On Computing Deltas of RDF/S Knowledge Bases *

Dimitris Zeginis
Computer Science Department, University of Crete, and Institute of Computer Science, FORTH-ICS, Greece
zeginis@ics.forth.gr

Yannis Tzitzikas
Computer Science Department, University of Crete, and Institute of Computer Science, FORTH-ICS, Greece
tzitzik@ics.forth.gr

Vassilis Christophides
Computer Science Department, University of Crete, and Institute of Computer Science, FORTH-ICS, Greece
christop@ics.forth.gr

ABSTRACT

The ability to compute the differences that exist between two RDF/S knowledge bases (KB) is an important step to cope with the evolving nature of the Semantic Web (SW). In particular, RDF/S Deltas can be employed to reduce the amount of data that need to be exchanged and managed over the network in order to build SW synchronization and versioning services. By considering Deltas as sets of change operations, in this paper we introduce various RDF/S differential functions which take into account inferred knowledge from an RDF/S KB. We first study their correctness in transforming a source to a target RDF/S KB in conjunction with the semantics of the employed change operations (i.e. with or without side-effects). Then we formally analyze desired properties of RDF/S Deltas such as, size minimality, semantic identity, redundancy elimination, reversibility and composability. Finally, we report experimental results regarding the computing time and size of produced Deltas over real and synthetic RDF/S KBs.

1. INTRODUCTION

In order to cope with the evolving nature of the Semantic Web (SW) we need effective and efficient support for building advanced SW synchronization and versioning services. RDF/S Deltas, reporting the differences that exist between two RDF/S Knowledge Bases (KB) have been proven to be crucial in order to reduce the amount of data that need to be exchanged in this respect over the network [14, 15, 2, 6].

Although RDF/S knowledge bases can be serialized in various text formats (e.g., XML[1], N-Triples[2], Trix[3]), a straightforward application of existing version control systems for software code, such as RCS[4] or CVS[5], or for XML data, such as [19], [7], is not a viable solution for computing RDF/S Deltas. This is mainly due to the fact that RDF Knowledge Bases, essentially represent graphs which (a) may feature several possible serializations (since there is no notion of edge ordering in [3]) and (b) are enriched with the semantics of RDFS specification (also including inferred edges from transitive relations such as subsumption [4]). For these reasons, several non text-based tools have been recently developed for comparing RDF/S graphs produced autonomously on the SW, as for example, SemVersion [27], PromptDiff [21], Ontoview [15], [8] and [2].

In most cases, the output of these tools is mainly exploited by humans, and thus an intuitive presentation of the produced Deltas (and other related issues) has received considerable attention. SemVersion [27] proposes two Diff algorithms: (a) one *structure-based* which returns a set-based difference of the triples explicitly forming the two graphs, and (b) one *semantic-aware* which also takes into account the triples inferred by the associated RDFS schemas. PromptDiff [21, 22, 20] is an ontology-versioning environment, that includes a version-differential algorithm (based on heuristic matchers [21, 22]), while the visualization of the computed difference between two ontologies is discussed in [20]. Ontoview [15] is an ontology management system, able to compare two ontology versions and highlight their differences. Notably, it allows users to specify the conceptual relations (i.e. equivalence, subsumption) between the different versions of an ontology concept. Moreover, [8, 10] introduce the notion of RDF/S molecules as the finest components to be used when comparing RDF/S graphs. Finally, tracking the evolution of ontologies when changes are preformed in more controlled environments (e.g. collaborative authoring tools) has been addressed in [16, 23, 31].

However, existing RDF/S differential tools have not yet focused on the formal properties (e.g. size) of the produced Deltas. In this paper we are interested in computing RDF/S Deltas as sets of change operations that enable us to successfully transform one RDF/S KB into the other under different update semantics (with or without side-effects). In this context, rather than storing complete snapshots of all KB versions, RDF/S Deltas between consecutive KB versions can be organized in a chain. If a delta's target version is

*This paper is an extended version of [30] published in ISWC'07.
This work was conducted while the third author was visiting the Laboratory for Foundations of Computer Science, University of Edinburgh, UK.
[1]www.w3.org/TR/rdf-syntax-grammar/
[2]www.w3.org/2001/sw/RDFCore/ntriples/
[3]www.w3.org/2004/03/trix/
[4]www.gnu.org/software/rcs
[5]www.cvshome.org

newer than its source version in the archive, then it is refereed as *forward delta*. A *backward delta* has the opposite orientation. Consider, for example, the two RDF/S KBs K and K' of Figure 1 and their standard representation as sets of explicitly defined triples [4]: what set of change operations could transform K to K' ($\Delta(K \rightarrow K')$) or vice versa ($\Delta(K' \rightarrow K)$)?

To answer this question we need to consider the semantics of the update primitives such as $Add(t)$ and $Del(t)$ where t is a triple involving any RDF/S predicate. By assuming a side-effect free semantics for these primitives, i.e. $Add(t)$ (resp. $Del(t)$) is a straightforward addition (resp. deletion) of t to the set $Triples(K)$, K' can be obtained by executing the following set Δ_e (e stands for *e*xplicit) of change operations:

$$\Delta_e = \{Del(TA\ subClassOf\ Person),\ Del(TA\ subClassOf\ Student), Del(Jim\ type\ TA),\ Add(TA\ subClassOf\ Graduate),\ Add(Student\ subClassOf\ Person),\ Add(Jim\ type\ Person)\}$$

Δ_e is actually composed of update operations over the explicit triples of K and K', and it is provided by the majority of existing RDF/S differential tools [2, 27, 8]. However, by assuming side-effects (on the inferred triples not represented in Figure 1) during the execution of the above update primitives, we can reach K' by applying on K the following set Δ_d (d stands for *d*ense) of change operations:

$$\Delta_d = \{Del(Jim\ type\ TA),\ Add(TA\ subClassOf\ Graduate),\ Add(Student\ subClassOf\ Person)\}$$

As we can easily observe, Δ_d has only three change operations in contrast to Δ_e that has six, given that inferred triples are also taken into account for the Delta computation. For example, $Del(TA\ subClassOf\ Person)$ is not included in Δ_d because it can be inferred from K'. As we can see in Figure 1, this differential function yields even smaller in size operation sets than the Δ_c (c stands for *c*losure) semantics-aware Delta of [27]. However, Δ_d cannot always successfully transform one RDF/S KB to another. Returning to our example of Figure 1, Δ_d cannot be used to migrate backwards from K' to K since $Del(Graduate\ subClassOf\ Person)$ is an operation not included in Δ_d. For this reason, we need to consider additional RDF/S differential functions involving inferred triples such as Δ_{dc} (dc stands for *d*ense & *c*losure) illustrated in Figure 1. Still the resulting sets of operations have at most the same size as those returned by Δ_c. Finally, we consider one more RDF/S differential function named Δ_{ed} (ed stands for *e*xplicit & *d*ense) illustrated in Figure 1. Still the resulting sets of operations have at most the same size as those returned by Δ_e.

Clearly, the storage requirements of the employed Deltas between consecutive RDF/S graph versions is crucial. *Small sized* Deltas yielding as less as possible update operations are quite beneficial in various backward and forward versioning policies. In extremis, Deltas should not report any change between two *semantically equivalent* RDF/S graphs (i.e. with the same set of explicit and inferred triples). Additionally, when executed, Deltas should at best result to a KB that is *redundancy free*. Furthermore, when we want to propagate changes across distributed RDF/S graph versions (i.e. synchronization), we also need Deltas that can be *reversed* and *composed* without materializing the involved intermediate RDF/S graph versions. In response to the above requirements, the main contributions of this paper are:

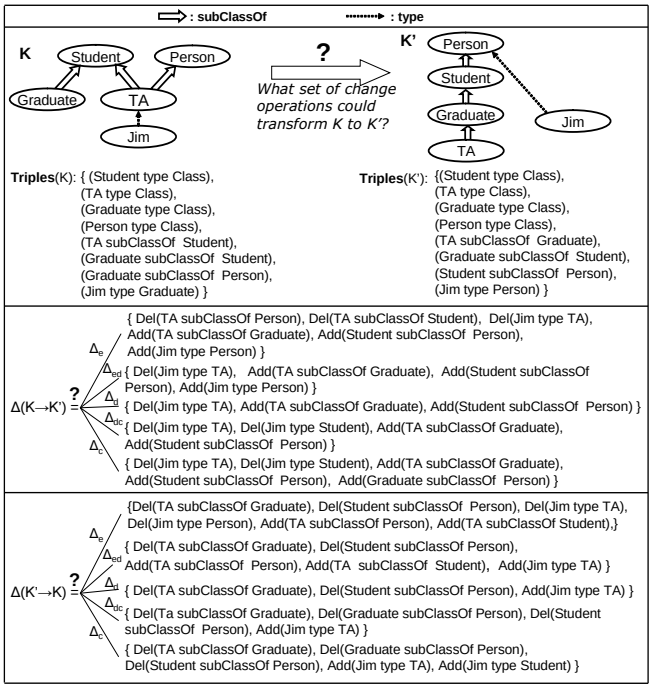


Figure 1: Transforming K to K' and vice versa

(a) We analyze five differential functions returning sets of changes operations, namely, *explicit* (Δ_e), *closure* (Δ_c), *dense* (Δ_d), *dense & closure* (Δ_{dc}), and *explicit & dense* (Δ_{ed}) .
(b) We consider two popular change operations semantics: one *plain* set-theoretic ($\mathcal{U}_p$), and another that involve *inference* and *redundancy elimination* ($\mathcal{U}_{ir}$).
(c) We study which combinations of change operation semantics and differential functions can be employed to *correctly* transform a source to a target RDF/S KB. In addition, we are interested in other useful properties such as semantic identity, non redundancy, reversibility and composition of RDF/S Deltas.

The rest of this paper is organized as follows. Section 2 introduce briefly RDF/S Knowledge Bases. Section 3 defines formally five RDF/S differential functions and discusses their size and time complexity. Section 4 elaborates on the change operations and their semantics, while Section 5 studies the properties of the introduced RDF/S Deltas. Please note that the proofs of all the Properties and Theorems presented in the paper can be found at [29]. Their experimental evaluation is presented in Section 6. Finally, Section 7 presents our conclusions and future research plans.

2. PRELIMINARIES: RDF/S KBS

In general, an RDF/S Knowledge Base (KB) is defined by a set of triples of the form (subject, predicate, object). Let $\mathcal{T}$ be the set of all possible triples that can be constructed from an infinite set of URIs (for resources, classes and properties) as well as literals [12]. Then, a KB can be seen as a finite subset K of $\mathcal{T}$, i.e. $K \subseteq \mathcal{T}$. Apart from the explicitly specified triples of a K, other triples can be inferred based on the RDF/S semantics [13]. For this reason, we introduce the notion of closure and reduction of RDF/S KBs.

The *closure* of a K, denoted by $C(K)$, contains all the triples that either are explicitly asserted or can be inferred from K by taking into account class or property assertions made by the associated RDFS schemas. Thus, we can consider that $C(K)$ is defined (and computed) by taking the reflexive and transitive closures of RDFS binary relations such as subClassof and type. It should be stressed that our work is orthogonal to the consequence operator of logic theories [9] actually employed to define the closure operator C. Specifically, if P denotes the powerset of all possible sets of triples of $\mathcal{T}$, then the closure operator can be defined as any function $C : P \rightarrow P$ that satisfies the following properties:

- $K \subseteq C(K)$ for all K, i.e. C is *extensive*
- If $K \subseteq K'$ then $C(K) \subseteq C(K')$, i.e. C is *monotonically increasing*
- $C(C(K)) = C(K)$ for all K, i.e. C is an *idempotent* function

If it holds $C(K) = K$, then we will call K *completed*. The elements of K will be called *explicit* triples, while the elements of $C(K) - K$ will be called *inferred*. We can now define an equivalence relation between two knowledge bases.

DEF. 1. Two knowledge bases K and K' are *equivalent*, denoted by $K \sim K'$, iff $C(K) = C(K')$.

The *reduction* of a K, denoted by $R(K)$, is the smallest in size set of triples such that $C(R(K)) = C(K)$. Let Ψ denote the set of all knowledge bases that have a unique reduction. Independently of whether the reduction of a K is unique or not, we can characterize a K as (semantically) *redundancy free*, and we can write $RF(K) = True$ (or just $RF(K)$), if it does not contain explicit triples which can be inferred from K. Formally, K is redundancy free if there is not any proper subset K' of K (i.e. $K' \subset K$) such that $K \sim K'$.

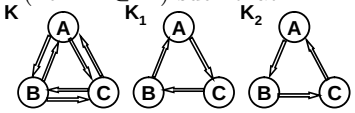


Figure 2: KB without unique reduction

It is worth noticing that the reduction of a K is not always unique. In general, uniqueness of the transitive reduction of a binary relation R is guaranteed only when R is antisymmetric and finite. Unfortunately, this is not the case of RDF/S KBs allowing cycles in the subsumption relations. For example, in Figure 2 we have $K \sim K_1 \sim K_2$, moreover $RF(K_1), RF(K_2)$, but $K_1 \neq K_2$.

3. RDF/S KBS DELTAS

In this Section, we formally define the five differential functions of RDF/S KBs introduced in Figure 1, namely, Δ_e, Δ_c, Δ_d, Δ_{dc} and Δ_{ed}.

$$
\begin{aligned}
\Delta_e(K \rightarrow K') &= \{Add(t) \mid t \in K' - K\} \cup \\
&\quad \{Del(t) \mid t \in K - K'\} \\
\Delta_c(K \rightarrow K') &= \{Add(t) \mid t \in C(K') - C(K)\} \cup \\
&\quad \{Del(t) \mid t \in C(K) - C(K')\} \\
\Delta_d(K \rightarrow K') &= \{Add(t) \mid t \in K' - C(K)\} \cup \\
&\quad \{Del(t) \mid t \in K - C(K')\} \\
\Delta_{dc}(K \rightarrow K') &= \{Add(t) \mid t \in K' - C(K)\} \cup \\
&\quad \{Del(t) \mid t \in C(K) - C(K')\} \\
\Delta_{ed}(K \rightarrow K') &= \{Add(t) \mid t \in K' - K\} \cup \\
&\quad \{Del(t) \mid t \in K - C(K')\}
\end{aligned}
$$

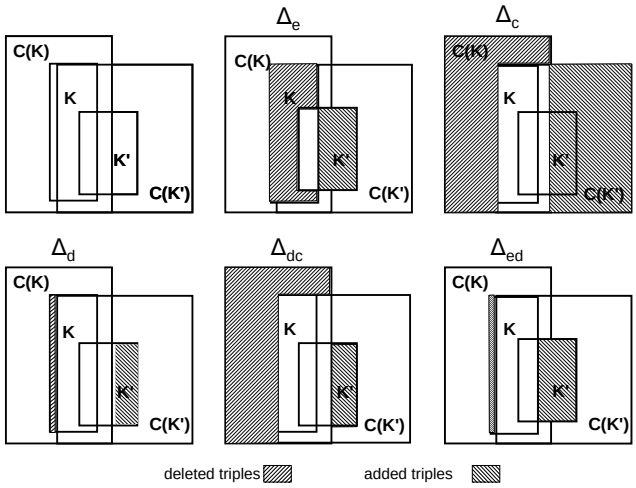


Figure 3: RDF/S KBs differential functions

Δ_e (where e stands for *e*xplicit) actually returns the set difference over the explicitly asserted triples, while Δ_c (where c stands for *c*losure) returns the set difference by also taking into account the inferred triples. As we mentioned in Section 1, existing approaches (e.g. [27]) are based on Δ_e and Δ_c. However, as we are interested in small sized Deltas, we introduce three novel differential functions namely Δ_d (where d comes from *d*ense), Δ_{dc} (dc comes from *d*ense & *c*losure) and Δ_{ed} (ed comes from *e*xplicit & *d*ense). It is not hard to see that Δ_d produce the smallest in size set of change operations. Figure 3 shows the Venn diagrams of the corresponding sets of triples to be added and deleted, in the general case of overlapping K and K. Unfortunately, and as we will see at Section 5, Δ_d can be actually applied to transform K to K only under certain conditions. For this reason, we additionally consider Δ_{dc} and Δ_{ed} yielding smaller in size deltas than Δ_c and Δ_e respectively. Δ_{dc} resembles to Δ_d regarding additions and Δ_c regarding deletions, while Δ_{ed} resembles to Δ_e regarding additions and Δ_d regarding deletions.

3.1 RDF/S Delta Size

Let $|\Delta(K \rightarrow K')|$ denote the *number* of change operations in $\Delta(K \rightarrow K')$. To keep notation simple we shall also use $|\Delta_x|$ to denote $|\Delta_x(K \rightarrow K')|$ for any $x \in \{e, c, d, dc, ed\}$.

LEMMA 1. For any pair of knowledge bases K and K' it holds:

$$
\begin{aligned}
|\Delta_d| &\leq |\Delta_{ed}| \leq |\Delta_e| \\
|\Delta_d| &\leq |\Delta_{dc}| \leq |\Delta_c|
\end{aligned}
$$

In a nutshell, Δ_d gives always the smallest in size Deltas, Δ_{dc} gives smaller Deltas than Δ_c, Δ_{ed} gives smaller Deltas than Δ_e, while Δ_{dc} is incomparable to Δ_{ed}. Figure 4(a) illustrates the Hasse diagram of the ordering relation of Delta sizes in the general case. Under specific conditions regarding K, K', $C(K)$ and $C(K')$ the above definitions may coincide.

PROP. 1. If $K \subseteq K'$ then the following relationships also hold:

$$
\begin{aligned}
|\Delta_e| &= |\Delta_{ed}| \\
|\Delta_d| &= |\Delta_{dc}|
\end{aligned}
$$

This case corresponds to the quite frequent scenario where K is stored in KB (and therefore it might be also redundancy free), and K' is derived from K by adding new triples. Hence, Δ_d and Δ_{dc} give the same in size Deltas, as also happens for Δ_e and Δ_{ed}. Figure 4(b) shows the corresponding Hasse diagram.

PROP. 2. If $K \supseteq K'$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{ed}|$$
$$|\Delta_c| = |\Delta_{dc}|$$

This case corresponds to the scenario where K' is generated only by deleting triples from K. Hence, Δ_d and Δ_{ed} (alternately Δ_c and Δ_{dc}) give the same in size Deltas. Figure 4(c) shows the corresponding Hasse diagram.

PROP. 3. If $K = C(K)$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_{ed}|$$

This case corresponds to the scenario where either the subsumption hierarchy of K consists of only one level (i.e. no triples can be inferred) or the closure of K is already materialized. Thus, Δ_d, Δ_{dc} and Δ_{ed} give the same in size Deltas. Figure 4(d) shows the corresponding Hasse diagram.

PROP. 4. If $K' = C(K')$ then the following relationships also hold:

$$|\Delta_c| = |\Delta_{dc}|$$
$$|\Delta_e| = |\Delta_{ed}|$$

This case corresponds to the scenario where either the subsumption hierarchy of K' consists of only one level (i.e. no triples can be inferred) or the closure of K' is already materialized. Thus, Δ_c and Δ_{dc} give the same in size Deltas, as also happens for Δ_e and Δ_{ed}. Figure 4(e) shows the corresponding Hasse diagram.

PROP. 5. If $K = C(K')$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_c| = |\Delta_{ed}| = 0$$
$$|\Delta_e| \geq 0$$

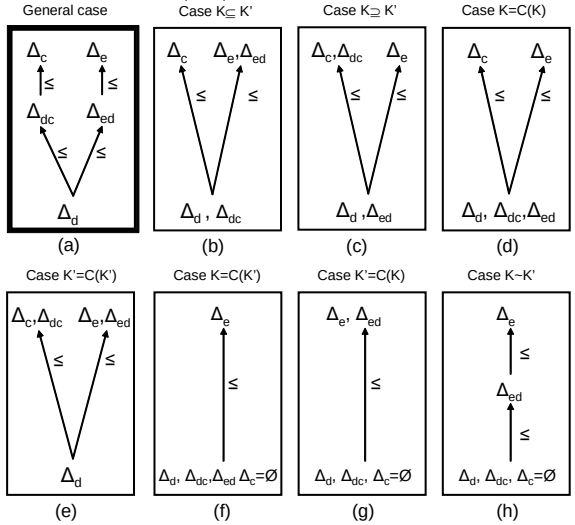


Figure 4: Ordering Relation of RDF/S Delta Sizes

In this case K is semantically equivalent to K'. This case arise when K contains explicit triples that became implicit in K'. Hence, unlike Δ_e, the differential function Δ_c, Δ_{dc}, Δ_c and Δ_{ed} give an empty result. Figure 4(f) shows the corresponding Hasse diagram.

PROP. 6. If $K' = C(K)$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_c| = 0$$
$$|\Delta_e| = |\Delta_{ed}| \geq 0$$

In this case K' contains explicit triples that was implicit in K'. Thus, Δ_d, Δ_{dc} and Δ_c give an empty result, while Δ_e and Δ_{ed} give the same non-empty result. Figure 4(g) shows the corresponding Hasse diagram.

PROP. 7. If $K \sim K'$ then the following relationships also hold:

$$|\Delta_d| = |\Delta_{dc}| = |\Delta_c| = 0$$
$$0 \leq |\Delta_{ed}| \leq |\Delta_e|$$

In this case K and K' are equivalent. Thus, Δ_c, Δ_{dc} and Δ_c give an empty result, while Δ_e and Δ_{ed} give a non-empty result with the Δ_{ed} producing a smaller in size result. Figure 4(h) shows the corresponding diagram.

3.2 RDF/S Deltas Complexity

Let's consider the $\Delta_x(K_1 \rightarrow K_2)$ of two RDF/S KBs K_1 and K_2 for any $x \in \{e, c, d, dc, ed\}$. Let's also consider the following sizes: $N_1 = |K_1|$, $N_2 = |K_2|$, $C_1 = |C(K_1)|$, $C_2 = |C(K_2)|$, as well as $N = \max(N_1, N_2)$, $C = \max(C_1, C_2)$ and $M = max(N_2, C_1)$. Note that in the worst case, $C_1 = N_1^2$ and $C_2 = N_2^2$. We study the cost of computing $\Delta_x(K_1 \rightarrow K_2)$ in the following three different settings:

(i) $C(K_1)$ and $C(K_2)$ are precomputed and stored along with K_1, K_2 in corresponding hashtables. Thus, we can decide in $O(1)$ whether a triple t belongs to any of these collections.

(ii) Only K_1 and K_2 are stored in hashtables, while $C(K_1)$ ($C(K_2)$) are computed on the fly with a cost $O(N_1^2)$ and ($O(N_2^2)$ respectively [28].

(iii) Only K_1 and K_2 are stored, but interval-based labels [5] have been precomputed (with a cost $O(K_1)$ and $O(K_2)$) to encode transitive subsumption relationships of the two KBs. Thus, we can check in $O(\gamma)$ whether $t \in C(K_1)$ (or $t \in C(K_2)$), where γ will be discussed later on.

Table 1 summarizes for each setting the time complexity of Δ_x. There are essentially two major cost factors: whether a Δ_x (a) requires the closure computation of the operant KBs and (b) requires the scan of the involved KB closures. Clearly, with the exception of Δ_e, Δ_c, Δ_d and Δ_{dc} need to compute the closure of both K_1 and K_2, while Δ_{ed} only of K_2 (as depicted in column ii) of Table 1). However, only Δ_c requires to scan both KB closures: Δ_d and Δ_{ed} do not imply any scan while Δ_{dc} scans only $C(K_1)$ (as depicted in column i) of Table 1). This is the reason why even when labels are used (see column iii) of Table 1) Δ_c and Δ_{dc} are more expensive to compute. In a nutshell, Δ_c is the most expensive to compute Delta, followed by Δ_{dc}, then by Δ_d and Δ_{ed} and finally by Δ_e.

$K_1 \rightarrow K_2$	(i) all stored	(ii) K's stored	(iii) K's & labels
Δ_e	$O(N)$	$O(N)$	$O(N)$
Δ_c	$O(C)$	$O(N^2)$	$O(N^2)$
Δ_d	$O(N)$	$O(N^2)$	$O(N\gamma)$
Δ_{dc}	$O(M)$	$O(N^2)$	$O(N_1^2)$
Δ_{ed}	$O(N)$	$O(N_2^2)$	$O(N\gamma)$

Table 1: Time complexity of RDF/S Deltas

Let's now discuss the factor γ. If t is an inferred triple by subsumption or instantiation relationships, we need to check whether $t \in C(K)$. When labeling schemes are used to encode subsumption relationships of RDF/S classes (or properties), subsumption checking (i.e. to check whether a class/property A is direct or indirect subclass/subproperty of a class/property B) can be performed in $O(1)$ just by comparing the involved class labels (no transitive closure computation is required). The same is true for instance checking (i.e. to check whether a resource r is direct or indirect instance of a class B). However, more that one labels per class are generated when the subsumption hierarchy is a DAG (and not a tree) [5]. Furthermore, resources may be classified under several classes not necessarily related thought subsumption relationships. In both cases more than one labels comparison is needed per subsumption or instance check. Since, in practice the number of additionally compared labels is small, the γ factor captures the incurred small overhead.

4. RDF/S KB UPDATE SEMANTICS

In this work, we focus on two *basic* change operations allowing to transform one KB to another, namely triple *addition* $Add(t)$ and *deletion* $Del(t)$ where $t \in T$. In this respect, a triple *update* is "split" into an addition and a deletion of triplets having the same *subject* and *predicate* (and thus keep both "old" and "new" values usually ignored by updates). The five differential functions presented in the Section 3, yield essentially *sets of atomic change operations*. More formally, for a $\Delta_x(K \rightarrow K')$ where $x \in \{e, c, d, dc, ed\}$, Δ_x^+ is used to denote the corresponding set of triple additions (i.e. *incremental* changes) and Δ_x^- the set of triple deletions (i.e. *decremental* changes). Obviously, Δ_x contains only sets of useful change operations reflecting the *net effect* of successive modifications over the same (explicit or inferred) triple of two KB versions. In other terms, Δ_x does not contain both $Add(t)$ and $Del(t)$ operations for a given $t \in T$. A Delta $\Delta(K \rightarrow K')$ is *useful* if it holds $\Delta^+ \cap \Delta^- = \emptyset$.

By defining RDF/S Deltas as sets of atomic change operations, we avoid to specify an execution order as in an edit-script (i.e. a sequence of triple additions or deletions). This design choice amends to a simpler computation of RDF/S Deltas (see Section 3.2) while provides the opportunity of applying alternative semantics of changes when transforming a source to a target KB (i.e. with or without side-effects on the KB closure). In the sequel, we will formally introduce the *semantics* of basic $Add(t)$ and $Del(t)$ operations (i.e. the exact pre and post-conditions of each operation) while in Section 5.6 we will investigate how a set of change operations can be transformed to an edit-script amendable to a sequential execution.

4.1 Semantics of Change Operations

Table 2 defines two alternative semantics for triple additions and deletions, namely, $\mathcal{U}_p$ (p comes from $p$lain), and $\mathcal{U}_{ir}$ (ir comes from $i$nference & $r$eduction).

- Under $\mathcal{U}_p$-semantics, change operations capture essentially plain set theoretic additions and deletions of triples. This implies that only the explicit triples are taken into account while inferred ones are ignored (as in a standard database context).
- Under $\mathcal{U}_{ir}$-semantics, change operations incur also interesting side-effects such as redundancy elimination and knowledge preservation. This implies that the updated KB will not contain any explicit triple which can be inferred, while preserves as much of the knowledge expressed in K as possible (reminiscent to the postulates of the AGM theory [1] regarding contraction, and compliant with the semantics of the RUL update language [18]).

$\mathcal{U}_p$-semantics is straitforward. We will illustrate in the sequel $\mathcal{U}_{ir}$-semantics using the example of Figure 1. If we apply on K the set Δ_{dc} under $\mathcal{U}_{ir}$-semantics, then we will indeed get K'. The insertion of ($Student\ subClassOf\ Person$) makes the triple ($TA\ subClassOf\ Person$) redundant, so the execution of Add($Student\ subClassOf\ Person$) will remove ($TA\ subClassOf\ \ Person$) from the KB. Analogously, the insertion of ($TA\ subClassOf\ Graduate$) makes the triple ($TA\ \ subClassOf\ \ Student$) redundant, while the deletion of the triple ($Jim\ \ type\ Student$) will add the triples ($Jim\ \ type\ \ Person$) and ($Jim\ \ type\ \ Student$). Finally, the deletion of the triple ($Jim\ \ type\ \ Student$) will not have any side effects.

Change Operation Semantics $\mathcal{U}_p$			
Operation		Pre-condition	Post-condition
$Add(t)$	1	$t \in K$	$K' = K$
	2	$t \in C(K) - K$	$K' = K \cup \{t\}$
	3	$t \notin C(K)$	$K' = K \cup \{t\}$
$Del(t)$	4	$t \in K$	$K' = K - \{t\}$
	5	$t \in C(K) - K$	$K' = K$
	6	$t \notin C(K)$	$K' = K$
Change Operation Semantics $\mathcal{U}_{ir}$			
$Add(t)$	7	$t \in K$	$K' = K$
	8	$t \in C(K) - K$	$K' = K$
	9	$t \notin C(K)$	$K' = R(K \cup \{t\})$
$Del(t)$	10	$t \in K$	$K' = R(C(K) - \{t\})$
	11	$t \in C(K) - K$	$K' = K$
	12	$t \notin C(K)$	$K' = K$

Table 2: $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ alternative change semantics

Returning to Table 2, for every incremental ($Add(t)$) or decremental ($Del(t)$) operation three different, and mutually exclusive, pre-conditions are examined, namely $t \in K$, $t \in C(K) - K$ and $t \notin C(K)$. The post-conditions of each case are specified. K (resp. K') denotes the knowledge base before (resp. after) the execution of a change operation. Notice that post-conditions define exactly what K' will be, unless the reduction is not unique.

In particular, let t be a triple whose addition is requested. If $t \in K$, then under both $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ semantics no change will be made i.e. $K' = K$ (recall that K is a *set* of triples). If $t \in C(K) - K$, then under $\mathcal{U}_p$-semantics, K' will indeed contain t however, under $\mathcal{U}_{ir}$-semantics we will have $K' = K$ because every triple that exists at $C(K) - K$ can be inferred (and $\mathcal{U}_{ir}$ aims at redundancy-free KBs). Finally, when requesting the addition of a triple $t \notin C(K)$ under $\mathcal{U}_p$, K' will contain t. Under $\mathcal{U}_{ir}$, K' will contain only the triples remaining after the elimination of redundant ones (i.e. those that can be inferred) once t is added to K.

Let's now consider the deletion of a triple t. If t belongs to K, then K' will not contain t under $\mathcal{U}_p$-semantics. Under $\mathcal{U}_{ir}$, K' will contain the triples that remain after deleting t from $C(K)$ and eliminating the redundant ones (note that

$C(K)$ is used in order to preserve as much knowledge as possible). Now if $t \in C(K) - K$, then this deletion request has no effect under both semantics. This means that under both semantics, *only explicit triples can be deleted.* This relieves us from having to decide which of the (possibly several) policies need to be adopted for reaching a K' whose closure does not contain t. Finally, if $t \notin C(K)$, then this deletion request has no effect as t is already out of K.

We can now define when a KB K satisfies a basic change operation under $\mathcal{U}_p$ and $\mathcal{U}_{ir}$-semantics.

DEF. 2. We will say that K *satisfies* under $\mathcal{U}_p$-semantics:
(a) an operation $Add(t)$, iff $t \in K$,
(b) an operation $Del(t)$, iff $t \notin K$

DEF. 3. We will say that K *satisfies* under $\mathcal{U}_{ir}$-semantics:
(a) an operation $Add(t)$, iff $t \in C(K)$,
(b) an operation $Del(t)$, iff $t \notin C(K)$

The above definitions stem directly form the pre and post conditions of operations depicted in Table 2. In general, we can say that a K satisfies an RDF/S delta Δ under $\mathcal{U}_y$, where $y \in \{p, ir\}$, iff K satisfies *every* addition of Δ^+ and deletion of Δ^-. Recall also at this point that the RDF/S deltas considered in our work consists only of useful change operations.

If $\mathcal{U}$ is a symbol that denotes the semantics of a particular change operation (i.e. $\mathcal{U}_p$,$\mathcal{U}_{ir}$), then we will use $\Delta^{\mathcal{U}}(K)$ to denote the result of applying a under $\mathcal{U}$ semantics. Notice that the result of applying a change operation is unique under $\mathcal{U}_p$-semantics. This is true also for $\mathcal{U}_{ir}$ if we are in Ψ (KBs with unique reduction).

DEF. 4. A delta $\Delta(K \rightarrow K')$ can be applied on K as follows:
(a)Under $\mathcal{U}_p$, $\Delta(K \rightarrow K')^{\mathcal{U}_p}(K) = (K - \Delta^-) \cup \Delta^+$
(b)Under $\mathcal{U}_{ir}$, $\Delta(K \rightarrow K')^{\mathcal{U}_{ir}}(K) = R((C(K) - \Delta^-) \cup \Delta^+)$

It is clear that the resulting KB always satisfies Δ_x under $\mathcal{U}_p$ and $\mathcal{U}_{ir}$-semantics for every $x \in \{e, c, d, dc, ed\}$. However, as we will see in the next Section, the target KB is not always a correct transformation of the source KB (see for example Table 3(C) for Δ_d).

5. FORMAL PROPERTIES OF RDF/S DELTAS

In this section, we investigate which of the five RDF/S differential functions (introduced in Section 3) actually produce *correct* Deltas and under what semantics of update primitives (presented in Section 4). In addition, we are interested in other useful properties such as *semantic identity*, *non-redundancy*, *reversibility* and *composability* of RDF/S Deltas. Finally, we elaborate on the execution semantics of RDF/S Deltas as SW update programs (i.e. sequences of change operations).

5.1 Properties of RDF/S Deltas

Let Δ_x be a differential function, and $\mathcal{U}_y$ be a change operation semantics. Obviously, a pair $(\Delta_x, \mathcal{U}_y)$ can be used for versioning or synchronizing RDF/S KBs only if it is correct.

DEF. 5. A pair $(\Delta_x, \mathcal{U}_y)$ is *correct* if for any pair of knowledge bases K and K', it holds $\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K) \sim K'$.

Apart from correctness, RDF/S Deltas may also satisfy the following properties.

Semantic Identity (P1): It is desirable to have a Δ_x that reports an empty result if its operand KBs are semantically equivalent: If $K \sim K'$ then $\Delta_x(K \rightarrow K') = \emptyset$.

Non-Redundancy (P2): It is also desirable that the execution of a Δ_x will result to a KB that is always redundancy free (independently of whether K and K' are redundancy free or not): $RF(\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K))$.

Weaker Non-Redundancy (P2.1): If $RF(K')$ then $RF(\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K))$. Notice that P2.1 is weaker than P2: if P2 holds then P2.1 holds too.

Reversibility (P3): The inverse of an atomic update operation is defined as: $Inv(Add(t)) = Del(t)$ and $Inv(Del(t)) = Add(t)$. A set of change operations U can be reversed as follows: $Inv(U) = \cup\{\ Inv(u) \mid u \in U\}$. A Δ_x is reversible if: $Inv(\Delta_x(K \rightarrow K')) = \Delta_x(K' \rightarrow K)$.

Composition (P4): Let two Deltas $\Delta_1 = \Delta_1^+ \cup \Delta_1^-$ and $\Delta_2 = \Delta_2^+ \cup \Delta_2^-$. Their composition, denoted as $\Delta_1 \circ \Delta_2$ is a Delta $\Delta = \Delta^+ \cup \Delta^-$ where:

- $\Delta^+ = (\Delta_1^+ \cup \Delta_2^+) - (\Delta_1^- \cup \Delta_2^-)$
- $\Delta^- = (\Delta_1^- \cup \Delta_2^-) - (\Delta_1^+ \cup \Delta_2^+)$

A Δ_x can be composed if:

$$\Delta_x(K_1 \rightarrow K_2) \circ ... \circ \Delta_x(K_{n-1} \rightarrow K_n) = \Delta_x(K_1 \rightarrow K_n)$$

5.2 Correctness of $(\Delta_x, \mathcal{U}_y)$-pairs

For identifying the pairs that are correct, Table 3 depicts for each of our four examples the result of applying Δ_d, Δ_c, Δ_e, Δ_{dc} and Δ_{ed} in both directions: $K \rightarrow K'$ and $K' \rightarrow K$. Then, for each of two update semantics:

- $\mathcal{U}_p$: is marked with Y when $\Delta_x(K \rightarrow K')^{\mathcal{U}_p}(K) \sim K'$ i.e. the transformation is *correct*. Otherwise, the column is marked with N. If the respective pair is correct and the result is RF it is also mentioned in the table.
- $\mathcal{U}_{ir}$: is marked with Y when $\Delta_x(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$ i.e the transformation is correct. Otherwise the column is marked with N. Since by definition in this case the resulting KB is always in a redundancy free state, we do not mark it as RF.

Those pairs that are marked with "N" w.r.t. correctness, essentially constitute a proof by counterexample of their incorrectness. For the rest of the pairs (i.e. those with a Y) we have to prove that they are *always* correct.

THEOREM 1. For any pair of valid knowledge bases K and K' it holds:

$$\Delta_e(K \rightarrow K')^{\mathcal{U}_p}(K) \sim \Delta_{ed}(K \rightarrow K')^{\mathcal{U}_p}(K) \sim \Delta_c(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim \Delta_{dc}(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$$

THEOREM 2. $\Delta_d(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$ iff (a) K is complete, or (b) $C(K) - K \subseteq C(K')$.

THEOREM 3. $\Delta_c(K \rightarrow K')^{\mathcal{U}_p}(K) \sim K'$ iff K is complete.

An interesting remark regarding Theorem 2 is that if $C(K) \subseteq C(K')$, then condition (b) holds. This implies that we could use the pair $(\Delta_d, \mathcal{U}_{ir})$ in cases we know that $C(K) \subseteq C(K')$. For example if K is an ontology O and K' is an ontology O' that specializes O, then we are sure that $C(K) \subseteq C(K')$. In such cases we can use Δ_d (or alternatively Δ_{dc}) which give the smallest in size Deltas (Δ_{dc} returns the same Delta).

5.3 Semantic Identify and RF of $(\Delta_x, \mathcal{U}_y)$-pairs

PROP. 8. If $K \sim K'$ then $\Delta_d(K \to K') = \Delta_c(K \to K') = \Delta_{dc}(K \to K') = \emptyset$.

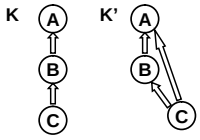


Figure 5: Two equivalent KBs

Note that Δ_e is not included in Prop. 8 because even if $K \sim K'$, it may be $K = K'$, $K \subset K'$, $K' \subset K$, or $K \not\subseteq K'$ and $K' \not\subseteq K$. In the example of Figure 5 we get $\Delta_e(K \to K') = \{Add(C\ subClassOf\ A)\}$ although $K \sim K'$. However, one can easily prove that: If K and K' are both redundancy free, and the knowledge bases considered have always a unique reduction, then $K \sim K' \Rightarrow \Delta_e(K \to K') = \emptyset$.

PROP. 9. If $K \sim K'$, $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_e(K \to K') = \emptyset$

Finally, it can be easily proved that if (a) $K' \subseteq K$ or (b) K and K' are both redundancy free, and the KBs considered have always a unique reduction, then $K \sim K' \Rightarrow \Delta_{ed}(K \to K') = \emptyset$.

PROP. 10. If $K \sim K'$, and (a) $K' \subseteq K$ or (b) $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_{ed}(K \to K') = \emptyset$

5.4 Reversibility of Δ_x

It is not hard to see that this property is satisfied only by those Deltas defining symmetric set differences on operant KBs (and their closures). This is the case of Δ_e, Δ_c and Δ_d. It should be stressed that the reverse of a Delta $Inv(\Delta_x(K \to K'))$ is correct only when the $\Delta_x(K' \to K)$ is correct (see section 5.2)

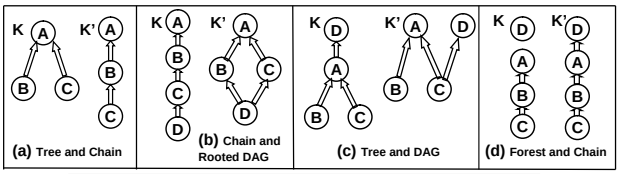
PROP. 11. For every pair of valid knowledge bases K, K' it holds:

$$Inv(\Delta_e(K \to K')) = \Delta_e(K' \to K)$$
$$Inv(\Delta_c(K \to K')) = \Delta_c(K' \to K)$$
$$Inv(\Delta_d(K \to K')) = \Delta_d(K' \to K)$$

Δ_{dc} and Δ_{ed} do not satisfy (P3). For example at the case (a) of Table 3 $\Delta_{ed}(K \to K') = \{Add(C\ subClassOf\ B)\}$, so $Inv(\Delta_{ed}(K \to K')) = \{Del(C\ \ subClassOf\ \ B)\}$, while $\Delta_{ed}(K' \to K) = \{Add(C\ subClassOf\ A), Del(C\ subClassOf\ B)\}$. This means that:

$$Inv(\Delta_{ed}(K \to K')) \neq \Delta_{ed}(K' \to K)$$

At the case (d) of Table 3 $\Delta_{dc}(K \to K') = \{Add(A\ subClassOf\ D)\}$, so $Inv(\Delta_{dc}(K \to K')) = \{Del(A\ subClassOf\ D)\}$, while



(a) Tree and Chain			
Delta	$K \to K'$	$\mathcal{U}_p$	$\mathcal{U}_{ir}$
Δ_e	$\{Add(C\ subClassOf\ B), Del(C\ subClassOf\ A)\}$	Y,*RF*	Y
Δ_c	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_d	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{dc}	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{ed}	$\{Add(C\ subClassOf\ B)\}$	Y	Y
	$K' \to K$		
Δ_e	$\{Add(C\ subClassOf\ A), Del(C\ subClassOf\ B)\}$	Y,*RF*	Y
Δ_c	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_d	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{dc}	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{ed}	$\{Add(C\ subClassOf\ A), Del(C\ subClassOf\ B)\}$	Y, *RF*	Y
(b) Chain and Rooted DAG			
Δ_e	$\{Add(C\ subClassOf\ A), Add(D\ subClassOf\ B), Del(C\ subClassOf\ B)\}$	Y, *RF*	Y
Δ_c	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_d	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{dc}	$\{Del(C\ subClassOf\ B)\}$	**N**	Y
Δ_{ed}	$\{Add(C\ subClassOf\ A), Add(D\ subClassOf\ B), Del(C\ subClassOf\ B)\}$	Y, *RF*	Y
	$K' \to K$		
Δ_e	$\{Add(C\ subClassOf\ B), Del(C\ subClassOf\ A), Del(D\ subClassOf\ B)\}$	Y, *RF*	Y
Δ_c	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_d	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{dc}	$\{Add(C\ subClassOf\ B)\}$	Y	Y
Δ_{ed}	$\{Add(C\ subClassOf\ B)\}$	Y	Y
(c) Tree and DAG			
Δ_e	$\{Add(C\ subClassOf\ D), Del(A\ subClassOf\ D)\}$	Y, *RF*	**N**
Δ_c	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D)\}$	**N**	Y
Δ_d	$\{Del(A\ subClassOf\ D)\}$	**N**	**N**
Δ_{dc}	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D)\}$	**N**	Y
Δ_{ed} s	$\{Add(C\ subClassOf\ D), Del(A\ subClassOf\ D)\}$	Y, *RF*	**N**
	$K' \to K$		
Δ_e	$\{Add(A\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_c	$\{Add(A\ subClassOf\ D), Add(B\ subClassOf\ D)\}$	Y	Y
Δ_d	$\{Add(A\ subClassOf\ D)\}$	Y	Y
Δ_{dc}	$\{Add(A\ subClassOf\ D)\}$	Y	Y
Δ_{ed}	$\{Add(A\ subClassOf\ D)\}$	Y	Y
(d) Forest and Chain			
Δ_e	$\{Add(A\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_c	$\{Add(A\ subClassOf\ D), Add(B\ subClassOf\ D), Add(C\ subClassOf\ D)\}$	Y	Y
Δ_d	$\{Add(A\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_{dc}	$\{Add(A\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_{ed}	$\{Add(A\ subClassOf\ D)\}$	Y, *RF*	Y
	$K' \to K$		
Δ_e	$\{Del(A\ subClassOf\ D)\}$	Y, *RF*	**N**
Δ_c	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_d	$\{Del(A\ subClassOf\ D)\}$	Y, *RF*	**N**
Δ_{dc}	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y, *RF*	Y
Δ_{ed}	$\{Del(A\ subClassOf\ D)\}$	Y, *RF*	**N**

Table 3: Examples

$\Delta_{dc}(K' \rightarrow K) = \{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$. Thus:

$$Inv(\Delta_{dc}(K \rightarrow K')) \neq \Delta_{dc}(K' \rightarrow K)$$

Generally Δ_{dc} and Δ_{ed} do not satisfy reversibility, unless K' is complete (i.e. $K' = C(K')$). This is due to the fact that when $K' = C(K')$, Δ_{dc} yields the same result as Δ_c and Δ_{ed} as Δ_e (see Figure 4.e).

5.4.1 Reversibility and Delta size

In order to speed-up access to forward and backward RDF/S KB versions we have to compute and store non-reversible Deltas (such as Δ_{ed} and Δ_{dc}) to both directions. However, since the only reversible Deltas are Δ_e and Δ_c whose size is bigger than Δ_{ed} and Δ_{dc}, it will be intresting to examine whether it is more beneficial to store Δ_e and Δ_c assuming only one direction or Δ_{ed} and Δ_{dc} in both directions. The following property proves that the former option is better than the latter.

PROP. 12. For any pair of knowledge bases K and K' it holds:

$$\begin{aligned} |\Delta_e(K \rightarrow K')| &\leq |\Delta_{ed}(K \rightarrow K')| + |\Delta_{ed}(K' \rightarrow K)| \\ |\Delta_c(K \rightarrow K')| &\leq |\Delta_{dc}(K \rightarrow K')| + |\Delta_{dc}(K' \rightarrow K)| \end{aligned}$$

5.5 Composition of Δ_x

The RDf/S Deltas that can be always composed are Δ_e and Δ_c. Note that the composition of a differential function $\Delta_x(K_1 \rightarrow K_2) \circ ... \circ \Delta_x(K_{n-1} \rightarrow K_n) = \Delta_x(K_1 \rightarrow K_n)$ is correct only when the $\Delta_x(K_1 \rightarrow K_n)$ is correct (see section 5.2)

PROP. 13. For any valid knowledge bases $K_1, K_2, K_3, ..., K_n$ it holds:

$$\begin{aligned} \Delta_e(K_1 \rightarrow K_2) \circ ... \circ \Delta_e(K_{n-1} \rightarrow K_n) &= \Delta_e(K_1 \rightarrow K_n) \\ \Delta_c(K_1 \rightarrow K_2) \circ ... \circ \Delta_c(K_{n-1} \rightarrow K_n) &= \Delta_c(K_1 \rightarrow K_n) \end{aligned}$$

It is not hard to see that this property is not satisfied by Δ_d, Δ_{dc} and Δ_{ed}. For example, for the KBs of the Figure 6 (a) for Δ_d we have:

$$\begin{aligned} \Delta_d(K_1 \rightarrow K_2) &= \{Add(B\ subClassOf\ A)\} \\ \Delta_d(K_2 \rightarrow K_3) &= \{Del(B\ subClassOf\ A)\} \\ \Delta_d(K_1 \rightarrow K_3) &= \{Add(C\ subClassOf\ A)\} \end{aligned}$$

If we compose the first two Deltas we get the empty result that is different from $\Delta_d(K_1 \rightarrow K_3)$:

$$\Delta_d(K_1 \rightarrow K_2) \circ \Delta_d(K_2 \rightarrow K_3) = \emptyset$$

The same result holds also for Δ_{dc}.

At the figure 6 (b) for Δ_{ed} we have:

$$\begin{aligned} \Delta_{ed}(K_1 \rightarrow K_2) &= \{Add(C\ subClassOf\ B)\} \\ \Delta_{ed}(K_2 \rightarrow K_3) &= \{Del(C\ subClassOf\ B)\} \\ \Delta_{ed}(K_1 \rightarrow K_3) &= \{Del(C\ subClassOf\ A)\} \end{aligned}$$

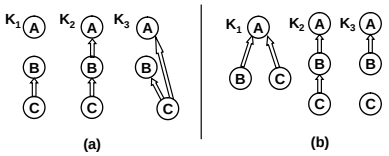


Figure 6: Composition of Δ_x

We can easily observe that the composition of the two Deltas yields an empty set that is different from $\Delta_{ed}(K_1 \rightarrow K_3)$:

$$\Delta_{ed}(K_1 \rightarrow K_2) \circ \Delta_{ed}(K_2 \rightarrow K_3) = \emptyset$$

Generally speaking, Δ_{dc} and Δ_{ed} can not be composed unless K' is complete. This is due to the fact that when $K' = C(K')$, Δ_{dc} yields the same result as Δ_c and Δ_{ed} as Δ_e (see Figure 4.e).

5.6 Streaming Execution of RDF/S Deltas

Regarding the execution of Deltas, so far we have considered a batch execution mode as defined in Def. 4. An alterative method would be to execute each operation in Δ^+ and Δ^- individually (according to the pre/post-conditions defined in Table 2). The rising question is whether these two alternative execution modes yield equivalent KBs. It is not hard to see that this is true only for $\mathcal{U}_p$ semantics.

As expected, the order of execution of change operations affect the resulting KB under $\mathcal{U}_{ir}$ semantics. In particular the resulting KB may not satisfy all change operations returned by a differential function (see Def. 3). For instance, for the KBs of Table 3 (d) we get $\Delta_{dc}(K' \rightarrow K) = \{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$ If the operations are executed in the order $\langle Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\rangle$ under $\mathcal{U}_{ir}$ semantics, then all of them will be satisfied and the result will be equivalent to K. Now consider the following execution order $\langle Del(B\ subClassOf\ D), Del(C\ subClassOf\ D), Del(A\ subClassOf\ D)\rangle$. In this case the operation $Del(B\ subClassOf\ D)$ does not change the K as it requests the deletion of an inferred triple and according to $\mathcal{U}_{ir}$ semantics an inferred triple cannot be deleted. The same will happen with the operation $Del(C\ subClassOf\ D)$. Finally, the operation $Del(A\ subClassOf\ D)$ will be executed and will cause the addition of the triple $(B\ subClassOf\ D)$. It is obvious that the operation $Del(B\ subClassOf\ D)$ is not satisfied by the resulting KB because it contains the triple $(B\ subClassOf\ D)$. The same problem occurs when K' contains a redundant triple e.g. $(B\ subClassOf\ D)$. A similar situation is encountered with Δ_c and with Δ_d when K and K' are not redundancy free.

Alg 1. **Execute**(K, M) where $M \subseteq \mathcal{S}$
(1) $UnSat = M$
(2) repeat
(3) $Sat = \emptyset$ //the set of all satisfied operations
(4) for i=1 to $|UnSat|$
(5) $u = UnSat[i]$ // gets the next operation
(6) $K_t = u^{\mathcal{U}_{ir}}(K)$ // applies the operation using $\mathcal{U}_{ir}$
(7) if K_t satisfies u then
(8) $Sat = Sat \cup \{u\}$

(9) $K = K_t$
(10) endif
(11) endfor
(12) $UnSat = UnSat - Sat$
(13) until $UnSat = \emptyset$

For these reasons, we devise a loop-based algorithm (`Alg1`) allowing to determine a sequence of change operations that when executed under $\mathcal{U}_{ir}$ semantics, the resulting KB guarantees satisfaction of all operations in the RDF/S Delta. Obviously, this algorithm will not terminate unless all delete operations in M are satisfied. This is true when M is derived from one of Δ_c, Δ_{dc} and Δ_d.

PROP. 14. For every $M \subseteq \mathcal{S}$ produced by Δ_d, Δ_{dc} or Δ_c Alg. 1 terminates.

Below we describe in brief the crux of the proof. Let Y be the satisfiable deletions and Z the unsatisfiable deletions at any point during the execution of the algorithm (i.e. the sets Sat and $UnSat$ respectively). We can prove that whenever $|Y| = 0$ we also have $|Z| = 0$. This guarantees that the algorithm always terminates since all elements of M are satisfied.

THEOREM 4. If $\{K, K'\} \subseteq \Psi$ then for every delta produced by Δ_d, Δ_{dc} and Δ_c there exists at least one sequence of change operations that when executed under $\mathcal{U}_{ir}$ the result is equivalent to the execution of the corresponding *set* of change operations under $\mathcal{U}_{ir}$. Alg. 1 produces such a sequence.

5.7 Summarizing the Results

The pairs that are always correct are: $(\Delta_e, \mathcal{U}_p)$, $(\Delta_{ed}, \mathcal{U}_p)$, $(\Delta_c, \mathcal{U}_{ir})$, $(\Delta_{dc}, \mathcal{U}_{ir})$. The pair $(\Delta_c, \mathcal{U}_p)$ is correct if K is complete, while the pair $(\Delta_d, \mathcal{U}_{ir})$ is correct only in the cases specified in Theorem 2. The Deltas that always satisfy *semantic identity* are: Δ_c, Δ_{dc} and Δ_d, while Δ_e and Δ_{ed} satisfy semantic identity only in the cases specified in Theorems 9 and 10 respectively. Pairs that rely on $\mathcal{U}_{ir}$ semantics always result to *non-redundant* RDF/S KBs, while the pairs $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_{ed}, \mathcal{U}_p)$ satisfy only *weaker non-redundancy*. The pair $(\Delta_c, \mathcal{U}_p)$ does not satisfy neither of the two properties. The Deltas that always *reversible* are Δ_e, Δ_c and Δ_d, while Δ_{dc} and Δ_{ed} are reversible only if K' is complete. Finally, Δ_e and Δ_c can be always *composed*, while Δ_{dc} and Δ_{ed} only in the case where K' is complete. Table 4 synopsizes the above results. Note that the properties (P1)-(P4) are examined only for the pairs $(\Delta_x, \mathcal{U}_y)$ that are always correct.

$(\Delta_x, \mathcal{U}_y)$	Correctness	(P1)	(P2)	(P2.1)	(P3)	(P4)
$(\Delta_e, \mathcal{U}_p)$	**Y**	Prop. 9	N	Y	Y	Y
$(\Delta_c, \mathcal{U}_p)$	Y [1]	Y	N	N	Y	Y
$(\Delta_d, \mathcal{U}_p)$	N	-	-	-	-	-
$(\Delta_{dc}, \mathcal{U}_p)$	N	-	-	-	-	-
$(\Delta_{ed}, \mathcal{U}_p)$	**Y**	Prop. 10	N	Y	Y [2]	Y [2]
$(\Delta_e, \mathcal{U}_{ir})$	N	-	-	-	-	-
$(\Delta_c, \mathcal{U}_{ir})$	**Y**	Y	Y	Y	Y	Y
$(\Delta_d, \mathcal{U}_{ir})$	Th.2	Y	Y	Y	Y	N
$(\Delta_{dc}, \mathcal{U}_{ir})$	**Y**	Y	Y	Y	Y [2]	Y [2]
$(\Delta_{ed}, \mathcal{U}_{ir})$	N	-	-	-	-	-

(1) if $K = C(K)$ (2) if $K = C(K)$

Table 4: Synopsis

6. EXPERIMENTAL EVALUATION

In this Section we experimentally measure the time required to compare real and synthetic RDF/S KBs of variable size and structure. In addition, we are interested in comparing the size (i.e. triples to be added or deleted) of the produced Deltas with respect to the different application setting (i.e. forward/backward, reverse deltas). It is worth mentioning that our experimental findings confirm the respective analytical evaluation (see Section 3 and 3.2) of the five differential functions while reveal useful information for the cases that such an analysis is not available.

All experiments were carried out in a PC with processor Intel Core2 Quad 2.4 Ghz, 2 GB Ram, running Windows Vista. We assume that both KBs have been loaded in main memory using a labeling schema for encoding transitive closures and thus net differential times are reported. The blank nodes are assumed to have different URLs and thus they are not compared for isomorphism.

6.1 Testbed

Our testbed comprises the following datasets:

Real Data Set: We used the RDF/S dumps from the Gene Ontology (GO) project[6] as a representative of large-scale evolving Semantic Web data. GO terms (i.e. genes) are exported in a simple RDF/S schema containing only one metaclass, instance of which are all the 53,574 classes representing the GO terms in the 2007-11-01 version of the dump. Subsumption relationships between classes are represented by user-defined properties (i.e. not by "subClassof" properties). GO employs in total 11 properties to describe genes. The GO RDF/S KB does not contain any redundant triples while it employs heavily blank nodes (e.g. to represent structured values as tuples). Specifically, 28,171 of the 53,574 instance resources are blank nodes. Additionally, there are 400,999 property instances, most of which connect a blank node to a literal value. It is worth noticing that GO curators usually reclassify terms as obsolete and thus they are not always deleted; this is done so that existing biological annotations do not have dangling reference during the time lag between the term being made obsolete and the reannotation of the entity.

Synthetic Data Set: As the GO RDF/S KB has rather a simple structure (i.e. without a significantly large number of inferred triples) we have also created and used a two synthetic data sets. Specifically, using the synthetic KB generator described in [26], we created a sequence of four KBs, $K_1, \ldots, K_4$, with 100, 200, 300, and 400 classes respectively. The number of properties $|E|$ in each KB is $3 * N$ where N is the number of classes. At the **simple** case, for each class 10 instances were created, while at the **complex** one 100 instances were created. In both data sets, for each property 10 property instances were created among randomly selected instances of the corresponding domain and range class. To simulate their evolution, we adopted a naming convention such that all classes, properties, and their instances in K_i are also present in K_{i+1} for each i=1..3. However, their subsumption structure may differ: classes which are higher in the subsumption hierarchy in version K_i may be found at a lower level in version K_{i+1}. This result to both additions

[6] www.geneontology.org/

and deletions of explicit triples (i.e. "subClassof" properties). The depth of the class subsumption hierarchy in each schema is 7, while the synthetic data set does not contain any redundant triple.

In order to capture the effect of poor vs rich subsumption structures in RDF/S KBs, we introduce the following metric regarding their inference potential.

DEF. 6. *The* inference strength *of a knowledge base K, denoted by $is(K)$, is defined as:*

$$is(K) = \frac{|C(K)| - |K|}{|K|}$$

Clearly, if $K = C(K)$ then $is(K) = 0$. As one would expect the greater this factor is, the greater is the number of new inferred triples in $C(K)$ w.r.t. K.

6.2 Experimental Results

For each compared RDF/S KB version we report its size and inference strength as well as the time (in seconds) required to compute the Deltas as well as their sizes (in triples). Specifically, for each Delta we report in parentheses the added and deleted triples ($(|\Delta^+|, |\Delta^-|)$) and then their total size ($|\Delta|$).

Table 5 and 6 report respectively, the Delta computing time and size for non successive versions of the Biological data set. Note that Deltas are computed for both forward and backward versions of RDF/S KBs. As we can observe, the produced Deltas exhibit almost similar sizes: Δ_e, Δ_{ed}, Δ_d and Δ_{dc} differ on at most 2 change operations. Only the size of Δ_c is bigger by at most 63 triples. This is due to the fact that the *is* factor is very small in this dataset. Specifically the range of the *is* factor for this data set is $[0.178, 0.229]$.

Tables 8 and 7 report respectively, the Delta computing time and size for four consequent versions of the simple Synthetic data set. We can easily observe the significant divergences on Delta sizes, which stem from the fact that the *is* factor is bigger at this data set with range [0.789, 0.826]. Specifically, on average the deltas produced by Δ_c are 63% bigger those of Δ_e. The deltas of Δ_{dc} are 16% bigger that those of Δ_e. Note however, that Δ_c Δ_d and Δ_{dc} satisfy properties like semantic identity and also non-redundancy when used with $\mathcal{U}_{ir}$-semantics. The delta of Δ_d is 1% smaller than the size of Δ_e and the delta of Δ_{ed} is 0.5% smaller than the size of Δ_e. The minor differences in $|\Delta_e|$, $|\Delta_{ed}|$ and $|\Delta_d|$ sizes are due to the fact that most changes occur at the explicit graph i.e. only few triples that are inferred by K became explicit in K' and vice versa. On the other hand, the significant divergences between $|\Delta_e|$ and $|\Delta_c|$ sizes are due to changes occurring higher at the subsumption hierarchy. For the same reason, the deletions reported by Δ_{dc} are more than those reported by Δ_e, Δ_{ed} and Δ_d. In general, for Δ_e, Δ_{ed} and Δ_d all kinds of changes affect in the same way the delta size, while for Δ_c additions or deletions that occur highly at the subsumption hierarchy affect more the delta size. Finally, for Δ_{dc} all additions have the same impact, but deletions that occur highly at the subsumption hierarchy affect more the size of the produced delta.

Our experimental findings are summarized in Figure 7 which confirms the ordering of Delta sizes illustrated in Figure 4. In addition, we observe that in most cases it holds $|\Delta_e| < |\Delta_c|$ and $|\Delta_{ed}| < |\Delta_{dc}|$. Recall also that Δ_e produces a big in size result if $K = C(K')$, while Δ_e and Δ_{ed} produce a big in size result if $K' = C(K')$.

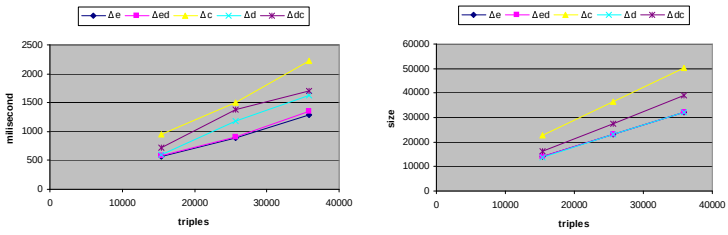


Figure 7: Delta time/size: simple Synthetic Data

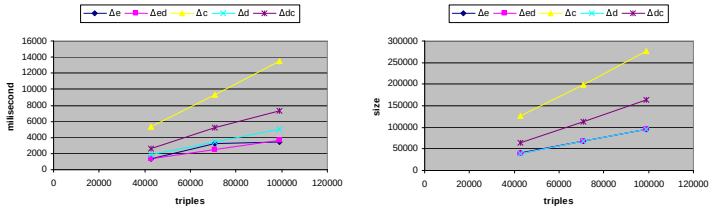


Figure 8: Delta time/size: complex Synthetic Data

Tables 9 and 10 report respectively, the Delta computing time and size for four successive versions of the complex Synthetic data set with a bigger inference strength than the previous with range [2.868, 2.987]. In this case, significant differences in Delta sizes are observed. Specifically, on average Δ_c is 212% bigger than Δ_e. Δ_{dc} is 57% bigger than Δ_e. Δ_d is 1.3% smaller than Δ_e. Δ_{ed} is 0.8% smaller than Δ_e. Significant divergences also observed in Delta computing time.

The experimental results of Figures 7 and 8 confirms the time complexity of Deltas presented in Table 1. We can observe that the execution time for Δ_c is always greater that the other ones. Furthermore, the execution times of Δ_e, Δ_d and Δ_{ed} are very close and clearly smaller than Δ_{dc}.

Finally, Figure 9 (a) depicts how the inference strength affects the Delta sizes, while Figure 9 (b) illustrates how the inference strength affects the size of Deltas required in order to be able to move both forward and backward to KB versions. Since Δ_e and Δ_c are reversible we report only the size of delta at one direction (i.e. $|\Delta(K \rightarrow K')|$), while for Δ_{dc} and Δ_{ed} we report the bisectional Delta size: $|\Delta(K \rightarrow K')|$ + $|\Delta(K' \rightarrow K)|$. In both Figures we are employing data from the all the three previously presented data sets. As we can see in Figure 9 (a), the less the inference strength is, the less the delta computing time and size differ. Last but not least, in the case of handling both forward and backward versions, Δ_e appear to be the cheaper in size solution, followed by Δ_{ed}, then by Δ_c and finally by Δ_{dc}. This relationship is not captured by our analytical study (see Prop. 12).

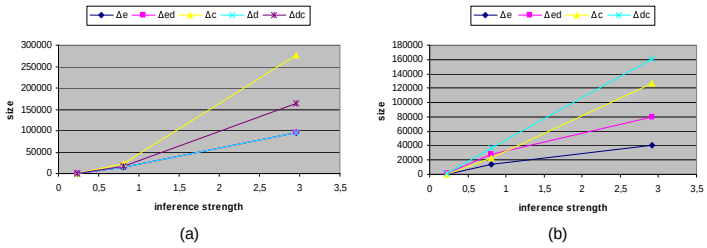


Figure 9: (a) Size of Forward Deltas (b) Size of Forward/Backward Deltas

7. CONCLUDING REMARKS

Most of the existing RDF/S differential tools [2, 27, 8] rely on the $(\Delta_e, \mathcal{U}_p)$ pair. Semversion [27] offers also $(\Delta_c, \mathcal{U}_p)$ which (as we have proved) yields correct results if K is complete. None of the works (theoretical or practical) has used Δ_d, Δ_{dc} or Δ_{ed} nor studied formally properties as their size, their correctness and non-redundancy w.r.t. the semantics of the change operations, as well as semantic identity, reversibility and composability of the produced Deltas. All Deltas studied in this paper are implemented in the context of the Semantic Web Knowledge Middleware[7].

The pairs that are always correct are: $(\Delta_e, \mathcal{U}_p)$, $(\Delta_{ed}, \mathcal{U}_p)$, $(\Delta_{dc}, \mathcal{U}_{ir})$ and $(\Delta_c, \mathcal{U}_{ir})$. We have also identified special cases where $(\Delta_d, \mathcal{U}_{ir})$ is correct given that it is the smallest in size Delta. Δ_e and Δ_{ed} require less computing time than Δ_c and Δ_{dc}, respectively. However, if $K \sim K'$ then Δ_e and Δ_{ed} may return a very big in size result while Δ_{dc} and Δ_c yield always an empty result. Note also that we have experimentally verified that Δ_e produces smaller in size results that Δ_c and that in most practical cases it holds $|\Delta_{ed}| < |\Delta_{dc}|$.

However, the cost of executing correctly the Deltas depends on the semantics of the change operations. Clearly, $\mathcal{U}_{ir}$ is most costly because it requires computing the closure and the reduction of a KB. This is the price to pay for keeping the resulting KBs in a redundancy-free state[8]. Figure 10 shows the ordering of the correct pairs according to the combined cost of computing and executing RDF/S Deltas.

In addition, we have proved that only $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_c, \mathcal{U}_{ir})$ can be reversed and composed to implement various forward/backward version policies over distributed RDf/S KB archives. In addition, as we have experimentally verified storing only Δ_e (or Δ_d) is more beneficial than storing Δ_{ed} (or Δ_{dc}) in both directions. As we have seen these requirements cannot be satisfied by smaller in size RDF/S Deltas such as Δ_{ed} (or Δ_{dc}) which incur a loss of information.

Finally, it is worth noticing that unlike our work, belief contraction-revision (e.g. [11, 17, 9]) theories consider KBs as logic theories and focus on what the result of applying a contraction/revision operation on a KB should be. In our setting, the target K' is known, so the focus is given on the discovery of changes between two consistent versions of KBs.

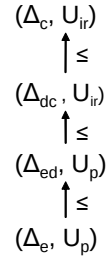


Figure 10: Cost-based ordering of the correct pairs

Acknowledgements

This work was partially supported by the EU projects CASPAR (FP6-2005-IST-033572) and KP-Lab (FP6-2004-IST-4). Many thanks to Yannis Theoharis for providing us the synthetic datasets.

8. REFERENCES

[1] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. "On the Logic of Theory Change: Partial Meet Contraction and Revision Functions". *J. Symb. Log.*, 50(2):510–530, 1985.

[2] T. Beners-Lee and D. Connoly. "Delta: An Ontology for the Distribution of Differences Between RDF Graphs", 2004. http://www.w3.org/DesignIssues/Diff (version: 2006-05-12).

[3] B. Berliner. "CVS II: Parallelizing Software Development". In *Procs of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990.

[4] D. Brickley and R. V. Guha. "RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation", February 2004. http://www.w3.org/TR/rdf-schema/.

[5] V. Christophides, D. Plexousakis, M. Scholl, and S. Tourtounis. "On Labeling Schemes for the Semantic Web". In *Procs. of WWW'03*, pages 544–555, Budapest, Hungary, May 2003.

[6] R. Cloran and B. Irwin. "Transmitting RDF graph deltas for a Cheaper Semantic Web". In *Procs. of SATNAC'2005*, South Africa, September 2005.

[7] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[8] L. Ding, T. Finin, A. Joshi, Y. Peng, P. da Silva, and D. McGuinness. "Tracking RDF Graph Provenance using RDF Molecules". In *Procs of ISWC'05*, Galway,Ireland, November 2005.

[9] G. Flouris. *"On Belief Change and Ontology Evolution"*. PhD thesis, Computer Science Department, University of Crete, Greece, 2006.

[10] J. Petersson F. Piazza P. Puliti G. Tummarello, C. Morbidoni. "RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications". In *1st Annual International Conference on Mobile and Ubiquitous Systems MobiQuitous*, Boston, MA, August 2004.

[11] P. Gärdenfors. "Belief Revision: An Introduction". In *Belief Revision*, pages 1–20. Cambridge University Press, 1992.

[12] C. Gutierrez, C. Hurtado, and A. Mendelzon. "Foundations of Semantic Web Databases". In *In 23 ACM Symposium on Principles of Database Systems (PODS)*, 2004.

[13] P. Hayes. "RDF Semantics, W3C Recommendation", February 2004. http://www.w3.org/TR/rdf-mt/.

[14] J. Heflin, J. Hendler, and S. Luke. "Coping with Changing Ontologies in a Distributed Environment". In *Procs of*

[7] 139.91.183.30:9090/SWKM

[8] Concerning the execution under $\mathcal{U}_{ir}$, related algorithms include [24], while a similar in spirit approach for RDF/S has already been implemented for the RUL language [18].

AAAI-99 Workshop on Ontology Management, Florida, July 1999.

[15] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. "Ontology versioning and change detection on the web". In *Procs of EKAW'02*, pages 197–212, Siguenza, Spain, Oct 2002.

[16] M. Klein and N. Noy. "A component-based framework for ontology evolution". In *InWorkshop on Ontologies and Distributed Systems at IJCAI-03*, Acapulco, Mexico, 2003.

[17] S. Konieczny and R. P. Perez. "Propositional Belief Base Merging or How to Merge Beliefs/Goals Coming from Several Sources and Some Links With Social Choice Theory". *European Journal of Operational Research*, 160(3):785–802, 2005.

[18] M. Magiridou, S. Sahtouris, V. Christophides, and M. Koubarakis. "RUL:A Declarative Update Language for RDF". In *Procs of ISWC'05*, pages 506–521, Galway, Ireland, Nov 05.

[19] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-Centric Management of Versions in an XMLWarehouse. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 581–590, Roma, Italy, 2001.

[20] M. Klein N. F. Noy, S. Kunnatur and M. A. Musen. "Tracking Changes During Ontology Evolution". In *Procs of ISWC'04*, pages 259–273, Hisroshima, Japan, November 2004.

[21] N. F. Noy and M. A. Musen. "PromptDiff: A Fixed-point Algorithm for Comparing Ontology Versions". In *Procs of AAAI-02*, pages 744–750, Edmonton, Alberta, July 2002.

[22] N. F. Noy and M. A. Musen. "Ontology versioning in an ontology management framework". *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[23] P. Plessers and O. De Troyer. "Ontology Change Detection Using a Version Log". In *Procs of ISWC'05*, pages 578–592, Galway,Ireland, November 2005.

[24] J. A. La Poutre' and J. van Leeuwen. "Maintenance of Transitive Closures and Transitive Reductions of Graphs". In *Procs of the Intern. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 106–120, July 1987.

[25] Y. Theoharis, V. Christophides, and G.Karvounarakis. "Benchmarking Database Representations of RDF/S Stores". In *Procs of ISWC'05*, pages 685–701, Galway, Ireland, Nov 05.

[26] Y. Theoharis, G. Georgakopoulos, and V. Christophides. On the Synthetic Generation of Semantic Web Schemas. In *Procs of the Joint ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases. Colocated with VLDB2007*, September 2007.

[27] M. Volkel, W. Winkler, Y. Sure, S. Ryszard Kruk, and M. Synak. "SemVersion: A Versioning System for RDF and Ontologies". In *Procs of ESWC'05.*, Heraklion, Crete, May 2005.

[28] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

[29] D. Zeginis. On computing deltas of rdf/s knowledge bases. Master's thesis, Computer Science Department, University of Crete, 2008.

[30] D. Zeginis, Y. Tzitzikas, and V. Christophides. "On the Foundations of Computing Deltas Between RDF Models". In *Proceedings of the 6th International Semantic Web Conference (ISWC-07)*, Busan, S. Korea, 2007.

[31] Z. Zhang, L. Zhang, C. Lin, Y. Zhao, and Y. Yu. "Data Migration for Ontology Evolution". In *Poster Proceedings ISWC'03*, Sanibel Island, Florida, USA, October 2003.

Table 5: Delta time for Biological Data

K			K'			Time of ($\Delta(K \rightarrow K')$)				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b1}	2898	0.229	K_{b2}	2964	0.229	0.281	0.289	0.329	0.296	0.312
K_{b3}	4493	0.225	K_{b4}	4816	0.223	0.594	0.603	0.765	0.625	0.648
K_{b5}	5994	0.218	K_{b6}	6068	0.218	0.967	0.973	1.046	0.975	0.998
K_{b7}	7340	0.205	K_{b8}	7399	0.205	1.119	1.128	1.354	1.133	1.149
K_{b9}	9028	0.196	K_{b10}	9217	0.196	1.665	1.682	1.763	1.698	1.718
K_{b11}	10806	0.182	K_{b12}	12772	0.182	1.973	2.021	2.090	1.979	2.042
K_{b13}	11680	0.178	K_{b14}	11779	0.178	2.181	2.196	2.300	2.187	2.249

Table 6: Delta size for Biological Data

Size of $\Delta(K \rightarrow K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b1}	K_{b2}	(79, 13) 92	(79, 13) 92	(94, 13) 107	(79, 13) 92	(79, 13) 92
K_{b3}	K_{b4}	(425, 102) 527	(425, 102) 527	(487, 103) 590	(425, 102) 527	(425, 103) 528
K_{b5}	K_{b6}	(92, 18) 110	(92, 18) 110	(107, 18) 125	(92, 18) 110	(92, 18) 110
K_{b7}	K_{b8}	(70, 11) 81	(70, 11) 81	(79, 11) 90	(70, 11) 81	(70, 11) 81
K_{b9}	K_{b10}	(250, 61) 311	(250, 61) 311	(286, 63) 349	(250, 61) 311	(250, 63) 313
K_{b11}	K_{b12}	(67, 4) 71	(67, 4) 71	(79, 4) 83	(67, 4) 71	(67, 4) 71
K_{b13}	K_{b14}	(117, 18) 135	(117, 18) 135	(134, 18) 152	(117, 18) 135	(117, 18) 135
Size of $\Delta(K' \rightarrow K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{b14}	K_{b13}	(18, 117) 135	(18, 117) 135	(18, 134) 152	(18, 117) 135	(18, 134) 152
K_{b12}	K_{b11}	(4, 67) 71	(4, 67) 71	(4, 79) 83	(4, 67) 71	(4, 79) 83
K_{b10}	K_{b9}	(61, 250) 311	(61, 250) 311	(63, 286) 349	(61, 250) 311	(61, 286) 347
K_{b8}	K_{b7}	(11, 70) 81	(11, 70) 81	(11, 79) 90	(11, 70) 81	(11, 79) 90
K_{b6}	K_{b5}	(18, 92) 110	(18, 92) 110	(18, 107) 125	(18, 92) 110	(18, 107) 125
K_{b4}	K_{b3}	(102, 425) 527	(102, 425) 527	(103, 487) 590	(102, 425) 527	(102, 487) 589
K_{b2}	K_{b1}	(13, 79) 92	(13, 79) 92	(13, 94) 107	(13, 79) 92	(13, 94) 107

Table 7: Delta size for simple Synthetic Data

Size of $\Delta(K \rightarrow K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{s1}	K_{s2}	(9,490, 4,509) 13,999	(9,490, 4,434) 13,924	(15,905, 6,932) 22,837	(9,430, 4,434) 13,864	(9,430, 6,932) 16,362
K_{s2}	K_{s3}	(14,089, 9,096) 23,185	(14,089, 9,022) 23,111	(22,937, 13,512) 36,449	(14,041, 9,022) 23,063	(14,041, 13,512) 27,553
K_{s3}	K_{s4}	(18,579, 13,635) 32,214	(18,579, 13,548) 32,127	(29,993, 20,422) 50,415	(18,519, 13,548) 32,067	(18,519, 20,422) 38,941
K_{s1}	K_{s4}	(19,461, 4,543) 24,004	(19,461, 4,495) 23,956	(34,960, 6,991) 41,951	(19,442, 4,495) 23,937	(19,442, 6,991) 26,433
K_{s4}	$C(K_{s4})$	(17,409, 0) 17,409	(17,409, 0) 17,409	(0, 0) 0	(0, 0) 0	(0, 0) 0
Size of $\Delta(K' \rightarrow K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
$K_{C(s4)}$	K_{s4}	(0, 17,409) 17,409	(0, 0) 0	(0, 0) 0	(0, 0) 0	(0, 0) 0
K_{s4}	K_{s1}	(4,543, 19,461) 24,004	(4,543, 19,442) 23,985	(6,991, 34,960) 41,951	(4,495, 19,442) 23,937	(4,495, 34,960) 39,455
K_{s4}	K_{s3}	(13,635, 18,579) 32,214	(13,635, 18,519) 32,154	(20,422, 29,993) 50,415	(13,548, 18,519) 32,067	(13,548, 29,993) 43,541
K_{s3}	K_{s2}	(9,096, 14,089) 23,185	(9,096, 14,041) 23,137	(13,512, 22,937) 36,449	(9,022, 14,041) 23,063	(9,022, 22,937) 31,959
K_{s2}	K_{s1}	(4,509, 9490) 13,999	(4,509, 9,430) 13,939	(6,932, 15905) 22,837	(4,434, 9430) 13,864	(4,434, 15,905) 20,339

Table 8: Delta time for simple Synthetic Data

K			K'			Time of ($\Delta(K \rightarrow K')$)				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{s1}	5,162	0.821	K_{s2}	10,267	0.789	0.570	0.583	0.952	0.594	0.718
K_{s2}	10,267	0.789	K_{s3}	15,389	0.806	0.889	0.903	1.499	1.187	1.388
K_{s3}	15,389	0.806	K_{s4}	20,460	0.826	1.296	1.352	2.217	1.623	1.702
K_{s1}	5,162	0.821	K_{s4}	20,460	0.826	1.077	1.088	1.624	1.109	1.092
K_{s4}	20,460	0.826	$C(K_{s4})$	37,369	0	1.225	1.358	1.782	1.462	1.639

Table 9: Delta time for Complex Synthetic Data

K			K'			Time of ($\Delta(K \rightarrow K')$)				
KB	Triples	$is(K)$	KB	Triples	$is(K')$	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l1}	14,250	2.970	K_{l2}	28,355	2.868	1.333	1.339	5.382	1.844	2.590
K_{l2}	28,355	2.868	K_{l3}	42,477	2.923	3.266	2.511	9.302	3.413	5.201
K_{l3}	42,477	2.923	K_{l4}	56,546	2.987	3.443	3.643	13.442	5.071	7.329
K_{l1}	14,250	2.970	K_{l4}	56,546	2.987	2.131	2.192	7.741	2.205	3.405

Table 10: Delta size for the Complex Synthetic Data

Size of $\Delta(K \rightarrow K')$						
K	K'	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l1}	K_{l2}	(27,182, 13,077) 40,259	(27,182, 12,372) 39,547	(90,207, 37,100) 127,307	(26,762, 12,372) 39,134	(26,762, 37,100) 63,862
K_{l2}	K_{l3}	(41,270, 27,148) 68,418	(41,270, 26,804) 68,074	(128,328, 71,344) 199,672	(40,952, 26,804) 67,756	(40,952, 71,344) 112,296
K_{l3}	K_{l4}	(54,975, 40,906) 95,881	(54,975, 40,549) 95,524	(167,279, 108,443) 275,722	(54,645, 40,549) 95,194	(54,645, 108,443) 163,088
K_{l1}	K_{l4}	(55,407, 13,111) 68,518	(55,407, 12,703) 68,110	(206,266, 37,339) 243,605	(55,388, 12,703) 68,091	(55,388, 37,339) 92,727
Size of $\Delta(K' \rightarrow K)$						
K'	K	Δ_e	Δ_{ed}	Δ_c	Δ_d	Δ_{dc}
K_{l4}	K_{l1}	(13,111, 55,407) 68,518	(13,111, 55,388) 68,499	(37,339, 206,266) 243,605	(12,703, 55,388) 68,091	(12,703, 206,266) 218,969
K_{l4}	K_{l3}	(40,906, 54,975) 95,881	(40,906, 54,645) 95,551	(108,443, 167,279) 275,722	(40,549, 54645) 95,194	(40,549, 167,279) 207,828
K_{l3}	K_{l2}	(27,148, 41,270) 68,418	(27,148, 40,952) 68,100	(71,344, 128,328) 199,672	(26,804, 40,952) 67,756	(26,804, 128,328) 155,132
K_{l2}	K_{l1}	(13,077, 27,182) 40,259	(13,077, 26,762) 39,839	(37,100, 90,207) 127,307	(12,372, 26,762) 39,134	(12,372, 90,207) 102,579