Updating RDFS: from Theory to Practice

Claudio Gutierrez[1], Carlos Hurtado[2], and Alejandro Vaisman[3]

[1] Universidad de Chile
[2] Universidad Adolfo Ibañez, Chile
[3] Universidad de la República, Uruguay

Abstract. There is a comprehensive body of theory studying updates and schema evolution of knowledge bases, ontologies, and in particular of RDFS. In this paper we turn these ideas into practice by presenting a feasible and practical procedure for updating RDFS. Along the lines of ontology evolution, we treat schema and instance updates separately, showing that RDFS instance updates are not only feasible, but also deterministic. For RDFS schema update, known to be intractable in the general abstract case, we show that it becomes feasible in real world datasets. We present for both, instance and schema update, simple and feasible algorithms.

1 Introduction

RDF has become one of the prime languages for publishing data on the Web, thanks to initiatives like Linked Data, Open Data, Datagovs, etc. The next step is to work on the evolution of such data, thus, facing the issue of information update. If one analyzes the data that is being published, the vocabulary used includes the core fragment of RDFS plus some OWL features. This poses strong challenges to the goal of updating such information. It is well-known that the problem of updating and schema evolution in Knowledge Bases is both, intractable and non-deterministic in the general case. For example, erasing a statement φ (that is, updating the knowledge base so that the statement φ can not be deduced from it) not only could take exponential time, but, there could be many different and equally reasonable solutions. Thus, there is no global solution and the problem has to be attacked by parts.

In this paper we study the problem of updating data under the RDFS vocabulary, considering the rest of the vocabulary as constant. Many proposals on updates in RDFS and light knowledge bases (e.g. DL-lite ontologies) have been presented and we discuss them in detail in Section 5. Nevertheless, such proposals have addressed the problem from a strictly theoretical point of view, making them –due to the inherent complexity of the general problem– hard or impossible to be used in practice.

Using the Katsuno-Mendelzon theoretical approach (from now on, K-M approach) for update and erasure [7], which has been investigated and proved fruitful for RDFS (see [2, 3, 5]), we show that updates in RDFS can be made practical. We are able to get this result by (a) following the approach typical in ontology evolution, where schema and instance updates are treated separately; (b) focusing on the particular form of the deductive rules of RDFS; and (c) considering blank nodes as constants (which for current big data sets is a rather safe assumption). In this paper we concentrate in the erasure operation ('deleting' a statement), because update (adding information) in RDFS, due to the positive logic nature of it, turns out to be almost trivial [5]. Our two main results are, a deterministic and efficient algorithm for updating instances, and a reduction of the update problem for schema to a graph theoretical problem in very small graphs.

Regarding instance update, we show that due to the particular form of the rules involved in RDFS [12], and using a case by case analysis, instance erasure (i.e., erasing data without touching the schema) is a deterministic process for RDFS, that is, it can be uniquely defined, hence opening the door to automate it. Then, we show that this process can be done efficiently, and reduces essentially to compute reachability in small graphs. We present pseudo-code of the algorithms that implement this procedure.

As for schema erasure, the problem is intrinsically non-deterministic, and worst, untractable in general. A trivial example is a chain of subclases $(a_i, \texttt{sc}, a_{i+1})$ from where one would like to erase the triple $(a_1, \texttt{sc}, a_n)$. The minimal solutions consist in deleting one of the triples. In fact, we show that in general, each solution corresponds bi-univocally to the well-known problem of finding minimal cuts for certain graphs constructed from the original RDF graph to be updated. This problem is known to be untractable. The good news here is that the graphs where the cuts have to be performed are very small (for the data we have, it is almost of constant size: see Table 1). They correspond essentially to the subgraphs containing triples with predicates subClassOf and subPropertyOf. Even better, the cuts have to be performed over each connected component of these graphs (one can avoid cuts between different connected components), whose size is proportional to the length of subClassOf (respectively subPropertyOf) chains in the original graph. We also present pseudo-code for this procedure.

The remainder of the paper is organized as follows. Section 2 reviews RDF notions and notations and the basics of the K-M approach to erasure. Section 3 studies the theoretical basis of the erasure operations proposed, and Section 4 puts to practice the ideas presenting algorithms for efficiently computing erasure in practice. Section 5 discusses related work. We conclude in Section 6.

2 Preliminaries

To make this paper self-contained we present in this section a brief review of basic notions on RDF, and theory of the K-M approach to update in RDFS. Most of the material in this section can be found in [4, 5, 12] with more detail.

Definition 1 (RDF Graph). *Consider infinite sets U (URI references); $B = \{N_j : j \in \mathbb{N}\}$ (Blank nodes); and L (RDF literals). A triple $(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an* RDF triple. *The union of U, B, L will be denoted by UBL.*

Dataset	# Triples	# Schema	#Instances	$\{\mathtt{sc}, \mathtt{sp}\}$-Chain-length	Most used voc.
bio2rdf (1)	2,024,177	685	1,963,738	3	type, label
data.gov.uk	22,504,895	16	22,503,962	1	type, value
bibsonomy	13,010,898	0	12,380,306	0	type, value
dbtune	58,920,361	418	58,248,647	7	type, label
geonames	9,415,253	0	9409247	0	type
uniprot	72,460,981	12295	72458497	4	type, reif.

Table 1. Statistics of triples in schema, instances and $\mathtt{sc}, \mathtt{sp}$ chains of some RDF datasets. (The difference between # triples and #(schema + instances) is due the predicates sameAs, sameClass, which being schema, do not have semantics in RDFS.)

An RDF graph *(just graph from now on) is a set of RDF triples. A* subgraph *is a subset of a graph. A graph is* ground *if it has no blank nodes.* □

A set of reserved words defined in RDF Schema (called the *rdfs-vocabulary*) can be used to describe properties like attributes of resources, and to represent relationships between resources. In this paper we restrict to a fragment of this vocabulary which represents the essential features of RDF and that contains the essential semantics (see [12]): [range], rdfs:domain [dom], rdf:type [type], rdfs: subClassOf [sc] and rdfs:subPropertyOf [sp]. The following set of rule schemas captures the semantics of this fragment [12]. In each rule schema, capital letters A, B, C, D, X, Y,... represent variables to be instantiated by elements of UBL.

GROUP A (Subproperty)

$$\frac{(A, \mathtt{sp}, B) \quad (B, \mathtt{sp}, C)}{(A, \mathtt{sp}, C)} \tag{1}$$

$$\frac{(A, \mathtt{sp}, B) \quad (X, A, Y)}{(X, B, Y)} \tag{2}$$

GROUP B (Subclass)

$$\frac{(A, \mathtt{sc}, B) \quad (B, \mathtt{sc}, C)}{(A, \mathtt{sc}, C)} \tag{3}$$

$$\frac{(A, \mathtt{sc}, B) \quad (X, \mathtt{type}, A)}{(X, \mathtt{type}, B)} \tag{4}$$

GROUP C (Typing)

$$\frac{(A, \mathtt{dom}, C) \quad (X, A, Y)}{(X, \mathtt{type}, C)} \tag{5}$$

$$\frac{(A, \mathtt{range}, D) \quad (X, A, Y)}{(Y, \mathtt{type}, D)} \tag{6}$$

Definition 2 (Proof Tree, Deduction). *Let G, H be RDF graphs, and t a triple. Then a* proof tree *of t from G is a tree constructed as follows: (1) The root is t; (2) The leaves are elements of G; (3) If t is a node, then t has children*

t_1, t_2 iff $\frac{t1 \quad t_2}{t}$ is the instantiation of a rule (see rules above). If t has a proof tree from G we will write $G \vdash t$.

A deduction of H from G is a set of proof trees from G, one for each $t \in H$. □

Definition 3 (Closure). *Let G be an RDF graph. The* closure *of G, denoted $cl(G)$, is the set of triples that can be deduced from G (under Definition 2), that is, $cl(G) = \{t : G \vdash t\}$.* □

The formalization of the K-M approach is based on the models of a theory. Thus we need the logical notion of a *model* of a formula (of an RDF graph). The model theory of RDF (given in [6]) follows standard classical treatment in logic with the notions of model, interpretation, and entailment, denoted $\models$ (see [4] for details). Also, throughout this paper we work with Herbrand models, which turn out to be special types of RDF graphs themselves. For a ground graph G, a Herbrand model of G is any RDF graph that contains $\mathrm{cl}(G)$ (in particular, $\mathrm{cl}(G)$ is a minimal model). $\mathrm{Mod}(G)$ will denote the set of such models of G. The deductive system presented is a faithful counterpart of these model-theoretic notions:

Proposition 1 (See [4, 12]). *(1) $G \models H$ iff $\mathrm{cl}(H) \subseteq \mathrm{cl}(G)$; (2) The deductive system of Definition 2 is sound and complete for $\models$ (modulo reflexivity of* `sc` *and* `sp`*).*[4]

2.1 Semantics of Erase in RDF

From a model-theoretic point of view, the K-M approach can be characterized as follows: for each model M of the theory to be changed, find the set of models of the sentence to be inserted that are 'closest' to M. The set of all models obtained in this way is the result of the change operation. Choosing an update operator then reduces to choosing a notion of closeness of models.

Working with positive theories like RDFS, the problem of adding positive knowledge (e.g. a triple, a graph H) to a given graph G is fairly straightforward. In fact, for ground graphs it corresponds to the union of the graphs. (See [5]). Thus, in what follows we concentrate in the 'erase' operation, that is, 'deleting' a triple t (or a graph H) from a given graph G. A standard approach in KB is to ensure that, after deletion, the statement t should not be derivable from G, and that the deletion should be minimal. The result should be expressed by another formula, usually in a more expressive language. We next characterize the erase operation using the K-M approach, which essentially states that, erasing statements from G means adding models to $\mathrm{Mod}(G)$, the set of models of G.

Definition 4 (Erase Operator). *The operator $\bullet$, representing the erasure, is defined as follows: for graphs G and H, the semantics of $G \bullet H$ is given by:*

$$\mathrm{Mod}(G \bullet H) = \mathrm{Mod}(G) \ \cup \bigcup_{m \in \mathrm{Mod}(G)} \min(((\mathrm{Mod}(H))^c, \leq_m) \qquad (7)$$

[4] As in [12], we are avoiding triples of the form $(a, \texttt{sc}, a)$ and $(b, \texttt{sp}, b)$, because this causes no harm to the core of the deductive system (see [12]).

where $(\ )^c$ *denotes complement. In words, the models of* $(G \bullet H)$ *are those of* G *plus the collection of models* $m_H \not\models H$ *such that there is a model* $m \models G$ *for which* m_H *is* $\leq_m$*-minimal among the elements of* $\mathrm{Mod}(H)^c$. □

The following standard notion of distance between models gives an order which is the one we will use in this paper. Recall that the the symmetric difference between two sets S_1 and S_2, denoted as $S_1 \oplus S_2$, is $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$.

Definition 5 (Order $\leq_m$). *Let* G, G_1, G_2 *be models of RDF graphs, and let* $\mathcal{G}$ *be a set of models of RDF graphs. Then : (1)* $G_1 \leq_G G_2$ *(*G_1 *is 'closer' to* G *than* G_2*) if and only if* $G_1 \oplus G \subseteq G_2 \oplus G$*; (2)* G_1 *is* $\leq_G$*-minimal in* $\mathcal{G}$ *if* $G_1 \in \mathcal{G}$*, and for all* $G_2 \in \mathcal{G}$*, if* $G_2 \leq_G G_1$ *then* $G_2 = G_1$. □

Representing faithfully in RDF the notions of erase defined above is not possible in the general case, given its lack of negation and disjunction. The next example illustrates the problems.

Example 1. Let us consider the graphs $G = \{(a, \mathtt{sc}, b), (b, \mathtt{sc}, c)\}$, and $H = \{(a, \mathtt{sc}, c)\}$. Any graph G' representing the result of this update cannot contain both $(a, \mathtt{sc}, b)$, and $(b, \mathtt{sc}, c)$, since this would derive $(a, \mathtt{sc}, c)$. Then, the result of the update should be $\{(a, \mathtt{sc}, b)\} \vee \{(b, \mathtt{sc}, c)\}$. Elaborating on this a little further, in Equation 7, $\mathrm{Mod}(H)^c$ are the models that cannot derive $(a, \mathtt{sc}, c)$. From these models, $\min((\mathrm{Mod}(H))^c, \leq_m)$ contains the ones at distance '1' from $\mathrm{Mod}(G)$, namely $\{\{(a, \mathtt{sc}, b)\}, \{(b, \mathtt{sc}, c)\}\}$. Any model that does not include $(a, \mathtt{sc}, b)$ or $(b, \mathtt{sc}, c)$ is at distance ≥ 2 from $Mod(G)$. Moreover, any model including both triples would not be in $(\mathrm{Mod}(H))^c$ since it would derive $(b, \mathtt{sc}, c)$. □

2.2 Approximating Erase in RDF

Knowing that it is not possible in general to find RDF graphs representing the new state after erasure, we study the 'closest' RDF formulas that express it. In this direction, we introduce the notion of erase candidate, which gives a workable characterization of erase (expressed previously only in terms of sets of models).

Definition 6 (Erase Candidate). *Let* G *and* H *be RDF graphs. An erase candidate of* $G \bullet H$ *is a maximal subgraph* G' *of* $\mathrm{cl}(G)$ *such that* $G' \not\models H$*. We denote* $\mathrm{ecand}(G, H)$ *the set of erase candidates of* $G \bullet H$. □

Example 2. For the RDF graph G of Figure 1 (a), the set $\mathrm{ecand}(G, \{(a, \mathtt{sc}, d)\})$ is shown in Figure 2 (a). □

The importance of $\mathrm{ecand}(G, H)$ resides in that it defines a partition of the set of models of $G \bullet H$, and a set of formulas whose disjunction represents erase:

Theorem 1 (See [5]). *Let* G, H *be RDF graphs.*

1. *If* $E \in \mathrm{ecand}(G, H)$*, then* $E \in \mathrm{Mod}(G \bullet H)$.
2. *If* $m \in \mathrm{Mod}(G \bullet H)$ *and* $m \notin \mathrm{Mod}(G)$*, then there is a unique* $E \in \mathrm{ecand}(G, H)$ *such that* $m \models E$.

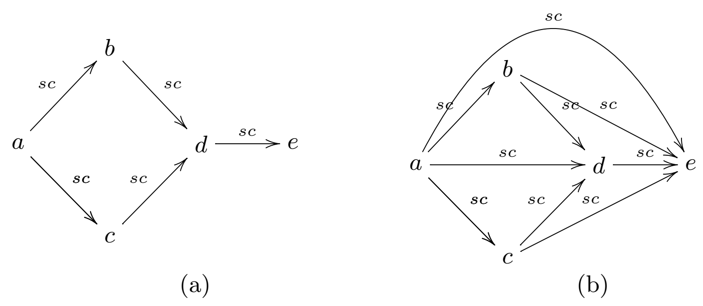

Fig. 1. (a) An RDF Graph G. (b) The closure of G.

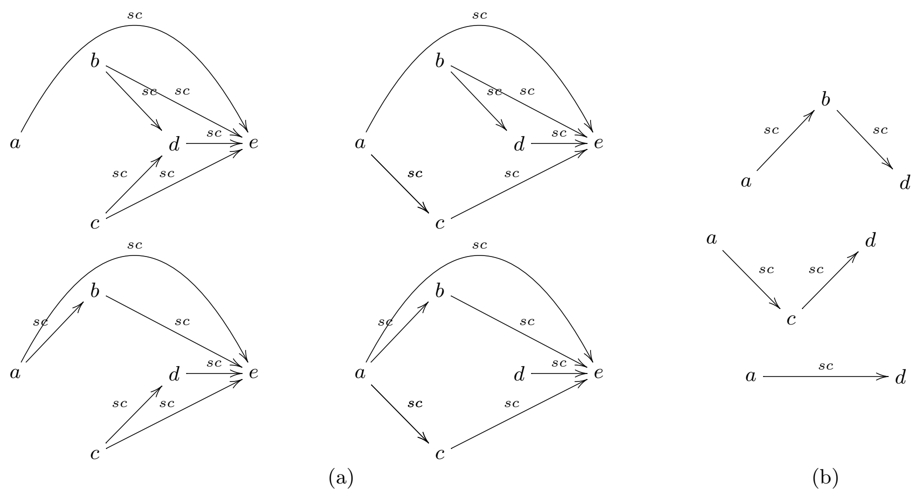

Fig. 2. (a) The set of erase candidates $\text{ecand}(G, \{(a, \texttt{sc}, d)\})$. (b) The set of minimal bases $\texttt{minbases}(\text{cl}(G), \{(a, \texttt{sc}, d)\})$

3. *For all formulas F of RDF, $(\bigcap_{E \in \text{ecand}(G,H)} E) \models F$ if and only if* $\text{Mod}(G \bullet H) \subseteq \text{Mod}(F)$.

Items (1) and (2) in Theorem 1 state that if we had disjunction in RDF, erasure could be expressed by the following finite disjunction of RDF graphs:

$$G \bullet H \;\; \text{“=”} \;\; G \vee E_1 \vee \cdots \vee E_n,$$

where E_j are the erase candidates of $G \bullet H$. Item (3) states the fact that all the statements entailed by $G \bullet H$ expressible in RDF are exactly represented by the RDF graph defined by the intersection of all the erase candidates graphs.

Note that the smaller the size of $\text{ecand}(G, H)$, the better the approximation to $G \bullet H$, being the limit the case when it is a singleton:

Corollary 1. *If* $\text{ecand}(G, H) = \{E\}$, *then* $(G \bullet H) \equiv E$. □

3 Computing the Erase in RDF

From the discussion above, it follows that approximating $G \bullet H$ reduces to find the erase candidates of this operation. For working purposes, it is easier to work with the 'complement' of them in $\mathrm{cl}(G)$, that we will call *delta candidates*:

Definition 7 (Delta Candidates $\mathrm{dcand}(G, H)$). *The set of delta candidates, denoted* $\mathrm{dcand}(G, H)$*, is the set of minimal graphs $D \subseteq \mathrm{cl}(G)$ such that $(\mathrm{cl}(G) \setminus D) \not\models H$.* $\square$

Thus, the relationship between delta and erase candidates is the following:

$$\mathrm{dcand}(G, H) = \{(\mathrm{cl}(G) \setminus E) : E \in \mathrm{ecand}(G, H)\}. \tag{8}$$

The remainder of this section provides a characterization of delta candidates, based in the notion of *proof tree* (Definition 2).

Definition 8 (Bases and Minimal Bases). *(1) The set of leaves of a proof tree (of H from G) is called the* base *of such proof.*

(2) A base *B of H from G, is a* minimal base *iff it is minimal under set-inclusion among all the bases of proofs of H from G (that is, for every base B' of H from G, it holds $B \subseteq B'$). We denote* $\texttt{minbases}(G, H)$ *the set of minimal bases of G, H.* $\square$

Example 3. For the graph G given in Figure 1 (a), the set $\texttt{minbases}(\mathrm{cl}(G), \{(a, \texttt{sc}, d)\})$ contains the graphs given in Figure 2 (b). $\square$

We now need to define the notion of a *hitting set*.

Definition 9 (Hitting Set). *A hitting set for a collection of sets $C_1, \ldots, C_n$ is a set C such that $C \cap C_i$ is non-empty for every C_i. C is called* minimal *if it is a minimal under set-inclusion.* $\square$

Theorem 2. *Let G, H, C be RDF graphs. Then, C is a hitting set for the collection of sets* $\texttt{minbases}(G, H)$ *iff $(\mathrm{cl}(G) \setminus C) \not\models H$. Moreover, C is a minimal hitting set iff $\mathrm{cl}(G) \setminus C$ is a maximal subgraph G' of cl(G) such that $G' \not\models H$.* $\square$

Proof. (sketch) Note that if C is a hitting set, its minimality follows from the maximality of its complement, $G \setminus C$, and vice versa. Hence we only have to prove that C is a hitting set for $\texttt{minbases}(G, H)$ iff $(G \setminus C) \not\models H$.

Now we are ready to give an operational characterization of delta candidates in terms of hitting sets and minimal bases.

Corollary 2. *Let G, H, C be RDF graphs. $C \in \mathrm{dcand}(G, H)$ if and only if C is a minimal hitting set for* $\texttt{minbases}(\mathrm{cl}(G), H)$*.* $\square$

Proof. Follows from the Definition 7, Theorem 2, and the observation that $C \subseteq \mathrm{cl}(G)$ is minimal iff $\mathrm{cl}(G) \setminus C$ is maximal. $\square$

3.1 Erasing a Triple from a Graph

Now we are ready to present algorithms to compute the delta candidates. We reduce computing erase candidates to finding minimal multicuts in certain directed graphs. The essential case is the deletion of one triple.

Definition 10 (Minimal Cut). *Let (V, E) be a directed graph. A set of edges $C \subseteq E$ disconnects two vertices $u, v \subseteq V$ iff each path from u to v in the graph passes through a vertex in C. In this case C is called a* cut. *This cut is* minimal *if the removal of any node from C does not yield another cut.*

Cuts can be generalized to sets of pairs of vertices yielding multicuts. *A* minimal multicut *for a set of pairs of nodes $(u_1, v_1), (u_2, v_2g), \ldots, (u_n, v_n)$ is a minimal set of edges that disconnects u_i and v_i, for all i.* □

For a triple t in a graph G, we will show that the graphs in dcand(G, t) correspond to certain cuts defined in two directed graphs derived from G, that we denote $G[\texttt{sc}]$ and $G[\texttt{sp}]$, defined as follows:

Definition 11 (Graphs $G[\texttt{sc}]$ and $G[\texttt{sp}]$). *Given an RDF graph G, we denote $G[\texttt{sc}] = (N, E)$ the directed graph defined in Table 2. For each triple of the form specified in the first column of the table, we have the corresponding edges in E. The set of nodes N is composed of all the nodes mentioned in the edges given in the table. The directed graph $G[\texttt{sp}]$ is defined similarly in Table 2. We use the letters n and m to refer to nodes in $G[\texttt{sc}]$ and $G[\texttt{sp}]$, respectively.* □

Triple in G	Edge in $G[\texttt{sc}]$
$(a, \texttt{sc}, b)$	(n_a, n_b)
$(a, \texttt{type}, b)$	$(n_{t,a}, n_b)$
Triple in G	**Edges in $G[\texttt{sp}]$**
$(p, \texttt{sp}, q)$	(m_p, m_q)
(a, p, b)	$(m_{a,b}, m_p)$
$(p, \texttt{dom}, c)$	$(m_p, m_{v,\texttt{dom}})$ for each $n_c \to^* n_v$ in $G[sc]$
$(p, \texttt{range}, c)$	$(m_p, m_{v,\texttt{range}})$ for each $n_c \to^* n_v$ in $G[sc]$

Table 2. Description of the construction of the graphs $G[\texttt{sc}]$ (above) and $G[\texttt{sp}]$ (below).

For an RDF triple t, the set of multicuts (set of pairs of nodes) associated to the erasure of t from an RDF graph G, is defined as follows:

Definition 12 (Set of edges $t[\texttt{sc}, G]$ and $t[\texttt{sp}, G]$). *The set $t[\texttt{sc}, G]$ contains the pairs of nodes (u, v) as described in Table 3 (second column) with u, v nodes in $G[\texttt{sc}]$. Analogously, we define $t[\texttt{sp}, G]$ using Table 3 (third column).* □

Example 4. Let us consider graph G on the left hand side of Figure 3. The center part and the right hand side show, respectively, the graphs $G[\texttt{sp}]$ and $G[\texttt{sc}]$, built according to Table 2. For example, for the triple $t = (d, \texttt{sc}, c)$, the sets of edges are $(d, \texttt{sc}, c)[\texttt{sc}, G] = \{(n_d, n_c)\}$ and $(d, \texttt{sc}, c)[\texttt{sp}, G] = \emptyset$. There are triples which

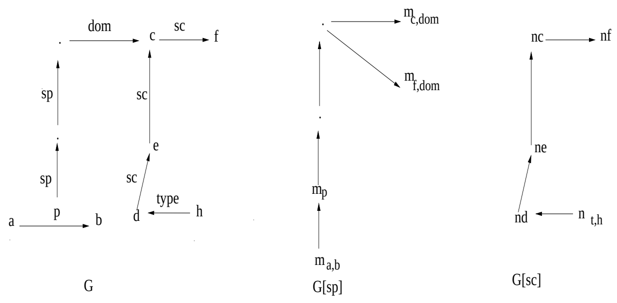

Fig. 3. An RDF graph G and its associated $G[\texttt{sc}]$ and $G[\texttt{sp}]$ graphs

give rise to multiple pairs of nodes. For example, for the triple $t = (a, \texttt{type}, c)$ and the graph in Figure 3, the sets contain the pairs $(a, \texttt{type}, c)[\texttt{sc}, G] = \{(n_{t,a}, n_c)\} \cap G[\texttt{sc}] = \emptyset$, and $(a, \texttt{type}, c)[\texttt{sp}, G] = \{(m_{ab}, m_{c,dom}), (m_{ba}, m_{c,dom})\}$. $\square$

$t \in G$	$t[\texttt{sc}, G]$	$t[\texttt{sp}, G]$
$(a, \texttt{sc}, b)$	(n_a, n_b)	–
$(a, \texttt{sp}, b)$	–	(m_a, m_b)
(a, p, b)	–	(m_{ab}, m_p)
$(a, \texttt{type}, c)$	$(n_{t,a}, n_c)$	pairs $(m_{a,x}, m_{c,\texttt{dom}})$ for all x
		pairs $(m_{x,a}, m_{c,\texttt{range}})$ for all x

Table 3. Construction of the pairs of nodes $t[\texttt{sc}, G]$ and $t[\texttt{sp}, G]$ associated to a triple t. The minimal multicuts of them in $G[\texttt{sc}]$ and $G[\texttt{sp}]$ respectively, will give the elements of $\text{dcand}(G, t)$ (Theorem 3).

The next theorem shows that computing the delta candidates can be reduced to compute minimal multicuts, in particular the set of cuts defined in Table 3 in the graphs defined in Table 2.

Theorem 3. *The elements of* $\text{dcand}(G, t)$ *are precisely the triples of* G *that correspond (according to the mapping in Table 2) to the minimal multicuts of* $t[\texttt{sc}, G]$ *in* $G[\texttt{sc}]$ *plus the minimal multicuts of* $t[\texttt{sp}, G]$ *in* $G[\texttt{sp}]$.

Proof. The proof is a case-by-case analysis of each form of t. For $t = (a, \texttt{dom}, c)$ or $t = (a, \texttt{range}, c)$, the set $\text{dcand}(G, t) = \{t\}$, because t cannot be derived by any rule, thus, $G \not\models t$ if and only if $t \notin G$.

Case $t = (a, sc, b)$. From the deduction rules in Section 2, t can be deduced from G if and only if there is a path in $G[\texttt{sc}]$ from n_a to n_b (note that the only rule that can derive t is (3)). Hence $\text{dcand}(G, t)$ is in correspondence with the set of the minimal cuts from n_a to n_b in $G[\texttt{sc}]$.

Case $t = (a, \texttt{sp}, b)$. This is similar to the previous one. ¿From the deduction rules, it follows that t can be only be deduced from G if there is a path in $G[\texttt{sp}]$ from m_a to m_b (by rule (1)). Thus $\mathrm{dcand}(G, t)$ is the set of the minimal cuts from m_a to m_b in $G[\texttt{sp}]$.

Case $t = (a, \texttt{type}, c)$. To deduce t we can use rules (4), (5) and (6). Rule (4) recursively needs a triple of the same form $(a, \texttt{type}, d)$ and additionally the fact that $(d, \texttt{sc}, c)$. Thus, t can be derived from G if there is path in $G[\texttt{sc}]$ from $n_{t,a}$ to n_c. Triple t can also be derived from (5) and (6). Let us analyze (5) (the other case is symmetric). We need the existence of triples (a, P, x) and $(P, \texttt{dom}, u)$ and $u \to^* c$ in $G[\texttt{sc}]$, i.e., $(u, \texttt{sc}, c)$. Then (a, P, x) can be recursively derived by rule(2) (and by *no other* rule); $(P, \texttt{dom}, u)$ should be present; and the last condition needs $(u, \texttt{sc}, c)$. Hence t can be derived if for some x there is a path from $m_{a,x}$ to $m_{c,dom}$ in $G[\texttt{sp}]$ (this explains the two last lines of Table 1).

Analyzing the rules, we can conclude that t is derivable from G if and only if we can avoid the previous forms of deducing it. That is, producing a minimal cut between $n_{t,a}$ and n_c in $G[\texttt{sc}]$ and a minimal multicut between the set of pairs $(m_{ax}, m_{c,dom})$ for all x, and the set of pairs $(m_{y,a}, m_{range,c})$ for all y, in the graph $G[\texttt{sp}]$.

Case $t = (a, p, b)$. Here, t can only be derived using rule (2). This needs the triples (a, q, b) and $(q, \texttt{sp}, p)$. With similar arguments as above, it can be shown that t can be derived from G iff there is path in $G[\texttt{sp}]$ from $m_{a,b}$ to m_p. Hence $\mathrm{dcand}(G, t)$ is the set of minimal cuts from $m_{a,b}$ to m_p in $G[\texttt{sp}]$. $\square$

The complexity of the above process is given essentially by the complexity of finding minimal multicuts:

Theorem 4. *Let G, H be ground RDF graphs, and t be a ground triple. The problem of deciding whether $E \in \mathrm{ecand}(G, t)$ is in PTIME.*

Proof. From Definition 6, the problem reduces to determine if $D = \mathrm{cl}(G) \setminus E$ is a delta candidate in $\mathrm{dcand}(G, t)$. Let $G' = \mathrm{cl}(G)$, G' can be computed in polytime. Theorem 3 shows that we have to test (i) whether $t[\texttt{sc}, G']$ is a minimal cut in $G'[\texttt{sc}]$ and (ii) whether $t[\texttt{sp}, G']$ is a minimal (multi)cut in $G'[\texttt{sp}]$. In both cases the test can be done in PTIME by simple reachability analysis in the graphs $G'[\texttt{sc}]$ and $G'[\texttt{sp}]$, respectively. Testing whether a set of edges S is a minimal cut for (v_1, u_1) in a graph (V, E) can be done performing polytime reachability analysis in the graph as follows. To test whether S is a cut, delete from E the edges in S, and test whether v_1 reaches u_1 in this new graph. To test minimality, do the same test for each set of edges $S' \subset S$ resulting from removing a single edge from S. S is minimal iff all of the S's are not cuts. We proceed similarly for testing if a set of edges is a minimal multicut. $\square$

3.2 Erasing a Graph from a Graph

The problem of computing erase candidates $\mathrm{ecand}(G, H)$ for the case where H has several triples can be easily reduced to the previous one when $H = \{t\}$.

Lemma 1. *Let G, H be ground RDF graphs in normal form (i.e. without redundancies, see [4]). (i) If $E \in \text{ecand}(G, H)$, then there exists a triple $t_i \in H$ such that $E \in \text{ecand}(G, \{t_i\})$; (ii) If $D \in \text{dcand}(G, H)$, then there exists a triple $t_i \in H$ such that $D \in \text{dcand}(G, \{t_i\})$.*

Proof. (i) Suppose $G \not\models H$, then there is a triple $t_i \in H$ such that $G \not\models t_i$, which yields $\text{ecand}(G, H) = \{G\} = \text{ecand}(G, \{t_i\})$. Now we assume that $G \models H$. That is $H \subseteq \text{nf}(G)$. Let $T = (H \setminus I)$. T is non-empty because $I \not\models H$ and $\text{nf}(E) = E$. Now if T has more than one triple, then we add one triple of T to I and obtain $I' \in \text{ecand}(G, H)$ which is greater than I contradicting that E is maximal. Therefore T must have exactly one triple, say t_j. In this case can be easily verified that $E = \text{ecand}(G, \{t_j\})$. (ii) Follows directly from (ii).

The intuition of Lemma 1 is that each delete candidate in $\text{dcand}(G, H)$ is also a delete candidate of $\text{dcand}(G, \{t_i\})$ for some triple t_i in H. Therefore, the problem of computing delete candidates reduces to finding the minimal sets among the delete candidates associated to each triple in H.

The following result, consequence of Lemma 1(ii), yields a basic procedure for computing delete candidates: find the minimal cuts for $\text{ecand}(G, \{t\})$ for each triple $t \in H$, and then find the minimum among them.

Proposition 2. *Let G and H be ground RDF graphs. Then*
$\text{dcand}(G, H) = \min\{D : D \in \bigcup_{t \in H} \text{dcand}(G, t)\}$.

4 Computing the Delta Candidates in Practice

We have seen that if we had disjunction, erase could be expressed as $G \bullet H = G \vee E_1 \vee \cdots \vee E_n$, where the E_i's are the erase candidates. From each E_i we get a delta candidate $D_i = \text{cl}(G) \setminus E_i$. In Section 3 we studied how to compute the D_i's borrowing standard machinery from graph theory. This computation is hard in the general case. In practice, however, there are two factors which turn this computation feasible. First, in RDFS data, as in most data models, schemas are small and stable, and data are large and dynamic. Second, what really matters when computing RDFS updates of schemas are small parts of the schema, essentially the length of the subclass and subproperty chains. Table 1 shows some examples of well-known RDF datasets.

Taking into account these observations, we present practical and feasible algorithms for updating RDF data. We concentrate in the case of a single triple which is the kernel of the operation (as can be deduced from Lemma 1).

4.1 Computing RDF Schema Erasure

We have already reduced the computation of erasure to that of computing the set $\text{ecand}(G, t)$. Algorithm 1 indicates the steps to be done. We have so far studied the decision problem related to computing the set of erase candidates. Generating $\text{ecand}(G, H)$ (respectively $\text{dcand}(G, H)$) requires, in the worst case,

Algorithm 1 Compute dcand(G, t) (General Case)

```
Input: Graph $G$, triple $t$
Output: dcand$(G, t)$
1: Compute $G := \mathrm{cl}(G)$
2: Compute $G[sc]$
3: Compute $G[sp]$
4: Compute $t[sc, G]$
5: Compute $t[sp, G]$
6: Compute minimal multicults for $t[sc, G]$ in $G[sc, G]$
7: Compute minimal multicults for $t[sp, G]$ in $G[sp, G]$
```

time exponential in the size of the graphs $G[sp]$ and $G[sc]$. Indeed, the number of cuts could be exponential. Standard algorithms on cut enumeration for directed graphs can be adapted to our setting [9].

The good news is that what really matters is the size of the maximal connected component of the graphs (one can avoid cuts between disconnected components). In our case, the size of the connected components of $G[sc]$ and $G[sp]$ are small, and a good estimation of it is the length of the maximal chain of `sc` and `sp` respectively (very small in most real-world datasets). Based on the above, Algorithm 2 would be a feasible one for updating schemas in most practical cases.

Algorithm 2 Update schema of G by erasing t

```
Input: Graph $G$, triple $t$
Output: $G \bullet t$
1: Choose an ordering on predicates (see e.g. [8], 2.3)
2: Compute dcand$(G, t)$
3: Order the elements $D \in$ dcand$(G, t)$ under this ranking
4: Delete the minimal $D$ from $G$
```

4.2 Computing RDF Instance Erasure

For the case of instances, the situation is optimal. Instance erasure assumes that the schema of the graph should remain untouched. In this setting, we will show that (i) the procedure is deterministic, that is, there is a unique choice of deletion (or in other words, dcand(G, t) has a single element); (ii) this process can be done efficiently.

Algorithm 3 computes dcand(G, t) for the case of instances. The key fact to note is that for instances t, that is, triples of the form $(a, \texttt{type}, b)$ or (a, p, b), where p does not belong to RDFS vocabulary, the minimal multicut is unique. For triples of the form (a, p, b), it follows from Table 3 that one has to cut paths from m_{ab} to m_p in $G[sp]$. Note that nodes of the form m_p are non-leaf ones, hence all edges in such paths in $G[sp]$ come from schema triples $(u, \texttt{sp}, v)$ (see Table

Algorithm 3 Compute dcand(G, t) (Optimized version for Instances)

```
Input:  Graph $G$, triple $t$
Output: dcand$(G, t)$
 1: Compute $G' := \mathrm{cl}(G)$
 2: Compute $G'[sc]$
 3: Compute $G'[sp]$
 4: Compute $t[sc, G']$
 5: Compute $t[sp, G']$
 6: if $t = (a, \mathtt{type}, b)$ then
 7:     $D \leftarrow \{(a, \mathtt{type}, z) \in G :\ n_z$ reaches $n_b$ in $G'[sc]\ \}$
 8:     $D \leftarrow D \cup \{(a, p, x) \in G :\ m_{ax}$ reaches $m_{b,dom}$ in $G'[sp]\ \}$
 9:     $D \leftarrow D \cup \{(y, p, a) \in G :\ m_{ya}$ reaches $m_{b,range}$ in $G'[sp]\ \}$
10: else
11:     if $t = (a, p, b)$ then
12:         $D \leftarrow \{(a, w, b) \in G :\ m_{ab}$ reaches $m_p$ in $G'[sp]\ \}$
13:     end if
14: end if
15: dcand$(G, t) \leftarrow D$
```

2). Because we cannot touch the schema, if follows that if there is such path, the unique option is to eliminate the edge m_{ab}, which corresponds to triples (a, w, b) in G. For triples of the form $(a, \mathtt{type}, b)$ the analysis (omitted here for the sake of space) is similar, though slightly more involved. The cost of Algorithm 3 is essentially the computation of the graphs $G[sc]$ and $G[sp]$ and then reachability tests. Once the triples are ordered by subject and object (time $O(n \lg n)$), the rest can be done in linear time.

5 Related Work

Updates have attracted the attention of the RDF community, although so far only addressing updates to the instance part of an RDF graph. Sarkar [14] identifies five update operators: and presented algorithms for two of them. Zhan [17] proposes an extension to RQL, and defines a set of update operators. Both works define updates in an operational way, and semantic issues are considered to a very limited extent. Ognyanov and Kiryakov [13] state that the two basic types of updates in an RDF repository are the addition and the removal of a statement (triple), and describe a graph updating procedure. Magiridou *et al* [11] introduce RUL, a declarative update language for RDF with three operations: *insert, delete* and *modify* Schema updates are not studied, however. SPARQL/Update [16] is an extension to SPARQL where update operations are performed over a collection of RDF graphs. The treatment is purely syntactic, not considering the semantics of RDFS vocabulary. In this sense, the work of the present paper can be considered as input for future enrichments of this language to include the semantics of RDF.

Konstantinidis *et al.* [8] introduce a framework for RDF/S ontology evolution, based on the belief revision principles of Success and Validity. The authors map RDF to First-Order Logic (FOL), and combine FOL rules (representing the RDF ontology), with a set of validity rules (which capture the semantics of the language), showing that this combination captures an interesting fragment of the RDFS data model. Finally, an ontology is represented as a set of positive *ground* facts, and an update is a set of negative *ground* facts. If the update causes side-effects on the ontology defined above, they chose the best option approach based on the principle of minimal change, for which they define an order between the FOL predicates. Note that the paper overrides the lack of disjunction and negation in RDF, by means of working with FOL. Opposite to this approach, in the present paper we remain within RDF.

Chirkova and Fletcher [2], building on [5] and in [8], present a preliminary study of what they call well-behaved RDF schema evolution (namely, updates that are unique and well-defined). They focus in the deletion of a triple from an RDF graph, and define a notion of determinism, showing that when deleting a triple t from a graph G, an update graph for the deletion exists (and is unique), if and only if t is not in the closure of G, or t is deterministic in G. Although closely related to our work, the proposal does not study instance and schema updates separately, and practical issues are not discussed.

There is an important corpus of work in the field of Description Logics ontologies. These knowledge bases are composed of two parts denoted TBox and ABox, expressing intensional and extensional knowledge, respectively. Currently existing work (discussed below) only address the case of updates to the extensional part (i.e., *instance* updates). De Giacomo *et al.* [3] study the non-expressibility problem for *erasure.* The non-expressibility problem states that given a fixed Description Logic $\mathcal{L}$, the result of an *instance level* update/erasure is not expressible in $\mathcal{L}$ (for update, this has already been proved by Liu *et al.* [10]). That is, they study the problem of updating only the ABox part of a DL ontology, assuming that the schema (i.e., the TBox) remains unchanged. For update they use the possible models approach [15], and for erasure, the K-M approach we use in this paper. They address the problem by means of approximation, building also in the ideas expressed in [5]. The authors show that, for a fragment of Description Logic, updates can be performed in polynomial time with respect to the sizes of the original knowledge base and the update formula. Finally, Calvanese *et al.* also study updates to ABoxes in DL-Lite ontologies. The authors present a classification of the existing approaches to evolution, and show that ABox evolution under what they define as *bold semantics* is uniquely defined [1].

6 Conclusions

In this paper, following the K-M approach to update in RDFS, we focus in bringing to practice the theory developed on this topic. Following the approach typical in ontology evolution, where schema and instance updates are treated separately, we proposed practical procedures for computing schema and instance RDF erasure.

One of the main results presented in this paper is that instance erasure is a deterministic and feasible process for RDFS. Further, we presented an algorithm to perform it. For schema erasure, the problem is non-deterministic and untractable in the general case. However, we show that since schemas are very small in practice, it can become tractable. Thus, we proposed an algorithm to compute schema updates, based on computing multicuts in graphs. Complexity analysis are provided in all cases.

Future work includes developing an update language for RDF based on the principles studied here, and implementing the proposal at big scale.

Acknowledgements. This work was partly done during a visit of Claudio Gutierrez to INCO, Univ. Republica, Uruguay, in 2010. Gutierrez thanks Project Fondecyt 1110287.

References

1. Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Evolution of DL-Lite knowledge bases. In *ISWC*, 2010.
2. Rada Chirkova and George H. L. Fletcher. Towards well-behaved schema evolution. In *WebDB*, 2009.
3. Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On instance-level update and erasure in description logic ontologies. *J. Log. Comput.*, 19(5):745–770, 2009.
4. Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, and Jorge Pérez. Foundations of semantic web databases. *Journal of Computer and System Sciences (to appear)*, 2010. This is the journal version of the paper with same title presented at the PODS Conference, 2004, Proc. PODS, pp. 95-106.
5. Claudio Gutiérrez, Carlos A. Hurtado, and Alejandro A. Vaisman. The meaning of erasing in RDF under the katsuno-mendelzon approach. In *WebDB*, 2006.
6. Patrick Hayes(Ed.). RDF semantics. *W3C Working Draft, October 1st., 2003.*
7. H. Katsuno and A. O. Mendelzon. On the difference between updating knowledge base and revising it. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 387–394, Cambridge, MA, 1991.
8. G. Konstantinidis, G. Flouris, G. Antoniou, and V. Christophides. A formal approach for RDF/S ontology evolution. In *ECAI*, pages 70–74, 2008.
9. Hung-Yau Lin, Sy-Yen Kuo, and Fu-Min Yeh. Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD. In *IEEE Symposium on Computers and Communications (ISCC 2003), 30 June - 3 July, Kiris-Kemer, Turkey*, 2003.
10. Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating description logic aboxes. In *KR*, pages 46–56, 2006.
11. M. Magiridou, S. Sahtouris, S. Christophides, and M Koubarakis. RUL: A declarative update language for RDF. In *International Semantic Web Conference*, pages 506–521, 2005.
12. Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal deductive systems for rdf. In *ESWC*, pages 53–67, 2007.

13. D. Ognyanov and A. Kiryakov. Tracking changes in RDF(S) repositories. In *EKAW'02*, pages 373–378, Spain, 2002.
14. S. Sarkar and H.C. Ellis. Five update operations for RDF. *Rensselaer at Hartford Technical Report, RH-DOES-TR 03-04*, 2003.
15. Marianne Winslett. Reasoning about action using a possible models approach. In *AAAI*, pages 89–93, 1988.
16. WWW Consortium. *SPARQL/Update: A language for updating RDF graphs*, 2008. http://www.w3.org/Submission/SPARQL-Update/.
17. Y. Zhan. Updating RDF. In *21st Computer Science Conference*, Rensselaer at Hartford, 2005.