Handling Inconsistencies Due to Class Disjointness in SPARQL Updates

Albin Ahmeti[1,3(✉)], Diego Calvanese[2], Axel Polleres[3], and Vadim Savenkov[3]

[1] Vienna University of Technology, Favoritenstraße 9, 1040 Vienna, Austria
albin.ahmeti@gmail.com
[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. The problem of updating ontologies has received increased attention in recent years. In the approaches proposed so far, either the update language is restricted to sets of ground atoms or, where the full SPARQL update language is allowed, the TBox language is restricted so that no inconsistencies can arise. In this paper we discuss directions to overcome these limitations. Starting from a DL-Lite fragment covering RDFS and concept disjointness axioms, we define three semantics for SPARQL instance-level (ABox) update: under cautious semantics, inconsistencies are resolved by rejecting updates potentially introducing conflicts; under brave semantics, instead, conflicts are overridden in favor of new information where possible; finally, the fainthearted semantics is a compromise between the former two approaches, designed to accommodate as much of the new information as possible, as long as consistency with the prior knowledge is not violated. We show how these semantics can be implemented in SPARQL via rewritings of polynomial size and draw first conclusions from their practical evaluation.

1 Introduction

RDF has become one of the most important data formats for interoperability, knowledge representation and querying. SPARQL, the W3C standardized language for managing RDF data [11], has grown to offer great power and flexibility of querying, including support for efficient reasoning, rooted in more than a decade of intensive research in description logics. With respect to updates however, SPARQL is currently far less mature. In particular, the interplay between updates and reasoning remains completely open.

In [1], we discussed semantics of SPARQL updates for RDFS ontologies, for the cases in which the knowledge base ABox is fully materialized or to the contrary, is reduced to its minimal core that cannot be derived using TBox axioms. The present paper continues this study of SPARQL updates focusing on the role of *inconsistency* in supporting SPARQL ABox updates over materialized stores. As a minimalistic ontology language allowing for inconsistencies, we consider RDFS$_\neg$, an extension of RDFS [12] with *class disjointness axioms* of the form $\{P$ `disjointWith` $Q\}$ from OWL [16].

H. Sack et al. (Eds.): ESWC 2016, LNCS 9678, pp. 387–404, 2016.
DOI: 10.1007/978-3-319-34129-3_24

As a running example, we assume a triple store G with an RDFS$_\neg$ ontology (TBox) $\mathcal{T}$ encoding an educational domain, asserting a range restriction plus mutual disjointness of the concepts like professor and student (we use Turtle syntax [2], in which dw abbreviates OWL's `disjointWith` keyword, and dom and rng respectively stand for the `domain` and `range` keywords of RDFS).

$\mathcal{T} = \{$`:studentOf` dom `:Student. :studentOf` rng `:Professor. :Professor` dw `:Student.` $\}$

Consider the following SPARQL update [8] request u in the context of the TBox $\mathcal{T}$:

```
INSERT {?X :studentOf ?Y} WHERE  {?X :attendsClassOf ?Y}
```

Consider an ABox with data on student tutors that happen to attend each other's classes: $\mathcal{A}_1 = \{$`:jim :attendsClassOf :ann. :ann :attendsClassOf :jim`$\}$. Here, u would create two assertions `:jim :studentOf :ann` and `:ann :studentOf :jim`. Due to the range and domain constraints in $\mathcal{T}$, these assertions result in clashes both for Jim and for Ann. Note that all inconsistencies are in the new data, and thus we say that u is *intrinsically inconsistent* for the particular ABox $\mathcal{A}_1$. We discuss how such updates can be fixed using SPARQL rewritings.

Now, let $\mathcal{A}_2$ be the ABox $\{$`:jim :attendsClassOf :ann. :jim a :Professor`$\}$. It is clear that after the update u, the ABox will become inconsistent with respect to $\mathcal{T}$ due to the property assertion `:jim :studentOf :ann`, implying that Jim is both a professor and a student which contradicts the disjointness axiom. In contrast to the previous case, the clash here is between the prior knowledge and the new data. Based on [1] we propose *three update semantics* for this case, and provide *efficient SPARQL rewriting algorithms* for implementing them in the RDFS$_\neg$ setting.

The topic of knowledge base updates is extremely broad. Our aim in this paper is to adapt the basic belief revision operators for efficient implementation of ABox updates expressed in SPARQL 1.1, in the presence of RDFS$_\neg$ TBox axioms. In contrast to our setting, most of existing works on knowledge base evolution consider updates based on sets of ground facts to be inserted or deleted. Restricting negation to class disjointness allowed us to keep the presentation clear. It is not difficult to lift our rewritings to theories with role disjointness, functionality and inequality (`owl:differentFrom`). We discuss related work in more detail in Sect. 6.

In the remainder of the paper, after some short preliminaries (Sect. 2) we discuss checking for intrinsic inconsistencies in Sect. 3. Then in Sect. 4 we present three semantics for dealing with general inconsistencies in the context of materialized triple stores. Sect. 5 describes our practical evaluation of the semantics. Finally, Sect. 6 puts our work in the context of existing research and provides concluding remarks.

2 Preliminaries

We introduce basic notions about RDF graphs, RDFS$_\neg$ ontologies, and SPARQL queries. We will use RDF and DL notation interchangeably, treating RDF graphs without non-standard RDFS$_\neg$ vocabulary use [19] as a sets of TBox and ABox assertions.

Table 1. *DL-Lite*$_{\text{RDFS}_\neg}$ assertions vs. RDF(S), where A, A' denote concept (or, class) names, P, P' denote role (or, property) names, Γ is the set of IRI constants (excl. the OWL/RDF(S) vocabulary) and $x, y \in \Gamma$. For RDF(S), we use abbreviations (rsc, sp, dom, rng, a) as introduced in [17].

TBox	RDFS$_\neg$	TBox	RDFS$_\neg$	TBox	RDFS$_\neg$	ABox	RDFS$_\neg$
1. $A' \sqsubseteq A$	A' sc A.	3. $\exists P \sqsubseteq A$	P dom A.	5. $A' \sqsubseteq \neg A$	A' dw A.	6. $A(x)$	x a A.
2. $P' \sqsubseteq P$	P' sp P.	4. $\exists P^- \sqsubseteq A$	P rng A.			7. $P(x, y)$	x P y.

Definition 1 (RDFS$_\neg$ ABox, TBox, Triple Store). *We call a set $\mathcal{T}$ of inclusion assertions of the forms 1–5 in Table 1 an (RDFS$_\neg$)* TBox*, a set $\mathcal{A}$ of assertions of the forms 6–7 in Table 1 an* (RDF) ABox*, and the union $G = \mathcal{T} \cup \mathcal{A}$ an (RDFS$_\neg$)* triple store*.*

Definition 2 (Interpretation, Satisfaction, Model, Consistency). *An* interpretation $\langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ *consists of a non-empty set $\Delta^\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$, which maps*

- *each atomic concept A to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$,*
- *each negation of atomic concept to $(\neg A^\mathcal{I}) = \Delta^\mathcal{I} \setminus A^\mathcal{I}$,*
- *each atomic role P to a binary relation $P^\mathcal{I}$ over $\Delta^\mathcal{I}$, and*
- *each element of Γ to an element of $\Delta^\mathcal{I}$.*

For expressions $\exists P$ and $\exists P^-$, the interpretation function is defined as $(\exists P)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \exists y.(x, y) \in P^\mathcal{I}\}$ and $(\exists P^-)^\mathcal{I} = \{y \in \Delta^\mathcal{I} \mid \exists x.(x, y) \in P^\mathcal{I}\}$, resp. An interpretation $\mathcal{I}$ satisfies *an inclusion assertion $E_1 \sqsubseteq E_2$ (of one of the forms 1–5 in Table 1), if $E_1^\mathcal{I} \subseteq E_2^\mathcal{I}$. Analogously, $\mathcal{I}$* satisfies *ABox assertions of the form $A(x)$, if $x^\mathcal{I} \in A^\mathcal{I}$, and of the form $P(x, y)$, if $(x^\mathcal{I}, y^\mathcal{I}) \in P^\mathcal{I}$. An interpretation $\mathcal{I}$ is called a* model *of a triple store G (resp., a TBox $\mathcal{T}$, an ABox $\mathcal{A}$), denoted $\mathcal{I} \models G$ (resp., $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$), if $\mathcal{I}$ satisfies all assertions in G (resp., $\mathcal{T}$, $\mathcal{A}$). Finally, G is called consistent, if it does not entail both $C(x)$ and $\neg C(x)$ for any concept C and constant $x \in \Gamma$, where entailment is defined as usual.*

As in [1], we treat only ABox updates with WHERE clauses restricted to unions of conjunctive queries (without projection) over DL ontologies:

Definition 3 (BGP, CQ, UCQ, Query Answer). *A* conjunctive query *(CQ) q, or* basic graph pattern (BGP)*, is a set of atoms of the form 6–7 from Table 1,*

where now $x, y \in \Gamma \cup \mathcal{V}$, $\mathcal{V}$ a countably infinite set of variables (written as '?'-prefixed alphanumeric strings). 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). An* answer *(under RDFS$_\neg$ 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 $ans_{rdfs}(q, G)$ (or simply $ans(q, G)$). The set of answers to a UCQ Q is $\bigcup_{q \in Q} ans(q, G)$.*

Query answering in the presence of ontologies is done either by rule-based pre-materialization of the ABox or by query rewriting. In the RDFS$_\neg$ case, materialization in polynomial time is feasible. Let $mat(G)$ be the triple store obtained from exhaustive application of the inference rules in Fig. 1 on a consistent triple store G. We also define a special notation chase$(q, \mathcal{T})$ to denote the "materialization" (also known as chase) of an ABox resp. a BGP q w.r.t. the TBox $\mathcal{T}$. We call all triples occurring in chase$(q, \mathcal{T})$ but not in q the *effects of q w.r.t. $\mathcal{T}$*.

We now adapt the semantics for SPARQL update operations from [1].

Definition 4 (SPARQL Update Operation, Simple Update of a Triple Store). *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*

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

Let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store then the *simple update* of G w.r.t. $u(P_d, P_i, P_w)$ is defined as $G_{u(P_d,P_i,P_w)} = (G \setminus \mathcal{A}_d) \cup \mathcal{A}_i$, where $\mathcal{A}_d = \bigcup_{\theta \in ans(P_w,G)} gr(P_d\theta)$, $\mathcal{A}_i = \bigcup_{\theta \in ans(P_w,G)} gr(P_i\theta)$, and $gr(P)$ denotes the set of ground triples in pattern P.

We call a triple store G (resp. the ABox of G) *materialized* if the equality $G \setminus \mathcal{T} = mat(G) \setminus \mathcal{T}$ holds. In this paper, we will always consider G to be materialized and focus on "materialization preserving" semantics for SPARQL update operations, which we dubbed $\mathbf{Sem}_2^{mat}$ in [1] and which preserves a materialized triple store. We recall the intuition behind $\mathbf{Sem}_2^{mat}$, given an update $u = (P_d, P_i, P_w)$: *(i)* delete the instantiations of P_d *along with all their causes*; *(ii)* insert the instantiations of P_i *plus all their effects*.

The notion of "causes" is made precise as follows. Given an ABox assertion A, $A^{\text{caus}} = \{B \mid A \in \text{chase}(\{B\}, \mathcal{T})\}$. In the definition of A^{caus}, if A is a class

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

$$\frac{?P\ \mathsf{sp}\ ?Q. \quad ?S\ ?P\ ?O.}{?S\ ?Q\ ?O.} \qquad \frac{?P\ \mathsf{rng}\ ?C. \quad ?S\ ?P\ ?O.}{?O\ \mathsf{a}\ ?C.} \qquad \frac{?S\ \mathsf{a}\ ?C,?D. \quad ?C\ \mathsf{dW}\ ?D.}{\bot}$$

Fig. 1. Minimal RDFS rules from [17]plus class disjointness "clash" rule from OWL2 RL [16].

membership `(x a C)` where $x \in \Gamma \cup \mathcal{V}$, then B is one of `(x a C')`, `(x P ?Y)`, `(?Y P x)` for some fresh variable $?Y$, class `C'` and role `P`. If A is a role participation assertion `(x R z)`, B is of the form `(x P z)`, for some role `P`. For a SPARQL triple (possibly with variables) C we use C^{caus} to denote a BGP computed in the same way as for the ABox assertion A above.

Definition 5 ($\mathbf{Sem}_2^{mat}$[1]). *Let $u(P_d, P_i, P_w)$ be an update operation. Then*

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

Here, $P_d^{\text{caus}} = \bigcup_{A \in atoms(P_d)} A^{\text{caus}}$; $P^{\text{eff}} = \text{chase}(P, \mathcal{T})$ and P_d^{fvars} is a pattern that binds variables occurring in P_d^{caus} but not in P_d to the constants from Γ occurring in G.

We refer to [1] for further details, but stress that as such, $\mathbf{Sem}_2^{mat}$ is not able to detect or deal with inconsistencies arising from extending G with instantiations of P_i. In what follows, we will discuss how this can be remedied.

Remark 1. Note that although the DELETE clause P_d is syntactically a BGP, its semantics is different. Namely, triples occurring in P_d are mutually independent (cf. Definition 4), so that for every $\theta \in ans(P_w, G)$, each atom in $P_d\theta \cap G$ is deleted from G no matter which other atoms of $P_d\theta$ occur in G. Therefore, P_d^{caus} is computed atom-wise, unlike CQ rewriting [4]. Note that $|A^{\text{caus}}| = O(||\mathcal{T}||)$ where $||\mathcal{T}||$ denotes the vocabulary size of $\mathcal{T}$: in each RDFS$_\neg$ derivation, a class membership assertion can occur at most once for each class in $\mathcal{T}$, and a role membership assertion can occur at most twice for every role in $\mathcal{T}$. Thus, $|P_d^{\text{caus}}| \leq 2|P_d| \cdot ||\mathcal{T}||$ and $|P_i^{\text{eff}}| \leq |P_i| \cdot ||\mathcal{T}||$, so both can be computed in poly-time. This underpins the polynomial complexity of our rewritings.

3 Checking Consistency of a SPARQL Update

In the literature on the evolution of DL-Lite knowledge bases [5,7], updates represented by pairs of ABoxes $\mathcal{A}_d, \mathcal{A}_i$ have been studied. However, whereas such update might be viewed to fit straightforwardly to the corresponding $\mathcal{A}_d, \mathcal{A}_i$ in Definition 4, it is typically assumed that $\mathcal{A}_i$ is consistent with the TBox, and thus one only needs to consider how to deal with inconsistencies between the update and the old state of the knowledge base. However, this a priori assumption may be insufficient for SPARQL updates, where concrete values for inserted triples are obtained from variable bindings in the WHERE clause, and depending on the bindings, the update can be either consistent or not. This is demonstrated by the update u from Sect. 1 which, when applied to the ABox $\mathcal{A}_1$, results in an inconsistent set $\mathcal{A}_i$ of insertions. We call this *intrinsic inconsistency* of an update *relative to a triple store* $G = \mathcal{T} \cup \mathcal{A}$.

Definition 6. *Let G be a triple store. The update u is said to be* intrinsically consistent w.r.t. *G if the set of new assertions $\mathcal{A}_i$ from Definition 4 generated by applying u to G, taken in isolation from the ABox of G, does not contradict the TBox of G. Otherwise, the update is said to be intrinsically inconsistent w.r.t. G.*

Algorithm 1. constructing a SPARQL ASK query to check intrinsic inconsistency (for the definition of P_i^{eff}, cf. Definition 5)

Input: RDFS$_\neg$ TBox $\mathcal{T}$, SPARQL update $u(P_d, P_i, P_w)$
Output: A SPARQL ASK query returning $True$ if u is intrinsically inconsistent

```
1 if ⊥ ∈ P_i^eff then
2 |   return ASK {}    //u contains clashes in itself, i.e., is inconsistent for any
  |   triple store
3 else
4 |   W := {FILTER(False)};    //neutral element w.r.t. union
5 |   foreach pair of triple patterns (?X a P), (?Y a R) in P_i^eff do
6 |   |   if P ⊑ ¬R ∈ T then
7 |   |   |   W := W UNION {{P_w θ1[?X ↦ ?Z]} . {P_w θ2[?Y ↦ ?Z]}} for a fresh
  |   |   |   ?Z
8 |   return ASK WHERE {W}
```

Intrinsic inconsistency of the update differs crucially from the inconsistency w.r.t. the old state of the knowledge base, illustrated by the ABox $\mathcal{A}_2$ from Sect. 1. This latter case can be addressed by adopting an update policy that prefers newer assertions in case of conflicts, as studied in the context of DL-Lite KB evolutions [5], which we will discuss in Sect. 4 below. Intrinsic inconsistencies however are harder to deal with, since there is no cue which assertion should be discarded in order to avoid the inconsistency. Our proposal here is thus to discard *all* mutually inconsistent pairs of insertions.

We first present an algorithm for checking intrinsic inconsistency by means of SPARQL ASK queries and then a safe rewriting algorithm. This rewriting is based on an observation that clashing triples can be introduced by a combination of two bindings of variables in the WHERE clause, as the example in the Sect. 1 (the ABox $\mathcal{A}_1$) illustrates. To handle such cases, two copies of the WHERE clause P_w are created by the rewriting in Algorithms 1 and 2, for each pair of disjoint concepts according to the TBox of the triple store. These algorithms use notation described in Remark 2 below.

Remark 2. Our rewriting algorithms rely on producing fresh copies of the WHERE clause. Assume θ, θ_1, θ_2, ... to be substitutions replacing each variable in a given formula with a distinct fresh one. For a substitution σ, we also define $\theta[\sigma]$ resp. $\theta_i[\sigma]$ to be an extension of σ, renaming each variable at positions not affected by σ with a distinct fresh one. For instance, let F be a triple $(?Z$ `:studentOf` $?Y)$. Now, $F\theta$ makes a variable disjoint copy of F: $?Z_1$ `:studentOf` $?Y_1$ for fresh $?Z_1, ?Y_1$. $F[?Z \mapsto ?X]$ is just a substitution of $?Z$ by $?X$ in F. Finally, $F\theta[?Z \mapsto ?X]$ results in $?X$ `:studentOf` $?Y_2$ for fresh $?Y_2$. We assume that all occurrences of $F\theta[\sigma]$ stand for syntactically the same query, but that $F\theta[\sigma_1]$ and $F\theta[\sigma_2]$, for distinct σ_1 and σ_2, can only have variables in $range(\sigma_1) \cap range(\sigma_2)$ in common. That is, the choice of fresh variables is defined by the parameterizing substitution σ. ■

Algorithm 2. Safe rewriting safe(u)

Input: RDFS$_\neg$ TBox $\mathcal{T}$, SPARQL update $u(P_d, P_i, P_w)$
Output: SPARQL update safe(u)

```
1 if ⊥ ∈ P_i^eff then
2 |   return u(P_d, P_i, FILTER(False))
3 W := {FILTER(False)};    //neutral element w.r.t. union
4 foreach pair of triple patterns (?X a P), (?Y a R) in P_i^eff do
5 |   if P ⊑ ¬R ∈ T then
6 |   |   //cf. Remark 2 for notation θ[...]
7 |   |   W := W UNION {P_w θ1[?X ↦?Y]} UNION {P_w θ2[?Y ↦?X]}}
8 return u(P_d, P_i, P_w MINUS {W})
```

Using this notation, the possibility of unifying two variables $?X$ and $?Y$ in P_w on a given triple store can be tested with the query $\{P_w\theta_1[?X \mapsto ?Z]\}\{P_w\theta_2[?Y \mapsto ?Z]\}$ where θ_1 and θ_2 are variable renamings as in Remark 2 and $?Z$ is a fresh variable.

In order to check the intrinsic consistency of an update, this condition should be evaluated for every pair of variables of P_w, the unification of which leads to a clash. A SPARQL ASK query based on this idea is produced by Algorithm 1. Note that it suffices to check only triples of the form $\{?X$ a $?C\}$ at line 5 of Algorithm 1, since disjointness conditions can only be formulated for concepts, according to the syntax in Table 1. Furthermore, since we are taking the facts in P_i^{eff} extended by all facts implied by $\mathcal{T}$, at line 6 of Algorithm 1 it suffices to check the disjointness conditions explicitly mentioned in $\mathcal{T}$ and not all those which are implied by $\mathcal{T}$. Note also that the DELETE clause P_d plays no role in this case, since we only consider clashes within inserted facts.

Example 1. Consider the update u from Sect. 1, in which the INSERT clause P_i can create clashing triples. To identify potential clashes, Sect. 1 first applies the inference rule for the range constraint, and computes $P_i^{\text{eff}} = \{?X$ `a :Student` . $?Y$ `a :Professor`$\}$. Now both variables $?X, ?Y$ occur in the triples of type (6) from Sect. 1 with clashing concept names. The following ASK query is produced by Sect. 1.

```
ASK WHERE { ?X :attendsClassOf ?Y . ?Y :attendsClassOf ?X1 }
```

(In this and subsequent examples we omit the trivial FILTER(*False*) union branch used in rewritings to initialize variables with disjunctive conditions, such as W in Algorithm 1) ■

Suppose that an insert is not intrinsically consistent for a given triple store. One solution would be to discard it completely, should the above ASK query return *True*. Another option which we consider here is to only discard those variable bindings from the WHERE clause, which make the INSERT clause P_i inconsistent. This is the task of the *safe rewriting* safe($\cdot$) in Algorithm 2, removing all variable bindings that participate in a clash between different triples of P_i. Let P_w be a WHERE clause, in which the variables $?X$ and $?Y$ should

not be unified to avoid clashes. With θ_1, θ_2 being "fresh" variable renamings as in Remark 2, Algorithm 2 uses the union of $P_w\theta_1[?X \mapsto ?Y]$ and $P_w\theta_2[?Y \mapsto ?X]$ to eliminate unsafe bindings that send $?X$ and $?Y$ to the same value.

Example 2. Algorithm 2 extends the WHERE clause of the update u from Sect. 1 as follows:

```
INSERT{?X :studentOf ?Y} WHERE{?X :attendsClassOf ?Y
 MINUS{{?X1 :attendsClassOf ?X} UNION {?Y :attendsClassOf ?Y2}}}
```

Note that the safe rewriting can make the update void. For instance, safe(u) has no effect on the ABox $\mathcal{A}_1$ from Sect. 1, since there is no cue, which of `:jim :attendsClassOf :ann`, `:ann :attendsClassOf :jim` needs to be dismissed to avoid the clash. However, if we extend this ABox with assertions both satisfying the WHERE clause of u and not causing undesirable variable unifications, safe(u) would make insertions based on such bindings. For instance, adding the fact `:bob :attendsClassOf :alice` to $\mathcal{A}_1$ would assert `:bob :studentOf :alice` as a result of safe(u). ■

A rationale for using MINUS rather than FILTER NOT EXISTS in Algorithm 2 (and also in a rewriting in forthcoming Sect. 4) can be illustrated by an update in which variables in the INSERT and DELETE clauses are bound in different branches of a UNION:

```
DELETE {?V a :Professor} INSERT {?X :studentOf ?Y}
WHERE {{?X :attendsClassOf ?Y} UNION {?V :attendsClassOf ?W}}
```

A safe rewriting of this update (abbreviating `:attendsClassOf` as `:aCo`) is

```
DELETE {?V a :Professor} INSERT {?X :studentOf ?Y}
WHERE { {{?X :aCo ?Y} UNION {?V :aCo ?W}}
        MINUS{ {{?X1 :aCo ?X} UNION {?V1 :aCo ?W1}}
         UNION {{?Y :aCo ?Y2} UNION {?V2 :aCo ?W2}} } }
```

It can be verified that with FILTER NOT EXISTS in place of MINUS this update makes no insertions on all triple stores: the branches {`?V1 :aCo ?W1`} and {`?V2 :aCo ?W2`} are satisfied whenever {`?X :aCo ?Y`} is, making FILTER NOT EXISTS evaluate to *False* whenever {`?X :aCo ?Y`} holds.

We conclude this section by formalizing the intuition of update safety. For a triple store G and an update $u = (P_d, P_i, P_w)$, let $[\![P_w]\!]^u_G$ denote the set of variable bindings computed by the query "SELECT$?X_1, \ldots, ?X_k$ WHERE P_w" over G, where $?X_1, \ldots, ?X_k$ are the variables occurring in P_i or in P_d.

Theorem 1. *Let $\mathcal{T}$ be a TBox, let u be a SPARQL update (P_i, P_d, P_w), and let query q_u and update* safe$(u) = (P_d, P_i, P'_w)$ *result from applying Algorithm 1 resp. Algorithm 2 to u and $\mathcal{T}$. Then, the following properties hold for an arbitrary RDFS$_\neg$ triple store $G = \mathcal{T} \cup \mathcal{A}$:*

(1) $q_u(G) = True$ *iff* $\exists \mu, \mu' \in [\![P_w]\!]^u_G$ *s.t.* $\mu(P_i) \wedge \mu'(P_i) \wedge \mathcal{T} \models \bot$;
(2) $[\![P_w]\!]^u_G \setminus [\![P'_w]\!]^u_G = \{\mu \in [\![P_w]\!]^u_G \mid \exists \mu' \in [\![P_w]\!]^u_G$ *s.t.* $\mu(P_i) \wedge \mu'(P_i) \wedge \mathcal{T} \models \bot\}$.

4 Materialization Preserving Update Semantics

In this section we discuss resolution of inconsistencies between triples already in the triple store and newly inserted triples. Our baseline requirement for each update semantics is formulated as the following property.

Definition 7 (Consistency-preserving). *Let G be a triple store and $u(P_d, P_i, P_w)$ an update. A materialization preserving update semantics Sem is called* consistency preserving *in RDFS$_\neg$ if the evaluation of update u, i.e., $G^{Sem}_{u(P_d,P_i,P_w)}$, results in a consistent triple store.*

Our consistency preserving semantics are respectively called *brave*, *cautious* and *fainthearted*. The brave semantics always gives priority to newly inserted triples by discarding all pre-existing information that contradicts the update. The cautious semantics is exactly the opposite, discarding inserts that are inconsistent with facts already present in the triple store; i.e., the cautious semantics never deletes facts unless explicitly required by the DELETE clause of the SPARQL update. Finally, the fainthearted semantics executes the update partially, only performing insertions for those variable bindings which do not contradict existing knowledge (again, taking into account deletions).

All semantics rely upon incremental update semantics $\mathbf{Sem}_2^{mat}$, introduced in Sect. 2, which we aim to extend to take into account class disjointness. Note that for the present section we assume updates to be intrinsically consistent, which can be checked or enforced beforehand in a preprocessing step by the safe rewriting discussed in Sect. 3. In this section, we lift our definition of update operation to include also updates (P_d, P_i, P_w) with P_w produced by the safe rewriting Algorithm 2 from some update satisfying Definition 4. What remains to be defined is the handling of clashes between newly inserted triples and triples already present in the triple store.

The intuitions of our semantics for a SPARQL update $u(P_d, P_i, P_w)$ in the context of an RDFS$_\neg$ TBox are as follows:

- *brave semantics* $\mathbf{Sem}^{mat}_{brave}$: *(i)* delete all instantiations of P_d and their causes, *plus all the non-deleted triples in G clashing with instantiations of triples in P_i to be inserted*, again also including the causes of these triples; *(ii)* insert the instantiations of P_i plus all their effects.
- *cautious semantics* $\mathbf{Sem}^{mat}_{caut}$: *(i)* delete all instantiations of P_d and their causes; *(ii)* insert all instantiations of P_i plus all their effects, *unless they clash with some non-deleted triples in G*: in this latter case, do not perform the update.
- *fainthearted semantics* $\mathbf{Sem}^{mat}_{faint}$: *(i)* delete all instantiations of P_d and their causes; *(ii)* insert those instantiations of P_i (plus all their effects) which *do not clash with non-deleted triples in G*.

Remark 3. Note that $\mathbf{Sem}_2^{mat}$ is not able to cope with so called "dangling" effects – that is, triples inserted at some point for the sake of materialization, whose causes have been subsequently deleted. As pointed out in [1], one way to

Algorithm 3. *Brave semantics* $\mathbf{Sem}^{mat}_{brave}$

Input: Materialized triple store $G = \mathcal{T} \cup \mathcal{A}$, SPARQL update $u(P_d, P_i, P_w)$
Output: $G^{\mathbf{Sem}^{mat}_{brave}}_{u(P_d,P_i,P_w)}$

$P'_d := P_d^{\text{caus}}$;
foreach *triple pattern* $(?X \ \mathtt{a} \ C)$ *in* P_i^{eff} **do**
  foreach C' *s.t.* $C \sqsubseteq \neg C' \in \mathcal{T}$ *or* $C' \sqsubseteq \neg C \in \mathcal{T}$ **do**
    if $(?X \ \mathtt{a} \ C') \notin P'_d$ **then**
      $P'_d := P'_d \ . \ \{?X \ \mathtt{a} \ C'\}^{\text{caus}}$
return $G_{u(P'_d, P_i^{\text{eff}}, \{P_w\} P_d^{fvars})}$

deal with this issue is to combine $\mathbf{Sem}_2^{mat}$ with marking of explicitly inserted triples. This approach was implemented as a semantics $\mathbf{Sem}_{1b}^{mat}$ in [1], splitting the ABox $\mathcal{A}$ into the explicit part $\mathcal{A}_{ex}$ and the implicit part $\mathcal{A}_{im} = \mathcal{A} \setminus \mathcal{A}_{ex}$. $\mathcal{A}_{ex}$ can be maintained, e.g., in a separate RDF graph using a straightforward update rewriting. Now, deleting P_d would not only retract P_d^{caus} from $\mathcal{A}$, but also the triples in $\text{chase}(P_d^{\text{caus}}, \mathcal{T}) \setminus \text{chase}(\mathcal{A}_{ex} \setminus P_d^{\text{caus}}, \mathcal{T})$. That is, the effects of P_d^{caus} are removed unless they can be derived from facts remaining in $\mathcal{A}$ after enforcing the deletion P_d. Such an aggressive removal of dangling triples can lead to counterintuitive behavior (cf. Example 9 in [1]), and requires maintaining the explicit ABox $\mathcal{A}_{ex}$, which is why we opted to preserve dangling effects in our rewritings.

We will now describe implementations of the three semantics above via SPARQL rewritings, which can be shown to be materialization preserving and consistency preserving.

4.1 Brave Semantics

The rewriting in Algorithm 3 implements the brave update semantics $\mathbf{Sem}^{mat}_{brave}$; it can be viewed as combining the idea of *FastEvol*[5] with $\mathbf{Sem}_2^{mat}$ to handle inconsistencies by giving priority to triples that ought to be inserted, and deleting all those triples from the store that clash with the new ones.

Example 3. Example 2 in Sect. 3 provided a safe rewriting safe(u) of the update u from Sect. 1. According to Algorithm 3, this safe update is rewritten to:

```
DELETE {?X a :Professor . ?X1 :studentOf ?X .
        ?Y a :Student . ?Y :studentOf ?Y1}
INSERT {?X :studentOf ?Y . ?X a :Student . ?Y a :Professor}
WHERE {{?X :attendsClassOf ?Y
MINUS{{?X2 :attendsClassOf ?X} UNION {?Y :attendsClassOf ?Y2}}}
OPTIONAL {?X1 :studentOf ?X} OPTIONAL {?Y :studentOf ?Y1} }
```

The DELETE clause removes potential clashes for the inserted triples. Note that also property assertions implying clashes need to be deleted, which introduces fresh variables $?X1$ and $?Y1$. These variables have to be bound in the WHERE

clause, and therefore P_d^{fvars} adds two optional clauses to the WHERE clause, which is a computationally reasonable implementation of the concept P^{fvars} from Definition 5. ■

The DELETE clause P'_d of the rewritten update is initialized in Algorithm 3 with the set P_d of triples from the input update. Rewriting ensures that also all "causes" of deleted facts are removed from the store, since otherwise the materialization will re-insert deleted triples. To this end, line 1 of Algorithm 3 adds to P'_d all facts from which P_d can be derived. Then, for each triple implied by P_i (that is, for each triple in P_i^{eff}) the algorithm computes the patterns of clashing triples and adds them to the DELETE clause P'_d, along with their causes. Note that it suffices to only consider disjointness assertions that are syntactically contained in $\mathcal{T}$ (and not those implied by $\mathcal{T}$), since we assume that the store G is materialized. Finally, the WHERE clause of the rewritten update is extended to satisfy the syntactic restriction that all variables in P'_d must be bound: bindings of "fresh" variables introduced to P'_d due to the domain or range constraints in $\mathcal{T}$ are provided by the part P_d^{fvars}, cf. Definition 5 and Example 3. The rewritten update is evaluated over the triple store, computing its new materialized and consistent state.

In the RDFS$_\neg$ ontology language and under the restriction that only ABox updates are allowed, the brave semantics is a belief revision operator [10,20], performing a minimal change of the RDF graph (which due to materialization can be seen both as a deductive closure of the formula representing the ABox as well as the minimal model of this formula). There is a unique way of resolving inconsistencies since the only deduction rule with more than one ABox assertion in the premise, is the clash due to class disjointness (Fig. 1): assuming intrinsic consistency, the choice of which class membership assertion to remove in order to avoid clash is univocal (new knowledge is always preferred).

Theorem 2. *Algorithm 3, given a SPARQL update u and a consistent materialized triple store $G = \mathcal{T} \cup \mathcal{A}$, computes a new consistent and materialized state w.r.t.* brave semantics. *The rewriting in lines 1–6 takes time polynomial in the size of u and $\mathcal{T}$.*

4.2 Cautious Semantics

Unlike $\mathbf{Sem}^{mat}_{brave}$, its *cautious* version $\mathbf{Sem}^{mat}_{caut}$ always gives priority to triples that are already present in the triple store, and dismisses any inserts that are inconsistent with it. We implement this semantics as follows: *(i)* the DELETE command does not generate inconsistencies and thus is assumed to be always possible; *(ii)* the update is actually executed only if the triples introduced by the INSERT clause do not clash with state of the triple graph *after all deletions have been applied*.

Cautious semantics thus treats insertions and deletions asymmetrically: the former depend on the latter but not the other way round. The rationale is that deletions never cause inconsistencies and can remove clashes between the old and the new data.

Algorithm 4. *Cautious semantics* $\mathbf{Sem}^{mat}_{caut}$

Input: Materialized triple store $G = \mathcal{T} \cup \mathcal{A}$, SPARQL update $u(P_d, P_i, P_w)$
Output: $G^{\mathbf{Sem}^{mat}_{caut}}_{u(P_d,P_i,P_w)}$

1 $W := \{\,\mathsf{FILTER}(\mathit{False})\}$ // *neutral element w.r.t. union*
2 **foreach** $(?X\ \mathsf{a}\ C) \in P_i^{\text{eff}}$ **do**
3 | **foreach** C' *s.t.* $C \sqsubseteq \neg C' \in \mathcal{T}$ *or* $C' \sqsubseteq \neg C \in \mathcal{T}$ **do**
4 | | $\Theta^{-}_{C'} := \{\,\mathsf{FILTER}(\mathit{False})\}$
5 | | **foreach** $(?Y\ \mathsf{a}\ C') \in P_d^{\text{caus}}$ **do**
6 | | | $\Theta^{-}_{C'} := \Theta^{-}_{C'}\ \mathsf{UNION}\ \{P_w\theta[?Y \mapsto ?X]\}$
7 | | $W := W\ \mathsf{UNION}\ \{\{?X\ \mathsf{a}\ C'\}\ \mathsf{MINUS}\ \{\Theta^{-}_{C'}\}\}$
8 $Q := \mathsf{ASK\,WHERE}\ \{\{P_w\}.\{W\}\};$
9 **if** $Q(G)$ **then**
10 | **return** G
11 **else**
12 | **return** $G^{\mathbf{Sem}^{mat}_{brave}}_{u(P_d,P_i,P_w)}$

As in the case of brave semantics, cautious semantics is implemented using rewriting, presented in Algorithm 4. First, the algorithm issues an ASK query to check that no clashes will be generated by the INSERT clause, provided that the DELETE part of the update is executed. If no clashes are expected, in which case the ASK query returns *False*, the brave update from the previous section is applied.

For a safe update $u = (P_d, P_i, P_w)$, the ASK query is generated as follows. For each triple pattern $\{?X\ \mathsf{a}\ C\}$ among the effects of P_i, at line 3 Algorithm 4 enumerates all concepts C' that are explicitly mentioned as disjoint with C in $\mathcal{T}$. As in the case of brave semantics, this syntactic check is sufficient due to the assumption that the update is applied to a materialized store; by the same reason also no property assertions need to be taken into account.

For each concept C' disjoint with C, we need to check that a triple matching the pattern $\{?X\ \mathsf{a}\ C'\}$ is in the store G and will not be deleted by u. Deletion happens if there is a pattern $\{?Y\ \mathsf{a}\ C'\} \in P_d^{\text{caus}}$ such that the variable $?Y$ can be bound to the same value as $?X$ in the WHERE clause P_w. Line 6 of Algorithm 4 produces such a check, using a copy of P_w, in which the variable $?Y$ is replaced by $?X$ and all other variables are replaced with distinct fresh ones. Since there can be several such triple patterns in P_d^{caus}, testing for clash elimination via the DELETE clause requires a disjunctive graph pattern $\Theta^{-}_{C'}$ constructed at line 6 and combined with $\{?X\ \mathsf{a}\ C'\}$ using MINUS at line 7.

Finally, the resulting pattern is appended to the list W of clash checks using UNION. As a result, $\{P_w\}.\{W\}$ queries for triples that are not deleted by u and clash with an instantiation of some class membership assertion $\{?X\ \mathsf{a}\ C\} \in P_i^{\text{eff}}$.

Theorem 3. *Algorithm 4, given a SPARQL update u and a consistent materialized triple store $G = \mathcal{T} \cup \mathcal{A}$, computes a new consistent and materialized state*

Algorithm 5. *Fainthearted semantics* $\mathbf{Sem}^{mat}_{faint}$

Input: Materialized triple store $G = \mathcal{T} \cup \mathcal{A}$, SPARQL update $u(P_d, P_i, P_w)$
Output: $G^{\mathbf{Sem}^{mat}_{faint}}_{u(P_d,P_i,P_w)}$

1 $W := P_w$
2 **foreach** *triple pattern* $(x \ \mathsf{a} \ C)$ *in* P_i^{eff} **do**
3 **foreach** C' *s.t.* $C \sqsubseteq \neg C' \in \mathcal{T}$ *or* $C' \sqsubseteq \neg C \in \mathcal{T}$ **do**
4 $\Theta^-_{C'} := \{\mathsf{FILTER}(False)\}$;
5 **foreach** $(z \ \mathsf{a} \ C') \in P_d^{\text{caus}}$ **do**
6 $\Theta^-_{C'} := \Theta^-_{C'}$ UNION $\{P_w\theta[z \mapsto x]\}$;
7 $W := \{W\}$ MINUS $\{x \ \mathsf{a} \ C'$ MINUS $\{\Theta^-_{C'}\}\}$;
8 $W := \{W\}$ UNION $\{P_w\theta_1 \ . \ P_d^{fvars}\theta_1\}$;
9 **return** $G_{u(P_d^{\text{caus}}\theta_1, \ P_i^{\text{eff}}, \ W)}$

w.r.t. cautious semantics. *The rewriting in lines 1–8 takes time polynomial in the size of* u *and* $\mathcal{T}$.

Example 4. Algorithm 4 rewrites the safe update safe(u) from Example 2 as follows:

```
ASK WHERE{{?X :attendsClassOf ?Y
 MINUS{{?X1 :attendsClassOf ?X} UNION {?Y :attendsClassOf ?Y2}}}
 .{{?Y a :Student} UNION {?X a :Professor}}}
```

Now, consider an update u' having both INSERT and DELETE clauses:

```
DELETE {?Y a :Professor} INSERT{?X a :Student}
WHERE {?X :attendsClassOf ?Y}
```

The update u' inserts a single class membership fact and thus is always intrinsically consistent. The ASK query in Algorithm 4 takes the DELETE clause of u' into account:

```
ASK WHERE {{?X :attendsClassOf ?Y}
.{{?X a :Professor} MINUS {?Z :attendsClassOf ?X }}}
```

■

4.3 Fainthearted Semantics

Our third, *fainthearted* semantics is meant to take an intermediate position between the cautious semantics and the brave one. A shortcoming of the cautious semantics is that massive update can be retracted because of only a few clashing triples. Not to discard an update completely in such a case, the user can decide either to override the existing knowledge — that is, opt for the brave semantics — or to apply insertions only for those variable bindings which are not clashing with the existing state, which is what the fainthearted semantics does.

Our realization of the idea of accommodating non-clashing inserts is based on *decoupling the insert and the delete* part of an update: whereas the delete is executed for *all* variable bindings satisfying the WHERE clause, one dismisses

the inserts for variable bindings that yield clashes with the state of the store *after the delete*. That is, we deviate from the notion of update as an atomic operation in a different way than in the safe rewriting where *both* deletions and insertions are dismissed for variable bindings leading to clashes. Our motivation for such a design decision is explained next.

Assume that for each variable binding μ returned by the WHERE pattern, we want to either insert $\mathrm{gr}(P_i\mu)$ along with deleting $\mathrm{gr}(P_d\mu)$, or dismiss μ altogether. As an example, consider the update u' from Example 4 and the ABox {`:jim :attendsClassOf :ann. :jim a :Professor. :bob :attendsClassOf :jim`}. With the variable binding $\mu_1 = [?X \mapsto$ `:jim`$, ?Y \mapsto$ `:ann`$]$ we insert `:jim a :Student` knowing that the clashing fact `:jim a :Professor` will be deleted by the binding $\mu_2 = [?X \mapsto$ `:bob`$, ?Y \mapsto$ `:jim`$]$. However, if the update is atomic, this anticipated deletion will only happen if $\mathrm{gr}(P_i\mu_2)$ does not introduce clashes. Assume this is the case (i.e. also {`:bob a :Professor`} is in the ABox): we have to look one more step ahead and check if this triple will be deleted by some variable binding μ_3, and so on. This behaviour could be realized with SPARQL path expressions, which would however stipulate severe syntactic restrictions on the WHERE clause P_w of the original update.

As mentioned above, our interpretation of fainthearted semantics assumes independence between the INSERT and DELETE parts of the update. To implement this, we rely on SPARQL's flexible handling of variable bindings. Namely, we rename the variables in the DELETE clause apart from the rest of the update, and put this renamed apart copy of the WHERE clause in a new UNION branch. The original WHERE clause is then rewritten (using MINUS operator, similarly to the case of cautious semantics) to ensure that insertions are only done for variable bindings where clashes are removed by the DELETE clause with some variable binding. The implementation can be found in Algorithm 5.

Example 5. The update u' from Example 4 is rewritten as follows by Algorithm 5:

```
DELETE {?Y1 a :Professor } INSERT {?X a :Student}
WHERE {{?X2 :attendsClassOf ?Y1} UNION {?X :attendsClassOf ?Y.
      {MINUS {?X a :Professor MINUS {?X3 :attendsClassOf ?X}}}}}
```

The first union branch binds the variables in the DELETE clause (both using fresh variables). The second branch binds the variable `?X` in the INSERT clause, using MINUS to remove variable bindings for which a non-deleted clash exists. The test that a clash will not be deleted is expressed using the inner MINUS operator. ■

We conclude with a claim of correctness and polynomial complexity of the rewriting, similar to those made for the brave and cautious semantics.

Theorem 4. *Algorithm 5, given a SPARQL update u and a materialized triple store $G = \mathcal{T} \cup \mathcal{A}$ w.r.t.* fainthearted semantics*, computes a new consistent and materialized state. The rewriting in lines 1–9 takes time polynomial in the size of u and $\mathcal{T}$.*

5 Experimental Evaluation

For each of the three semantics discussed in the previous section, we provided a preliminary implementation using the Jena API (http://jena.apache.org) and evaluated them against Jena TDB triple store which implements the latest SPARQL 1.1 specification. As before, for computing the initial materialization of a triple store $mat(G)$ we rely on-board, forward-chaining materialization in Jena TDB using the minimal RDFS rules as in Fig. 1.

For our experiments, we used the data generated by the EUGen generator [15] of for the size range of 5 to 50 Universities. We opted for using this generator as it extends the LUBM ontology [9] with chains of subclasses, making the rewritings more challenging. In our case we have used the default of $i = 20$ subclasses for each LUBM concept (e.g., `Subj`i`Students`) and made such subclasses pairwise disjoint. Moreover, we have added more disjointness axioms where appropriate, e.g., `:AssociateProfessor dw :FullProfessor`. All these TBox axioms are merged with our previous reduced RDFS version of LUBM used in our previous work [1]. To compare the experimental results with the previous work, for our experiments we adapted the seven updates from [1]. Our prototype, as well as files containing the data, ontology, and the updates used for experiments, are made available on a dedicated Web page[1].

The results summarized in Table 2 show that the LUBM 50 dataset (507 MB uncompressed, 8.7 M triples after materialization) can be handled in seconds on a quad-core Intel i7 3.20 GHz machine with 16 GB RAM. For each of the three semantics, we have compared the time elapsed for rewriting and for the evaluation of the resulting update. The last line in Table 2 is the evaluation time for the original, non-rewritten update. One can notice that brave semantics $\mathbf{Sem}^{mat}_{brave}$ is often the most expensive one, since it performs most modifications. When the number of inconsistent inserts is low though, the situation is different, and the brave semantics slightly outperforms the fainthearted semantics $\mathbf{Sem}^{mat}_{faint}$ (Update #6 and #7), due to the more complex checks in the WHERE clause produced by Algorithm 5. For the cautious semantics $\mathbf{Sem}^{mat}_{caut}$, the numbers in the table are construction and evaluation time of the ASK query checking for the feasibility of update (cf. Algorithm 4). In case this ASK query returns $False$, the runtime of brave semantics should be added in order to obtain the total runtime of the update. Update #4 demonstrates that $\mathbf{Sem}^{mat}_{caut}$ can perform significantly worse than $\mathbf{Sem}^{mat}_{faint}$ when the number of instantiations in the original WHERE clause is high. This is because the ASK query in $\mathbf{Sem}^{mat}_{caut}$ looks for instantiations of the WHERE clause which can lead to clashes with the existing tuples (using a conjunctive condition), whereas $\mathbf{Sem}^{mat}_{faint}$ reduces the set of solutions of the original WHERE clause using MINUS, which is apparently more efficient in the Apache TDB.

[1] http://dbai.tuwien.ac.at/user/ahmeti/sparqlupdate-inconsistency-resolver/.

Table 2. Evaluation results in seconds for LUBM 50

Update #	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$\mathbf{Sem}^{mat}_{brave}$	12,4	14,8	0,1	22,1	46,0	15,3	13,6
$\mathbf{Sem}^{mat}_{caut}$	0,3	0,2	0,2	44,0	0,2	3,9	2,3
$\mathbf{Sem}^{mat}_{faint}$	2,2	2,8	0,01	17,4	3,3	16,7	15,3
Original	0,2	0,2	0,2	10,2	0,2	6,6	5,4

6 Related Work and Conclusions

In this paper we have taken a step further from our previous work, in combining SPARQL Update and RDFS entailment by adding concept disjoints as a first step towards dealing with inconsistencies in the context of SPARQL Updates. We distinguish the case of intrinsic inconsistency, localized within instantiations of the INSERT clause of a SPARQL update, and the usual case when the new information is inconsistent with the old knowledge. In the former case, our solution was to discard all solutions of the WHERE query that participate in an inconsistency. For the latter case, we discussed several reconciliation strategies, well suited for efficient implementation in SPARQL. Our preliminary implementation shows the feasibility of all proposed approaches on top of an off-the-shelf triple store supporting SPARQL and SPARQL update (Apache TDB).

The problem of knowledge based update and belief revision has been extensively studied in the literature, although not in the context of SPARQL updates where facts to be deleted or inserted come from a query. As argued in Sect. 4.1, brave semantics implements the most established approach of adapting the new information fully via a minimal change, which is feasible in the setting of fixed RDFS$_\neg$ TBoxes. Also semantics deliberating between accepting and discarding change are known (see [10] for a survey). In [18] an approach involving user interaction to decide whether to accept or reject an individual axiom is considered, with some part of the update being computed automatically in order to ensure its consistency. We do not consider interactive procedures here (although they clearly make sense in the case of more complex TBoxes or for TBox updates). Instead, we rely on the resolution strategies which are simple for the user to understand and can be efficiently encoded in SPARQL. In a practical KB editing system, one should probably combine the two approaches, e.g. for resolving the intrinsic inconsistency. Likewise, the approaches [3,7,13] consider grounded updates only, whereas our focus is on implementation of updates in SPARQL. The approach in [7] captures RDFS and several additional types of constraints and is close in spirit to our brave semantics.

Intrinsic consistency of an update is a common assumption in knowledge base update (e.g. [5–7,14]), which can be easily violated in the case of SPARQL updates. It is worth noting that our resolution strategy for intrinsic inconsistency called safe rewriting can be combined with all three update semantics using just the basic SPARQL operators.

Much interesting work remains to be done in order to optimize rewritten updates. Moreover, we plan to further extend our work towards increasing coverage of more expressive logics and OWL profiles, namely additional axioms from OWL2 RL or OWL 2 QL [16].

Acknowledgements. This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT12-SEE, and EU IP project Optique (*Scalable End-user Access to Big Data*), grant agreement n. FP7-318338.

References

1. Ahmeti, A., Calvanese, D., Polleres, A.: Updating RDFS ABoxes and TBoxes in SPARQL. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 441–456. Springer, Heidelberg (2014)
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, February 2014
3. Benferhat, S., Bouraoui, Z., Papini, O., Würbel, E.: A prioritized assertional-based revision for DL-Lite knowledge bases. In: Fermé, E., Leite, J. (eds.) JELIA 2014. LNCS, vol. 8761, pp. 442–456. Springer, Heidelberg (2014)
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. Autom. Reasoning **39**(3), 385–429 (2007)
5. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of $DL-Lite$ knowledge bases. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 112–128. Springer, Heidelberg (2010)
6. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On instance-level update and erasure in description logic ontologies. J. Log. Comput. **19**(5), 745–770 (2009)
7. Flouris, G., Konstantinidis, G., Antoniou, G., Christophides, V.: Formal foundations for RDF/S kb evolution. Knowl. Inf. Syst. **35**(1), 153–191 (2013)
8. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 update. W3C Recommendation, World Wide Web Consortium, March 2013
9. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. J. Web Seman. **3**(2–3), 158–182 (2005)
10. Hansson, S.: A survey of non-prioritized belief revision. Erkenntnis **50**(2–3), 413–427 (1999)
11. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium, March 2013
12. Hayes, P., Patel-Schneider, P.: RDF 1.1 semantics. W3C Recommendation, World Wide Web Consortium, February 2014
13. Kharlamov, E., Zheleznyakov, D., Calvanese, D.: Capturing model-based ontology evolution at the instance level: the case of DL-Lite. J. Comput. Syst. Sci. **79**(6), 835–872 (2013)
14. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating description logic aboxes. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) KR, pp. 46–56. AAAI Press (2006)
15. Lutz, C., Seylan, I., Toman, D., Wolter, F.: The combined approach to OBDA: taming role hierarchies using filters. In: Kagal, L., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 314–330. Springer, Heidelberg (2013)

16. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: Owl 2 web ontology language profiles, 2nd edn. W3C Recommendation, World Wide Web Consortium, December 2012
17. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal deductive systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 53–67. Springer, Heidelberg (2007)
18. Nikitina, N., Rudolph, S., Glimm, B.: Interactive ontology revision. Web Seman. Sci. Serv. Agents World Wide Web **12–13**, 118–130 (2012). reasoning with context in the Semantic Web
19. Polleres, A., Hogan, A., Delbru, R., Umbrich, J.: RDFS and OWL reasoning for linked data. In: Rudolph, S., Gottlob, G., Horrocks, I., van Harmelen, F. (eds.) Reasoning Weg 2013. LNCS, vol. 8067, pp. 91–149. Springer, Heidelberg (2013)
20. Winslett, M.: Updating Logical Databases. Cambridge University Press, Cambridge (2005)