Dealing with Inconsistencies due to Class Disjointness in SPARQL Update

Albin Ahmeti[1,3], Diego Calvanese[2], Axel Polleres[3], and Vadim Savenkov[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. 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) atomic updates, or, where the full SPARQL Update language is allowed, the TBox language is restricted to RDFS where no inconsistencies can arise. In this paper we discuss directions to overcome these limitations. Starting from a DL-Lite fragment covering RDFS and concept/class disjointness axioms, we define two semantics for SPARQL Update: under cautious semantics, inconsistencies are resolved by rejecting *all* updates potentially introducing conflicts; under brave semantics, instead, conflicts are overridden in favor of new information where possible. The latter approach builds upon existing work on the evolution of DL-Lite knowledge bases, setting it in the context of generic SPARQL updates.

1 Introduction

This paper initiates the study of SPARQL updates in the context of potential inconsistencies: as a minimalistic ontology language allowing for inconsistencies, we consider RDFS$_\neg$, an extension of RDFS [8] with *class disjointness axioms* of the form $\{P$ `disjointWith` $Q\}$ from OWL [10], sometimes referred to as *negative inclusions* or NIs [4], as the corresponding description logic encoding of this statement is $P \sqsubseteq \neg Q$.

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 [6] 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: $\mathcal{A}_1 = \{$`:jimmy :attendsClassOf :ann. :ann :attendsClassOf :jimmy`$\}$. Here, u would create two assertions `:jimmy :studentOf :ann` and `:ann :studentOf :jimmy`. Due to the range and domain constraints in $\mathcal{T}$, these assertions result in clashes both for Jimmy and for Ann. Note that all inconsistencies

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 [11].

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.

are in the new data, and thus we say that u is *intrinsically inconsistent* for the particular ABox $\mathcal{A}_1$. Our solution for such updates will be to discard problematic assignments but keep those that cause no clashes.

Now, let $\mathcal{A}_2$ be the ABox {`:jimmy :attendsClassOf :ann. :jimmy a :Professor`}. It is clear that after the update u, the ABox will become inconsistent, due to the property assertion `:jimmy :studentOf :ann`, implying that Jimmy is both a professor and a student which contradicts the TBox disjointness axiom. In contrast to the previous case, the clash now is between the prior knowledge and the new data. We propose *two update semantics*, extending our previous work [1] for dealing with such inconsistencies and provide *rewriting algorithms* for implementing them using the basic constructs of the SPARQL language (e.g., making use of the UNION, MINUS, FILTER, and OPTIONAL operators).

In the remainder of the paper, after some short preliminaries (Sec. 2) we discuss checking of intrinsic inconsistencies in Sec. 3, and then in Sec. 4 we present two semantics for dealing with general inconsistencies in the context of materialized triple stores. An overview of related work and concluding remarks can be found in Sec. 5.

2 Preliminaries

We introduce basic notions about RDF graphs, RDFS$_\neg$ ontologies, and SPARQL queries. In this paper we use RDF and DL notation interchangeably, treating RDF graphs that do not use non-standard RDFS$_\neg$ vocabulary [12] as sets of TBox and ABox assertions.

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 $\Delta^\mathcal{I}$,*

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

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

Fig. 1. Minimal RDFS rules from [11] plus class disjointness “clash” rule from OWL2 RL [10].

– *each negation of atomic concept A 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 $(\exists P)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x,y) \in P^{\mathcal{I}}\}$ resp. $(\exists P^-)^{\mathcal{I}} = \{y \in \Delta^{\mathcal{I}} \mid \exists x.(x,y) \in P^{\mathcal{I}}\}$. 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 restricted SPARQL queries corresponding to (unions of) conjunctive queries without non-distinguished variables 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 $x, y \in \Gamma \cup \mathcal{V}$.*[4] *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 $ans_{\mathsf{rdfs}}(q, G)$ (or simply $ans(q, G)$). The set of answers to a UCQ Q is $\bigcup_{q \in Q} ans(q, G)$.*

We also recall from [1], that query answering in the presence of ontologies can be done either by rule-based pre-materialization of the ABox or by query rewriting. Let $rewrite(q, \mathcal{T})$ be the UCQ resulting from applying $PerfectRef$ [3] (or, equivalently, the stripped-down version from [1, Alg.1]) to a query q and let $G = \mathcal{T} \cup \mathcal{A}$ be a triple store. Furthermore, let $mat(G)$ be the triple store obtained from exhaustive application of the inference rules in Fig. 1 on a consistent triple store G, and—analogously—let $chase(q, \mathcal{T})$ refer to "materialization" w.r.t. $\mathcal{T}$ applied to a CQ q. The next result transfers from [1] to consistent RDFS$_\neg$ stores.

Proposition 1. *Let $G = \mathcal{T} \cup \mathcal{A}$ be a consistent triple store, and q a CQ. Then, $ans(q, G) = ans(rewrite(q, \mathcal{T}), \mathcal{A}) = ans(q, mat(G))$.*

We have used this previously to define the semantics of SPARQL update operations.

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*

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

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.

[4] $\mathcal{V}$ is a countably infinite set of variables (written as '`?`'-prefixed alphanumeric strings).

For the sake of readability of the algorithms, we assume that all update operations $u(P_d, P_i, P_w)$ in this paper *contain no constants* in the BGPs P_d and P_i, and that all property assertions $(?X \ \mathsf{p} \ ?Y)$ in P_d have distinct variables $?X$ and $?Y$. Enhancing our results to updates with constants and variable equalities in P_d and P_i is not difficult, but requires distinguishing special cases: e.g., instead of replacing the variable y in a pattern Q by z, the expression $Q \ \mathsf{FILTER}(y = z)$ can be used in the case when y is a constant; instead of $Q(y) \ \mathsf{MINUS} \ P$ for a variable y, $Q \ \mathsf{FILTER\ NOT\ EXISTS} \ P$ should be used for ground Q.

We call a triple store or (ABox) *materialized* if in each state it always guarantees $G \setminus \mathcal{T} = mat(G) \setminus mat(\mathcal{T})$. In the present paper, we will always 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 *plus all their causes*; *(ii)* insert the instantiations of P_i *plus all their effects*.

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

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

Here, given a pattern P, $P^{\mathrm{caus}} = flatten(rewrite(P, \mathcal{T}))$; $P^{\mathrm{eff}} = chase(P, \mathcal{T})$ and $P^{fvars} = \{?v \ \mathsf{a} \ \texttt{rdfs:Resource} \mid ?v \in \mathcal{V}(P^{caus}) \setminus \mathcal{V}(P)\}$, where $flatten(\cdot)$ computes the set of all triples occurring in the UCQ $rewrite(P, \mathcal{T})$, which in our case allows us to obtain all possible "causes" of each atom in P_d, and "$\texttt{?v}$ a $\texttt{rdfs:Resource}$" is a shortcut for a pattern that binds $?v$ to any $x \in \Gamma$ 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 P_i and G. In what follows, we will discuss how this can be remedied.

3 Checking Consistency of a SPARQL Update

Within previous work on the evolution of DL-Lite knowledge bases [4], updates given in the form of 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 Def. 4, in [4] it is 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. This a priori assumption may be insufficient for SPARQL updates though, 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 Sec. 1 which, when applied to the ABox $\mathcal{A}_4$, 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 Def. 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. Def. 5)

```
Input: RDFS¬ 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 $\bot \in P_i^{\text{eff}}$ 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 $P_i^{\text{eff}}$ do
6         if $P \sqsubseteq \neg R \in \mathcal{T}$ then
7             $W := W$ UNION $\{\{P_w\theta_1[?X \mapsto ?Z]\} . \{P_w\theta_2[?Y \mapsto ?Z]\}\}$ 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 Sec. 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 [4], which we will discuss in Sec. 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 Sec. 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 Rem. 1 below.

Remark 1. 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. ■

Now, 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 Rem. 1 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 Alg. 1. Note that it suffices to

Algorithm 2: Safe rewriting $\mathit{safe}(u)$

```
   Input: RDFS$_\neg$ TBox $\mathcal{T}$, SPARQL update $u(P_d, P_i, P_w)$
   Output: SPARQL update $\mathit{safe}(u)$
1  if $\bot \in P_i^{\text{eff}}$ then
2      return $u(P_d, P_i, \mathsf{FILTER}(\mathit{False}))$
3  $W := \{\mathsf{FILTER}(\mathit{False})\}$;    //neutral element w.r.t. union
4  foreach pair of triple patterns $(?X\ \mathsf{a}\ P),\ (?Y\ \mathsf{a}\ R)$ in $P_i^{\text{eff}}$ do
5      if $P \sqsubseteq \neg R \in \mathcal{T}$ then
6          //cf. Rem. 1 for notation $\theta[\ldots]$
7          $W := W\ \mathsf{UNION}\ \{P_w\theta_1[?X \mapsto ?Y]\}\ \mathsf{UNION}\ \{P_w\theta_2[?Y \mapsto ?X]\}\}$
8  return $u(P_d, P_i, P_w\ \mathsf{MINUS}\ \{W\})$
```

check only triples of the form $\{?X\ \mathsf{a}\ ?C\}$ at line 5 of Alg. 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 Alg. 1 it suffices to check the disjointness conditions explicitly mentioned in $\mathcal{T}$ and not all those which are implied by $\mathcal{T}$.

Example 1. Consider the update u from Sec. 1, in which the INSERT clause P_i can create clashing triples. To identify potential clashes, Alg. 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 Table 1 with clashing concept names. The following ASK query is produced by Alg. 1.

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

(In this and subsequent examples we omit the trivial $\mathsf{FILTER}(\mathit{False})$ union branch used in rewritings to initialize variables with disjunctive conditions, such as W in Alg. 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* $\mathit{safe}(\cdot)$ in Alg. 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 Rem. 1, Alg. 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 a same value.

Example 2. Alg. 2 extends the WHERE clause of the update u from Sec. 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, $\mathit{safe}(u)$ has no effect on the ABox $\mathcal{A}_1$ from Sec. 1, since there is no cue, which of `:jimmy :attendsClassOf :ann`, `:ann :attendsClassOf :jimmy` 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, $\mathit{safe}(u)$

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 $\mathit{safe}(u)$. ■

A rationale for using MINUS rather than FILTER NOT EXISTS in Alg. 2 (and also in a rewriting in forthcoming Sec. 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 $[\![P_w]\!]_G^u$ 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 $\mathit{safe}(u) = (P_d, P_i, P'_w)$ result from applying Alg. 1 resp. Alg. 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) $q_u(G) = \mathit{True}$ iff $\exists \mu, \mu' \in [\![P_w]\!]_G^u$ s.t. $\mu(P_i) \wedge \mu'(P_i) \wedge \mathcal{T} \models \bot$;
(2) $[\![P_w]\!]_G^u \setminus [\![P'_w]\!]_G^u = \{\mu \in [\![P_w]\!]_G^u \mid \exists \mu' \in [\![P_w]\!]_G^u$ 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 two consistency preserving semantics are respectively called *brave* and *cautious*. 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.

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: $G^{\mathbf{Sem}^{mat}_{brave}}_{u(P_d,P_i,P_w)}$
1   $P'_d := P_d^{\text{caus}}$;
2   foreach triple pattern $(?X\ \mathsf{a}\ C)$ 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           if $(?X\ \mathsf{a}\ C') \notin P'_d$ then
5               $P'_d := P'_d \,.\, \{?X\ \mathsf{a}\ C'\}^{\text{caus}}$
6   return $G_{u(P'_d, P_i^{\text{eff}}, \{P_w\} P_d^{\mathit{fvars}})}$
```

Both semantics rely upon incremental update semantics $\mathbf{Sem}_2^{mat}$, introduced in Sec. 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 Sec. 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 Alg. 2 from some update satisfying Def. 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, perform neither deletions nor insertions.

For a SPARQL update u, we will define rewritings of u implementing the above semantics, which can be shown to be materialization preserving and consistency preserving.

4.1 Brave Semantics

The rewriting in Alg. 3 implements the brave update semantics $\mathbf{Sem}^{mat}_{brave}$; it can be viewed as combining the idea of *FastEvol* [4] 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.

The DELETE clause P'_d of the rewritten update is initialized with the set P_d of triples from the input update u. Rewriting ensures that also all "causes" of deleted facts are removed from the store, since otherwise deleted triples will be re-inserted by materialization. To this end, line 1 of Alg. 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 clashing patterns 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 all that are implied by $\mathcal{T}$), since we assume that the triple store is materialized.

Finally, the WHERE clause of the rewritten update is extended to satisfy the syntactic restriction that all variables in P'_d must have matches in the WHERE clause: bindings of "fresh" variables introduced to P'_d by eventual domain or range constraints are provided by the part P_d^{fvars}, cf. Def. 5 and Ex. 3 below. The rewritten update is evaluated over the triple store, computing its new materialized and consistent state.

Example 3. Ex. 2 in Sec. 3 provided a safe rewriting $\mathit{safe}(u)$ of the update u from Sec. 1. According to Alg. 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, and the respective triples in P'_d contain 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 P_w of $\mathit{safe}(u)$, which is a computationally reasonable implementation of the concept P^{fvars} from Def. 5. ■

Theorem 2. *Alg. 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.

4.2 Cautious Semantics

Unlike $\mathbf{Sem}^{mat}_{\mathit{brave}}$, its *cautious* version $\mathbf{Sem}^{mat}_{\mathit{caut}}$ 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.

As in the case of brave semantics, cautious semantics is implemented using rewriting, presented in Alg. 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 Alg. 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.

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

For each concept C' disjoint from C, we need to check that a triple matching the pattern $\{?X$ a $C'\}$ is in the store G and will not be deleted by u. Deletion happens if there is a pattern $\{?Y$ 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 Alg. 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 $\{?X$ 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 $\{?X$ a $C\} \in P_i^{\text{eff}}$.

Theorem 3. *Alg. 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 w.r.t.* cautious semantics.

Example 4. Alg. 4 rewrites the safe update $safe(u)$ from Ex. 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 Alg. 4 takes the DELETE clause of u' into account:

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

■

5 Discussion and Conclusions

In this paper, we have taken a step further from our previous work, in combining SPARQL Update and RDFS entailment by also adding class/concept disjointness as a first step towards dealing with inconsistencies in the context of SPARQL Update. As discussed throughout the paper, previous approaches to handle inconsistencies in DL KB evolution (e.g., [4, 5, 9]) have assumed that the set of ABox assertions to be inserted is intrinsically consistent w.r.t. the TBox, and thus inconsistencies are treated only w.r.t. the old state of the knowledge base. As we have shown, this assumption is not trivially verifiable in the context of SPARQL updates, where DELETE/INSERT atoms are instantiated by a WHERE clause, and clashing triples could be instantiated within the same INSERT operation. We have addressed this problem by providing means to check whether a SPARQL update is intrinsically consistent and defining a safe rewriting that removes intrinsic clashes during inserts on-the-fly.

Next, taken that the problem of intrinsic consistency is solved, we have demonstrated how to extend the approach of [4] to SPARQL updates. We have defined a materialization and consistency preserving rewriting for SPARQL updates that essentially combines the ideas of [4] and our previous work on SPARQL updates under RDFS for materialized triple stores [1], dealing with clashes due to class disjointness axioms in a brave manner. That is, we overwrite inconsistent information in the triple store in favor of information being inserted. Alternatively, we have also defined a dual consistency-preserving update semantics that on the contrary discards insertions that would lead to inconsistencies.

Besides practical evaluation of the proposed algorithms, we plan to further extend our work towards increasing coverage of more expressive logics and OWL profiles, namely additional axioms from OWL 2 RL or OWL 2 QL [10]. Also, it could be useful to investigate further semantics, allowing for compromises between fully discarding the inconsistent old data and refusing the entire update due to clashes, and lift our methods to work with stores that are not fully materialized.

The consideration of negative information is an important issue also in other related works on knowledge base updates: for instance, the seminal work on database view maintenance by Gupta et al. [7] is also used in the context of materialized views using Datalog rules with stratified negation. Likewise, let us mention the work of Winslett [13] on formula-based semantics to updates, where negation is also considered.

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: Proc. of the 13th Int. Semantic Web Conf. (ISWC). LNCS, vol. 8796, pp. 441–456 (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 (Feb 2014), available at http://www.w3.org/TR/turtle/

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. J. of Automated Reasoning 39(3), 385–429 (2007)
4. 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)
5. 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), `http://dx.doi.org/10.1093/logcom/exn051`
6. 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/`
7. 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)
8. Hayes, P., Patel-Schneider, P.: RDF 1.1 semantics. W3C Recommendation, World Wide Web Consortium (Feb 2014), available at `http://www.w3.org/TR/rdf11-mt/`
9. 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), `http://dblp.uni-trier.de/db/conf/kr/kr2006.html#LiuLMW06`
10. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: Owl 2 web ontology language profiles (second edition). W3C Recommendation, World Wide Web Consortium (Dec 2012), available at `http://www.w3.org/TR/owl2-profiles/`
11. 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)
12. 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)
13. Winslett, M.: Updating Logical Databases. Cambridge University Press (2005)