ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-70

A Formal Approach for RDF/S Ontology Evolution

George Konstantinidis and **Giorgos Flouris** and **Grigoris Antoniou** and **Vassilis Christophides**[1]

Abstract. In this paper, we consider the problem of ontology evolution in the face of a change operation. We devise a general-purpose algorithm for determining the effects and side-effects of a requested elementary or complex change operation. Our work is inspired by belief revision principles (i.e., validity, success and minimal change) and allows us to handle any change operation in a provably rational and consistent manner. To the best of our knowledge, this is the first approach overcoming the limitations of existing solutions, which deal with each change operation on a per-case basis. Additionally, we rely on our general change handling algorithm to implement specialized versions of it, one per desired change operation, in order to compute the equivalent set of effects and side-effects.[2]

1 INTRODUCTION

Stored knowledge, in any knowledge-based application, may need to change due to various reasons, including changes in the modeled world, new information on the domain, newly-gained access to information previously unknown, and other eventualities [11]. Here, we consider the case of ontologies expressed in RDF/S (as most of the Semantic Web Schemas (85,45%) are expressed in RDF/S [14]) and introduce a formal framework to handle the evolution of an ontology given a change operation.

We pay particular attention to the semantics of change operations which can, in principle, be either *elementary* (involving a change in a single ontology construct) or *composite* (involving changes in multiple constructs) [5]. Even though RDF/S does not support negation, the problem is far from trivial as inconsistencies may rise due to the validity rules associated with RDF/S ontologies. In fact, naive set-theoretical addition or removal of ontological constructs (i.e., direct application of a change) has been acknowledged as insufficient for ontology evolution [4, 6, 12].

Most of the implemented ontology management systems (e.g., [1, 2, 8, 13]), are designed using an ad-hoc approach, that solves the problems related to each change operation on a per-case basis. More specifically, they explicitly define a finite, and thus incomplete, set of change operations that they support, and have determined, a priori, the semantics of each such operation. Hence, given the lack of a formal methodology, the designers of these systems have to determine, in advance, all the possible invalidities that could occur in reaction to a change, the various alternatives for handling any such possible invalidity, and to pre-select the preferable solutions for implementation per case [6]; this selection is hard-coded into the systems' implementations. This approach requires a highly tedious, case-based reasoning which is error-prone and gives no formal guarantee that the cases and options considered are exhaustive.

To overcome these limitations, we propose an ontology evolution framework and elaborate on its formal foundations. Our methodology is driven by ideas and principles of the *belief revision* literature [3]. In particular, we adopt the *Principle of Success* (every change operation is actually implemented) and the *Principle of Validity* (resulting ontology is *valid*, i.e., it satisfies all the validity constraints of the underlying language). Satisfying both these requirements is not trivial, as the straightforward application of a change operation upon an ontology can often lead to invalidity, in which case certain additional actions (*side-effects*) should be executed to restore validity. Sometimes, there may be more than one ways to do so, in which case a selection mechanism should be in place to determine the "best" option. In this paper, we employ a technique inspired by the *Principle of Minimal Change* [3] (stating that the appropriate result of changing an ontology should be as "close" as possible to the original).

The general idea of our approach is to first determine all the invalidities that any given change (elementary or composite) could cause upon the updated ontology, using a formal, well-specified validity model, and then to determine the best way to overcome potential invalidity problems in an automatic way, by exploring the various alternatives and comparing them using a selection mechanism based on an ordering relation on potential side-effects. In particular, our formal approach is parameterizable to this relation, thus providing a customizable way to guarantee the determination of the "best" result. Although our framework is general, in this paper we focus on a fragment of the RDF/S model which exhibits interesting properties for deciding query containment and minimization [10]. To the best of our knowledge, our implementation is the first one that allows processing any type of change operation, and in a fully automatic way.

2 PROBLEM FORMULATION

2.1 Modeling RDF/S, ontologies and updates

In order to abstract from the syntactic peculiarities of the underlying language and develop a uniform framework, we will map RDF to First-Order Logic (FOL). Table 1 (restricted for presentation purposes) shows the FOL representation of certain RDF statements.

The language's semantics is not carried over during the mapping, so we need to combine the FOL representation with a set of *validity rules* that capture such semantics. For technical reasons, we assume that all constraints can be encoded in the form of (or can be broken down into a conjunction of) DEDs (disjunctive embedded dependencies), which have the following general form:

$$\forall \vec{u} P(\vec{u}) \rightarrow \vee_{i=1,\ldots,n} \exists \vec{v_i} Q_i(\vec{u}, \vec{v_i}) \qquad \text{(DED)}$$

where:

- $\vec{u}$,$\vec{v_i}$ are tuples of variables

[1] Institute of Computer Science, FO.R.T.H., Heraklion, Greece, email:gconstan,fgeo,antoniou,christop@ics.forth.gr
[2] This work was partially supported by the EU projects CASPAR (FP6-2005-IST-033572) and KP-LAB (FP6-2004-IST-27490).

Table 1. Representation of RDF facts using FOL predicates

RDF triple	Intuitive meaning	Predicate
C rdf:type rdfs:Class	C is a class	$CS(C)$
P rdf:type rdf:Property	P is a property	$PS(P)$
x rdf:type rdfs:Resource	x is a class instance	$CI(x)$
P rdfs:domain C	domain of property	$Domain(P, C)$
P rdfs:range C	range of property	$Range(P, C)$
C_1 rdfs:subClassOf C_2	IsA between classes	$C\_IsA(C_1, C_2)$
P_1 rdfs:subPropertyOf P_2	IsA between properties	$P\_IsA(P_1, P_2)$
x rdf:type C	class instantiation	$C\_Inst(x, C)$
x P y	property instantiation	$PI(x, y, P)$

- P, Q_i are conjunctions of relational atoms of the form $R(w_1, ..., w_n)$ and equality atoms of the form $(w = w')$, where $w_1, ..., w_n, w, w'$ are variables or constants
- P may be the empty conjunction

We employ DEDs, as they are expressive enough for capturing the semantics of different RDF fragments and other simple data models which are appropriate for our purposes in this paper. Moreover, DEDs will prove suitable for constructing a convenient mechanism for detecting and repairing invalidities.

Table 2 shows some rules that are used to capture the semantics of the various RDF constructs (e.g., R11 captures IsA transitivity), as well as the restrictions imposed by our RDF model (e.g., R8 captures that the domain of a property should be unique). It should be stressed that the semantics of the language captured by tables 1 and 2 essentially corresponds to a fragment of the standard RDF/S data model[3] in which there is a clear role distinction between ontology primitives, no cycles in the subsumption relationships, while property subsumption respects corresponding domain/range subsumption relationships. Such a fragment has been first studied in [10] in an effort to provide a group of sound and complete algorithms for query containment and minimization while it is compatible with W3C guidelines[4] for devising restricted fragments of the RDF/S data model. Similarly, the general-purpose change handling algorithm presented in this paper can be also applied to other fragments of RDF/S (see also [7, 9]) or the standard RDF/S semantics.

In Table 2, Σ denotes the set of constants in our language. We equip our FOL with closed semantics, i.e., CWA (*closed world assumption*). This means that, for two formulas p, q, if $p \nvdash q$ then $p \vdash \neg q$. Abusing notation, for two sets of ground facts U, V, we will say that U implies V ($U \vdash V$) to denote that $U \vdash p$ for all $p \in V$. Any expression of the form $P(x_1, ..., x_k)$ is called a *positive ground fact* where P is a predicate of arity k and $x_1, ..., x_k$ are constant symbols. Any expression of the form $\neg P(x_1, ..., x_k)$ is called a *negative ground fact* iff $P(x_1, ..., x_k)$ is a positive ground fact. L denotes the set of all well-formed formulae that can be formed in our FOL. We denote by L^+ the set of positive ground facts, L^- the set of negative ground facts and set $L^0 = L^+ \cup L^-$, called the set of ground facts of the language. We define:

- An *ontology* is a set $K \subseteq L^+$
- An *update* is a set $U \subseteq L^0$

In simple words, an ontology is any set of positive ground facts whereas an update is any set of positive or negative ground facts. Applying an update to an ontology should result in the incorporation of the update in the ontology.

By definition, ontologies have two properties: (a) they are always consistent (in the purely logical sense) and (b) they imply only the positive ground facts that are already in the ontology. The above two properties together with the CWA semantics, imply that:

- $P(x) \in K \Leftrightarrow K \vdash P(x) \Leftrightarrow K \nvdash \neg P(x)$
- $P(x) \notin K \Leftrightarrow K \vdash \neg P(x) \Leftrightarrow K \nvdash P(x)$

An application of these properties is that updating K with $\neg P(x)$ corresponds to contracting $P(x)$ from K, because "incorporating" $\neg P(x)$ in an ontology could be achieved only by removing $P(x)$ from K. Therefore, updating an ontology with negative ground facts corresponds to contraction/erasure in the standard terminology, whereas updating an ontology with positive ground facts corresponds to revision/update in the standard terminology.

Table 2. Validity Rules

Rule ID/Name	Integrity Constraint	Intuitive Meaning
R2 Domain Applicability	$\forall x, y \in \Sigma$: $Domain(x, y) \rightarrow PS(x) \wedge CS(y)$	Domain applies to properties; the domain of a property is a class
R4 C_IsA Applicability	$\forall x, y \in \Sigma$: $C\_IsA(x, y) \rightarrow CS(x) \wedge CS(y)$	Class IsA applies between classes
R6 C_Inst Applicability	$\forall x, y \in \Sigma$: $C\_Inst(x, y) \rightarrow CI(x) \wedge CS(y)$	Class Instanceof applies between a class instance and a class
R8 Domain is unique	$\forall x, y, z \in \Sigma$: $Domain(x, y) \rightarrow \neg Domain(x, z) \vee (y = z)$	The domain of a property is unique
R10 Domain and Range exists	$\forall x \in \Sigma, \exists y, z \in \Sigma$: $PS(x) \rightarrow Domain(x, z) \wedge Range(x, y)$	Each property has a domain and a range
R11 C_IsA Transitivity	$\forall x, y, z \in \Sigma$: $C\_IsA(x, y) \wedge C\_IsA(y, z) \rightarrow C\_IsA(x, z)$	Class IsA is Transitive
R12 C_IsA Irreflexivity	$\forall x, y, \in \Sigma$: $C\_IsA(x, y) \rightarrow \neg C\_IsA(y, x)$	Class IsA is Irreflexive
R15 Determining C_Inst	$\forall x, y, z \in \Sigma$: $C\_Inst(x, y) \wedge C\_IsA(y, z) \rightarrow C\_Inst(x, z)$	Class instance propagation
R17 Property Instance_of and Domain	$\forall x, y, z, w \in \Sigma$: $PI(x, y, z) \wedge Domain(z, w) \rightarrow C\_Inst(x, w)$	Instanceof between properties reflect in their sources/domains
R23 P_IsA Irreflexivity	$\forall x, y \in \Sigma$: $P\_IsA(x, y) \rightarrow \neg P\_IsA(y, x)$	Property IsA is Irreflexive

2.2 Updating under constraints

We say that an ontology K *satisfies* a validity rule c, iff $K \vdash c$. Obviously for a set C of validity rules, K satisfies C ($K \vdash C$) iff $K \vdash c$ for all $c \in C$. It is easy to see that for a simple constraint of the form $c = \forall u P(u) \rightarrow Q(u)$, where P, Q are simple positive predicates and u is a variable, it holds that:

$$K \vdash c \text{ iff for all constants } x : K \vdash \{\neg P(x)\} \text{ or } K \vdash \{Q(x)\}.$$

This can be easily extended to the general case. Suppose that $c = \forall \vec{u} P(\vec{u}) \rightarrow \vee_{i=1,...,n} \exists \vec{v_i} Q_i(\vec{u}, \vec{v_i})$, where $P(\vec{u}) = P_1(\vec{u}) \wedge \ldots \wedge P_k(\vec{u})$ for some $k \geqslant 0$ and $Q_i(\vec{u}, \vec{v_i}) = Q_{i1}(\vec{u}, \vec{v_i}) \wedge \ldots \wedge Q_{im}(\vec{u}, \vec{v_i})$ for some $m > 0$ depending on i. Then $K \vdash c$ iff for all tuples of constants x at least one of the following is true (note that in case of obvious reference to tuples of constants or variables we will be omitting the $'\neg'$ symbol):

- There is some $j : 0 < j \leqslant k$ such that $K \vdash \{\neg P_j(x)\}$.

[3] http://www.w3.org/TR/rdf-concepts/
[4] http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#technote

- There is some $i : 1 \leqslant i \leqslant n$ and some tuple of constants z such that for all $j = 1, 2, ..., m$ $K \vdash \{Q_{ij}(x, z)\}$.

We can conclude that $K \vdash c$ iff for all tuples of constants x at least one of the following sets is implied by K:

- $\{\neg P_j(x)\}, 0 < j \leqslant k$
- $\{Q_{i1}(x, z) \wedge Q_{i2}(x, z) \wedge ... \wedge Q_{im}(x, z)\}, 1 \leqslant i \leqslant n, z$:constant

Based on the above observation, we define the *component set* of c with respect to some tuple of constants x as follows:

$$Comp(c, x) = \{\{\neg P_j(x)\}|0 < j \leqslant k\} \cup \{\{Q_{i1}(x, z) \wedge Q_{i2}(x, z) \wedge ... \wedge Q_{im}(x, z)\} \,|1 \leqslant i \leqslant n, z : constant\}$$

Prop. 1 will subsequently help us define a valid ontology.

Prop. 1 *$K \vdash c$ iff for all constants x there is some $V \in Comp(c, x)$ such that $K \vdash V$.*

Def. 1 *Consider a FOL language L and a set of validity rules C. An ontology K will be called valid with respect to L and C iff K is consistent and it satisfies the validity rules C.*

Note that a valid ontology, by our rules of Table 2, contains all its implicit knowledge as well (i.e., it is closed with respect to inference). Due to the special characteristics of our framework (e.g., CWA, the form of rules, etc), one does not need to employ full FOL reasoning to determine whether an ontology K is valid (i.e., using Def. 1 and Prop. 1); instead, we can use the specialized procedure described below (Prop. 2).

Prop. 2 *A ground fact $P(x)$, added to an ontology K, would violate rule c, iff there is some set V and tuple of constants $\vec{u}$ for which $\neg P(x) \in V$ and $V \in Comp(c, \vec{u})$ and for all $V' \in Comp(c, \vec{u})$, $V \neq V'$ it holds that $K \nvdash V'$.*

As an example, consider the ontology of Fig. 1(a). The original ontology in our case, per Table 1, is: $K = \{ CS(A), CS(B), CI(a), CI(b), PS(P), Domain(P, A), Range(P, B), PI(a, b, P), C\_Inst(a, A), C\_Inst(b, B)\}$ and the update is: $U = \{Domain(P, D)\}$. To detect rule violations in an automated way, according to Prop. 2, we must find all the rules that contain $\neg Domain(x, y)$, set $x = P, y = D$, and determine whether some other component for the particular instantiation is implied by the ontology. If the answer is no, then the addition of $Domain(P, D)$ would violate the particular instantiation of this rule. In our case, this is true for rule $R2.2$ (domain applicability), for $x = P, y = D$ and rule $R8$ (unique domain) for $x = P, y = D, z = A$ as well as for $x = P, y = A, z = D$ (see also Table 3 for some rules in their component set format). Moreover, it violates rule $R17$ for $x = a, y = b, z = P, w = D$.

One nice property of our detection mechanism is that it provides an immediate way to restore invalidities as well, i.e., generate potential side-effects that would restore the violation. In particular, the violation that Prop. 2 detects can be restored by making any of the elements of $Comp(c, \vec{u})$ true in the ontology. At this point note that when a $Q_{ij}(x, z)$ in some set $V \in Comp(c, x)$ is an equality of the form $w = w'$, then the truth value of this equality is revealed as soon as we instantiate this rule's variables to constants. Therefore, by evaluating an equality as a tautology ($\top$) or contradiction ($\bot$) and replace it accordingly in the rule's instances, we are able to eliminate all the equality atoms from the components sets. Without equalities, the elements of $Comp(c, x)$ contain only positive and negative ground facts, so they are updates in our terminology. This is a very useful remark, as we will subsequently take advantage of the elements of $Comp(c, x)$, applying them as updates.

In our example, the validity of rule $R2.2$, for $x = P, y = D$ can be restored iff either $\{\neg Domain(P, D)\}$ or $\{CS(D)\}$ are added as additional updates (side-effects) to the ontology. Note that side-effects could trigger side-effects of their own if violating any rules.

Table 3. Some validity rules in component set form

Rule ID/Name	Components of the rule
R2 Domain Applicability	$R2.1 : \forall x, y \in \Sigma : Comp(R2.1, (x, y))=$ $\{\{\neg Domain(x, y)\}, \{PS(x)\}\}$ $R2.2 : \forall x, y \in \Sigma : Comp(R2.2, (x, y))=$ $\{\{\neg Domain(x, y)\}, \{CS(y))\}\}$
R8 Domain is unique	$\forall x, y, z \in \Sigma : Comp(R8, (x, y, z))=$ $\{\{\neg Domain(x, y)\}, \{\neg Domain(x, z)\}, \{(y = z)\}\}$
R10 Domain and Range exists	$R10.1 : \forall x \in \Sigma, \exists z \in \Sigma :$ $Comp(R10.1, (x, z))=$ $\{\{\neg PS(x)\}, \{Domain(x, z)\}\}$ $R10.2 : \forall x \in \Sigma, \exists y \in \Sigma :$ $Comp(R10.1, (x, y))=$ $\{\{\neg PS(x)\}, \{, Range(x, y)\}\}$
R17 Property Instance_of and Domain	$\forall x, y, z, w \in \Sigma :$ $Comp(R17, (x, y, z, w)) =$ $\{\{\neg PI(x, y, z)\}$ $,\{\neg Domain(z, w)\}, \{C\_Inst(x, w)\}\}$

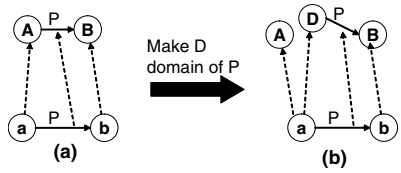

Figure 1. Adding a new domain to a property.

2.3 Selection of side-effects: ordering

If there were no validity rules or we were not interested in the result being a valid ontology the most rational way to perform an update would be to simply apply the changes in U upon K.

Def. 2 *The raw application of an update U to an ontology K is denoted by $K + U$ and is the following ontology: $K + U = \{P(x) \in L^+ | P(x) \in K \cup U and \neg P(x) \notin U\}$*

When a set of changes (i.e., an update U) is raw applied to a valid ontology K, some of the changes that appear in U may be void, i.e., they don't need to be performed because they are already implemented (implied) by the original ontology. We define an operator which, given a resulting ontology K' that an update would produce on a valid ontology K, calculates the actual effects of the update:

Def. 3 *For K a valid ontology and K' an ontology: $Delta(K, K') = \{P(x) \in L^0 | K' \vdash \{P(x)\}$ and $K \nvdash \{P(x)\}\}$*

Delta function is some kind of "edit distance"[5] between K and K'; if $K' = K + U$, then $Delta$ represents the actual changes that U enforces upon K. Thus, $K + U = K + Delta(K, K + U) = K'$, so $Delta(K, K + U)$ produces the same result as U when applied upon an ontology; however they may be different as U could contain void changes.

[5] Note that the term "edit-distance" is usually used for sequences and not sets (i.e., edit scripts)

As already mentioned, the raw application of an update would not work for our case, because it may not respect the validity constraints of the language. Thus, applying an update involves the application of some side-effects. In some cases, it may not be possible to find adequate side-effects for the update at hand; such updates are called *infeasible* and cannot be executed. For example, any inconsistent update (such as, $U = \{CS(A), \neg CS(A)\}$) is infeasible.

In most cases though, an update has several possible alternative sets of side-effects, which implies that a selection should be made. Consider an update U with alternative side-effects U_1 and U_2. Then, the set of changes that should be raw applied on the initial ontology, in order to reach a valid result, is either $U \cup U_1$ or $U \cup U_2$. According to the Principle of Minimal Change we should choose the one which causes the "mildest" effects upon the ontology; to determine the "relative mildness" (or "relative cost") of such effects, we define an ordering $\leqslant$ between updates. Note that this ordering should depend on K itself: for example, the "cost" of removing an IsA relation between A and B should depend on the importance of the concepts A, B in the RDF graph itself. The following conditions have proven necessary for an ordering to produce "rational" results.

Def. 4 *An ordering $\leqslant_K$ is called update-generating iff the following conditions hold:*

- ***Delta Antisymmetry**: For any U, U': $U \leqslant_K U'$ and $U' \leqslant_K U$ implies $Delta(K, K + U) = Delta(K, K + U')$.*
- ***Transitivity**: For any U, U', U'': $U \leqslant_K U'$ and $U' \leqslant_K U''$ implies $U \leqslant_K U''$.*
- ***Totality**: For any U, U': $U \leqslant_K U'$ or $U' \leqslant_K U$.*
- ***Conflict Sensitivity**: For any U, U': $U \leqslant_K U'$ iff $Delta(K, K + U) \leqslant_K Delta(K, K + U')$.*
- ***Monotonicity**: For any U, U': $U \subseteq U'$ implies $U \leqslant_K U'$.*

Similarly, an ordering scheme $\{\leqslant_K | K : \text{a valid ontology}\}$ is called update-generating iff $\leqslant_K$ is update-generating for all valid ontologies K.

For our RDF case we introduced a particular update-generating ordering, which is based on the ordering shown in Table 4 (among the positive and negative predicates presented in Table 1 for simplicity). The details of the expansion of this ordering to refer to ground facts and sets of ground facts (i.e., updates) is omitted due to space limitations. In short, the general idea is that an update U_1 is "cheaper" (or preferable) than U_2 (denoted by $U_1 \leqslant_K U_2$) iff the "most expensive" predicate used in update U_1, is "cheaper" than the "most expensive" predicate used in update U_2 where the predicates' relative preference is determined by the order shown in Table 4. Ties are resolved using cardinality considerations and/or the relative importance of the predicate's arguments in the original ontology (details omitted). Our ordering was based on the results of experimentation on various alternative orderings and results to an efficient and intuitive implementation. Nonetheless, our algorithm works with any update-generating ordering; each different ordering would model and impose a different global evolution policy on our algorithm. Fig. 1(b) depicts the outcome of the requested update with respect to our ordering.

Table 4. Ordering of predicates

$PI <_p C\_Inst <_p P\_IsA <_p C\_IsA <_p \neg PI <_p \neg C\_Inst <_p \neg P\_IsA <_p \neg C\_IsA <_p \neg Domain <_p \neg Range <_p \neg CI <_p \neg PS <_p \neg CS <_p Domain <_p Range <_p CI <_p PS <_p CS$

Def. 5 *Consider a FOL language L, a set of integrity constraints C, some update-generating ordering scheme $\leqslant$ and a change operator $\bullet : L^+ \times L^0 \rightarrow L^+$. A change operation $K \bullet U$ will be called rational with respect to $\leqslant$ iff it satisfies the following properties for all ontologies K and updates U:*

- *Limit Cases: If K is not a valid ontology or U is an infeasible update, then: $K \bullet U = K$.*
- *General Case: If K is a valid ontology and U is a feasible update, then:*
  - *Principle of Success: $K \bullet U \vdash U$*
  - *Principle of Validity: $K \bullet U$ is a valid ontology*
  - *Principle of Minimal Change: For all valid ontologies K' such that $K' \vdash U$, it holds that $Delta(K, K \bullet U) \leqslant_K Delta(K, K')$*

Def. 5 dictates that applying a *rational* change operator between an update and an ontology should result (in the general case) in a valid ontology (Principle of Validity), which implies the update (Principle of Success). Moreover, for any other valid ontology K' that could be an alternative result, the set of (non-void) side-effects leading to K' (captured by the $Delta$ function) is more "expensive" than the set of (non-void) side-effects leading to the result of the rational change operation. In effect, the rational change operator applies the minimum, with respect to $\leqslant_K$, set of effects and side-effects.

3 ALGORITHMS

3.1 General-purpose algorithm

We now present our general-purpose algorithm shown in Table 5. For a given language L (predicates and rules) and update-generating ordering $\leqslant$, the function takes as inputs the ontology K, an update U, and the set ESE (initially empty) of already considered effects and side-effects. The algorithm identifies all invalidities caused by the predicates of the update, appends the update with each possible alternative side-effect separately and calls itself. Upon returning, it compares the different alternatives with the "min" function (which implements $\leqslant$). Upon termination it returns the effects (U) and the minimal (per $\leqslant_K$) set of side-effects of U upon K. The output of the algorithm, if not infeasible, is ready to be raw applied to K, as stated in Theorem 1.

Table 5. Function Update: $(U, K, ESE) \rightarrow Delta(K, K \bullet U)$

$STEP1$: If $U \cup ESE$ is inconsistent, then return `INFEASIBLE`.
$STEP2$: If $(K \cup ESE) \vdash U$, then return $\emptyset$
$STEP3$: Take an arbitrary ground fact $P(x) \in U \setminus ESE$ such that $K \nvdash \{P(x)\}$
$STEP4$: Find a rule r, such that there is some set V and tuple of constants $\vec{y}$ for which $\neg P(x) \in V$ and $V \in Comp(r, \vec{y})$ and for all $V' \in Comp(r, y)$, $V \neq V'$ it holds that $V' \nsubseteq U \cup ESE$.
$STEP5$: If there is no such rule, then return $\{P(x)\} \cup$ `Update`$(U, K, ESE \cup \{P(x)\})$.
$STEP6$: Otherwise, select (arbitrarily) one such rule, say r, and return $min\{$`Update`$(U \cup V', K, ESE)\}$, where the min is taken over all $V' \in Comp(r, \vec{y}), V' \neq V$.

Theorem 1 *The raw application of the output of $Update(K, U, \emptyset)$ to K implements a rational change operation.*

The complexity of our algorithm depends on the parameters (language, rules, ordering). When considering an infinite number of constants in our language, we could have an infinite number of rule instances or a rule instance with an infinite size (when $\exists$ in a DED stands before a free variable). However, for our parameters, we can limit Σ to the finite set of all the names appearing in K or U, plus an extra auxiliary constant. The intuition behind this choice is that our ordering guarantees that no solution involving more than one auxiliary constant names (i.e., names not in K or U) could ever be selected for implementation (per $\leqslant$). This fact and Theorem 2 below guarantee termination of the algorithm:

Theorem 2 *Consider the language and the ordering defined in section 2, an ontology K and an update U. Then, if we restrict the set of constants of our language to a finite set, before calling $Update(K, U, \emptyset)$, the latter terminates.*

3.2 Special-purpose algorithms

In practice, our general-purpose algorithm will be applied for a particular language and ordering. For such a case, it makes sense to trade generality for computational efficiency and develop special-purpose algorithms that would produce the same output as the general-purpose one for the particular set of parameters. For this purpose, we singled out a number of useful change operations and developed a special-purpose algorithm for each one (for the particular language and ordering at hand). Since there is an infinite number of possible updates, this effort is inherently incomplete, and we will necessarily have to resort to the general-purpose algorithm for unconsidered operations.

This approach may seem to reintroduce the drawbacks of ad-hoc approaches mentioned earlier, but this is not the case: the special-purpose algorithms have been formally proven to be equivalent to the general algorithm (i.e., they are rational change operators) for the specific operation they tackle. This can be proved by running the general-purpose algorithm for all relevant states of an ontology and verifying its output against the output of the special-purpose algorithms. Thus, special-purpose algorithms are more efficient than the general-purpose algorithm, but use the same general approach as a formal foundation; moreover, any unforseen operation can anyway be dealt with by the general algorithm. As an example, Table 6 shows such an algorithm for adding a domain to a property, and Prop. 3 shows the relevant result.

Prop. 3 *Consider the language and the ordering defined in section 2, an ontology K and the update $U = \{Domain(P, D)\}$ (for any $P, D \in \Sigma$). Then, the output of the algorithm in Table 5 with the above inputs is identical to the output of the algorithm in Table 6 with the same inputs.*

Table 6. Add Domain Algorithm

If PS(P) doesn't exist does not already exist in K:
  Add to output $\{PS(P), Range(P, rdfs:Resource)\}$
If CS(D) doesn't exist does not already exist in K:
  Add to output $\{CS(D)\}$
Remove the old Domain, e.g., add to output $\{\neg Domain(P, A)\}$
Add the new Domain, i.e., add to output $\{Domain(P, D)\}$
If P is instantiated by a property instance, say $PI(a, b, P)$
  Verify that (the new domain) D has as its instance the resource a
  If not, add to output an instance relationship from a to D.

4 CONCLUSIONS

In this paper, we studied the problem of updating an ontology in the face of new information. We criticized the currently used paradigm of selecting a number of operations to support and determining the proper effects of each operation on a per-case basis, and proposed a formal framework to describe updates and their effects, as well as a general-purpose algorithm to perform those updates. Our methodology is inspired by the general belief revision principles of Validity, Success and Minimal Change [3]. The end result is an algorithm that is highly parameterizable, both in terms of the language used and in terms of the implementation of the Principle of Minimal Change.

Our methodology exhibits a "faithful" behavior with respect to the various choices involved, regardless of the particular ontology or update at hand. It lies on a formal foundation, issuing a solid and customizable method to handle any type of change on any ontology, including operations not considered at design time. In addition, it avoids resorting to the error-prone per-case reasoning of other systems, as all the alternatives regarding an update's side-effects can be derived from the language's rules themselves, in an exhaustive and provably correct manner. Although our general algorithm can be applied to a variety of languages, in this paper we elaborated on a specific fragment of RDF. This restriction allowed the development of special-purpose algorithms which provably exhibit behavior identical to the general-purpose one (so they are rational change operators), but also enjoy much better computational properties.

Our approach was recently implemented in a large scale real-time system, as part of the ICS-FORTH Semantic Web Knowledge Middleware (SWKM), which includes a number of web services for managing RDF/S knowledge bases[6]. Future work includes experimental evaluation of the algorithms' performance and the incorporation of techniques and heuristics that would further speed up our algorithms. We also plan to evaluate the feasibility of applying the same methodology in richer languages, such as OWL Lite (notice that for highly complex languages, the development of a complete set of validity rules may be difficult).

REFERENCES

[1] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, 'OilEd: A Reasonable Ontology Editor for the Semantic Web', *Ki 2001: Advances in AI: Joint German/Austrian Conference on AI, Vienna, Austria, September 19-21, 2001: Proceedings*, (2001).
[2] T. Gabel, Y. Sure, and J. Voelker, 'KAON–ontology management infrastructure', *SEKT informal deliverable*, **3**(1).
[3] P. Gärdenfors, 'Belief Revision: An Introduction', *Belief Revision*, **29**, 1–28, (1992).
[4] P. Haase and Y. Sure, 'D3.1.1.b state of the art on ontology evolution', Technical report, (2004).
[5] M. Klein and N.F. Noy, 'A component-based framework for ontology evolution', *Workshop on Ontologies and Distributed Systems at IJCAI*, (2003).
[6] G. Konstantinidis, G. Flouris, G. Antoniou, and V. Christophides, 'Ontology evolution: A framework and its application to rdf', in *Proceedings of the Joint ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases (SWDB-ODBIS-07)*, (2007).
[7] S. Munoz, J. Perez, and C. Gutierrez, 'Minimal deductive systems for rdf', in *Proceedings of the 4^{th} European Semantic Web Conference*, (2007).
[8] N. Noy, R. Fergerson, and M. Musen, 'The knowledge model of Protégé-2000: Combining interoperability and flexibility', *Lecture Notes in Artificial Intelligence (LNAI)*, **1937**, 17–32.
[9] J.Z. Pan and I. Horrocks, 'Metamodeling Architecture of Web Ontology Languages', *Proceedings of the Semantic Web Working Symposium*, **149**, (2001).
[10] G. Serfiotis, I. Koffina, V. Christophides, and V. Tannen, 'Containment and minimization of rdf/s query patterns', in *Proceedings of the 4^{th} International Semantic Web Conference (ISWC-05)*, (2005).
[11] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic, 'User-driven ontology evolution management', *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW*, (2002).
[12] L. Stojanovic and B. Motik, 'Ontology Evolution within Ontology Editors', *Proceedings of OntoWeb-SIG3 Workshop*, 53–62, (2002).
[13] Y. Sure, J. Angele, and S. Staab, 'OntoEdit: Multifaceted Inferencing for Ontology Engineering', *Journal on Data Semantics*, **1**(1), 128–152.
[14] Y. Theoharis, Y. Tzitzikas, D. Kotzinos, and V. Christophides, 'On graph features of semantic web schemas', *IEEE Transactions on Knowledge and Data Engineering*, **20**(5), (2008).

[6] http://athena.ics.forth.gr:9090/SWKM