBox $\mathcal{T}$
Output: Union (set) of conjunctive queries

$P := \{q\}$
repeat
  $P' := P$
  foreach $q \in P'$ **do**
    foreach g *in* q **do** // expansion
      foreach *inclusion axiom* I *in* $\mathcal{T}$ **do**
        if I *is applicable to* g **then**
          $P := P \cup \{q[g/\,\mathrm{gr}(g, I)]\}$
until $P' = P$
return P

Table 1. DL-Lite axioms vs. RDF(S)

	DL_{RDFS}	RDFS
1	$A_1 \sqsubseteq A_2$	$\mathsf{A_1}$ rdfs:subClassOf $\mathsf{A_2}$.
2	$\exists P \sqsubseteq A$	P rdfs:domain A.
3	$\exists P^- \sqsubseteq A$	P rdfs:range A.
4	$P_1 \sqsubseteq P_2$	$\mathsf{P_1}$ rdfs:subPropertyOf $\mathsf{P_2}$.
5	$A(x)$	x rdf:type A.
6	$P(x, y)$	x P y.

Table 2. Semantics of $\mathrm{gr}(g, I)$ in Alg.1

g	I	$\mathrm{gr}(g/I)$
$A(x)$	$B \sqsubseteq A$	$B(x)$
$A(x)$	$\exists P \sqsubseteq A$	$P(x, \_)$
$A(x)$	$\exists P^- \sqsubseteq A$	$P(\_, x)$
$P_1(x, y)$	$P_2 \sqsubseteq P_1$	$P_2(x, y)$

Here, '_' stands for a "fresh" variable.

DBpedia which stores certain entailed triples, but not all (cf. the example in [16]). In this paper we try to argue for a more systematic approach for dealing with updates in the context of RDFS entailments. More specifically, we will distinguish between two kinds of triple stores, that is (i) reduced RDF Stores, that is, redundancy-free RDF stores that do not store any assertional (ABox) triples entailed by others already, and (ii) materialized RDF stores, which store all entailed ABox triples explicitly. Next, we propose alternative update samentics that preserve the respective types (i) and (ii) of triple stores, and discuss possible implementation strategies, partially inspired by query rewriting techniques that have become popular in the context of ontology-based data access (OBDA) [11] and DL-Lite [3].

For the sake of this paper we will assume RDFS ontologies to be static and only assertional statements (ABoxes) to be updated. Along these lines, we will define a fragment of the SPARQL 1.1 Update language, that allows to Add/Delete assertional ABox statements, but not terminological statements (using the RDFS vocabulary). As already shown in [8], erasure of ABox statements is deterministic in the context of RDFS, but insertion and particularly the interplay of DELETE/INSERT in SPARQL 1.1 Update has not been considered therein.

The remainder of this paper continues with preliminaries (RDFS, SPARQL, DL-Lite, SPARQL update operations) in Section 2. We then present alternative update semantics along with possible implementation strategies in Section 3, and subsequently discuss future and related work (Section 4) before we conclude (Section 5).

2 Preliminaries

Since we will draw from ideas coming from OBDA and DL-Lite, we will use RDF graphs and RDFS ontologies compatible with DL-Lite ontologies in the present paper.

Definition 1 (RDFS ontology, ABox, TBox, triple store). *Let A be an atomic concept (or class, resp.) name and P be an atomic role (or, property, resp.) name, and Γ be a set of constants which in context of RDF(S) 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.*[4] *Let $x, y \in \Gamma$, then we call a set of inclusion axioms of the forms 1-4 in Tab. 1 a* RDFS ontology*, or* (RDFS) TBox*, whereas we call a set of assertions of the forms 5+6 in Tab. 1 an* (RDF) ABox*, where we view RDF and DL notation interchangeably. Finally, we call a container $G = \mathcal{T} \cup \mathcal{A}$ holding a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ an* (RDFS) triple store*.*

That is, more informally, we treat any RDF graph consisting of triples without non-standard RDFS vocabulary use as a pair of ABox and TBox triples, where, for the purposes of this paper, we will consider the latter fixed/immutable. Further to this, we will treat conjunctive queries (with only distinguished variables) and SPARQL basic graph patterns (BGPs) interchangeably in this paper as follows. Note that we do not provide details on more complex patterns (OPTIONAL, NOT EXISTS, FILTER, etc.) in

[4] That is, we assume no "non-standard use" [16] of these vocabularies. We may also assume concept names, role names and constants mutually disjoint (or distinguish them in the sense of "punning" [13] based on their position in atoms or RDF triples).

SPARQL 1.1 which are defined on top of BGP matching, but refer the reader to [9] for details on the semantics of these patterns.

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 from Tab. 1, where x, y are either constants from Γ or variables (written as alphanumeric strings preceded by '?'). A* union of conjunctive queries *(UCQ) Q, or UNION pattern, is a set of conjunctive queries. By $Var(q)$ (or $Var(Q)$, resp.) we denote the set of variables occurring in q (or Q, resp.).*

This definition allows only restricted forms of general SPARQL BGPs that correspond to conjunctive queries along the lines of Description Logics; that is, we do not allow, e.g., queries with variables in predicate positions nor "terminological" queries, e.g. `{?X rdfs:subClassOf ?Y.}`. We neither consider blank nodes separately in this paper.[5] 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 Description Logics) as follows.

Definition 3 (Interpretation, satisfaction, model). *An* interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ *consists of a non-empty set $\Delta^{\mathcal{I}}$ called the object domain, and an interpretation function $\cdot^{\mathcal{I}}$ which maps*

- *each atomic concept A to a subset of the domain $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,*
- *each atomic role P to a binary relation over the domain $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,*
- *each element of Γ to an element of $\Delta^{\mathcal{I}}$.*

For concept descriptions the interpretation function is defined as follows:

- $(\exists R)^{\mathcal{I}} = \left\{ x \in \Delta^{\mathcal{I}} \,\middle|\, \exists y.(x,y) \in R^{\mathcal{I}} \right\}$
- $(\exists R^-)^{\mathcal{I}} = \left\{ y \in \Delta^{\mathcal{I}} \,\middle|\, \exists x.(x,y) \in R^{\mathcal{I}} \right\}$

An interpretation $\mathcal{I}$ satisfies *an inclusion axiom $E_1 \sqsubseteq E_2$ (of one of the forms 1-4 in Tab. 1) if $E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$. Analogously, $\mathcal{I}$ satisfies an ABox assertion of the form*

- *$A(x)$ if $x^{\mathcal{I}} \in A^{\mathcal{I}}$*
- *$P(x,y)$ if $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in P^{\mathcal{I}}$*

Finally, an interpretation $\mathcal{I}$ is called a model *of a triple store $G = \mathcal{T} \cup \mathcal{A}$, denoted $\mathcal{I} \models G$, if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$, that is, if $\mathcal{I}$ satisfies all terminological axioms in $\mathcal{T}$ and all ABox assertions in $\mathcal{A}$.*

Definition 4 (Query answers). *For a CQ q (or, UCQ Q, resp.) and a triple store G, a substitution θ from variables in $Var(q)$ to constants in Γ such that $q\theta$ is true (or, there exists a $q \in Q$ with $q\theta$ is true) in every model of G is called an* answer *(under RDFS Entailment) to q, and we denote the set of all answers by $ans_{\text{RDFS}}(q, G)$ (or, $ans_{\text{RDFS}}(Q, G)$, resp.).*

Since we only treat RDFS entailment, but no other entailment regimes in this paper, in the following we will drop the subscript and simply write ans instead of ans_{RDFS}.

The following results follow immediately from e.g. [8, 14] & [3] and show that query answering under RDF can be done by either query rewriting or materialization.

[5] 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.

Proposition 1. *Let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store, q be a CQ. Further, let $rewrite(q, \mathcal{T})$ be as per Alg. 1 above, whereas by $materialize(G) = \mathcal{T} \cup \mathcal{A}'$ we denote the triple store obtained from exhaustive application of the following inference rules on G:*[6]

$$\frac{\texttt{?P rdfs:subPropertyOf ?Q. ?S ?P ?O.}}{\texttt{?S ?Q ?O.}} \qquad \frac{\texttt{?C rdfs:subClassOf ?D. ?S rdf:type ?C.}}{\texttt{?S rdf:type ?D.}}$$

$$\frac{\texttt{?P rdfs:domain ?C. ?S ?P ?O.}}{\texttt{?S rdf:type ?C.}} \qquad \frac{\texttt{?P rdfs:range ?C. ?S ?P ?O.}}{\texttt{?O rdf:type ?C.}}$$

Then, $ans(q, G) = ans(rewrite(q, \mathcal{T}), G \setminus \mathcal{T}) = ans(q, materialize(G) \setminus \mathcal{T})$.

Various existing triple stores (such as e.g. BigOWLIM [2]) have the option to perform materialization directly upon loading data. Accordingly, we will call *materialized triple stores* all triple stores that in each state always guarantee $G = materialize(G)$. On the other extreme, this suggests alternatively to discuss triple stores that do not store *any* redundant ABox triples. By $reduce(G)$ we will denote the hypothetical operator that produces the reduced "core" of G, and we call a triple store *reduced* if $G = reduce(G)$. While we do not provide the algorithm to compute $reduce(G)$, we note that this core is uniquely determined in our setting whenever $\mathcal{T}$ is acyclic (which is usually a safe assumption)[7]; it could be naïvely computed by exhaustively "marking" each triple that can be inferred from applying any of the rules in Prop. 1 and subsequently removing all marked elements of $\mathcal{A}$. The following observation follows trivially.

Lemma 1. *Let $G = \mathcal{T}$ be a triple store with an empty ABox, then G is both reduced and materialized.*

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

Definition 5 (SPARQL update operation). *Let P_d, 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*

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

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 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 $\mathcal{A}_d$ and $\mathcal{A}_i$ to be deleted from and inserted into G, respectively. A naïve update semantics follows straightforwardly.

Definition 6 (Naïve update semantics). *Let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store, $u(P_d, P_i, P_w)$ be an update operation, then $G_{u(P_d,P_i,P_w)} = (G \setminus \mathcal{A}_d) \cup \mathcal{A}_i$, where*

$$\mathcal{A}_d = \bigcup_{\theta \in ans(P_w, G)} P_d\theta \quad \textit{and} \quad \mathcal{A}_i = \bigcup_{\theta \in ans(P_w, G)} P_i\theta \ .$$

As easily seen, this naïve semantics neither preserves materialized nor reduced triple stores; we leave it to the reader to confirm this observation on the update from Example 4 both on the reduced triple store from Example 1 and, resp., on the materialized triple store from Example 3.

[6] These rules correspond to rules 2)b), 3)b), 4)a)+b) from [14]; since we neither consider blank nodes nor terminological inferences, we do not need the further rules from [14].

[7] We note that even in the case when the TBox is cyclic we could define a deterministic way to remove redundancies, e.g. by preserving the lexicographically smallest ABox statements only within a cycle. That is, given TBox $A \sqsubseteq B \sqsubseteq C \sqsubseteq A$ and ABox $A(x), C(x)$; we would delete $C(x)$ and retain $A(x)$ only, to preserve reducedness.

3 Materialized/Reduced Preserving Update Semantics

In this section, we will investigate alternative update semantics under RDFS entailment that either preserve materialized or reduced stores and discuss in how far these semantics can – similar to query answering – be implemented in off-the-shelf SPARQL 1.1 triple stores by simple rewriting techniques. We will look into the following possible semantics:

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

Sem$_1^{mat}$: Another materialized-preserving semantics could follow the intention to
- delete the instantiations of P_d *plus all their causes*;
- insert the instantiations of P_i *plus all their effects*.

Sem$_2^{mat}$: This semantics is also materialized-preserving but extends **Sem$_1^{mat}$** with the intention to additionally (recursively) delete "dangling" effects for instantiations of P_d, that is, effect of to-be-deleted triples that would not be implied any longer *by any non-deleted triples* after deletion.

Sem$_0^{red}$: Again, the baseline for a reduced-preserving semantics would be to apply the naïve semantics, followed by (re-)reducing the triple store.

Sem$_1^{red}$: This reduced-preserving semantics extends **Sem$_0^{red}$** by additionally deleting the causes of instantiations of P_d.

The definitions of semantics **Sem$_0^{mat}$** and **Sem$_0^{red}$** are straightforward.

Definition 7 (Baseline materialized- and reduced-preserving update semantics). *Let G and $u(P_d, P_i, P_w)$ be as in Def. 6 above. Then, analogously to Def. 6, we define **Sem**$_0^{mat}$ as*

$$G^{\textbf{\textit{Sem}}_0^{mat}}_{u(P_d,P_i,P_w)} = materialize(G_{u(P_d,P_i,P_w)})$$

*and **Sem**$_0^{red}$ as*

$$G^{\textbf{\textit{Sem}}_0^{red}}_{u(P_d,P_i,P_w)} = reduce(G_{u(P_d,P_i,P_w)})$$

Let us proceed with a quick "reality-check" on these two baseline semantics by means of our running example.

Example 5. By referring back to the update from Example 4: as easily seen, neither **Sem$_0^{mat}$** (executed on the materialized triple store of Example 3), nor **Sem$_0^{red}$** (executed on the reduced triple store of Example 1) would have *any* effect.

As this behavior is quite arguable, which is why we will proceed with discussing how the other intentions could be implemented or, resp., what implications such alternative update semantics would have.

Alternative Materialized-Preserving Semantics. Let us first look into materialized-preserving semantics in more detail. As it turns out, **Sem$_1^{mat}$** can be achieved fairly straightforwardly, building upon similar rewriting ideas as query rewriting. However, as we will argue, there might be reasons why one wants to adopt the other semantics.

As we have seen, in the setting of RDFS we can use Alg. 1 rewrite to expand a CQ q to a UCQ that considers all its "causes". A slight variation can of course 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 Alg. 1 to a simple set; we denote this flattening operation on a set of sets S as $flatten(S)$.

Likewise, we can easily define a "turned around" rewriting that computes all the "effects" of a pattern by modifying Alg. 1 such that it simply considers Table 2 with the first and third column "swapped" (and '_' just matching any variable or constant).[8] Let us call the resulting algorithm $rewrite_{eff}$. Likewise, we could have defined $rewrite_{eff}$ algorithm as a modification of $materialize(G)$ applied to BGPs instead of to the graph itself.[9] For the sake of completeness, in the same sense we can also analogously to $reduce(G)$ define an algorithm $reduce(P, \mathcal{T})$ which takes a pattern as input and reduces it wrt. a TBox $\mathcal{T}$.

Using these considerations, we can thus define both rewritings that consider all causes, as well as rewritings that consider all effects:

Definition 8 (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 $P^{eff} = flatten(rewrite_{eff}(P, \mathcal{T}))$.*

This leads (almost) straightforwardly to a rewriting-based definition of $\mathbf{Sem}_1^{mat}$:

Definition 9. *Let $u(P_d, P_i, P_w)$ be an update operation, then*

$$G_{u(P_d,P_i,P_w)}^{\mathbf{Sem}_1^{mat}} = G_{u(P_d^{caus},P_i^{eff},\{rewrite(P_w)\}\{P_d^{fvars}\})}$$

where $P_d^{fvars} = \{$ `?x a rdfs:Resource.` $|$ *for each* `?x` $\in Var(P_d^{caus_{P_d}}) \setminus Var(P_d)\}$.

The only tricky part in this definition is the rewriting of the WHERE clause, where $rewrite(P_w)$ is joined[10] with a new pattern P_d^{fvars} that binds the "free" variables (i.e., the new variables denoted by '_' in Tab. 2, which were introduced by Alg. 1) in the rewritten DELETE clause, P_d^{caus}. Here `?x a rdfs:Resource.` stands as a shortcut for the UCQ `{{?x ?x`$_p$ `?x`$_o$`} UNION {?x`$_s$ `?x ?x`$_o$`} UNION {?x`$_s$ `?x`$_p$ `?x}}` which binds any term occurring in the triple store.

Example 6. Getting back to the materialized version of the triple store from Example 3, the update from Example 4 would, according to $\mathbf{Sem}_1^{mat}$ be rewritten to

```
DELETE { ?X a :Child. ?X :hasFather ?x1. ?X :hasMother ?x2.
         ?X :hasParent ?x3. }
INSERT { ?Y a :Mother. ?Y a :Parent. }
WHERE  { { ?X :hasMother ?Y. }
         { ?x1 a rdfs:Resource. ?x2 a rdfs:Resource.
           ?x3 a rdfs:Resource.} }
```

[8] We note that this could be viewed equivalently as simply applying the inference rules from Prop. 1 exhaustively to q.

[9] Please note that it is not our intention to provide optimized algorithms here, but just to convey the feasibility of this rewriting.

[10] 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. [9].

with the resulting triple store containing the following triples.

```
:jane a :Mother, :Parent. :jack a :Father, :Parent.
```

It is easy to argue that $\mathbf{Sem}_1^{mat}$ is materialized-preserving. However, this semantics might still result in potentially non-intuitive behaviors. For instance, in this semantics the following subsequent calls of INSERTs and DELETEs might leave "traces" as shown by the following example.

Example 7. Assuming $G = O_{fam}$ from Example 1 with an empty ABox. Under $\mathbf{Sem}_1^{mat}$ semantics the following sequence of updates would leave as a trace the resulting triples as in Example 6, which would not be the case under the naïve semantics.

```
DELETE{} INSERT {:joe :hasMother :jane;:hasFather :jack} WHERE{};
DELETE {:joe :hasMother :jane;:hasFather :jack} INSERT{} WHERE{};
```

$\mathbf{Sem}_2^{mat}$ tries to address the issue of such "traces", but can no longer be formulated by such a relatively straightforward rewriting. For the present, preliminary paper we leave out a detailed definition/implementation capturing the intention of $\mathbf{Sem}_2^{mat}$; an obvious starting point would be the "Delete and Rederive" algorithm [7], but whether $\mathbf{Sem}_2^{mat}$ can be implemented by rewriting to SPARQL update operations following the naïve semantics is open. We emphasize though, that a semantics that achieves the intention of $\mathbf{Sem}_2^{mat}$ would still potentially run into arguable cases, since it might run into removing explicitly added assertions, whenever removed assertions cause these, as shown by the following example.

Example 8. Assuming a materialized-preserving update semantics implementing the intention of $\mathbf{Sem}_2^{mat}$, the behaviour of the following update sequence is left open:

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

Note that the second and third update operation in this example show that even insertion and immediate subsequent deletion of a single triple is not idempotent for $\mathbf{Sem}_2^{mat}$.

In a strict reading of $\mathbf{Sem}_2^{mat}$'s intention, `:x a :Parent.` would count as a dangling effect, since it is an effect of the triple in the second insertion with no other causes in the triple store, and thus should be removed upon the third update operation. Nonetheless, implementations of (materialized) triple stores may make a distinction between implicit and explicitly inserted triples (e.g. by storing explicit and implicit triples in separate graph); however, 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? The authors of the present paper tend towards avoiding such distinction, but we have more discussion of such rather philosophical aspects of possible SPARQL update semantics on our agenda; for now, let us rather turn our attention to the potential alternatives for reduced-preserving semantics.

Alternative Reduced-Preserving Semantics. Again, similar to $\mathbf{Sem}_2^{mat}$, for both the baseline semantics $\mathbf{Sem}_0^{red}$ and $\mathbf{Sem}_1^{red}$ we leave it open whether it can be implemented by rewriting to SPARQL update operations following the naïve semantics, i.e., without the need to apply $reduce(G)$ over the whole triple store after each update; a strategy to avoid calling $reduce(G)$ would roughly include the following steps:

– delete the instantiations P_d *plus all the effects of instantiations* of P_i, which will be implied anyways upon the new insertion, thus preserving reduced;
– insert instantiations of P_i only if they are *not implied*, that is, neither already implied by the current state of G or all their causes in G were to be deleted.

We leave further investigation of whether these steps can be cast into one or several update requests directly by rewriting techniques to future work.

Rather, we show that we can capture the intention of $\mathbf{Sem}_1^{red}$ by a relatively straightforward extension of the baseline semantics.

Definition 10 ($\mathbf{Sem}_1^{red}$). *Let $u(P_d, P_i, P_w)$ be an update operation, then*

$$G^{\mathbf{Sem}_1^{red}}_{u(P_d,P_i,P_w)} = reduce(G_{u(P_d^{caus},P_i,\{rewrite(P_w)\}\{P_d^{fvars}\})})$$

where P_d^{caus} and P_d^{fvars} are as before.

Example 9. Getting back to the reduced version of the triple store from Example 1, the update from Example 4 would, according to $\mathbf{Sem}_1^{red}$ be rewritten to

```
DELETE { ?X a :Child. ?X :hasFather ?x1. ?X :hasMother ?x2.
         ?X :hasParent ?x3. }
INSERT { ?Y a :Mother. }
WHERE  { { ?X :hasMother ?Y. }
         { ?x1 a rdfs:Resource. ?x2 a rdfs:Resource.
           ?x3 a rdfs:Resource.} }
```

with the resulting triple store after (re-)reduction containing the following triple.

```
:jane a :Mother.
```

Note that in a reduced store there is no need to delete effects of P_d, which make the considerations that lead us to $\mathbf{Sem}_2^{mat}$ irrelevant for a reduced-preserving semantics, as shown in following example.

Example 10. Under $\mathbf{Sem}_1^{red}$, as opposed to $\mathbf{Sem}_1^{mat}$, the update sequence of Example 7 would leave no traces. However, note that the deletion of `:joe :hasParent :jack` in Example 9 might be viewed as non-intuitive, plus the update sequence of Example 8 would likewise result in an empty ABox, thus neither guaranteeing idempotence of single triple insertion followed by immediate deletion.

We finally observe that the rewriting for $\mathbf{Sem}_1^{red}$ is almost identical to $\mathbf{Sem}_1^{mat}$, but reduced-preservation depends on a separate post-processing step not readily available in off-the-shelf triple stores, whereas $\mathbf{Sem}_1^{mat}$ could be directly executed by rewriting on any triple store, preserving materialization.

4 Further Related Work and Possible Future Directions

There has already been work on updates over TBox-es in the context of tractable ontologies such as RDFS [8] and DL-Lite [4]. Even though such approaches give deterministic answers in general while computing updates, they are either limited to

DELETEs or INSERTs, but 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 fill the gap by introducing different rewriting semantics and taking into account both materialized and reduced stores. In the future, we plan to extend our work in the context of DL-Lite where we could reuse off-the-shelf, thoroughly studied rewriting techniques, and at the same time benefiting from a more expressive query language. Expanding beyond the simple language at the intersection of RDFS and DL-Lite towards more features of DL-Lite or coverage of less restricted RDF graphs and BGPs would impose new challenges: for instance, consistency checking and consistency-preserving updates (such as treated in [4]) do not yet play a role in the setting of RDFS; we are looking forward to investigate extensions in these directions. Besides, we are planning to implement all the proposed semantics and to test them against existing triple stores.

As for further related works, in the context of reduced stores, let us also refer to [15], where the cost of redundancy elimination under various (rule-based) entailment regimes – including RDFS – is discussed in detail. In the area of database theory, there has been a lot of work on updating logical databases: Winslett [17] distinguishes between model-based and formula-based updates; our approach clearly falls in the latter category, more concretely, it could be viewed as sets of propositional knowledge base updates [10] generated by SPARQL instantiating DELETE/INSERT templates. Moreover, in the domain of updates over views, the "Delete and Rederive" algorithm [7] is closely related to our discussion about "dangling" effects (particularly, to our proposed semantics $\mathbf{Sem}_2^{mat}$) where the tuples to be deleted from (recursive) materialized views are determined if only there are no tuples in relational stores that map to the corresponding tuples in the view, i.e. there is no alternative derivation for the tuples in the view; as mentioned above, it is on our agenda to investigate whether such behavior can be achieved in terms of update-rewriting techniques rather than full re-derivations. 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 updated semantics discussed herein. Finally, it is worth mentioning that similar work has been done also in classical AI, e.g., in [1], where the main idea is to choose the maximal subset of propositions from a set that do not imply a certain proposition (which is going to be deleted), hereby coinciding with the delete rewriting of our $\mathbf{Sem}_1^{mat}$ semantics.

5 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 in depth how to combine SPARQL 1.1 Entailment Regimes with the new SPARQL 1.1 Update language. While we acknowledge that our results are at a preliminary stage and we have been operating on a very restricted language in this paper, we could demonstrate that even in such a restricted setting (only capturing minimal RDFS entailments, only allowing ABox updates, restricting BGPs) the definition of a SPARQL 1.1 Update semantics under

entailments is a non-trivial task. While we proposed several possible semantics, neither of those might seem intuitive for all possible use cases. So, while a "one-size-fits-all" update semantics might not exist, we believe that this is a timely topic that deserves increased attention. Particularly, in the light of the increased importance that RDF and SPARQL are experiencing also in dynamic domains where data is continuously updated (dealing with dynamics in Linked Data, querying sensor data or stream reasoning) further research in this area is needed.

Acknowledgments This work has been funded by the Vienna Science and Technology Fund (WWTF, project ICT12-015), and by the Vienna PhD School of Informatics.

References

1. Alchourron, C.E., Gardenfors, P., Makinson, D.: On the logic of theory change: Contraction functions and their associated revision functions. Theoria 48, 14–37 (1982)
2. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: A family of scalable semantic repositories. Semantic Web 2(1), 33–42 (2011)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
4. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of DL-Lite knowledge bases. In: ISWC. pp. 112–128 (2010)
5. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 Update (Mar 2013), W3C Recommendation
6. Glimm, B., Ogbuji, C., Hawke, S., Herman, I., Parsia, B., Polleres, A., Seaborne, A.: SPARQL 1.1 Entailment Regimes (Mar 2013), W3C Recommendation
7. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In: ACM SIGMOD Int'l Conf. on Management of Data. pp. 157–166. ACM (1993)
8. Gutierrez, C., Hurtado, C., Vaisman, A.: Updating RDFS: from theory to practice. In: 8th Extended Semantic Web Conf. (ESWC 2011) (2011)
9. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (Mar 2013), W3C Recommendation
10. Katsuno, H., Mendelzon, A.O.: A unified view of propositional knowledge base updates. In: IJCAI. pp. 1413–1419 (1989)
11. Kontchakov, R., Rodriguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: A short course. In: Reasoning Web 2013, LNCS, vol. 8067, pp. 194–229. Springer (2013)
12. Mallea, A., Arenas, M., Hogan, A., Polleres, A.: On Blank Nodes. In: Proceedings of the 10th International Semantic Web Conference (ISWC 2011). LNCS, vol. 7031. Springer (Oct 2011)
13. Motik, B.: On the Properties of Metamodeling in OWL. Journal of Logic and Computation 17(4), 617–637 (2007)
14. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. In: ESWC. pp. 53–67 (2007)
15. Pichler, R., Polleres, A., Skritek, S., Woltran, S.: Complexity of redundancy detection on RDF graphs in the presence of rules, constraints, and queries. Semantic Web – Interoperability, Usability, Applicability 4(4) (2013)
16. Polleres, A., Hogan, A., Delbru, R., Umbrich, J.: RDFS & OWL reasoning for linked data. In: Reasoning Web 2013, LNCS, vol. 8067, pp. 91–149. Springer (Jul 2013)
17. Winslett, M.: Updating Logical Databases. Cambridge University Press (2005)