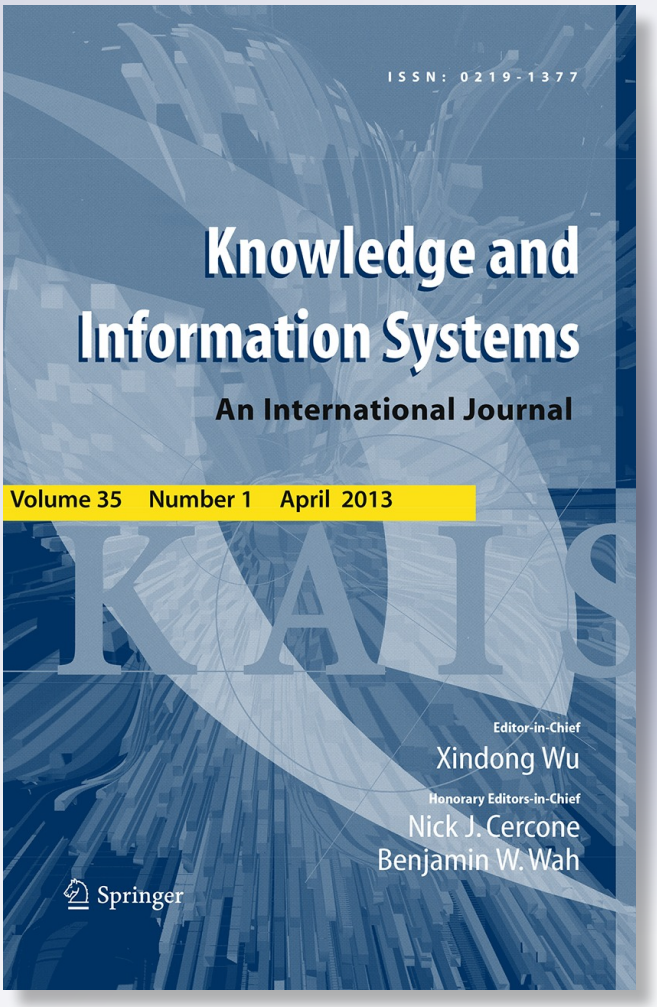
Formal foundations for RDF/S KB evolution

Giorgos Flouris, George Konstantinidis, Grigoris Antoniou & Vassilis Christophides

Knowledge and Information Systems
An International Journal

ISSN 0219-1377
Volume 35
Number 1

Knowl Inf Syst (2013) 35:153-191
DOI 10.1007/s10115-012-0500-2



Springer

Your article is protected by copyright and all rights are held exclusively by Springer-Verlag London Limited. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your work, please use the accepted author's version for posting to your own website or your institution's repository. You may further deposit the accepted author's version on a funder's repository at a funder's request, provided it is not made publicly available until 12 months after publication.

Author's personal copy

Knowl Inf Syst (2013) 35:153–191
DOI 10.1007/s10115-012-0500-2

REGULAR PAPER

Formal foundations for RDF/S KB evolution

Giorgos Flouris · George Konstantinidis · Grigoris Antoniou · Vassilis Christophides

Received: 15 April 2011 / Revised: 3 February 2012 / Accepted: 23 April 2012 /
Published online: 17 May 2012
© Springer-Verlag London Limited 2012

Abstract There are ongoing efforts to provide declarative formalisms of integrity constraints over RDF/S data. In this context, addressing the evolution of RDF/S knowledge bases while respecting associated constraints is a challenging issue, yet to receive a formal treatment. We provide a theoretical framework for dealing with both schema and data change requests. We define the notion of a rational change operator as one that satisfies the belief revision principles of Success, Validity and Minimal Change. The semantics of such an operator are subject to customization, by tuning the properties that a rational change should adhere to. We prove some interesting theoretical results and propose a general-purpose algorithm for implementing rational change operators in knowledge bases with integrity constraints, which allows us to handle uniformly any possible change request in a provably rational and consistent manner. Then, we apply our framework to a well-studied RDF/S variant, for which we suggest a specific notion of minimality. For efficiency purposes, we also describe specialized versions of the general evolution algorithm for the RDF/S case, which provably have the same semantics as the general-purpose one for a limited set of (useful in practice) types of change requests.

Keywords RDF/S evolution · Integrity constraints · Curated KBs · Evolution in the presence of integrity constraints · Principle of Minimal Change · Change operator

G. Flouris (✉) · G. Antoniou · V. Christophides
Institute of Computer Science, Foundation for Research and Technology-Hellas,
Heraklion, Greece
e-mail: fgeo@ics.forth.gr

G. Konstantinidis
University of Southern California, Los Angeles, CA, USA

G. Antoniou
University of Huddersfield, Huddersfield, UK

V. Christophides
University of Crete, Heraklion, Greece

Author's personal copy

1 Introduction

Recently, we have been witnessing an explosion on the number and size of curated knowledge bases (KBs),[1] published in RDF/S [10,47] in the context of the W3C Linked Open Data Initiative.[2] Several works [11,12,22,43,49,55,60] have acknowledged the need for introducing *integrity constraints* in KBs (and RDF/S KBs in particular). This is motivated by the ongoing discussion about the rule level of the Semantic Web [22], as well as by the need to support various applications, such as semantic interoperability [43], the integration of ontologies with relational databases [49,60], query optimization [55] and efficient query answering [11,12].

Given that RDF/S does not impose any constraints on the data, any application-specific constraints (e.g., functional properties) or semantics (e.g., acyclicity in subsumptions) can only be captured using declarative formalisms for representing constraints on top of RDF/S data. In this paper, we will use the term *RDF/S KB* to denote possibly interlinked and populated RDF/S ontologies (and their instance data) with associated integrity constraints.

RDF/S KBs are often subject to change for various reasons, including changes to the modeled world, new information on the domain (e.g., due to extracted metadata from text [32]), newly-gained access to information previously unknown or classified (e.g., due to entity resolution or disambiguation [51]), and other eventualities [28,57,61]. To address this problem, we propose a change framework that supports arbitrarily complex change requests in RDF/S KBs, as well as customizable semantics for the change operator. We consider both *schema change requests* (affecting the schema part of the RDF/S KB, i.e., classes and properties) and *data change requests* (affecting the data part, i.e., individuals), as well as combinations of the two. Note that the problem of determining the result of a change operation is complicated by the constraints associated with RDF/S KBs, because the result should be *valid*, i.e., it should satisfy the associated *validity model*, represented by the integrity constraints.

Our approach is driven by ideas of the *belief revision* literature [31]. In particular, we adopt the principles of *Success* (a change request must be implemented if possible) and *Validity* (the result should satisfy the validity model). To apply a given change request to an RDF/S KB, we first check whether a direct application would result to a valid KB. For example, if the integrity constraints state that the class subsumption hierarchy must be acyclic, then the addition of a class subsumption causing a cycle would result to an invalidity. In such a case, additional changes (called *side-effects*) should be applied on top of the original change request, to enforce validity; in our example, the cycle could be broken by removing one of the subsumptions causing it. In most cases, there are various alternatives for the side-effects to be used, so a *selection mechanism* is necessary to determine the best (i.e., preferred) option; such a mechanism should implement the *Principle of Minimal Change* [31], which states that a change operation should have the minimal possible impact upon an RDF/S KB (under some application-specific notion of minimality). Any change operator respecting these principles will be called *rational*.

As RDF/S KBs are usually backended by relational databases, we will use the relational model as an abstraction to formalize KBs and integrity constraints. In particular, we consider *DED constraints* [20] (*disjunctive embedded dependencies*), which can capture several interesting types of constraints, including constraints mainly used in the relational context, such as primary key and foreign key constraints (used, e.g., in [43]), and constraints used in the

[1] http://www.w3.org/wiki/DataSetRDFDumps.

[2] http://linkeddata.org/.

Author's personal copy

RDF/S and ontological context, such as acyclicity and transitivity constraints (as in [55]) and cardinality constraints (used in [49]). In addition, DEDs can be used to detect and resolve invalidities efficiently, using only syntactical manipulations which avoid the need to perform standard (and inefficient) reasoning. Our mechanism for selecting the *preferred* (minimal) side-effects is modeled as a *user-defined ordering* between the various solutions; as a result, it can be freely defined and customized depending on the needs of the application at hand. Using these abstractions, we provide a *general framework for changing a KB in the presence of integrity constraints*; our approach will be initially described using the general abstraction and then we will show how these ideas can be applied to RDF/S.

In addition, we describe a *general-purpose algorithm* which, given a customization (i.e., a specific set of integrity constraints and selection mechanism), *implements a rational change operator* for the given setting. The algorithm uses a recursive process. In each step of the recursion one violated integrity constraint is selected, and we determine all the possible ways to resolve this violation. Each resolution option may have unforeseen implications, so the resolution that will lead to the most preferable overall result (set of side-effects) cannot be, in general, determined a priori. Therefore, we have to consider each of the different options, as side-effects, in alternative recursive branches. Then, another violated integrity constraint is selected (possibly different for each branch) and the process is repeated. At the end of the recursion, there will be no violated constraints, and the accumulated side-effects (one set per recursion branch) are compared using the selection mechanism; the preferred one is singled out and returned. Due to its generality, this algorithm is NP-hard even for the RDF/S case; for this reason, we also develop *efficient specializations* of the general-purpose algorithm, which exhibit the same behavior and semantics, but are applicable only for a specific setting.

This paper is structured as follows: in Sect. 2 we present an example that helps identifying the main challenges associated with the problem we consider; this example will be used for illustration purposes throughout the paper. In Sect. 3, we provide a general modeling of the problem of changing KBs in the presence of integrity constraints. In Sects. 4, 5, 6, we formalize the three guiding principles of this work (Success, Validity and Minimal Change, respectively), as well as the validity model and the selection mechanism. In Sect. 7, we provide an algebraic viewpoint for our framework, by defining rational change operators and proving that each different parameterization (validity model and selection mechanism) defines uniquely a rational change operator. In Sect. 8, we describe the general algorithm that can be used to implement a rational change operator and show its correctness. Section 9 shows how the abstract ideas of the previous sections can be applied to the case of changes upon RDF/S KBs, and Sect. 10 describes efficient algorithms that specialize the general-purpose one to implement a rational change operator for the RDF/S setting. Section 11 provides a summary of related work, whereas Sect. 12 concludes the paper. Finally, we include an Appendix containing the proofs of all the formal results appearing in the paper. This paper is an extended and highly revised version of [40]; additional contributions include a better formalization and proofs, revised algorithms, more complete comparison to related work, more detailed description of the special-purpose algorithms and a number of complexity results.

2 Motivating example

As explained above, RDF/S KBs are backended by relational databases, so we will abstract and formalize our framework for RDF/S KB evolution using relational concepts. For illustration purposes, we will use throughout the paper the example of Table 1, which shows

Author's personal copy

Table 1 Motivating example

Paper
DB_Paper
RDF_Paper
Cool_Paper

PublishedIn	
DB_Paper	KAIS
RDF_Paper	KAIS
Cool_Paper	Cool_Journal

Journal
KAIS
Cool_Journal

Cites	
DB_Paper	Cool_Paper
RDF_Paper	DB_Paper
Cool_Paper	RDF_Paper

a simple relational database with 4 tables, indicating papers that are published in journals, as well as the papers' citations.

Let us suppose that we would like to impose some constraints on the above schema, which are formally expressed as shown below:

- $\sigma_1 = \forall x, y PublishedIn(x, y) \rightarrow Paper(x)$
- $\sigma_2 = \forall x, y PublishedIn(x, y) \rightarrow Journal(y)$
- $\sigma_3 = \forall x, y Cites(x, y) \rightarrow Paper(x)$
- $\sigma_4 = \forall x, y Cites(x, y) \rightarrow Paper(y)$
- $\sigma_5 = \forall x Paper(x) \rightarrow \exists y PublishedIn(x, y)$
- $\sigma_6 = \forall x, y, z PublishedIn(x, y) \wedge PublishedIn(x, z) \rightarrow (y = z)$
- $\sigma_7 = \forall x Paper(x) \rightarrow \exists y(Cites(x, y) \wedge (x \neq y))$

The intuitive meaning of these constraints is that papers are published in journals (σ_1), and papers cite papers (σ_2). Each paper must be published (σ_5), but the same paper cannot be published to two different journals (σ_6). Finally, each paper must cite at least one other paper (σ_7).

Let us now suppose that a user notices that $Cool_Paper$ is not actually published in $Cool_Journal$, and issues an update request to delete $PublishedIn(Cool_Paper, Cool_Journal)$. If we go ahead and implement this change, we notice that σ_5 will be violated, because $Cool_Paper$ will not be associated to any journal via the $PublishedIn$ relationship any more. This is not acceptable, per the Principle of Validity; on the other hand, the Principle of Success forces us to implement this change, as it is the user's desire to have the specific tuple removed. Our only way out of the deadlock is to implement other changes, in addition to the one explicitly requested by the user, in order to resolve the constraint violation. These changes are called *side-effects*. In this particular example, we have two options:

1. To decide that $Cool_Paper$ is actually published in some other journal, and introduce a new tuple, in this case $PublishedIn(Cool_Paper, KAIS)$. Further checks would determine that no other constraint is violated and the process can stop, having as side-effect only the addition of the above tuple.
2. To decide that $Cool_Paper$ is an erroneous record (i.e., that $Cool_Paper$ is not a paper at all) and must be deleted. In this case, deleting the tuple $Paper(Cool_Paper)$ causes further problems, namely that all the tuples in $Cites$ that involve $Cool_Paper$ must be deleted (otherwise, σ_3, σ_4 will be violated). Continuing recursively, we realize that the deletion of $Cites(DB_Paper, Cool_Paper)$ would further cause σ_7 to be violated, because now DB_Paper does not cite any other paper, causing further side-effects, which can be resolved as above.

Author's personal copy

In the above example, it makes sense to follow the first solution, as it causes a much smaller set of side-effects; this is in accordance to the Principle of Minimal Change.

Note that the above problem could be resolved via user interaction, i.e., by informing the user about the violation that his change is about to cause and asking him to resolve it manually; this is the approach usually followed by ontology editors and other interactive tools [42,58]. However, this could quickly lead to user frustration, as well as poor update results, especially in case of complex datasets and/or constraints where the user is unable to understand or predict the full ramifications of his choices.

In the following sections, we will elaborate in more details the above process, and formalize our approach for determining the change result.

3 Problem abstraction

3.1 Knowledge bases and change requests

As explained above, we will use the relational model as an abstraction mechanism for our framework. Thus, knowledge will be expressed using expressions of the form $p(\vec{a})$, where p is a predicate symbol and $\vec{a}$ is a vector of constants $(a_1, \ldots, a_n)$; we will call these expressions *ground facts*. Expressions of the form $\neg p(\vec{a})$ will be called *negated ground facts*. For simplicity of notation, we will often use the symbols $g, \neg g$ to denote (negated) ground facts.

A *Knowledge Base (KB)* is a finite set of positive ground facts. Under the relational notation, the KB of the example of Sect. 2 is: $\mathcal{K} = \{Paper(DB_Paper), Paper(RDF_Paper), Paper(Cool_Paper), PublishedIn(DB_Paper, KAIS), PublishedIn(RDF_Paper, KAIS), PublishedIn(Cool_Paper, Cool_Journal), Journal(KAIS), Journal(Cool_Journal), Cites(DB_Paper, Cool_Paper), Cites(RDF_Paper, DB_Paper), Cites(Cool_Paper, RDF_Paper)\}$.

Per the standard relational semantics, for a KB $\mathcal{K}$ and a ground fact g, it holds that $\mathcal{K} \vdash g$ iff $g \in \mathcal{K}$. Thus, in the above example we have: $\mathcal{K} \vdash Journal(KAIS)$ and $\mathcal{K} \nvdash Cites(DB_Paper, RDF_Paper)$. This is in accordance with the closed world semantics adopted by relational databases, but also by standard RDF/S query languages [54]. The semantics can be easily extended to logical formulas, e.g., $\mathcal{K} \vdash g \rightarrow g'$ iff $g \notin \mathcal{K}$ or $g' \in \mathcal{K}$. For a set of formulas Φ, $\mathcal{K} \vdash \Phi$ iff $\mathcal{K} \vdash \phi$ for all $\phi \in \Phi$.

A *change request* $\mathcal{C}$ is a request to add and/or remove information (ground facts) to/from the KB and will be modeled as a finite set of (possibly negated) ground facts. Positive ground facts correspond to additions, whereas negated ones correspond to deletions; for example $\mathcal{C} = \{g_1, g_2, \neg g_3\}$ is a request to add g_1, g_2 and remove g_3. In the motivating example, the request to remove $PublishedIn(Cool_Paper, Cool_Journal)$ is modeled as: $\mathcal{C} = \{\neg PublishedIn(Cool_Paper, Cool_Journal)\}$.

3.2 Integrity constraints and change operators

Integrity constraints will be represented using DEDs [20], in particular the slightly richer class $DED^{\neq}$, which also supports inequality axioms. Formulas in $DED^{\neq}$ are of the form $\forall \vec{x}\, \hat{p}(\vec{x}) \rightarrow \vee_{i=1,\ldots,n} \exists \vec{y}_i \hat{q}_i(\vec{x}, \vec{y}_i)$, where $\vec{x}, \vec{y}_i$ are tuples of variables and $\hat{p}(\vec{x}), \hat{q}_i(\vec{y}_i)$ are conjunctions of ground facts and (in)equality atoms of the form $(w = w')$, $(w \neq w')$, where w, w' are variables or constants (note that $\hat{p}$ may be the empty conjunction).

The class of constraints $DED^{\neq}$ is expressive enough for capturing several types of constraints, including foreign key constraints (see, e.g., σ_5 in Sect. 2), primary key

Author's personal copy

constraints (e.g., σ_6), inclusion dependencies (e.g., $\sigma_1, \sigma_2, \sigma_3, \sigma_4$), transitive, symmetric or acyclic relations, cardinality constraints and others [20]; note that this covers the needs that have appeared in the related literature in both the ontological and the relational setting (e.g., [1,2,11,43,49,55,60]) as well as the needs of our RDF/S modelling (see Sect. 9.3). Moreover, DEDs will prove suitable for constructing a convenient mechanism for detecting and repairing invalidities.

A *validity model* is a finite set of DED constraints $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. Given a constraint σ and an assignment of the variables in $\vec{x}$ to constants $\vec{a}$, we define the *grounded instance of* (or simply *instance of*) σ with respect to $\vec{a}$ (denoted by $\sigma(\vec{a})$), to be the formula that is produced by replacing all variables from $\vec{x}$ in σ by their corresponding assignment in $\vec{a}$. Returning to our motivating example, $\sigma_3(Cool_Paper, RDF_Paper) = Cites(Cool_Paper, RDF_Paper) \rightarrow Paper(Cool_Paper)$. As we will see later, the detection and resolution of violations will be based on constraint instances.

Regarding the instances of constraints that contain (in)equality axioms, a special note is necessary. In this work, we employ the *unique name assumption*, which means that different constants correspond (by default) to different real-world entities. This assumption is suitable for the RDF/S context, where different URIs correspond to different resources (same for literals). This assumption means that, given two different constants a_1, a_2, the (in)equality axiom $a_1 = a_2$ ($a_1 \neq a_2$) always evaluates to $false$ ($true$), whereas $a_1 = a_1$ ($a_1 \neq a_1$) obviously evaluates to $true$ ($false$), respectively.

Thanks to this assumption, equality axioms can always be eliminated from constraint instances. For example, $\sigma_6(Cool_Paper, KAIS, Cool_Journal) = PublishedIn(Cool_Paper, KAIS) \wedge PublishedIn(Cool_Paper, Cool_Journal) \rightarrow false$, because $KAIS = Cool_Journal$ evaluates to false; thus, $\sigma_6(Cool_Paper, KAIS, Cool_Journal)$ is equivalent to $\neg PublishedIn(Cool_Paper, KAIS) \vee \neg PublishedIn(Cool_Paper, Cool_Journal)$. On the other hand, $\sigma_6(Cool_Paper, KAIS, KAIS) = true$, because $KAIS = KAIS$ evaluates to false.

This is not entirely true for inequality axioms: we can eliminate the inequality axioms that involve universally quantified variables and/or constants, but not those that involve existentially quantified variables. For example, $\sigma_7(Cool_Paper) = Paper(Cool_Paper) \rightarrow \exists y(Cites(Cool_Paper, y) \wedge (Cool_Paper \neq y))$.

We say that an RDF/S KB $\mathcal{K}$ *satisfies* the (instance of the) constraint σ ($\sigma(\vec{a})$), iff $\mathcal{K} \vdash \sigma$ ($\mathcal{K} \vdash \sigma(\vec{a})$). Similarly, $\mathcal{K}$ *satisfies* a validity model $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ (denoted by $\mathcal{K} \vdash \Sigma$) iff $\mathcal{K} \vdash \sigma_i$ for $i = 1, \ldots, n$. A constraint instance (or constraint, or validity model) is *violated* by a KB $\mathcal{K}$ iff it is not satisfied. A KB that satisfies a validity model Σ is called a *valid KB* with respect to Σ. It is trivial to note that, for a KB $\mathcal{K}$, $\mathcal{K} \vdash \sigma$ iff $\mathcal{K} \vdash \sigma(\vec{a})$ for all $\vec{a}$; similarly, $\mathcal{K} \vdash \Sigma$ iff $\mathcal{K} \vdash \sigma(\vec{a})$ for all $\sigma \in \Sigma$ and $\vec{a}$.

A *change operator* $\bullet$ is an operator that takes in the input a KB and a change request and returns a new KB. In this work we are interested in *rational change operators*, which satisfy the principles of Success, Validity and Minimal Change, introduced in the next sections.

4 Principle of success

The Principle of Success informally states that the change request should be implemented. Thus, given a change request $\mathcal{C} = \{g_1, \ldots, g_n, \neg g'_1, \ldots, \neg g'_m\}$ and a KB $\mathcal{K}$, a change operator $\bullet$ *respects the Principle of Success* iff $g_i \in \mathcal{K} \bullet \mathcal{C}$, $g'_j \notin \mathcal{K} \bullet \mathcal{C}$ for all $i = 1, \ldots, n$, $j = 1, \ldots, m$. This condition can be more compactly (and equivalently) formulated as: $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{C}$. Obviously, the above condition cannot be satisfied if the change request requires both the

Author's personal copy

addition and the deletion of the same ground fact, i.e., if $g, \neg g \in \mathcal{C}$, so we restrict our attention to change requests for which this does not hold. It is easy to devise operators that respect the Principle of Success. The following operator will be of special interest for the following:

Definition 4.1 The *raw application* of a change request $\mathcal{C}$ upon a KB $\mathcal{K}$ (denoted by $\mathcal{K}+\mathcal{C}$) is defined as: $\mathcal{K}+\mathcal{C} = (K \cup \{g | g \in \mathcal{C}\}) \setminus \{g | \neg g \in \mathcal{C}\}$.

As an example, taking the KB $\mathcal{K}$ and change request $\mathcal{C}$ of Sect. 2 (see Sect. 3.1 for their representation using ground facts), we get: $\mathcal{K}_1 = \mathcal{K}+\mathcal{C} = \{Paper(DB_Paper), Paper(RDF_Paper), Paper(Cool_Paper), PublishedIn(DB_Paper, KAIS), PublishedIn(RDF_Paper, KAIS), Journal(KAIS), Journal(Cool_Journal), Cites(DB_Paper, Cool_Paper), Cites(RDF_Paper, DB_Paper), Cites(Cool_Paper, RDF_Paper)\}$.

It is trivial to show that the operator $+$ respects the Principle of Success. In fact, the raw application is the straightforward way to apply a change request when no integrity constraints are present. However, in the case of associated constraints, this naive way of applying a change request gives us no guarantees that the result will be valid (e.g., $\mathcal{K}_1$ violates $\sigma_5(Cool_Paper)$). In the next section, we will refine $+$ to satisfy the Principle of Validity as well.

5 Principle of validity

The Principle of Validity states that the resulting KB should be valid. Formally, given a validity model Σ, a KB $\mathcal{K}$ and a change request $\mathcal{C}$, a change operator respects the Principle of Validity iff $\mathcal{K} \bullet \mathcal{C} \vdash \Sigma$. To develop a change operator that satisfies the Principles of Success and Validity, we will start with the raw application operator $(+)$ and adapt it to take into account the validity model. The underlying idea is that, given a KB $\mathcal{K}$ and a change request $\mathcal{C}$, we consider $\mathcal{K}_1 = \mathcal{K}+\mathcal{C}$; if $\mathcal{K}_1$ is not valid, we apply additional changes to it, called *side-effects*, to make it valid. To do so, we must first identify each *invalidity*, i.e., each constraint instance $\sigma(\vec{a})$ for which $\mathcal{K}_1 \nvdash \sigma(\vec{a})$, and determine how it can be resolved, i.e., identify the possible side-effect(s) that could be applied upon $\mathcal{K}_1$ to guarantee that the result will satisfy $\sigma(\vec{a})$. Repeating this process for all invalidities, we will eventually reach a KB $\mathcal{K}_n$ which is valid. In the following subsections, we describe the above process in detail.

5.1 Detection and resolution of invalidities

The DED form of constraints and the semantics described in Sect. 3.1 allow both the easy detection of an invalidity, and the determination of all possible options for repairing it, using just syntactical manipulations over the violated constraint instances.

This can be easily seen with an example: let us take the constraint instance $\sigma_1(Cool_Paper, Cool_Journal)$ from Sect. 2. Then, per our semantics, given some KB $\mathcal{K}$, $\mathcal{K} \vdash \sigma_1(Cool_Paper, Cool_Journal)$ iff $PublishedIn(Cool_Paper, Cool_Journal) \notin \mathcal{K}$ or $Paper(Cool_Paper) \in \mathcal{K}$. If $\mathcal{K} \nvdash \sigma_1(Cool_Paper, Cool_Journal)$, then none of the above conditions holds, so we can resolve this invalidity by making one or more of those conditions true, i.e., by removing $PublishedIn(Cool_Paper, Cool_Journal)$ from $\mathcal{K}$, or by adding $Paper(Cool_Paper)$ to $\mathcal{K}$. For the KB of Sect. 2, $\mathcal{K} \vdash \sigma_1(Cool_Paper, Cool_Journal)$ because $Paper(Cool_Paper) \in \mathcal{K}$.

Author's personal copy

The idea can be easily extended to DEDs that contain existential quantifiers. For the case of $\sigma_5(Cool_Paper), \mathcal{K} \vdash \sigma_5(Cool_Paper)$ iff $Paper(Cool_Paper) \notin \mathcal{K}$ or there is some constant b such that $PublishedIn(Cool_Paper, b) \in \mathcal{K}$. As before, if $\mathcal{K} \nvdash \sigma_5(Cool_Paper)$ then the invalidity can be resolved by removing $Paper(Cool_Paper)$ from $\mathcal{K}$, or by adding $PublishedIn(Cool_Paper, b)$ to $\mathcal{K}$ for some constant b. Note that when existential quantifiers are involved, we have many potential side-effects, namely one for each different constant b. In our motivating example, $\mathcal{K} \vdash \sigma_5(Cool_Paper)$ because $PublishedIn(Cool_Paper, Cool_Journal) \in \mathcal{K}$, but $\mathcal{K}_1 \nvdash \sigma_5(Cool_Paper)$ because $Paper(Cool_Paper) \in \mathcal{K}_1$ and there is no tuple of the form $PublishedIn(Cool_Paper, b) \in \mathcal{K}_1$ for any b. The two resolution options described in Sect. 2 (adding $PublishedIn(Cool_Paper, KAIS)$ or removing $Paper(Cool_Paper)$), are in accordance to the options discussed above.

Finally, note that equality axioms in a constraint need not be considered, because they are eliminated in each of its instances (see Sect. 3.2). Inequality axioms can appear inside existential quantifiers, in which case they simply overrule some of the options. For example, if $\sigma_7(Cool_Paper)$ is violated, we can remove $Paper(Cool_Paper)$ or we can add $Cites(Cool_Paper, b)$ for some constant $b \neq Cool_Paper$.

The above process determines the *different potential side-effects* that can be applied upon a KB in order to resolve a particular invalidity. Each of these side-effects is a set $\mathcal{S}$ of additions or deletions of ground facts, so $\mathcal{S}$ will be formally modeled as a set of possibly negated ground facts; note that this set is usually a singleton (e.g., in our examples it is always a singleton), but in some cases (depending on the form of the constraint) it could contain more than one additions/deletions.

We define the *resolution set* of a constraint instance $\sigma(\vec{a})$ (denoted by $Res(\sigma(\vec{a}))$) to be the set of all $\mathcal{S}_i$ that can be used to resolve a violation of the constraint instance. These are easy to identify by the constraint's syntax: we first transform the constraint instance into its disjunctive normal form,[3] and use each disjunct to form a $\mathcal{S}_i$, by putting all conjuncts in said disjunct in $\mathcal{S}_i$. For example, $Res(\sigma_1(Cool_Paper, Cool_Journal)) = \{ \{\neg PublishedIn(Cool_Paper, Cool_Journal)\}, \{Paper(Cool_Paper)\} \}$, because $\sigma_1(Cool_Paper, Cool_Journal)$ can be written as: $\neg PublishedIn(Cool_Paper, Cool_Journal) \vee Paper(Cool_Paper)$.

Based on the analysis above, it is easy to show the following propositions:[4]

Proposition 5.1 *Consider a KB $\mathcal{K}$ and a constraint instance $\sigma(\vec{a})$. Then $\mathcal{K} \vdash \sigma(\vec{a})$ iff there exists some $\mathcal{S} \in Res(\sigma(\vec{a}))$ such that $\mathcal{K} \vdash \mathcal{S}$.*

Proposition 5.2 *Consider a KB $\mathcal{K}$ and a constraint instance $\sigma(\vec{a})$. If $\mathcal{K} \nvdash \sigma(\vec{a})$ then for any $\mathcal{S} \in Res(\sigma(\vec{a}))$ it holds that $\mathcal{K} + \mathcal{S} \vdash \sigma(\vec{a})$. Moreover, this is the minimal way (with respect to $\subseteq$) to resolve this invalidity, i.e., for any $\mathcal{S}'$ such that $\mathcal{K} + \mathcal{S}' \vdash \sigma(\vec{a})$, there is some $\mathcal{S} \in Res(\sigma(\vec{a}))$ such that for all (possibly negated) ground facts $g \in \mathcal{S}$ it holds that $\mathcal{K} \vdash g$ or $g \in \mathcal{S}'$.*

Proposition 5.1 shows how to detect an invalidity. Proposition 5.2 shows how a detected invalidity can be resolved, and verifies that our resolution method is both correct and complete, in the sense that any other side-effects used to resolve a given invalidity would have to include at least the side-effects in some $\mathcal{S} \in Res(\sigma(\vec{a}))$ (except from the ground facts that are already implied by $\mathcal{K}$).

[3] http://en.wikipedia.org/wiki/Disjunctive_normal_form.

[4] Proofs for all propositions can be found in the Appendix.

Author's personal copy

In practice, it makes sense to assume that the KB being changed is valid to begin with. Under this assumption, any invalidities in $\mathcal{K}+\mathcal{C}$ will be introduced due to the raw application of $\mathcal{C}$, so it limits the constraint instances that need to be considered for violation. In the example of Sect. 2, the change request $\mathcal{C} = \{\neg PublishedIn(Cool_Paper, Cool_Journal)\}$ could only cause the violation of $\sigma_5(Cool_Paper)$, i.e., this is the only constraint instance that needs to be checked for violation after the raw application of $\mathcal{C}$. Formally:

Proposition 5.3 *Consider a valid KB $\mathcal{K}$, a change request $\mathcal{C}$, and a constraint instance $\sigma(\vec{a})$. If $\mathcal{K}+\mathcal{C} \nvdash \sigma(\vec{a})$, then there is some $\mathcal{S} \in Res(\sigma(\vec{a}))$ and some (positive or negative) ground fact g such that $g \in \mathcal{C}$ and $\neg g \in \mathcal{S}$.*

Note that the condition of Proposition 5.3 is necessary but not sufficient for a constraint violation.

5.2 Refining raw application

Now the process outlined in the beginning of this section can be described as follows: given a valid KB $\mathcal{K}$ and a change request $\mathcal{C}$, we first compute $\mathcal{K}_1 = \mathcal{K}+\mathcal{C}$. Then, we select one constraint instance that is violated by the result (say $\sigma(\vec{a})$) and one possible set of side-effects that resolve it (say $\mathcal{S} \in Res(\sigma(\vec{a}))$), and apply it on $\mathcal{K}_1$, to get $\mathcal{K}_2 = \mathcal{K}_1 + \mathcal{S}$. Then we repeat the process, selecting another violated constraint instance and a corresponding set of side-effects, until reaching a valid KB, which is returned as the result.

There are certain things that need to be noted in the above process. First, we make the assumption that $\mathcal{K}$ is valid, so as to use Proposition 5.3 and check a small number of constraint instances for violation. Second, more than one violations can be caused by a single change request, and side-effects applied in previous steps may also cause violations of their own. Therefore, at each step, we should check the constraints that are possibly violated by the change request, as well as by all the side-effects that have so far been applied. This cascading effect (side-effects causing side-effects of their own), may lead to complicated sequences of side-effects and resolutions which are difficult to foresee in the general case.

Another important point is that the applied side-effects should not conflict with the change request itself. In the example of Sect. 2, the violation of $\sigma_5(Cool_Paper)$ could also be resolved by the addition of $PublishedIn(Cool_Paper, Cool_Journal)$, and this is also included in $Res(\sigma_5(Cool_Paper))$; nevertheless, this option should be ignored, because it directly contradicts the change request to remove $PublishedIn(Cool_Paper, Cool_Journal)$, so considering it would violate the Principle of Success. A similar comment is that one should also not apply side-effects that conflict with previously applied side-effects, as this could cause violations to previously resolved constraint instances, eventually leading the process into an infinite loop.

Another problem with the refined raw application is that it does not specify the order in which the violated constraint instances will be considered, nor a selection process for the side-effects to apply (as there will usually be more than one options). The former issue will be addressed in Sect. 8, where we will show that the presented process satisfies the required principles, regardless of order. The latter issue is related to the Principle of Minimal Change and will be resolved using a *selection mechanism*, to be described in Sect. 6.

One could devise change requests for which the above process cannot lead to any result. For example, the change request: $\mathcal{C}_{bad} = \{PublishedIn(DB_Paper, Cool_Journal), \neg Paper(DB_Paper)\}$ specifies that DB_Paper was published in $Cool_Journal$, but, at the same time, requires the deletion of DB_Paper. It is obvious that any operator that respects the Principle of Success would lead to a result containing $PublishedIn(DB_Paper,$

Author's personal copy

$Cool_Journal$), and not containing $Paper(DB_Paper)$; such a result cannot respect the Principle of Validity (as it does not satisfy $\sigma_1(DB_Paper, Cool_Journal)$). The case studied in Sect. 4 where there is some ground fact g such that $g, \neg g \in \mathcal{C}$ is a similar case. Change requests with this property are called *infeasible*. Formally:

Definition 5.1 Consider a validity model Σ. A change request $\mathcal{C}$ is called *feasible* with respect to Σ iff there is some valid KB $\mathcal{K}$ such that $\mathcal{K} \vdash \mathcal{C}$. It is called *infeasible* otherwise.

6 Principle of minimal change

The Principle of Minimal Change informally states that the result of a change operation should be as close as possible to the original KB. Thus, to formally define this principle, we need to (a) define the notion of "distance" between KBs, and, (b) devise a way to "compare distances", in order to determine the preferred result. Given these tools, we can then modify the process described in Sect. 5.2 so that it does not select the side-effects to apply during each resolution randomly, but in such a way that the accumulated impact upon the KB is minimal (according to the distance and the comparison method discussed). The following subsections explain in detail these notions.

6.1 Deltas

The most straightforward way to determine the distance between KBs is using *deltas*, such as those introduced in [7,21,53,62,64]. A delta is actually a description of the differences between two KBs. In our context, we formalize deltas as sets of (possibly negated) ground facts:

Definition 6.1 Consider two KBs $\mathcal{K}_1, \mathcal{K}_2$. We define the *delta* between $\mathcal{K}_1, \mathcal{K}_2$, denoted by $\Delta(\mathcal{K}_1, \mathcal{K}_2)$ (or simply Δ, when $\mathcal{K}_1, \mathcal{K}_2$ are irrelevant, or obvious from the context) as follows: $\Delta(\mathcal{K}_1, \mathcal{K}_2) = \{g | g \in \mathcal{K}_2 \setminus \mathcal{K}_1\} \cup \{\neg g | g \in \mathcal{K}_1 \setminus \mathcal{K}_2\}$

It is trivial to see that $\Delta(\mathcal{K}_1, \mathcal{K}_2) = \{g | \mathcal{K}_2 \vdash g \; and \; \mathcal{K}_1 \nvdash g\} \cup \{\neg g | \mathcal{K}_2 \vdash \neg g \; and \; \mathcal{K}_1 \nvdash \neg g\}$ and that $\mathcal{K}_1 + \Delta(\mathcal{K}_1, \mathcal{K}_2) = \mathcal{K}_2$. Thus, $\Delta(\mathcal{K}_1, \mathcal{K}_2)$ contains exactly the changes that must be raw applied upon $\mathcal{K}_1$ to get $\mathcal{K}_2$. In this sense, $\Delta(\mathcal{K}_1, \mathcal{K}_2)$ captures accurately the notion of distance.

6.2 Comparing deltas

Our next step is to find a way to compare deltas. Consider a KB $\mathcal{K}$, a change request $\mathcal{C}$, and a set of deltas which represent the candidate sets of changes that could be applied to respect the Principles of Success and Validity; the comparison is used to determine which delta represents the minimal change, i.e., which of the candidate deltas is preferred for application. To do this, we will use an ordering $\leq_{\mathcal{K}}$ between deltas, with the intuitive meaning that $\Delta_1 \leq_{\mathcal{K}} \Delta_2$ iff Δ_1 represents a set of changes that are preferable to apply upon $\mathcal{K}$, compared to Δ_2. Recall that this comparison is needed to allow us to make the optimal set of choices for the side-effects to apply during the resolutions of the violated constraint instances. The ordering depends on $\mathcal{K}$; this is necessary, because some changes (ground facts) in a delta may be considered more, or less, important, depending on the contents of the KB itself. For example, in the RDF/S context, the removal of a class that is low in the class hierarchy may be preferable than the removal of a class that is higher in the hierarchy (see also Sect. 9.4).

Author's personal copy

Some properties need to be imposed on said ordering for it to be suitable as a selection mechanism. First, we require it to be *total*, so that all deltas are comparable. Second, we impose *antisymmetry*, to guarantee that there will not be more than a single minimum. Third, we require the order to be *wellfounded*, i.e., that there is always a minimum. These three properties together guarantee that any set of deltas has a single minimum, so we will always be able to determine the most preferable delta out of a set of deltas.

The fourth required property is *transitivity*, dictated by the intuitive fact that if Δ_1 is preferred over Δ_2, and Δ_2 is preferred over Δ_3, then Δ_1 should be preferred over Δ_3. Fifth, we impose the *monotonicity* property, which intuitively states that adding ground facts to a delta cannot make it most preferable, i.e., if $\Delta_1 \subseteq \Delta_2$ then $\Delta_1 \preceq_{\mathcal{K}} \Delta_2$; this is dictated by our intuition behind the Principle of Minimal Change.

Combining the above properties, we conclude that $\preceq_{\mathcal{K}}$ should be a *well-order* that satisfies the monotonicity property; such orderings will be called *change-generating*:

Definition 6.2 Consider a KB $\mathcal{K}$. An ordering $\preceq_{\mathcal{K}}$ is called *change-generating* iff for all $\Delta_1, \Delta_2, \Delta_3$:

- **Totality**: $\Delta_1 \preceq_{\mathcal{K}} \Delta_2$ or $\Delta_2 \preceq_{\mathcal{K}} \Delta_1$.
- **Antisymmetry**: $\Delta_1 \preceq_{\mathcal{K}} \Delta_2$ and $\Delta_2 \preceq_{\mathcal{K}} \Delta_1$ implies $\Delta_1 = \Delta_2$.
- **Wellfoundedness**: for any non-empty set of deltas $\mathcal{Z}$, there is some $\Delta \in \mathcal{Z}$ such that there is no $\Delta' \in \mathcal{Z}$ for which $\Delta' <_{\mathcal{K}} \Delta$.
- **Transitivity**: $\Delta_1 \preceq_{\mathcal{K}} \Delta_2$ and $\Delta_2 \preceq_{\mathcal{K}} \Delta_3$ implies $\Delta_1 \preceq_{\mathcal{K}} \Delta_3$.
- **Monotonicity**: $\Delta_1 \subseteq \Delta_2$ implies $\Delta_1 \preceq_{\mathcal{K}} \Delta_2$.

Note that a simple cardinality-based comparison is not change-generating, because it is not wellfounded. However, it is easy to show that a change-generating order can always be defined; for example, we could use cardinality comparison, and apply some wellorder in case of a tie (wellorders can always be defined, per the wellordering theorem[5]), e.g., by assigning priorities on the addition/removal of tuples from certain relations. An example of a change-generating ordering that is useful for the RDF/S context will appear in Sect. 9.4.

Now consider a set of deltas $\mathcal{Z}$ which are candidates for application upon a KB $\mathcal{K}$ in response to a change request $\mathcal{C}$; then, as explained above, a change-generating ordering $\preceq_{\mathcal{K}}$ allows us to select the most preferable delta to apply upon $\mathcal{K}$; such a delta will be denoted by $min_{\mathcal{K}}(\mathcal{Z})$. Similarly, a family of change-generating orderings $\preceq = \{\preceq_{\mathcal{K}} | \mathcal{K}$: valid KB, $\preceq_{\mathcal{K}}$: change-generating$\}$ allows us to select the most preferable delta to apply upon each valid KB and will be called a *selection mechanism*.

Note that the selection mechanism selects deltas, rather than resulting KBs; this is reasonable, because the distance of two KBs $\mathcal{K}, \mathcal{K}'$ is determined by $\Delta(\mathcal{K}, \mathcal{K}')$. However, our ultimate goal is to select the KB whose delta (distance) from the original KB is minimal. For this reason, we will often abuse notation and write $\mathcal{K}_1 \preceq_{\mathcal{K}} \mathcal{K}_2$ to denote that $\Delta(\mathcal{K}, \mathcal{K}_1) \preceq_{\mathcal{K}} \Delta(\mathcal{K}, \mathcal{K}_2)$; similarly, for a set of KBs Ω, we write $min_{\mathcal{K}}(\Omega)$ to denote the KB $\mathcal{K}_0$ for which $\Delta(\mathcal{K}, \mathcal{K}_0) = min_{\mathcal{K}}(\{\Delta(\mathcal{K}, \mathcal{K}') | \mathcal{K}' \in \Omega\})$.

6.3 Formalizing the principle of minimal change

The Principle of Minimal Change dictates that a change operator should select the KB which is closest to the original KB, over all other potential results, i.e., results that satisfy other conditions related to the change operator (in our case, the Principles of Success and Validity). So, given a KB $\mathcal{K}$ and a change request $\mathcal{C}$, and supposing a set Ω of potential results, the Principle of Minimal Change states that $\mathcal{K} \bullet \mathcal{C} = min_{\mathcal{K}}(\Omega)$.

[5] http://en.wikipedia.org/wiki/Well-ordering_theorem.

Author's personal copy

To modify the process described in Sect. 5.2 to respect the Principle of Minimal Change, we need to make sure that the resolution selected in each step will lead to the preferred delta. Note that we require the delta as a whole to be preferred, not each resolution in isolation, therefore a greedy strategy selecting the locally preferred resolution (i.e., the set of side-effects $\mathcal{S} \in Res(\sigma(\vec{a}))$ that is preferred according to $\preceq$) would not work. Instead, we need to consider all possible options independently to form the set of candidate deltas, $\mathcal{Z}$, and then select the preferred one (according to $\preceq$). Details on this process will appear in Sect. 8.

7 Rational change operators

Rational change operators are those which satisfy the Principles of Success, Validity and Minimal Change. Formally:

Definition 7.1 Consider a validity model Σ and a selection mechanism $\preceq$. A change operator $\bullet$ will be called *rational* with respect to Σ, $\preceq$ iff for all valid KBs $\mathcal{K}$ and feasible change requests $\mathcal{C}$ it holds that $\mathcal{K} \bullet \mathcal{C} = min_{\mathcal{K}}(\Omega)$, where $\Omega = \{\mathcal{K}' | \mathcal{K}' \vdash \mathcal{C}, \mathcal{K}' \vdash \Sigma\}$. Moreover, whenever $\mathcal{K}$ is invalid or $\mathcal{C}$ is infeasible, it holds that $\mathcal{K} \bullet \mathcal{C} = \mathcal{K}$.

Note that infeasible change requests have been exempted from the requirement to satisfy the principles. Also, the input KB is assumed to be valid (this is necessary to use Proposition 5.3). According to Definitions 5.1 and 6.2 a rational change operator always exists. In addition, it is unique, because by the definition of $\preceq$, there is a unique minimum:

Proposition 7.1 *For any given validity model Σ and selection mechanism $\preceq$, there is a unique rational change operator.*

A rational change operator cannot be easily defined in a declarative way, because one needs to take into account all possible combinations of KBs and change requests, and all possible invalidities that a change request could cause on any given KB. On the other hand, Proposition 7.1 shows that each selection mechanism uniquely identifies a rational change operator, so one could indirectly use this feature to define a rational change operator by defining a selection mechanism. A procedural way to define rational change operators will be explained in Sect. 8 below.

8 Algorithm

In this section, we will formalize and elaborate on the algorithm sketch provided in Sect. 6.3. In a nutshell, we first raw apply $\mathcal{C}$ upon $\mathcal{K}$, and check for constraint instance violations. Then, for each such violation, $\sigma(\vec{a})$, we explore *all possible* resolutions, i.e., all different $\mathcal{S} \in Res(\sigma(\vec{a}))$. Each different resolution spawns a new recursive branch, and the above process is repeated recursively. Eventually, a multitude of different sets of resolutions are generated, each corresponding to a delta; we compare those deltas using $\preceq$, and return the preferred (minimal) one.

More specifically, the algorithm will be based on a recursive function, $Change$ (see Algorithm 1), which takes as input the set $\mathcal{C}_{\text{rem}}$ of ground facts that remain to be applied to the KB, a valid KB $\mathcal{K}_0$, and the most preferred delta (Δ_{pref}) that has been calculated so far for $\mathcal{C}_{\text{rem}}$, so as to stop exploring potential solutions if they are already less preferred than Δ_{pref} (exploiting the monotonicity and transitivity properties of $\preceq_{\mathcal{K}}$). The $Change$ function returns

Author's personal copy

Algorithm 1: Change Function ($Change(\mathcal{C}_{rem}, \mathcal{K}_0, \Delta_{pref})$)

1: **if** there is a ground fact g such that $g, \neg g \in \mathcal{C}_{rem}$ or $\Delta_{pref} \leq_{\mathcal{K}_0} \Delta(\mathcal{K}_0, \mathcal{K}_0 + \mathcal{C}_{rem})$ **then**
2: **return** Δ_∞
3: **end if**
4: **if** $\mathcal{K}_0 + \mathcal{C}_{rem}$: valid **then**
5: **return** $\Delta(\mathcal{K}_0, \mathcal{K}_0 + \mathcal{C}_{rem})$
6: **end if**
7: Take a (possibly negated) ground fact $g \in \mathcal{C}_{rem}$ such that there exists a constraint instance $\sigma(\vec{a})$ for which $\neg g \in \mathcal{S}$ for some $\mathcal{S} \in Res(\sigma(\vec{a}))$ and $\mathcal{K}_0 + \mathcal{C}_{rem} \nvdash \sigma(\vec{a})$
8: **if** there is at least one $\mathcal{S}' \in Res(\sigma(\vec{a}))$ such that $\mathcal{S}' \neq \mathcal{S}$ **then**
9: **for all** $\mathcal{S}' \in Res(\sigma(\vec{a}))$, $\mathcal{S}' \neq \mathcal{S}$ **do**
10: $\Delta_{pref} := min_{\mathcal{K}_0}(\{\Delta_{pref}, \texttt{Change}(\mathcal{C}_{rem} \cup \mathcal{S}', \mathcal{K}_0, \Delta_{pref})\})$
11: **end for**
12: **else**
13: **return** Δ_∞
14: **end if**
15: **return** Δ_{pref}

the preferred set of ground facts that are required to apply $\mathcal{C}_{\text{rem}}$ to $\mathcal{K}_0$; this set contains the ground facts in $\mathcal{C}_{\text{rem}}$, plus all their side-effects, but does not contain any ground facts already implied by $\mathcal{K}_0$. For implementation purposes, we assume a special delta, denoted by Δ_∞, which is used to mark unacceptable resolution branches and is assumed to be less preferable than any other delta.

In the first call to $Change$, $\mathcal{C}_{\text{rem}}$ is set equal to the original change request $\mathcal{C}$, $\mathcal{K}_0$ is the original KB $\mathcal{K}$ (and does not change between the recursive calls) and Δ_{pref} is set to Δ_∞, so as to allow any solution to constitute the first potentially preferred result; hence, Δ_{pref} is either equal to Δ_∞ (for as long as no complete solution has been found), or equal to the most preferred of the complete solutions that have been found so far. So, the first call ($Change(\mathcal{C}, \mathcal{K}, \Delta_\infty)$) returns the preferred set of effects and side-effects (delta) for the original change request upon $\mathcal{K}$. Note that this call should be made through another procedure that will check whether $\mathcal{K}$ is valid and will, upon return of $Change$, apply its output to $\mathcal{K}$ (using raw application), unless the output is Δ_∞ (denoting that the original update was infeasible—see also Proposition 8.1). This procedure is simple and omitted.

$Change$ relies on a recursive process. In line 1 we check whether it makes sense to continue exploring this recursive branch. If $\mathcal{C}_{\text{rem}}$ is contradictory, then the previously selected resolution options contradict each other, so the current resolution branch must be dropped. Similarly, if the cost of applying $\mathcal{C}_{\text{rem}}$ is larger than Δ_{pref}, then the current branch cannot lead to a preferred solution, so it need not be explored further. In both cases, the branch is rejected by returning Δ_∞ (see line 2). If, on the other hand, $\mathcal{K}_0 + \mathcal{C}_{\text{rem}}$ is valid (line 4), then a solution has been reached, so the recursion stops, returning the effects and side-effects found ($\Delta(\mathcal{K}_0, \mathcal{K}_0 + \mathcal{C}_{\text{rem}})$).

If none of the above is true, then there are still violations that need to be considered. One of them is arbitrarily selected in line 7 (say $\sigma(\vec{a})$); note how Proposition 5.3 is used here to avoid checking all constraint instances. To resolve the given violation, we need to take one or more $\mathcal{S}' \in Res(\sigma(\vec{a}))$, different from $\mathcal{S}$, i.e., different from the one that allowed us to identify the violation and is part of $\mathcal{C}_{\text{rem}}$. If there are no such $\mathcal{S}'$, then we return Δ_∞ in line 13 (this can happen only when $Res(\sigma(\vec{a}))$ is a singleton). If such an $\mathcal{S}'$ exists, then it is added to our side-effects. Given that we cannot know a priori which $\mathcal{S}'$ will lead to the preferred solution, we consider them all separately: each $\mathcal{S}'$ is added to the current $\mathcal{C}_{\text{rem}}$, and a new instance of $Change$ (in effect, a new recursive subtree) is spawned to calculate the consequences of this

Author's personal copy

choice, i.e., $\mathcal{S}'$ (lines 9-11). The returned delta of each alternative is compared with the best that has been found so far (Δ_{pref}), and the next $\mathcal{S}'$ is considered. Eventually, one branch will return the preferred delta, which will be recursively propagated to the caller through line 15.

In all comparisons, any delta is preferable over Δ_∞; if any two alternatives are equally preferred, then they must be equal (antisymmetry). Upon termination, the output of the algorithm (say Δ_{out}) is the delta that should be raw applied upon $\mathcal{K}$ (to produce the result, $\mathcal{K} + \Delta_{\text{out}}$), unless Δ_∞ is returned, in which case the original change is infeasible.

Example 8.1 Let us see how this algorithm would handle the example of Sect. 2. Initially, $Change$ will be called with parameters: $\mathcal{C}_{\text{rem}} = \{\neg PublishedIn(Cool_Paper, Cool_Journal)\}$, $\mathcal{K}_0 = \mathcal{K}$ and $\Delta_{pref} = \Delta_\infty$. The check of line 1 will fail, so the algorithm will proceed to compute $\mathcal{K}_0 + \mathcal{C}_{\text{rem}}$ (line 4); let's denote this result by $\mathcal{K}_1$. As explained before, $\mathcal{K}_1$ is not valid because it violates $\sigma_5(Cool_Paper)$, so the check of line 4 will fail and line 7 will identify this violation and set: $g = \neg PublishedIn(Cool_Paper, Cool_Journal)$, $\sigma(\vec{a}) = \sigma_5(Cool_Paper)$ and $\mathcal{S} = \{PublishedIn(Cool_Paper, Cool_Journal)\}$. For the particular constraint instance it holds that: $Res(\sigma_5(Cool_Paper)) = \{\{\neg Paper(Cool_Paper)\}, \{PublishedIn(Cool_Paper, KAIS)\}, \{PublishedIn(Cool_Paper, Cool_Journal)\}$. Thus, the FOR loop of line 9 will iterate over the first two sets of side-effects (which also correspond to the two options described in Sect. 2).

For the sake of this example, let's assume that $\mathcal{S}' = \{PublishedIn(Cool_Paper, KAIS)\}$ is considered first. Then, line 10 will call the function: $Change(\{\neg PublishedIn(Cool_Paper, Cool_Journal), PublishedIn(Cool_Paper, KAIS)\}, \mathcal{K}, \Delta_\infty)$. In this new recursive call, the check of line 1 is again false, but the check of line 4 is true, because no rule is violated for the new KB, as $Cool_Paper$ appears now to be published in some other journal ($KAIS$). Thus, line 5 will return $\{\neg PublishedIn(Cool_Paper, Cool_Journal), PublishedIn(Cool_Paper, KAIS)\}$; this corresponds to a candidate full solution for the set of side-effects to apply upon the original KB.

After the recursive call returns, the original $Change$ function will execute line 10 and, since $\Delta_{pref} = \Delta_\infty$, it will set $\Delta_{pref} = \{\neg PublishedIn(Cool_Paper, Cool_Journal), PublishedIn(Cool_Paper, KAIS)\}$. Then, the second $\mathcal{S}'$ will be considered, namely $\mathcal{S}' = \{\neg Paper(Cool_Paper)\}$, spawning a new recursive call: $Change(\{\neg PublishedIn(Cool_Paper, Cool_Journal), \neg Paper(Cool_Paper)\}, \mathcal{K}, \Delta_{pref})$.

In this new recursive call, line 1 will determine whether $\{\neg PublishedIn(Cool_Paper, Cool_Journal), PublishedIn(Cool_Paper, KAIS)\} \leq_{\mathcal{K}_0} \{\neg PublishedIn(Cool_Paper, Cool_Journal), \neg Paper(Cool_Paper)\}$. This, of course, depends on the definition of $\leq_{\mathcal{K}_0}$, so let's assume that the comparison is based on the cardinality of the deltas (smaller deltas are preferred); in case of equal cardinality, one prefers the deltas that remove (rather than add) tuples; in case of a further tie, the actual tuples are considered, taking into account both the relations and the constants involved (details are irrelevant and omitted). For the particular selection mechanism, the check fails, so we proceed with line 4, which identifies an invalidity. In particular, line 7 will identify that $\sigma_3(Cool_Paper, RDF_Paper)$ is violated because for $g = \neg Paper(Cool_Paper)$, $\mathcal{S} = \{Paper(Cool_Paper)\}$ the conditions of line 7 hold.

The only $\mathcal{S}' \in Res(\sigma_3(Cool_Paper, RDF_Paper)$ for which $\mathcal{S} \neq \mathcal{S}'$ is $\mathcal{S}' = \{\neg Cites(Cool_Paper, RDF_Paper)\}$. Thus, a new recursive call will be spawned, namely: $Change(\{\neg PublishedIn(Cool_Paper, Cool_Journal), \neg Paper(Cool_Paper), \neg Cites(Cool_Paper, RDF_Paper)\}, \mathcal{K}, \Delta_{pref})$. However, for this recursive call, the check of line 1 will succeed because $\Delta_{pref} = \{\neg PublishedIn(Cool_Paper, Cool_Journal), PublishedIn(Cool_Paper, KAIS)\} \leq_{\mathcal{K}_0} \{\neg PublishedIn(Cool_Paper, Cool_Journal),$

Author's personal copy

$\neg Paper(Cool_Paper), \neg Cites(Cool_Paper, RDF_Paper)\}$, as the former has cardinality 2, whereas the latter has cardinality 3. This will force the final recursive call to return Δ_∞.

Line 10 of the old recursive call will now compare Δ_∞ with Δ_{pref} and keep Δ_{pref} to return to the original call, which, in turn will also keep Δ_{pref} in its own comparison of line 10, eventually returning it. Thus, the result of said change operation is $\mathcal{K} + \Delta_{pref}$, i.e., $\{Paper(DB_Paper), Paper(RDF_Paper), Paper(Cool_Paper), PublishedIn(DB_Paper, KAIS), PublishedIn(RDF_Paper, KAIS), PublishedIn(Cool_Paper, KAIS), Journal(KAIS), Journal(Cool_Journal), Cites(DB_Paper, Cool_Paper), Cites(RDF_Paper, DB_Paper), Cites(Cool_Paper, RDF_Paper)\}$.

It can be shown that the *Change* function described in Algorithm 1 implements a rational change operator:

Proposition 8.1 *Consider a validity model* Σ*, a selection mechanism* $\preceq$*, and the corresponding rational change operator* $\bullet$*. Consider also some valid KB* $\mathcal{K}$ *and a change request* $\mathcal{C}$*. Suppose that the call* $Change(\mathcal{C}, \mathcal{K}, \Delta_\infty)$ *terminates with output* Δ_{out}*. If* $\mathcal{C}$*: feasible, then* $\Delta_{\text{out}} \neq \Delta_\infty$ *and* $\mathcal{K} + \Delta_{\text{out}} = \mathcal{K} \bullet \mathcal{C}$*. If* $\mathcal{C}$*:infeasible then* $\Delta_{\text{out}} = \Delta_\infty$*.*

Note that the *Change* function (Algorithm 1) does not specify the order in which the rules are considered in line 7 (see also the related discussion in Sect. 5.2); Proposition 8.1 shows that this is not an issue, as the algorithm will report the correct result, regardless of the order.

The computational properties (termination and complexity) of Algorithm 1 depend on the actual validity model and selection mechanism employed. Since these are not specified in the above general discussion, one cannot predict, e.g., the complexity of comparing deltas in lines 1 and 10, which could range to anything from constant to undecidable, depending on the actual selection mechanism. Similarly, the validity checking in lines 4, 7 depends on the actual constraints considered; an extensive complexity analysis for this problem under various assumptions appears in [1,15]. In the next sections, we will confine ourselves to the RDF/S context and show how the presented framework can be applied to RDF/S KB evolution, as well as the related termination results.

9 Tailoring the framework to RDF/S

To apply our framework to a given evolution context (such as RDF/S KB evolution), we need to:

- determine the predicates and constants that will be used to model the information (ground facts) in the corresponding KBs;
- define, using $DED^{\neq}$ constraints, the validity model that expresses the constraints/semantics of said context;
- define a selection mechanism that captures the intuition of the knowledge engineer regarding the preferred delta to apply upon a KB for the specific context.

In this section, we will perform this exercise for the RDF/S context under the semantics proposed in [55]; a similar approach can be used for other contexts (including other RDF/S formalizations).

9.1 An introduction to RDF/S

RDF [47] uses *resources* to represent real-world entities. *Triples* of the form (*subject*, *predicate*, *object*) are used to describe resources. The set $\mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$ is the set of all triples,

Author's personal copy

where **U**, **L** are two disjoint and infinite sets denoting the URIs (identifiers for resources) and literals, respectively.

RDFS [10] introduces *typing* and *inference* semantics to RDF. Typing semantics determines whether a resource is a particular object in the real world (i.e., it is an *individual*), or it is a collection of other resources (i.e., it is a *class*) or it is a binary relation between resources (i.e., it is a *property*) [55]. Inference semantics allows us to infer knowledge that is not explicit in the KB (e.g., implicit subsumption relations) [55]. In addition, RDFS introduces special URIs (e.g., *rdfs*:*subClassOf*), which allow triples to describe, for example, subsumption and instantiation relationships, or to determine the domain and range of a property. For more details on typing, inference, and the semantics of the various RDFS constructs, see [10,55].

The term RDF/S refers to RDF that is enhanced with RDFS semantics. An *RDF/S KB* $\mathcal{K}$ is defined as a finite set of RDF triples that adhere to the semantics of RDFS.

For simplicity, we focus on the semantical information of RDF/S, so certain features like bags, lists, comments, reification, blank nodes etc are not considered in this version of the work. Moreover, metaclasses (collections of classes) and metaproperties (collections of properties) [55] can be easily handled, and are omitted for simplicity. The ideas presented in this paper can easily be extended to include these features (see also [39]).

9.2 Modeling RDF/S KBs using ground facts

The semantical information contained in RDF/S triples can easily be mapped to ground facts. The basic idea is that each triple type is associated with some predicate that models this type of information, whereas the constants used are the URIs and literals ($\mathbf{U} \cup \mathbf{L}$). We define the set $Sp \subset \mathbf{U}$ which contains all the RDF/S URIs which have special semantics, such as: *rdf*:*type*, *rdfs*:*Class*, *rdf*:*Property*, *rdfs*:*Resource*, *rdfs*:*subClassOf*, *rdfs*:*subPropertyOf*, *rdfs*:*domain*, *rdfs*:*range*, etc. Constants in Sp will be called *special* URIs, whereas URIs in $\mathbf{U} \setminus Sp$ will be called *custom* URIs.

The used predicates are those shown in Table 2. The predicates Cl, Pr, Ind are used to determine the type of a given custom URI. The rest of the predicates ($CSub$, $PSub$, Dom, Rng, CI, PI) are used to determine various relationships between URIs and/or literals (subsumption, domain, instantiation etc), as described in the table.

Table 3 shows how the different RDF/S triples are associated with particular ground facts and allows the transformation of a set of triples into a set of ground facts (and vice-versa).

Table 2 Predicates for RDF/S modeling

Predicate	Intuitive meaning
$Cl(A)$	A is a custom URI, and represents a class
$Pr(A)$	A is a custom URI, and represents a property
$Ind(A)$	A is a custom URI, and represents an individual
$CSub(A, B)$	A is a subclass of B
$PSub(A, B)$	A is a subproperty of B
$Dom(A, B)$	B is the domain of property A
$Rng(A, B)$	B is the range of property A
$CI(A, B)$	A is an instance of class B
$PI(A, B, C)$	A has property C with value B (i.e., the pair (A, B) is an instance of property C)

Author's personal copy

Table 3 Association of RDF/S triples with ground facts

RDF/S triple	Ground fact
$(A,\ rdf{:}type,\ rdfs{:}Class)$	$Cl(A)$
$(A,\ rdf{:}type,\ rdf{:}Property)$	$Pr(A)$
$(A,\ rdf{:}type,\ rdfs{:}Resource)$	$Ind(A)$
$(A,\ rdf{:}type,\ B)$	$CI(A, B)$
$(A,\ rdfs{:}subClassOf,\ rdfs{:}Resource)$	$CSub(A, rdfs{:}Resource)$
$(A,\ rdfs{:}subClassOf,\ B)$	$CSub(A, B)$
$(A,\ rdfs{:}subPropertyOf,\ B)$	$PSub(A, B)$
$(A,\ rdfs{:}domain,\ rdfs{:}Resource)$	$Dom(A, rdfs{:}Resource)$
$(A,\ rdfs{:}domain,\ B)$	$Dom(A, B)$
$(A,\ rdfs{:}range,\ rdfs{:}Resource)$	$Rng(A, rdfs{:}Resource)$
$(A,\ rdfs{:}range,\ rdfs{:}Literal)$	$Rng(A, rdfs{:}Literal)$
$(A,\ rdfs{:}range,\ B)$	$Rng(A, B)$
$(A,\ B,\ C)$	$PI(A, C, B)$

Note that the constants A, B that appear in Table 3 are assumed to be custom URIs, whereas C (that appears in the last row) may be a custom URI or a literal. These associations follow the semantics described in [55]. The typing semantics of RDF/S is determined by the three special RDF/S resources, *rdfs*:*Class*, *rdf*:*Property*, *rdfs*:*Resource*, as shown in the first three rows of Table 3. The other triple types express factual information, like class/property subsumption, the specification of the domain/range of a property, and class/property instantiation. In the last row, B in triple $(A,\ B,\ C)$ is the property which is being instantiated by the pair (A, C) (associated with the ground fact $PI(A, C, B)$).

Using Table 3, a set of triples can easily be transformed into a set of ground facts (and vice-versa). The only thing that should be noted is that the originally provided triples may be incomplete, in the sense that certain implicit information may be missing. For example, seeing the triple $(A,\ rdf{:}type,\ rdfs{:}Class)$ we conclude that A is a class and add $Cl(A)$, according to Table 3; however, all classes are subclasses of *rdfs*:*Resource*, but this fact is often omitted (i.e., the triple $(A,\ rdfs{:}subClassOf,\ rdfs{:}Resource)$ may not be in the RDF/S KB, so $CSub(A, rdfs{:}Resource)$ will not be added), even though it is actually an indispensable part of the knowledge that A is a class. Similarly, implicit (transitive) instantiations or subsumptions are often omitted. Identifying this missing information and adding it to the RDF/S KB can be done in a post-processing phase that would use rules to identify which triples (or ground facts) follow from other triples (or ground facts). As this is a purely technical process that poses no research challenges, it is omitted.

Note that we could alternatively use logical propositions to express certain triples; for example, class subsumption between two classes A, B is often captured using a formula of the form $\forall x\, A(x) \rightarrow B(x)$. This approach is often called *schema-aware* because the relations used for the representation are the classes and properties of the schema.

On the other hand, our approach is *schema-agnostic*, because the same relations are used for the representation of any RDF/S KB, regardless of the classes or properties appearing in it. The schema-agnostic approach is more adequate for our purposes because it allows us to capture assertions of the form "A is a class", and, consequently, support operations such as the addition and removal of classes, properties etc [26].

Author's personal copy

9.3 An RDF/S validity model

As already mentioned, this work uses the RDF/S model which was proposed in [55] in an effort to provide a group of sound and complete algorithms for RDF/S query containment and minimization. This model enforces a clear role distinction between types (classes, properties, individuals), requires explicitly specified and unique domains/ranges for properties, no cycles in the subsumptions, while property subsumption and instantiation respects corresponding domain/range subsumption/instantiation relationships. Similar semantics for RDF/S have also been recognized and suggested in [52] in an effort to provide compatibility between RDF/S and OWL DL.

Table 4 shows the integrity constraints that we impose on RDF/S KBs to capture the semantics of RDF/S (e.g., $R10.1$ ensures transitivity of class subsumption), as well as the restrictions imposed by [55] (e.g., $R13$ guarantees that the domain of a property is unique). For simplicity, Table 4 uses the additional predicates URI, Lit to represent custom URIs and literals, respectively, i.e., $URI(x)$ ($Lit(x)$) is true iff $x \in \mathbf{U} \setminus Sp$ ($x \in \mathbf{L}$); $URI(x)$, $Lit(x)$ should be replaced by $true$ or $false$ depending on the value of x in each constraint instance.

In Table 4, constraints $R1.1 - R7.3$ determine the required types for the constants that appear in each of the predicates. Constraints $R8.1 - R8.3$ guarantee the role distinction between different custom URIs, i.e., that classes, properties and individuals are disjoint. The special URI *rdfs*:*Resource* is the root of the class hierarchy, so all individuals are instantiated under it, and all classes are subsumed by it ($R9.1$, $R9.2$). The constraints $R10.1 - R11.2$ guarantee that $CSub$, $PSub$ are transitive and irreflexive; the former property is imposed by RDFS, whereas the latter stems from the requirement expressed in [55] for acyclic hierarchies. Note that $\perp$, appearing in $R10.2$, $R11.2$ is the logical falsehood; to transform those constraints to the standard $DED^{\neq}$ form, we can replace $\perp$ with, e.g., $(x \neq x)$. The existence and uniqueness of the domain/range of a property is imposed through $R12 - R14$. Inheritance of instantiation is imposed using $R15$, $R16$. Finally, the domain and range of a property should be respected by subsumed properties and by property instances, as described by $R17.1 - R18.2$.

9.4 An RDF/S selection mechanism

In this subsection, we describe the third and final step towards applying our evolution approach, which is to propose a selection mechanism suitable for RDF/S. To do so, we consider an ordering over predicates and define that the preferred delta is the one that contains less changes involving the least preferable predicates; in case of a tie, a further ordering over constants (URIs and literals) allows us to define an ordering over ground facts that have the same predicate and break the tie (to satisfy the requirement for the ordering to be wellfounded).

Even though the assumptions and intuitions underlying our proposal are reasonable for most applications (and, in particular, for the evolution of curated RDF/S KBs), we understand that there may be applications where it fares poorly; in fact, we argue that no single selection mechanism is suitable for all evolution contexts/applications, because it should reflect the application's peculiarities. Therefore, our proposal should be viewed as a general guideline, and as an example of how the intuition regarding a selection mechanism can be formalized and used to define a rational change operator.

The basic idea is that certain changes on the KB are more disruptive than others. For example, deleting a class is a more important change than deleting a class subsumption relationship, so side-effects containing ground facts of the form $\neg Cl(A)$ are less preferable than

Author's personal copy

Table 4 Integrity constraints for RDF/S

Integrity constraint	Intuitive meaning
$R1.1: \forall x Cl(x) \rightarrow URI(x)$ $R1.2: \forall x Pr(x) \rightarrow URI(x)$ $R1.3: \forall x Ind(x) \rightarrow URI(x)$	Allowable constants for Cl, Pr, Ind
$R2.1: \forall x, y CSub(x, y) \rightarrow Cl(x)$ $R2.2: \forall x, y CSub(x, y) \rightarrow Cl(y) \vee (y = rdfs{:}Resource)$	Typing of $CSub$
$R3.1: \forall x, y PSub(x, y) \rightarrow Pr(x)$ $R3.2: \forall x, y PSub(x, y) \rightarrow Pr(y)$	Typing of $PSub$
$R4.1: \forall x, y Dom(x, y) \rightarrow Pr(x)$ $R4.2: \forall x, y Dom(x, y) \rightarrow Cl(y) \vee (y = rdfs{:}Resource)$	Typing of Dom
$R5.1: \forall x, y Rng(x, y) \rightarrow Pr(x)$ $R5.2: \forall x, y Rng(x, y) \rightarrow Cl(y) \vee (y = rdfs{:}Resource) \vee (y = rdfs{:}Literal)$	Typing of Rng
$R6.1: \forall x, y CI(x, y) \rightarrow Ind(x)$ $R6.2: \forall x, y CI(x, y) \rightarrow Cl(y) \vee (y = rdfs{:}Resource)$	Typing of CI
$R7.1: \forall x, y, z PI(x, y, z) \rightarrow Ind(x)$ $R7.2: \forall x, y, z PI(x, y, z) \rightarrow Ind(y) \vee Lit(y)$ $R7.3: \forall x, y, z PI(x, y, z) \rightarrow PS(z)$	Typing of PI
$R8.1: \forall x, y Cl(x) \wedge Pr(y) \rightarrow (x \neq y)$ $R8.2: \forall x, y Cl(x) \wedge Ind(y) \rightarrow (x \neq y)$ $R8.3: \forall x, y Pr(x) \wedge Ind(y) \rightarrow (x \neq y)$	Classes, properties and individuals are disjoint
$R9.1: \forall x Cl(x) \rightarrow CSub(x, rdfs{:}Resource)$ $R9.2: \forall x Ind(x) \rightarrow CI(x, rdfs{:}Resource)$	Semantics for *rdfs:Resource* (root of the class hierarchy)
$R10.1: \forall x, y, z CSub(x, y) \wedge CSub(y, z) \rightarrow CSub(x, z)$ $R10.2: \forall x, y CSub(x, y) \wedge CSub(y, x) \rightarrow \bot$	Semantics for $CSub$ (transitive, irreflexive)
$R11.1: \forall x, y, z PSub(x, y) \wedge PSub(y, z) \rightarrow PSub(x, z)$ $R11.2: \forall x, y PSub(x, y) \wedge PSub(y, x) \rightarrow \bot$	Semantics for $PSub$ (transitive, irreflexive)
$R12: \forall x PS(x) \rightarrow \exists y, z(Dom(x, y) \wedge Rng(x, z))$	Each property has a domain and a range
$R13: \forall x, y, z Dom(x, y) \wedge Dom(x, z) \rightarrow (y = z)$	Unique property domain
$R14: \forall x, y, z Rng(x, y) \wedge Rng(x, z) \rightarrow (y = z)$	Unique property range
$R15: \forall x, y, z CI(x, y) \wedge CSub(y, z) \rightarrow CI(x, z)$	Class instance propagation
$R16: \forall x, y, z, w PI(x, y, z) \wedge PSub(z, w) \rightarrow PI(x, y, w)$	Property instance propagation
$R17.1: \forall x, y, z, w PSub(x, y) \wedge Dom(x, z) \wedge Dom(y, w) \rightarrow CSub(z, w) \vee (z = w)$ $R17.2: \forall x, y, z, w PSub(x, y) \wedge Rng(x, z) \wedge Rng(y, w) \rightarrow CSub(z, w) \vee (z = w)$	Subsumption between properties reflects in their domains/ranges
$R18.1: \forall x, y, z, w PI(x, y, z) \wedge Dom(z, w) \rightarrow CI(x, w)$ $R18.2: \forall x, y, z, w PI(x, y, z) \wedge Rng(z, w) \rightarrow CI(y, w) \vee (Lit(y) \wedge (w = rdfs{:}Literal))$	Correct property instantiation

Author's personal copy

Table 5 Ordering of predicates ($\leq_P$)

$PI \leq_P CI \leq_P PSub \leq_P CSub \leq_P \neg PI \leq_P \neg CI \leq_P \neg PSub \leq_P \neg CSub \leq_P \neg Dom \leq_P \neg Rng \leq_P \neg Ind \leq_P \neg Pr \leq_P \neg Cl \leq_P Dom \leq_P Rng \leq_P Ind \leq_P Pr \leq_P Cl$

side-effects containing ground facts of the form $\neg CSub(B, C)$. This intuition is encoded as an order (denoted by $\leq_P$) over predicates and negated predicates, and is shown in Table 5. The table shows that the less preferred changes are the addition of new resources (classes, properties, individuals), followed by the introduction of new domains/ranges of properties; this is based on the idea that adding new, artificial resources as side-effects is a change that should be avoided if possible (same for new domains/ranges). Deleting resources and domains/ranges follows; thus, it is preferable to delete an existing resource (or domain/range) than to add a new one. On the contrary, adding subsumption/instantiation relationships is preferable than deleting said relationships because it is intuitively preferable to enhance our knowledge with new facts, rather than delete existing facts. Deleting/adding instantiation relationships is preferable than deleting/adding subsumption relationships (as the latter refer to the schema part of the RDF/S KB), but all such changes are preferable than the addition/deletion of resources or domains/ranges.

The idea behind $\leq_P$ is driven one step further to allow the comparison of deltas: intuitively, Δ_1 is preferable over Δ_2 iff Δ_1 contains less important changes than Δ_2, i.e., if the ground facts in Δ_1 use less important predicates, according to $\leq_P$. Thus, the comparison is not based on the number of ground facts that a delta (as a whole) contains, but on the number of the "important" ground facts that it contains, as determined by the predicates they use. So, in order to compare Δ_1, Δ_2, we start with the least preferred predicate (according to $\leq_P$, i.e., predicate Cl) and count the number of ground facts in Δ_1, Δ_2 that use Cl (less is preferable). In case of a tie, we proceed to the next predicate (Pr in this case) and repeat the process until we reach a conclusion. A consequence of this fact is that a delta containing any number of less disruptive changes is more preferable than a delta containing even a single more disruptive one, e.g.: $\{\neg CSub(A_1, B_1), \neg CSub(A_2, B_2), PSub(A_3, B_3)\}$ is preferred over $\{\neg Cl(A)\}$. This is similar to the notion of *component-cardinality repairs* [1] that has been used in the context of database repairs; unlike component-cardinality repairs however, where all predicates are considered of equal importance, here we impose a strict ordering between predicates (and their negations).

In some cases, it could happen that two deltas contain the same number of ground facts for all predicates; in such cases, we need a more fine-grained criterion to determine the preferred one. This criterion is based on a comparison of the individual ground facts based on their arguments (used constants), and identifies the least preferred delta as the one that contains the most important ground fact. To compare ground facts that use the same predicate, we consider the constants that they use, and determine their importance based on the constants' position in the corresponding hierarchy. This is based on the intuition that, e.g., a class that is high in the class subsumption hierarchy represents an abstract, general concept, that is usually important conceptually, so it is less prone to change; therefore, deleting such a class should be less preferred than deleting a lower-level class. Similar arguments can be given for other predicates: for example, a class subsumption is more important if the subsumed class represents a concrete concept (low in the hierarchy) whereas the subsuming class represents an abstract concept (high in the hierarchy). In rare cases, it could happen that this criterion is not a tie-breaker either (e.g., when comparing the deletion of two sibling classes); to resolve

Author's personal copy

such ties we could use any arbitrary wellorder over $\mathbf{U} \cup \mathbf{L}$. We will denote this ordering using $\leq_{\mathbf{UL}}$ Note that such an ordering always exists, by the well-ordering theorem.

To formalize the above ideas we need to define some notions. First, we say that a constant $A \in \mathbf{U} \cup \mathbf{L}$ *appears* in an RDF/S KB $\mathcal{K}$ iff there is some ground fact $p(A_1, \ldots, A_n) \in \mathcal{K}$ such that $A = A_i$ for some $i = 1, 2, \ldots, n$. For $A, B \in \mathbf{U}$, we say that A is a *direct subclass* of B in $\mathcal{K}$ iff $CSub(A, B) \in \mathcal{K}$ and there is no C such that $CSub(A, C) \in \mathcal{K}$ and $CSub(C, B) \in \mathcal{K}$. Similar notions can be defined for properties, as well as for instantiation relationships (e.g., an individual A is a *direct instance* of B iff $CI(A, B) \in \mathcal{K}$ and there is no C such that $CI(A, C) \in \mathcal{K}$ and $CSub(C, B) \in \mathcal{K}$). We use the general term *direct sub-resource* to describe those notions. We say that $A \in \mathbf{U}$ is a *top resource* in $\mathcal{K}$ iff A appears in $\mathcal{K}$ and there is no $B \in \mathbf{U}$ such that A is a direct sub-resource of B. The intuitive meaning of these definitions in RDF/S KBs are obvious: direct sub-resources are those pairs of resources that are related through a direct (i.e., non-redundant) subsumption or instantiation relation, whereas a top resource is one that is no sub-resource of any resource.

A *path from A to the top* in $\mathcal{K}$ is a sequence S of the form $S = \langle A_1, A_2, \ldots, A_n \rangle$ where $A_1 = A$, A_n is a top resource in $\mathcal{K}$ and for all $i = 1, 2, \ldots, n-1$, it holds that A_i is a direct sub-resource of A_{i+1}. The *length* of $S = \langle A_1, A_2, \ldots, A_n \rangle$ is n. Note that if A does not appear in $\mathcal{K}$, or if it is a literal, then there is no path from A to the top. If A is a top resource, then the only path from A to the top is $\langle A \rangle$, whose length is 1. In general however, there may be many top resources and/or more than one paths from any given A to (each of) the top resource(s); in such cases, we are interested in the shortest path from A to the top (given that $\mathcal{K}$ is finite and no cycles are allowed in the subsumption relationships by the integrity constraints, there will always be a shortest path). Given a constant A (URI or literal) we set $Dist(A)$ to be the length of the shortest path from A to the top; if no such path exists (e.g., if A does not appear in $\mathcal{K}$), we set $Dist(A) = 0$. Note that $Dist$ represents the position of a constant in its corresponding hierarchy, in the sense that resources higher in the hierarchy have a lower $Dist$.

To define the wellorder $\leq_{\mathbf{UL}}$ that is necessary as the ultimate tie-breaker, we will treat URIs and literals as strings and use the so-called *shortlex* order.[6] The shortlex order compares the length of two strings (shortest one comes first); in case of equal size, the standard lexicographic ordering is used to determine the order. Shortlex can be easily shown to be a wellorder.[7]

Combining the above, the fine-grained ordering on ground facts is defined as shown in Table 6. Whenever two ground facts are using a different predicate, their order is determined using $\leq_P$; if they use the same predicate, we resort to comparing the constants involved using $Dist$ and $\leq_{\mathbf{UL}}$, as shown in Table 6.

Now we have all the necessary formalisms to define our selection mechanism. In Definition 9.1 below, Δ^q is a set containing the ground facts from Δ that use the (possibly negated) predicate q.

Definition 9.1 Consider two deltas Δ_1, Δ_2 and a valid KB $\mathcal{K}$. We define an ordering $\leq_{\mathcal{K}}$ such that $\Delta_1 \leq_{\mathcal{K}} \Delta_2$ iff any of the following is true:

1. There is some (possibly negated) predicate q such that $|\Delta_1^q| < |\Delta_2^q|$ and for all predicates q' such that $q <_P q'$ it holds that $|\Delta_1^{q'}| = |\Delta_2^{q'}|$.
2. For all (possibly negated) predicates q it holds that $|\Delta_1^q| = |\Delta_2^q|$ and there is some $g \in \Delta_1 \setminus \Delta_2$, such that $g <_G g'$ for all $g' \in \Delta_2 \setminus \Delta_1$.

[6] http://en.wikipedia.org/wiki/Shortlex_order.

[7] http://en.wikipedia.org/wiki/Lexicographical_order.

Author's personal copy

Table 6 Ordering of ground facts ($\leq_G$)

Predicate arity	Ground fact order
Arity 1:	Assuming $q(a_1), q(a_2)$ such that $q(a_1) \neq q(a_2)$ then:
$q(x)$	$q(a_1) <_G q(a_2)$ iff $Dist(a_1) > Dist(a_2)$;
(Cl, Pr, Ind)	if tied, $q(a_1) <_G q(a_2)$ iff $a_1 <_{\mathbf{UL}} a_2$.
Arity 2:	Assuming $q(a_1, b_1), q(a_2, b_2)$ such that $q(a_1, b_1) \neq q(a_2, b_2)$ then:
$q(a, b)$	$q(a_1, b_1) <_G q(a_2, b_2)$ iff $Dist(a_1) > Dist(a_2)$;
$(CSub, PSub, Dom, Rng, CI)$	if tied, $q(a_1, b_1) <_G q(a_2, b_2)$ iff $Dist(b_1) < Dist(b_2)$;
	if tied, $q(a_1, b_1) <_G q(a_2, b_2)$ iff $a_1 <_{\mathbf{UL}} a_2$);
	if tied, $q(a_1, b_1) <_G q(a_2, b_2)$ iff $b_1 <_{\mathbf{UL}} b_2$).
Arity 3:	Assuming $q(a_1, b_1, c_1), q(a_2, b_2, c_2)$ such that $q(a_1, b_1, c_1) \neq$
$q(x, y, z)$	$q(a_2, b_2, c_2)$ then:
(PI)	$q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $Dist(a_1) > Dist(a_2)$;
	if tied, $q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $Dist(b_1) > Dist(b_2)$;
	if tied, $q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $Dist(c_1) < Dist(c_2)$;
	if tied, $q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $a_1 <_{\mathbf{UL}} a_2$);
	if tied, $q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $b_1 <_{\mathbf{UL}} b_2$);
	if tied, $q(a_1, b_1, c_1) <_G q(a_2, b_2, c_2)$ iff $c_1 <_{\mathbf{UL}} c_2$).

3. $\Delta_1 = \Delta_2$.

This definition formally captures the intuitive description provided above. Our definitions guarantee that the ordering $\leq_{\mathcal{K}}$ is change-generating, so it can be used to define a selection mechanism. Formally:

Proposition 9.1 *The set* $\preceq = \{\leq_{\mathcal{K}} | \mathcal{K} : \textit{valid KB}\}$ *(where* $\leq_{\mathcal{K}}$ *as in Definition 9.1) is a selection mechanism.*

Note that some of the orderings imposed by $\leq_P$ (e.g., $Dom \leq_P Rng$), may seem artificial, but are necessary in order for $\leq_{\mathcal{K}}$ to be antisymmetric (thus, change-generating, cf. Definition 6.2). However, a detailed examination of our integrity constraints and the algorithm itself can reveal that, in practical change scenarios, determining the preferred delta cannot boil down to such a comparison. Similarly, using shortlex is arguably an arbitrary choice, but necessary to guarantee antisymmetry when all else fails; for this reason, it constitutes our last resort in the rare case when two deltas are very similar. An alternative approach here would be to drop the antisymmetry requirement from Definition 6.2; this would essentially allow deltas to be equally preferable, so the change operator could return more than one results. This scenario could be viable only for applications where it is desirable (and possible) to directly involve the knowledge engineer in the evolution process. We plan to consider such a relaxation as a future work.

9.5 Algorithm: termination and complexity

Algorithm 1 can be used as-is for the RDF/S setting, with the modeling, validity model and selection mechanism as described in the previous subsections. One thing that should be noted is that, given a KB $\mathcal{K}$ and a change request $\mathcal{C}$, we only need to consider constants that appear in $\mathcal{K}$ and $\mathcal{C}$, plus one "fresh" URI (i.e., a URI not appearing in $\mathcal{K}$, $\mathcal{C}$), say γ, that

Author's personal copy

is the minimum (according to $\preceq_{\mathbf{UL}}$) of the custom URIs that do not appear in $\mathcal{K}$ or $\mathcal{C}$; we denote this set of constants by Γ. This is a useful assumption for both intuitive and practical purposes, but is also correct from a formal point of view, because even if we considered arbitrary URIs, all solutions (potential change results) containing them would be rejected as non-preferred due to the preference ordering (see Proposition 9.2). As a result, the FOR loop in lines 9-11 of Algorithm 1 need only consider those $\mathcal{S}'$ which use constants from Γ (because all other $\mathcal{S}'$ cannot possibly lead to the correct result). We denote this modified algorithm by $Change_{RDF}$ (the pseudocode is almost identical to Algorithm 1 and omitted). This modification guarantees termination, without jeopardizing correctness:

Proposition 9.2 *Consider the validity model Σ defined in Sect. 9.3, the selection mechanism $\preceq$ defined in Sect. 9.4, and the corresponding rational change operator $\bullet$. Consider also some valid RDF/S KB $\mathcal{K}$ and a change request $\mathcal{C}$. Then, the call $Change_{RDF}(\mathcal{C}, \mathcal{K}, \Delta_\infty)$ terminates. Supposing that the output of $Change_{RDF}(\mathcal{C}, \mathcal{K}, \Delta_\infty)$ is Δ_{out}, then if $\mathcal{C}$: feasible, then $\Delta_{\text{out}} \neq \Delta_\infty$ and $\mathcal{K} + \Delta_{\text{out}} = \mathcal{K} \bullet \mathcal{C}$; if $\mathcal{C}$:infeasible then $\Delta_{\text{out}} = \Delta_\infty$.*

The proof of Proposition 9.2 exploits the fact that the validity model and selection mechanism for the RDF/S setting are fixed in order to show the correctness and termination of $Change_{RDF}$ (note that termination in particular is not guaranteed for the general case—see Sect. 8). Similarly, we can show that for the particular setting the algorithm is, in the worst-case scenario, exponential, because the recursive tree of evaluation generated by the $Change_{RDF}$ function can be exponential in size. However, it should be emphasized that this is an inherent property of the problem setting, rather than an artifact of the proposed solution. To see that, let us consider the easier subproblem "find a set of side-effects that would lead to a valid KB"; it is easy to note that this problem is equivalent to finding one set of ground facts satisfying all constraints, i.e., it is a satisfiability problem, which for the particular $DED^{\neq}$ constraints (Table 4) is equivalent to SAT; thus, implementing a rational change operator is NP-hard.

This issue is addressed using *special-purpose algorithms*, in which we trade generality for computational efficiency to develop optimized (and fast) algorithms that partially implement rational change operators for a given setting (RDF/S in our case); this process will be discussed in the next section.

10 Special-purpose algorithms

Special-purpose algorithms are based on the idea that, once we fix the setting (predicates, validity model, selection mechanism), we can determine the preferred way to handle certain types of change requests at design time, without having to recursively check all possible options; this leads to more efficient implementations. On the other hand, special-purpose algorithms do not enjoy the same generality as the general-purpose one, because they work only for the given setting, and only for the given types of change requests; for other change requests, one has to resort to the general-purpose algorithm.

In order to make sure that the special-purpose algorithms respect our principles (Success, Validity and Minimal Change), one should verify that they produce the same results as the general-purpose one for the change requests that they tackle, i.e., that each of them (partially) implements a rational change operator. Towards this end, the development of the general-purpose algorithm and the related theory is an essential first step, as (i) it allows proving that a special-purpose algorithm exhibits the required properties, something that would not be possible without the theoretical framework presented in the previous sections, and, (ii) it allows us to handle any unforeseen change requests.

Author's personal copy

Algorithms 2, 3, 4, 5, 6 show five of the special-purpose algorithms that we developed. Each of these algorithms corresponds to one type of change request, namely:

Algorithm 2: Add Class Instantiation ($Add_CI(\mathcal{K}, I, C)$)

```
1: if CI(I, C) ∈ K then
2:    return ∅
3: else
4:    Δ := {CI(I, C)}
5: end if
6: if Ind(I) ∉ K then
7:    Δ0 := Add_Ind(K, I)
8:    if Δ0 = Δ∞ then
9:       return Δ∞
10:   else
11:      Δ := Δ ∪ Δ0
12:   end if
13: end if
14: if Cl(C) ∉ K then
15:   Δ0 := Add_Cl(K, C)
16:   if Δ0 = Δ∞ then
17:      return Δ∞
18:   else
19:      Δ := Δ ∪ Δ0
20:   end if
21: end if
22: for all CSub(C, A) ∈ K do
23:   Δ := Δ ∪ {CI(I, A)}
24: end for
25: return Δ
```

- $Add_CI(\mathcal{K}, I, C)$ corresponds to the change request $\mathcal{C} = \{CI(I, C)\}$, i.e., it implements the operation $\mathcal{K} \bullet \{CI(I, C)\}$.
- $Add_Cl(\mathcal{K}, C)$ corresponds to $\mathcal{C} = \{Cl(C)\}$.
- $Rem_Ind(\mathcal{K}, I)$ corresponds to $\mathcal{C} = \{\neg Ind(I)\}$.
- $Rem_Dom(\mathcal{K}, P, C)$ corresponds to $\mathcal{C} = \{\neg Dom(P, C)\}$.
- $Rem_PSub(\mathcal{K}, P_1, P_2)$ corresponds to $\mathcal{C} = \{\neg PSub(P_1, P_2)\}$.

In total, we developed 18 special-purpose operations, each of which corresponds to one of the 18 types of singular change requests that can be defined (due to space limitations, we do not describe them all here—a full list can be found in [39]). One could define more, if interested in addressing some other type of change request in an efficient manner.

The algorithms have the same general structure: first, we check whether the change request that corresponds to said algorithm is already implied by the KB, in which case we return without reporting any effects or side-effects (this could happen, for example, if we are asked to add an already existing class). Otherwise, we determine which constraints from Table 4 can be violated via said change request. In most cases, we can determine (at design time) the best invalidity resolution to follow for each violated constraint, so the selection mechanism is hard-coded in the algorithm. For example, in Algorithm 5, the removal of a domain would violate rule $R12$, which could be resolved either by the removal of the associated property or by the addition of a new domain; however, we can show that the latter option will always lead to non-preferred deltas (due to the $\leq_P$ ordering), so we only consider the former. In other cases, it is not clear which is the best option, as, e.g., in Algorithm 6 where several options are evaluated before determining the result.

Author's personal copy

Algorithm 3: Add Class ($Add_Cl(\mathcal{K}, C)$)

if $Cl(C) \in \mathcal{K}$ **then**
  return $\emptyset$
else
  $\Delta := \{Cl(C), CSub(C, rdfs{:}Resource)\}$
end if
if C: is not a custom URI ($C \notin \mathbf{U} \setminus Sp$) **then**
  return Δ_∞
end if
if $Pr(C) \in \mathcal{K}$ **then**
  return $\Delta \cup Rem_Pr(\mathcal{K}, C)$
end if
if $Ind(C) \in \mathcal{K}$ **then**
  return $\Delta \cup Rem_Ind(\mathcal{K}, C)$
end if
return Δ

Algorithm 4: Remove Individual ($Rem_Ind(\mathcal{K}, I)$)

if $Ind(I) \notin \mathcal{K}$ **then**
  return $\emptyset$
else
  $\Delta := \{\neg Ind(I)\}$
end if
for all $CI(I, A) \in \mathcal{K}$ **do**
  $\Delta := \Delta \cup \{\neg CI(I, A)\}$
end for
for all $PI(I, A, B) \in \mathcal{K}$ **do**
  $\Delta := \Delta \cup \{\neg PI(I, A, B)\}$
end for
for all $PI(A, I, B) \in \mathcal{K}$ **do**
  $\Delta := \Delta \cup \{\neg PI(A, I, B)\}$
end for
return Δ

Algorithm 5: Remove Domain ($Rem_Dom(\mathcal{K}, P, C)$)

if $Dom(P, C) \notin \mathcal{K}$ **then**
  return $\emptyset$
else
  return $\{\neg Dom(P, C)\} \cup Del_Pr(\mathcal{K}, P)$
end if

Algorithm 6: Remove Property Subsumption ($Rem_PSub(\mathcal{K}, P_1, P_2)$)

if $PSub(P_1, P_2) \notin \mathcal{K}$ **then**
  return $\emptyset$
end if
Find all sets $S_i := \{PSub(A_{j-1}, A_j) | j = 1, \ldots, n_i)\}$ such that $PSub(A_{j-1}, A_j) \in \mathcal{K}$ for all $j = 1, \ldots, n_i$ and $A_0 = P_1, A_{n_i} = P_2, n_i \geq 1$
Find all minimal hitting sets of the family $\{S_i\}$, and name them $\Delta_1, \ldots, \Delta_n$
$\Delta := min_{\mathcal{K}}(\{\Delta_i | i = 1, \ldots, n\})$
return Δ

Taking Algorithm 2 as an illustrative example, we note that the addition of $CI(I, C)$ can only cause the violation of constraints $R6.1$, $R6.2$ and $R15$. These are checked and resolved

Author's personal copy

Table 7 Special-purpose algorithms and their complexity

Complexity	Special-purpose algorithms
$O(\|\mathcal{K}\|)$	Add_Cls, Add_Pr, Add_Ind, Add_CI, Rem_Pr, Rem_Ind, Rem_Dom, Rem_Rng, Rem_PI
$O(\|\mathcal{K}\|^2)$	Add_Dom, Add_Rng, Add_PI, Rem_Cl, Rem_CI
NP-complete	Add_CSub, Add_PSub, Rem_CSub, Rem_PSub

in lines 6–13, 14–21 and 22–24, respectively. In particular, line 6 checks whether $R6.1$ is violated; if so, the only possible resolution is by adding $Ind(I)$, so another special purpose algorithm is called to perform this operation (line 7). Similarly, line 14 checks whether $R6.2$ is violated and, if so, it resolves it by calling $Add_Cl(\mathcal{K}, C)$ (line 15). Finally, line 22 determines all the instances of $R15$ that are violated, and resolves the violation by adding the corresponding CI ground fact (line 23); note that the other resolution option (removing $CSub(C, A)$) is not considered, because it will always lead to non-preferred deltas (adding $CI(I, A)$ has no further side-effects, and $CI <_P CSub$). This argumentation is the sketch of the proof showing that Algorithm 2 is equivalent to the general-purpose one (for the specific change request), so it implements part of a rational change operator. More details on the proof of this correctness result (as well as the corresponding results for the other special-purpose algorithms) are omitted due to lack of space.

Note that it is not correct to use the above algorithms for more complicated change requests. For example, the change request $\mathcal{C} = \{CI(I, C), \neg Ind(I)\}$ should not be handled through a call to $Add_CI(\mathcal{K}, I, C)$ followed by a call to $Rem_Ind(\mathcal{K}, I)$; the above sequence of operations would return some result, whereas $\mathcal{C}$ itself is infeasible. The same holds even for feasible change requests. For example, consider the removal of a property subsumption relationship, say $\neg PSub(P_1, P_2)$. Algorithm 6 will determine the shortest path of subsumptions between P_1, P_2 and remove these as side-effects. If, however, a change request contains an additional subsumption removal (say $\neg PSub(P_3, P_4)$), then taking the shortest paths between P_1, P_2 and P_3, P_4 may give a non-preferred compound set of side-effects. An interesting subject of future work is to determine conditions of irrelevance, i.e., conditions under which complicated change requests can be handled as sequences of independent (and simpler) change requests without jeopardizing the correctness of the change result.

It is trivial to show that the above algorithms exhibit much better computational complexity than the general purpose one. In fact, half of the algorithms (9 out of 18) exhibit linear complexity with respect to the size of $\mathcal{K}$ ($O(|\mathcal{K}|)$), some (5) quadratic ($O(|\mathcal{K}|^2)$), whereas the rest (4) can be shown to be NP-complete. Table 7 summarizes the related complexity results.

NP-completeness appears in the algorithms that are related to the subsumption relationships. For example, in Algorithm 6, due to transitivity ($R10.2$), we must find all subsumption paths that connect P_1 and P_2, and remove one link in each; by the selection mechanism, a minimal number of links must be removed. As the paths may have intersections in the case of DAG hierarchies, the problem of finding such a minimal set is actually an instance of the Minimum Hitting Set problem (see line 5), which is an NP-complete problem. The same problem appears in all operations that directly or indirectly involve the removal of a subsumption relationship in a DAG hierarchy. Fortunately, few real-world schemata are DAGs, and even in those, most pairs of classes are connected with a single (or very few) subsumption path(s) [63]; thus, these algorithms are expected to be efficient in practical cases.

Author's personal copy

11 Related work

This work is related to different research fields, including ontology evolution [28], belief revision [31], database repairs [1,4], ontology debugging [28] and updating databases through views [5,37].

Ontology evolution deals with adapting an ontology to changes in its domain or conceptualization [28]. The integrity constraints considered are special constraints on certain relations (e.g., acyclicity/transitivity of subsumption) or property-related constraints (e.g., functional properties); in many cases though, there are no constraints considered at all. In addition, most of the works in this field employ simple and informal methodologies, with limited customization capabilities, and/or address only changes upon the data part of an ontology.

A detailed survey of the field appears in [28]. Some works address the problem using ontology editors [58] or other frameworks that help the engineers decide and implement manually the required changes in their preferred manner [42]. However, it has been argued ([40,58]) that manual application of changes is insufficient. In response to this need, some works like [6,30,38,50,59] have proposed and implemented change semantics, by determining, a priori, the side-effects necessary for each type of change request. This approach requires a highly tedious and error-prone design process, during which all possible problems (invalidities) caused by the supported requests need to be anticipated and resolved a priori, giving no formal guarantees that all cases have been considered or that the various resolution choices made are consistent, as there is no general theoretic treatment to determine the desired result. In certain cases, some flexibility is provided to independently customize [30] the semantics of some of the operations; this is similar to our selection mechanism, but it is restricted to certain operations only. Similarly, in [45], one can explicitly define the semantics of change operators in an event-driven manner. Given the infinite number of potential change requests however, the set of supported changes in all these works is necessarily incomplete [38], unlike in our work. In [46], a declarative approach for the evolution of RDF/S KBs is presented, which can handle all possible changes on the data part of an RDF/S KB, but it is based on fixed semantics, and cannot handle the schema part. In [13], the authors adopt the set of constraints found in [33] and study the "determinism" of changes by providing a characterization of when a deletion of a single triple can be applied unambiguously. In addition, they provide an algorithm which rejects changes that would cause side-effects. Note that our work can also handle side-effects, as well as changes involving more than a single triple.

Belief revision also addresses the problem of dynamic knowledge, but it is usually applied for standard logical formalisms like propositional or first-order logic [31]. In addition, most of the works in belief revision do not consider integrity constraints; as a result, the Principle of Validity in that context amounts to making sure that the resulting KB is consistent (in the standard logical sense).

The use of belief revision approaches for ontology evolution has been advocated in several works, but in several cases, the study remained at a theoretical level without proposing specific algorithms [25]. In addition, many results were negative, i.e., it was shown that certain belief revision approaches cannot be applied to many ontological languages, including RDF/S [29]. An algorithm inspired by belief revision for DL ontologies appears in [19], but it deals only with the data part. In [34], a similar belief-revision-inspired algorithm for changing RDF/S KBs is presented; however, in that setting, no integrity constraints are considered, so additions are trivial and the focus is on deletions, which are non-trivial due to the RDFS inference rules. Our work did not try to adapt any existing belief revision algorithm for the RDF/S setting, but was inspired by ideas presented in that context, such as the principles of Success, Validity and Minimal Change.

Author's personal copy

The field of database repairs addresses the problem of maintaining the integrity of data whenever some abnormal situation (e.g., a failed transaction) leads to an integrity violation [1,4,44]. Various terms have been used to describe this field, like minimal-change integrity maintenance [15], data reconciliation [23] or data cleaning [24]. In the context of database repairing, the invalidity is not necessarily caused by a change in the data (e.g., it could be caused by a change in the integrity constraints, a disk crash, an aborted transaction, or some other reason) and the focus is on repairing, i.e., the cause of the invalidity is not considered; as a result, the Principle of Success is irrelevant. In addition, even though several useful types of constraints (subsets of DEDs) have been identified (see [1] for a list), most works deal with primary and foreign key constraints only.

In many works related to database repairing, there are no guarantees on the distance of the resulting database from the original one (i.e., the Principle of Minimal Change does not apply), but others consider some distance metrics (e.g., symmetric difference [1]). Using such a metric, a repair is defined as a database instance which satisfies the integrity constraints and has minimal distance (according to said metric) from the original inconsistent database. Different metrics are important for different contexts (e.g., in the data warehousing context, subset-repairs are used [1]). *Repair checking techniques* [15] can be used to determine whether a given database is a repair of another. An extensive complexity analysis for these problems under various assumptions appears in [1,15].

On the other hand, only a few efforts have been made in the direction of *computing repairs*. In [14], repair checking algorithms are adapted to non-deterministically compute repairs. Another algorithm for computing repairs, based on a cost model, appears in [9], but it is sensitive to the order in which invalidities are considered and resolved. In [16], an algorithm which uses a flexible distance model to compute repairs is presented; however, the types of constraints used there are only a subset of DEDs.

An approach similar to database repairing advocates the introduction of an extra layer that allows reasoning and consistent querying over the invalid database without explicitly repairing it. This approach is called consistent query answering [8] and could be used for invalid ontologies as well [3,36]. However, this method is more useful when we have no control over the data (so we cannot repair it), which is not the case in our setting. Furthermore, it is restrictive, because it considers only common knowledge over all possible repairs associated with the original data: a tuple is an answer to a given query over a database iff it is an answer in every repair of said database. As a result, we cannot set any preferred repairs, or prevent certain repairs from being considered.

The field of ontology debugging addresses the problem of resolving invalidities in ontologies [28]. Like with database repairs, the cause of the invalidity is not considered during ontology debugging, so the Principle of Success is irrelevant. Ontology debugging consists of ontology diagnosis (identifying inconsistencies and other modeling errors) and ontology repair (repairing such modeling errors). Most works in the field deal with the former problem (diagnosis). In addition, ontology debugging usually deals with inconsistencies or incoherencies only (see [27] for a definition of inconsistency and incoherency), i.e., general integrity constraints are not considered. Surveys of existing ontology debugging approaches can be found at [28,35]. Our work can be applied for ontology diagnosis and repair if we drop the Principle of Success from our requirements and adapt our algorithm accordingly; we plan to explore this research path as a future work.

The problem of updating databases through views [5,17,37] addresses the problem of updating a database view on which updates cannot be issued directly (because, e.g, the view is virtual). In this case, the underlying database needs to change in a way that the evaluation of the view on the new database will give the updated view instance per the user's intention.

Author's personal copy

This is similar to our work in the sense that it requires changing the database in a way that the end result satisfies a certain condition (in this case, the changing of the view in a certain way; in our case, the satisfaction of the integrity constraints). Initial studies have focused on the characterization of "side-effect-free" updates [18,37], which are not always possible. For example, we might want to delete a tuple from a view instance, which comes from a join of two tuples in the base tables. Unless we change the join attribute or delete one of the latter tuples, the update cannot be implemented. This led researchers to restrict the kinds of updates supported [37], to develop frameworks to detect and present side-effects to the user [48], let the user encode some kind of ad-hoc resolution rules in the face of alternative side-effects [56], or relax the constraint that the view should materialize only the user's update and nothing more [41] (this is equivalent to implementing not the user's update but another one which is "close" to it). To the best of our knowledge, there is no approach related to the field of updating databases through views that enforces all updates (adhering to our Principle of Success), or that parameterizes the system with some "preference" mechanism in order to automatically resolve ambiguity.

12 Conclusion

We studied the problem of evolving KBs in the face of new information, while respecting the associated integrity constraints. We applied our work to RDF/S KBs with integrity constraints, considering schema and data change requests, as well as change requests involving any mixture of schema and data operations. We proposed a formal framework to describe such changes and their effects, as well as a general-purpose algorithm that identifies the effects and side-effects of a change request and implements a change operator. Our methodology is inspired by the general belief revision principles of Success, Validity and Minimal Change [31]. The end result is a general-purpose algorithm that is parameterizable, both in terms of the application context (language and validity model) and in terms of the implementation of the Principle of Minimal Change (selection mechanism). Using our framework, the knowledge engineer only needs to specify the change request, and does not need to address how any potential invalidities will be resolved; the system will automatically identify and apply any necessary side-effects, depending on the parameterization.

Our algorithm avoids resorting to the error-prone, per-case reasoning of other systems, as all the alternatives regarding the side-effects of a change request can be derived from the integrity constraints themselves, in an exhaustive and provably correct manner. In addition, it can support all imaginable operations, including operations not considered at design time. Finally, it can handle quite complex types of constraints ($DED^{\neq}$ [20]).

We applied our ideas on RDF/S, using a specific validity model inspired by [55]. This allowed us to develop simpler, special-purpose variations of the general-purpose algorithm, which provably return the same result for specific change requests in a much more efficient manner. Note that the general-purpose algorithm can be still relied upon for change requests that are not considered by the special-purpose ones.

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 KBs.[8] Future work includes experimental evaluation of the algorithms' performance and the incorporation of heuristics for improving their efficiency.

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

Author's personal copy

Appendix: Proofs of propositions

Proof of Propostion 5.1 By the construction of $Res(\sigma(\vec{a}))$, a constraint instance $\sigma(\vec{a})$ can be written as the disjunction of the conjunction of the elements in each $\mathcal{S} \in Res(\sigma(\vec{a}))$. Thus, the result follows from standard reasoning and our semantics. □

Proof of Propostion 5.2 By Definition 4.1, it follows that $\mathcal{K} + \mathcal{S} \vdash \mathcal{S}$, so by Proposition 5.1, $\mathcal{K} + \mathcal{S} \vdash \sigma(\vec{a})$. For the second part, by Proposition 5.1 it follows that there is some $\mathcal{S} \in Res(\sigma(\vec{a}))$ such that $\mathcal{K} + \mathcal{S}' \vdash \mathcal{S}$. So take some (possibly negated) ground fact $g \in \mathcal{S}$. Given that $\mathcal{K} + \mathcal{S}' \vdash \mathcal{S}$ it follows that $\mathcal{K} + \mathcal{S}' \vdash g$; using Definition 4.1 it is trivial to conclude that $\mathcal{K} \vdash g$ or $g \in \mathcal{S}'$. □

Proof of Propostion 5.3 Since $\mathcal{K}$ is valid, there is some $\mathcal{S} \in Res(\sigma(\vec{a}))$ such that $\mathcal{K} \vdash \mathcal{S}$. By our hypothesis, $\mathcal{K} + \mathcal{C} \nvdash \sigma(\vec{a})$ so $\mathcal{K} + \mathcal{C} \nvdash \mathcal{S}$. Thus, there is some (possibly negated) ground fact $g \in \mathcal{S}$ such that $\mathcal{K} + \mathcal{C} \nvdash g$; on the other hand, $\mathcal{K} \vdash g$. Combining these facts and the definition of raw application, it is trivial to show the result. □

Proof of Propostion 7.1 Take some KB $\mathcal{K}$ and change request $\mathcal{C}$. Suppose initially that $\mathcal{K}$ is valid and $\mathcal{C}$ is feasible. Since $\mathcal{C}$ is feasible, there is some valid KB $\mathcal{K}$ such that $\mathcal{K} \vdash \mathcal{C}$, i.e., $\Omega \neq \emptyset$. By wellfoundedness of $\preceq_{\mathcal{K}}$, Ω has at least one minimum; by totality and antisymmetry, this minimum is unique. We conclude that whenever $\mathcal{K}$ is valid and $\mathcal{C}$ is feasible there is exactly one result that satisfies the conditions of Definition 7.1; the same is obviously true if $\mathcal{K}$ is not valid, or $\mathcal{C}$ is not feasible. Thus, there is a unique rational change operator with respect to Σ, $\preceq$. □

Proof of Propostion 8.1 According to Algorithm 1, each recursive call can return at lines 2, 5, 13 or 15. If any specific call returns through line 15, it means that it had spawned at least one new call to *Change* (in line 10). If any call returns through lines 2 or 13, then it does not spawn a new call and it returns Δ_∞. If any call returns through line 5, then it does not spawn a new call and it returns some $\Delta \neq \Delta_\infty$. We will call the recursive calls returning through line 15 "intermediate" calls, the calls returning through lines 2 or 13 "rejected", and the calls returning through line 5 "accepted". □

Moreover, we set $\Delta_{\min} = \Delta(\mathcal{K}, \mathcal{K} \bullet \mathcal{C})$.
For presentation clarity, we will show some intermediate conclusions before showing the proposition. The proposition follows from conclusions #5, #6 below.

Conclusion #1: If $\Delta_{\text{out}} \neq \Delta_\infty$ is returned, then there is at least one accepted call.

Proof If all the calls are either rejected or intermediate, then all executions of line 10 in intermediate calls will compare Δ_∞ with Δ_∞; thus all intermediate calls will also return Δ_∞. Thus, the algorithm will return Δ_∞, a contradiction by our hypothesis. Therefore, there is at least one accepted call. □

Conclusion #2: If an accepted call returns Δ_{out}, then $\Delta_{\text{out}} \neq \Delta_\infty$, $\mathcal{K} + \Delta_{\text{out}} \vdash \mathcal{C}$, $\mathcal{K} + \Delta_{\text{out}} \vdash \Sigma$, and $\Delta_{\min} \preceq_{\mathcal{K}} \Delta_{\text{out}}$.

Proof In any given accepted call, the condition of line 4 is true and the returned delta will be $\Delta_{\text{out}} = \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) \neq \Delta_\infty$. It is trivial to show that $\mathcal{K} + \Delta_{\text{out}} = \mathcal{K} + \mathcal{C}_{\text{rem}}$. Thus, by the fact that the condition of line 4 is true, we conclude that $\mathcal{K} + \Delta_{\text{out}} \vdash \Sigma$. Moreover, note that we never remove ground facts from $\mathcal{C}_{\text{rem}}$, thus, given that the initial value of $\mathcal{C}_{\text{rem}}$ (in the first

Author's personal copy

recursive call) is $\mathcal{C}$, we conclude that in any recursive call, it holds that $\mathcal{C}_{\text{rem}} \supseteq \mathcal{C}$. Thus, by our hypotheses and the definition of raw application, $\mathcal{K} + \Delta_{\text{out}} = \mathcal{K} + \mathcal{C}_{\text{rem}} \vdash \mathcal{C}$. Given the above facts and the definition of the rational change operator, the condition $\Delta_{\text{min}} \leq_{\mathcal{K}} \Delta_{\text{out}}$, holds. □

Conclusion #3: If $\Delta_{\text{out}} \neq \Delta_{\infty}$ is returned, then $\mathcal{K} + \Delta_{\text{out}} \vdash \mathcal{C}$ and $\mathcal{K} + \Delta_{\text{out}} \vdash \Sigma$. Furthermore, $\Delta_{\text{min}} \leq_{\mathcal{K}} \Delta_{\text{out}}$.

Proof By conclusion #1, there is at least one accepted call. If the first call to *Change* is an accepted one, then the result is obvious by conclusion #2. If the first call to *Change* is a rejected call, then there is no accepted call (a contradiction). If the first call is an intermediate call, then it will return the preferred of all deltas returned through line 10. Given that all deltas are preferred over Δ_{∞}, it follows that the returned delta is a delta returned by one of the accepted calls, so the result follows from conclusion #2. Thus, the conclusion holds in all cases. □

Conclusion #4: If $\mathcal{C}$ is feasible, then there is at least one accepted call which returns Δ_{min}.

Proof If the first call is a rejected call, then the condition of line 1 is true. Note that, since $\Delta_{pref} = \Delta_{\infty}$, it cannot be the case that $\Delta_{pref} \leq_{\mathcal{K}_0} \Delta(\mathcal{K}_0, \mathcal{K}_0 + \mathcal{C}_{\text{rem}})$. Thus there is some g such that $g, \neg g \in \mathcal{C}_{\text{rem}} = \mathcal{C}$, i.e., $\mathcal{C}$ is infeasible, a contradiction. □

If the first call is an accepted call, then for the returned delta (Δ_{out}), it holds that $\Delta_{\text{out}} = \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C})$. Take some ground fact $g \in \Delta_{\text{out}}$; then $g \notin \mathcal{K}$, but $g \in \mathcal{K} + \mathcal{C}$, so $g \in \mathcal{C}$, i.e., by the Principle of Success, $g \in \mathcal{K} \bullet \mathcal{C}$, thus, by definition, $g \in \Delta_{\text{min}}$. Using similar arguments, we can show that if $\neg g \in \Delta_{\text{out}}$, then $\neg g \in \Delta_{\text{min}}$. Thus, $\Delta_{\text{out}} \subseteq \Delta_{\text{min}}$, so $\Delta_{\text{out}} \leq_{\mathcal{K}} \Delta_{\text{min}}$, by the monotonicity of $\leq_{\mathcal{K}}$. By conclusion #3, $\Delta_{\text{min}} \leq_{\mathcal{K}} \Delta_{\text{out}}$. By antisymmetry of $\leq_{\mathcal{K}}$, we conclude that $\Delta_{\text{out}} = \Delta_{\text{min}}$.

If the first call is an intermediate call, then it will pass through line 7. Suppose that line 7 selected ground fact $g_1 \in \mathcal{C}$ for which there is a constraint instance $\sigma_1(\vec{a}_1)$, such that $\neg g_1 \in \mathcal{S}_1$ for some $\mathcal{S}_1 \in Res(\sigma_1(\vec{a}_1))$ and $\mathcal{K} + \mathcal{C} \nvdash \sigma_1(\vec{a}_1)$.

Given that $\mathcal{K} \bullet \mathcal{C} \vdash \sigma_1(\vec{a}_1)$, there is some $\mathcal{S}'_1 \in Res(\sigma_1(\vec{a}_1))$ such that $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{S}'_1$. Suppose that $\mathcal{S}_1 = \mathcal{S}'_1$. Then $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{S}_1$. But $\neg g_1 \in \mathcal{S}_1$, so $\mathcal{K} \bullet \mathcal{C} \vdash \neg g_1$. On the other hand, $g_1 \in \mathcal{C}$ (by line 7), which is a contradiction because $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{C}$ (by the Principle of Success). Thus $\mathcal{S}_1 \neq \mathcal{S}'_1$, so the algorithm will pass through line 10 for $\mathcal{S}'_1$.

Let us consider the new recursive call to *Change* which uses $\mathcal{S}'_1$; for this call, it holds that $\mathcal{C}_{\text{rem}} = \mathcal{C} \cup \mathcal{S}'_1$. Take some $g \in \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}})$. Then, $g \notin \mathcal{K}$ so $g \in \mathcal{C}_{\text{rem}}$. If $g \in \mathcal{C}$, then $g \in \Delta_{\text{min}}$ (because $\mathcal{K} + \Delta_{\text{min}} = \mathcal{K} \bullet \mathcal{C} \vdash g$ by the Principle of Success and $g \notin \mathcal{K}$); similarly, if $g \in \mathcal{S}'_1$ then $g \in \Delta_{\text{min}}$ (because, by construction, $\mathcal{K} + \Delta_{\text{min}} = \mathcal{K} \bullet \mathcal{C} \vdash g$ and $g \notin \mathcal{K}$). Thus $g \in \Delta_{\text{min}}$; using similar arguments we can show that if $\neg g \in \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}})$ then $\neg g \in \Delta_{\text{min}}$. It follows that $\Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) \subseteq \Delta_{\text{min}}$. Consequently, $\Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) \leq_{\mathcal{K}} \Delta_{\text{min}}$.

Suppose that in this call, $g, \neg g \in \mathcal{C}_{\text{rem}}$. Then, since $\mathcal{C}$ is feasible to begin with, and the construction of $\mathcal{C}_{\text{rem}}$ it holds that $g \in \mathcal{C}$ and $\neg g \in \mathcal{S}'_1$ (or vice-versa). But this is impossible by the definition of $\mathcal{S}'_1$. Similarly, suppose that $\Delta_{pref} \leq_{\mathcal{K}} \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) \leq_{\mathcal{K}} \Delta_{\text{min}}$. Note that Δ_{pref} is either equal to Δ_{∞} (a contradiction), or it has been returned by an accepted call, so by Conclusion #2, it follows that $\Delta_{\text{min}} \leq_{\mathcal{K}} \Delta_{pref}$. Thus, $\Delta_{\text{min}} = \Delta_{pref}$ and Δ_{pref} has been returned by some accepted call, so the conclusion has been proven. Thus, let us suppose that the condition of line 1 is not true.

If the condition of line 4 is true, then the call is accepted and $\Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}})$ will be returned, so by Conclusion #2 $\Delta_{\text{min}} \leq_{\mathcal{K}} \Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}})$, and by the fact above (i.e.,

Author's personal copy

$\Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) \leq_{\mathcal{K}} \Delta_{\text{min}}$) and antisymmetry, it follows that $\Delta(\mathcal{K}, \mathcal{K} + \mathcal{C}_{\text{rem}}) = \Delta_{\text{min}}$, so the conclusion has been proven. Thus, let us suppose that the condition of line 4 is not true either.

Then, the algorithm will pass through line 7. As above, suppose that line 7 selected ground fact $g_2 \in \mathcal{C}_{\text{rem}}$ for which there is a constraint instance $\sigma_2(\vec{a_2})$, such that $\neg g_2 \in \mathcal{S}_2$ for some $\mathcal{S}_2 \in Res(\sigma_2(\vec{a_2}))$ and $\mathcal{K} + \mathcal{C}_{\text{rem}} \nvdash \sigma_2(\vec{a_2})$.

Given that $\mathcal{K} \bullet \mathcal{C} \vdash \sigma_2(\vec{a_2})$, there is some $\mathcal{S}_2' \in Res(\sigma_2(\vec{a_2}))$ such that $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{S}_2'$. Suppose that $\mathcal{S}_2 = \mathcal{S}_2'$. Then $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{S}_2$. But $\neg g_2 \in \mathcal{S}_2$, so $\mathcal{K} \bullet \mathcal{C} \vdash \neg g_2$. On the other hand, by line 7, $g_2 \in \mathcal{C}_{\text{rem}} = \mathcal{C} \cup \mathcal{S}_1'$. If $g_2 \in \mathcal{C}$, then we have a contradiction because $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{C}$ (by the Principle of Success). If $g_2 \in \mathcal{S}_1'$, then by the definition of $\mathcal{S}_1'$, $\mathcal{K} \bullet \mathcal{C} \vdash \mathcal{S}_1'$. So, in both cases, $\mathcal{K} \bullet \mathcal{C} \vdash g_2$, a contradiction. Thus, $\mathcal{S}_2 \neq \mathcal{S}_2'$, so the algorithm will pass through line 10 for $\mathcal{S}_2'$.

Therefore, this call is an intermediate call, and consider the new recursive call to *Change* (spawned through line 10) which uses $\mathcal{S}_2'$ (as in the previous case). Repeating the same arguments as above, we conclude that the new call is either a rejected call (in which case the conclusion is proven, because there is some other accepted call returning Δ_{min}—see above), or it is an accepted call that returns Δ_{min}, or it is an intermediate call; in the latter case, there is one spawned call for which the above properties hold, so we can repeat the above argumentation. Given that the algorithm terminates, there will be a finite number of intermediate calls; thus, repeating the argument a finite number of times, we will eventually reach an accepted or rejected call, so the conclusion is proved.

Conclusion #5: If $\mathcal{C}$: feasible, then $\Delta_{\text{out}} \neq \Delta_{\infty}$ and $\mathcal{K} + \Delta_{\text{out}} = \mathcal{K} \bullet \mathcal{C}$.

Proof By conclusion #4, there is at least one accepted call which returns Δ_{min}. For any other accepted call, it holds (by Conclusion #2), that the produced delta is less preferable (according to $\leq_{\mathcal{K}}$) than Δ_{min}, so it will be eventually rejected when compared with Δ_{min} (either in line 1 or in line 10). Thus, $\Delta_{\text{out}} = \Delta_{\text{min}}$. The result now follows easily. □

Conclusion #6: If $\mathcal{C}$: infeasible then $\Delta_{\text{out}} = \Delta_{\infty}$.

Proof Suppose that there is an accepted call which returns Δ_{out}. Then, by Conclusion #2, $\mathcal{K} + \Delta_{\text{out}} \vdash \Sigma$, $\mathcal{K} + \Delta_{\text{out}} \vdash \mathcal{C}$, so by Definition 5.1, $\mathcal{C}$ is feasible, a contradiction. So, there is no accepted call, i.e., all calls are either intermediate or rejected. Therefore, it is easy to see that the algorithm will return Δ_{∞}.

Conclusions #5, #6 show the proposition. □

Proof of Propostion 9.1 Firstly, we note that the shortlex order ($\leq_{\mathbf{UL}}$) is obviously total, transitive, antisymmetric and wellfounded[9] (i.e., a wellorder). The same is true for $\leq_P$. Given these, and the definition of $\leq_G$ through Table 6, it is trivial to see that $\leq_G$ is also total, transitive and antisymmetric. For wellfoundedness, consider a set of (possibly negated) ground facts $G = \{g_i\}$ and suppose that G has no minimal. By antisymmetry, it follows that there is an infinite sequence $g_1, g_2, \ldots$ such that $g_1 >_G g_2 >_G \ldots$. Since there is a finite number of predicates, there will be some index, say n_1, such that all g_i for $i > n_1$ use the same (possibly negated) predicate. Furthermore, $\mathcal{K}$ is finite, as well as valid, so the subsumption relationships are acyclic ($R10.2$, $R11.2$); thus all distances are finite. Therefore, there will be some index, say n_2, such that all g_i for $i > n_2$ use the same (possibly negated) predicate, and

[9] http://en.wikipedia.org/wiki/Lexicographical_order.

Author's personal copy

the constant(s) used in this predicate has (have) the same $Dist$. Thus, the comparison boils down to comparing the constants using $\leq_{\mathbf{UL}}$, which is wellfounded, so there is a minimum. Concluding, $\leq_G$ is a wellorder.

Now consider some valid RDF/S KB $\mathcal{K}$. To show that $\preceq$ is a selection mechanism, we need to show that $\leq_{\mathcal{K}}$ is change-generating.

Totality: consider Δ_1, Δ_2. If there is some (possibly negated) predicate q such that $|\Delta_1^q| \neq |\Delta_2^q|$ then the result is obvious. If $|\Delta_1^q| = |\Delta_2^q|$ for all q, then, if $\Delta_1 = \Delta_2$, the result is obvious. So suppose that $|\Delta_1^q| = |\Delta_2^q|$ for all q and $\Delta_1 \neq \Delta_2$. Then, set $\Delta = (\Delta_1 \setminus \Delta_2) \cup (\Delta_2 \setminus \Delta_1)$. It follows that $\Delta \neq \emptyset$. Given that $\leq_G$ is a wellorder, there is exactly one minimum (according to $\leq_G$) in Δ, say g. If $g \in \Delta_1 \setminus \Delta_2$, then $\Delta_1 <_{\mathcal{K}} \Delta_2$, whereas if $g \in \Delta_2 \setminus \Delta_1$, then $\Delta_2 <_{\mathcal{K}} \Delta_1$.

Antisymmetry: consider Δ_1, Δ_2 such that $\Delta_1 \leq_{\mathcal{K}} \Delta_2$ and $\Delta_2 \leq_{\mathcal{K}} \Delta_1$, and suppose that $\Delta_1 \neq \Delta_2$. If there is some (possibly negated) predicate q such that $|\Delta_1^q| \neq |\Delta_2^q|$ and for all predicates q' such that $q <_P q'$ it holds that $|\Delta_1^{q'}| = |\Delta_2^{q'}|$, then either $\Delta_1 <_{\mathcal{K}} \Delta_2$ or $\Delta_2 <_{\mathcal{K}} \Delta_1$, a contradiction. So, $|\Delta_1^q| = |\Delta_2^q|$ for all q. Consider the set $\Delta = (\Delta_1 \setminus \Delta_2) \cup (\Delta_2 \setminus \Delta_1)$. Since $\Delta_1 \neq \Delta_2$, it follows that $\Delta \neq \emptyset$. Since $\leq_G$ is a wellorder, there is exactly one minimum (according to $\leq_G$) in Δ, say g. If $g \in \Delta_1 \setminus \Delta_2$, then $\Delta_1 <_{\mathcal{K}} \Delta_2$, whereas if $g \in \Delta_2 \setminus \Delta_1$, then $\Delta_2 <_{\mathcal{K}} \Delta_1$, both of which contradict with our hypothesis. Thus, $\Delta_1 = \Delta_2$.

Transitivity: consider Δ_1, Δ_2, Δ_3 such that $\Delta_1 \leq_{\mathcal{K}} \Delta_2$ and $\Delta_2 \leq_{\mathcal{K}} \Delta_3$.

If $\Delta_1 = \Delta_2$ or $\Delta_2 = \Delta_3$, then the result is obvious.

Suppose that (a) there is some predicate q such that $|\Delta_1^q| < |\Delta_2^q|$ and for all predicates q_0 such that $q <_P q_0$ it holds $|\Delta_1^{q_0}| = |\Delta_2^{q_0}|$ and (b) there is some predicate q' such that $|\Delta_2^{q'}| < |\Delta_3^{q'}|$ and for all predicates q_0' such that $q' <_P q_0'$ it holds $|\Delta_2^{q_0'}| = |\Delta_3^{q_0'}|$. Then, obviously, there is some predicate q'' (which is either q or q', depending on whether $q <_P q'$ or $q' <_P q$ or $q = q'$) such that $|\Delta_1^{q''}| < |\Delta_3^{q''}|$ and for all predicates q_0'' such that $q'' <_P q_0''$ it holds $|\Delta_1^{q_0''}| = |\Delta_3^{q_0''}|$. Thus, $\Delta_1 \leq_{\mathcal{K}} \Delta_3$.

Suppose now that (a) there is some predicate q such that $|\Delta_1^q| < |\Delta_2^q|$ and for all predicates q_0 such that $q <_P q_0$ it holds $|\Delta_1^{q_0}| = |\Delta_2^{q_0}|$ and (b) that for all predicates q' it holds that $|\Delta_2^{q'}| = |\Delta_3^{q'}|$. Then, $|\Delta_1^q| < |\Delta_3^q|$ and for all predicates q_0 such that $q <_P q_0$ it holds that $|\Delta_1^{q_0}| = |\Delta_3^{q_0}|$ so $\Delta_1 \leq_{\mathcal{K}} \Delta_3$.

Similarly, if we suppose that (a) for all predicates q it holds that $|\Delta_1^q| = |\Delta_2^q|$ and (b) that there is some predicate q' such that $|\Delta_2^{q'}| < |\Delta_3^{q'}|$ and for all predicates q_0' such that $q' <_P q_0'$ it holds that $|\Delta_2^{q_0}| = |\Delta_3^{q_0'}|$, then we can show that $\Delta_1 \leq_{\mathcal{K}} \Delta_3$.

Now, consider the case that for all predicates q it holds that $|\Delta_1^q| = |\Delta_2^q| = |\Delta_3^q|$ and $\Delta_1' \neq \Delta_2'$ and $\Delta_2' \neq \Delta_3'$. Then the comparison boils down to comparing ground facts, and the ground facts that are relevant for the comparison of Δ_1, Δ_2, Δ_3 are those that belong to at least one, but not all, of Δ_1, Δ_2, Δ_3, so set $\Delta_0 = (\Delta_1 \cup \Delta_2 \cup \Delta_3) \setminus (\Delta_1 \cap \Delta_2 \cap \Delta_3)$. Since Δ_1, Δ_2, Δ_3 are different, it follows that $\Delta_0 \neq \emptyset$. Suppose that g_0 is the minimal (with respect to $\leq_G$) element of Δ_0; then, by the construction of Δ_0, there are some $i, j \in \{1, 2, 3\}$, $i \neq j$ such that $g_0 \in \Delta_i$, $g_0 \notin \Delta_j$, i.e., $g_0 \in \Delta_i \setminus \Delta_j$. Given that g_0 is minimal (with respect to $\leq_G$) in Δ_0, $g_0 <_{\mathcal{K}} g'$ for all $g' \in \Delta_j \setminus \Delta_i \subseteq \Delta_0$, so $\Delta_i \leq_{\mathcal{K}} \Delta_j$. Thus, $g_0 \in \Delta_i \setminus \Delta_j$ implies $\Delta_i \leq_{\mathcal{K}} \Delta_j$. Given this argumentation, and the fact that $\Delta_1 <_{\mathcal{K}} \Delta_2$, $\Delta_2 <_{\mathcal{K}} \Delta_3$ (by our hypotheses and the antisymmetry property of $\leq_{\mathcal{K}}$), it follows that:

- If $g_0 \in \Delta_1$ then:
  - If $g_0 \notin \Delta_3$ then $\Delta_1 \leq_{\mathcal{K}} \Delta_3$.
  - If $g_0 \in \Delta_3$ then $g_0 \notin \Delta_2$, so $\Delta_3 <_{\mathcal{K}} \Delta_2$, a contradiction.

Author's personal copy

– If $g_0 \notin \Delta_1$ then:

  - If $g_0 \in \Delta_2$ then $\Delta_2 \leq_{\mathcal{K}} \Delta_1$, a contradiction.
  - If $g_0 \notin \Delta_2$ then $g_0 \in \Delta_3$, so $\Delta_3 \leq_{\mathcal{K}} \Delta_2$, a contradiction.

Thus, $\Delta_1 \leq_{\mathcal{K}} \Delta_3$.

Wellfoundedness: consider any set of deltas $\mathcal{Z} = \{\Delta_i\}$ and suppose that $\mathcal{Z}$ has no minimal. Given the antisymmetry property of $\leq_{\mathcal{K}}$, this can happen only if $\mathcal{Z}$ is infinite and there is an infinite sequence, say $\Delta_1, \Delta_2, \ldots \in \mathcal{Z}$, such that $\Delta_{i+1} <_{\mathcal{K}} \Delta_i$ for all i.

Take Cl, the least preferred predicate according to $\leq_P$. Given that $\Delta_{i+1} <_{\mathcal{K}} \Delta_i$, it follows (by the definition of $\leq_{\mathcal{K}}$) that $|\Delta_{i+1}^{Cl}| \leq |\Delta_i^{Cl}|$. Moreover, each Δ_i is finite, so each Δ_i^{Cl} is finite also, so eventually, there will be some index n_{Cl} for which $|\Delta_{i+1}^{Cl}| = |\Delta_i^{Cl}|$ for all $i > n_{Cl}$. Using the same argumentation for the next predicate according to $\leq_P$ (namely Pr), and for indexes $i > n_{Cl}$, we can similarly find an index $n_{Pr} \geq n_{Cl}$ such that $|\Delta_{i+1}^{Pr}| = |\Delta_i^{Pr}|$ for all $i > n_{CS}$ (of course, it also holds that $|\Delta_{i+1}^{Cl}| = |\Delta_i^{Cl}|$ for all $i > n_{CS} \geq n_{Cl}$). Repeating the same process for all predicates, we will eventually find some index, say n such that for all $i > n$ and all predicates q it holds that $|\Delta_i^q| = |\Delta_{i+1}^q|$, i.e., for all $i, j > n$ it holds that $|\Delta_i| = |\Delta_j|$. Set $n^{\Delta} = |\Delta_i|$ for some $i > n$.

Now set $\Delta^1 = \bigcup_{i>n} \Delta_i \setminus \bigcap_{i>n} \Delta_i$. Since $\leq_G$ is wellfounded, Δ^1 has a minimal, say g_1. By the definition of Δ^1, there is some $k > n$ such that $g_1 \in \Delta_k$. Take any $m > k > n$. If $g_1 \notin \Delta_m$, then $\Delta_k <_{\mathcal{K}} \Delta_m$, a contradiction, so $g_1 \in \Delta_m$, for all $m > k$. Thus, there is some index, say $n_1 \geq n$ for which $g_1 \in \Delta_i$ for all $i > n_1$. Similarly, we define $\Delta^2 = \Delta^1 \setminus \{g_1\}$, find the minimal of Δ^2, say g_2, and prove that there is some index, say $n_2 \geq n_1$ for which $g_2 \in \Delta_i$ for all $i > n_2$. Continuing this recursive process, for any $k > 1$, we define the set $\Delta^k = \Delta^{k-1} \setminus \{g_{k-1}\}$ (where g_{k-1} is the minimal of Δ^{k-1}), we find the minimal, say g_k, and prove that there is some index, say $n_k \geq n_{k-1}$ for which $g_k \in \Delta_i$ for all $i > n_k$.

If there is some k for which $\Delta^k = \emptyset$, then Δ^1 was finite to begin with; given that each Δ_i is finite, $\bigcap_{i>n} \Delta_i$ is finite, thus, since Δ^1 is finite it follows that $\bigcup_{i>n} \Delta_i$ is finite also, which implies that not all Δ_i can be different, i.e., there is some index m for which $\Delta_i = \Delta_j$ for $i, j > m$, a contradiction. Therefore, this process can be performed for all $k = 1, 2, \ldots$.

Now set $j = n^{\Delta} + 1$; by construction, it follows that for any $i > n$ $j, g_1, \ldots, g_j \in \Delta_i$ (where $g_1, \ldots, g_j$ selected as above), and all the $g_1, \ldots, g_j$ are different, so $|\Delta_i| \geq j > n^{\Delta} = |\Delta_i|$, a contradiction. We conclude that there is no such sequence, i.e., we have reached a contradiction, so $\mathcal{Z}$ has a minimum.

Monotonicity: consider Δ_1, Δ_2 such that $\Delta_1 \subseteq \Delta_2$. If $\Delta_1 = \Delta_2$ then obviously $\Delta_1 \leq_{\mathcal{K}} \Delta_2$. If $\Delta_1 \subset \Delta_2$, then for all (possibly negated) predicates q it holds that $|\Delta_1^q| \leq |\Delta_2^q|$ and there is at least one (possibly negated) predicate q_0 such that $|\Delta_1^q| < |\Delta_2^q|$. Thus, $\Delta_1 \leq_{\mathcal{K}} \Delta_2$. □

Proof of Propostion 9.2 Take any valid KB $\mathcal{K}$ and feasible change request $\mathcal{C}$. Consider the sets $\Gamma_{\mathcal{K}}, \Gamma_{\mathcal{C}} \subseteq \mathbf{U} \cup \mathbf{L}$, which contain all the constants that appear in $\mathcal{K}$, $\mathcal{C}$, respectively, as well as the custom URI $\gamma \in \mathbf{U} \setminus Sp$ such that γ is the minimum, according to $\leq_{\mathbf{UL}}$ of all constants in $\mathbf{U} \setminus (\Gamma_{\mathcal{K}} \cup \Gamma_{\mathcal{C}} \cup Sp)$. Set $\Gamma = \Gamma_{\mathcal{K}} \cup \Gamma_{\mathcal{C}} \cup \{\gamma\}$.

Now consider a constraint instance $\sigma(\vec{a})$ from Table 4, such that $\vec{a}$ contains at least one constant that does not appear in $\mathcal{K}$ or $\mathcal{C}$. In order for $\mathcal{K} + \mathcal{C}$ to violate $\sigma(\vec{a})$ it should be the case that all the predicates in the antecedent of the constraint must be true. By the form of the constraints in Table 4 it follows that all the universally quantified variables in all constraints appear in predicates in the antecedent of said constraint. Combining these two facts, we can show that $\mathcal{K} + \mathcal{C} \vdash \sigma(\vec{a})$, because $\vec{a}$ contains at least one constant that does not appear in $\mathcal{K}$ or $\mathcal{C}$, so not all antecedents of $\sigma(\vec{a})$ can be true.

Author's personal copy

Given the modification of $Change_{RDF}$ in line 9, for all calls in $Change_{RDF}(\mathcal{C}_{\text{rem}}, \mathcal{K}_0, \Delta_{pref})$ it will hold that $\mathcal{C}_{\text{rem}}, \mathcal{K}_0, \Delta_{pref}$ contain only constants from Γ. Therefore, line 7 will never have to consider constraints containing constants that are not in Γ. Moreover, any given recursive branch cannot consider the same constraint instance twice, because, once we consider one constraint instance, it is resolved by adding side-effects to $\mathcal{C}_{\text{rem}}$; furthermore, we only add ground facts to $\mathcal{C}_{\text{rem}}$, and if we add contradictory ground facts, line 1 will stop the recursion, so a resolution cannot be undone in subsequent recursive calls. Since $\mathcal{K}, \mathcal{C}$ are finite, Γ is finite also, so, even if a recursive branch considers all possible constraint instances once, it will still have to consider a finite number of constraint instances, so it will have a finite length (equal to the number of different constraint instances considered). Similarly, due to the modification in line 9, the number of recursive branches spawned by each call will be finite. As a result, the total number of recursive calls is finite. Similarly, each individual call to $Change_{RDF}$ will terminate for the same reasons. We conclude that the call $Change_{RDF}(\mathcal{C}, \mathcal{K}, \Delta_\infty)$ terminates.

It remains to show that the algorithm returns the correct result upon termination. Using Proposition 8.1, it suffices to show that the modification of line 9 (i.e., ignoring some of the $\mathcal{S}' \in Res(\sigma(\vec{a}))$) does not jeopardize correctness, i.e., none of the ignored branches could lead to the correct result. Equivalently, we need to show that $\mathcal{K} \bullet \mathcal{C}$ contains only constants from Γ.

Suppose that this is not true, and that $\mathcal{K} \bullet \mathcal{C}$ contains at least one constant that does not appear in Γ. Set $\Delta = \Delta(\mathcal{K}, \mathcal{K} \bullet \mathcal{C})$, and let $\gamma_1, \gamma_2, \ldots, \gamma_m$ be the constants that appear in $\mathcal{K} \bullet \mathcal{C}$ but are not in Γ. Set Δ_γ the delta that occurs from Δ by replacing $\gamma_i \in \mathbf{U} \setminus Sp$ by γ and by dropping all ground facts that contain some $\gamma_j \notin \mathbf{U} \setminus Sp$.

By definition, $\mathcal{K} + \Delta \vdash \mathcal{C}$. Given that Δ_γ occurs from Δ by editing the ground facts that contain $\gamma_1, \gamma_2, \ldots, \gamma_m$ only, and that it replaces (some of) these constants with γ, as well as the fact that $\gamma, \gamma_1, \gamma_2, \ldots, \gamma_m$ do not appear in $\mathcal{C}$ by definition, the above fact implies that $\mathcal{K} + \Delta_\gamma \vdash \mathcal{C}$.

Similarly, $\mathcal{K}+\Delta \vdash \Sigma$ implies that $\mathcal{K}+\Delta_\gamma \vdash \Sigma$. This is true, intuitively, because all invalidities can be resolved using ground facts from $\Gamma_\mathcal{K} \cup \Gamma_\mathcal{C}$ only, except from the invalidities that involve the existential quantifier, for which any other custom URI can be used (but γ is preferred, by definition). More formally, let's assume, for the sake of contradiction, that there is some constraint instance $\sigma(\vec{a})$ such that $\mathcal{K}+\Delta_\gamma \nvdash \sigma(\vec{a})$. If $\sigma(\vec{a}) = R1.1(A)$ then $Cl(A) \in \mathcal{K}+\Delta_\gamma$ and A is not a custom URI. If $A \in \Gamma_\mathcal{K} \cup \Gamma_\mathcal{C}$ then by the construction of Δ_γ it follows that $Cl(A) \in \mathcal{K}+\Delta$ as well, i.e., $\mathcal{K}+\Delta \nvdash R1.1(A)$, a contradiction. If $A = \gamma$ then γ is a custom URI by definition, so $R1.1(A)$ is not violated. If $A \notin \Gamma$ then it cannot be the case that $Cl(A) \in \mathcal{K}+\Delta_\gamma$ because A does not appear in $\mathcal{K}$ or in Δ_γ. We conclude that, in any case, $\mathcal{K}+\Delta_\gamma \vdash R1.1(A)$. We can use similar arguments for all constraints in Table 4. Thus, $\mathcal{K}+\Delta_\gamma \vdash \Sigma$.

Finally, it is easy to note that for all predicates q it holds that $|\Delta_\gamma^q| \leq |\Delta^q|$. If $|\Delta_\gamma^q| < |\Delta^q|$ for some q, then $\Delta_\gamma <_\mathcal{K} \Delta$, a contradiction by the definition of Δ. So suppose that $|\Delta_\gamma^q| = |\Delta^q|$ for all q. Then, all γ_i are custom URIs. In addition, $Dist(\gamma) = Dist(\gamma_i) = 0$, because $\gamma, \gamma_1, \gamma_2, \ldots, \gamma_m$ do not appear in $\mathcal{K}$. Therefore, the comparison between Δ, Δ_γ boils down to comparing the constants using $\leq_{\mathbf{UL}}$; by the definition of γ, this implies that $\Delta_\gamma <_\mathcal{K} \Delta$. This is a contradiction by the definition of Δ. We conclude that $\mathcal{K} \bullet \mathcal{C}$ contains only constants from Γ. The result follows from Proposition 8.1. □

References

1. Afrati F, Kolaitis P (2009) Repair checking in inconsistent databases: algorithms and complexity. In: Proceedings of the 12th international conference on database theory (ICDT-09)
2. Akhtar W, Cortés-Calabuig A, Paredaens J (2010) Constraints in rdf. In: Proceedings of the 4th international conference on semantics in data and knowledge bases, SDKB'10, pp 23–39

Author's personal copy

3. Alexopoulos P, Wallace M, Kafentzis K, Askounis D (2011) Ikarus-onto: a methodology to develop fuzzy ontologies from crisp ones. Knowl Inf Syst 1–29. doi:10.1007/s10115-011-0457-6
4. Arenas M, Bertossi L, Chomicki J (1999) Consistent query answers in inconsistent databases. In: Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS-99), pp 68–79
5. Bancilhon F, Spyratos N (1981) Update semantics of relational views. ACM Trans Database Syst (TODS) 6(4):557–575
6. Bechhofer S, Horrocks I, Goble C, Stevens R (2001) Oiled: a reason-able ontology editor for the semantic web. In: Proceedings of advances in AI: joint German/Austrian conference on AI
7. Berners-lee T, Connolly D (2004) Delta: an ontology for the distribution of differences between rdf graphs. RDF Graphs, World Wide Web. http://www.w3.org/DesignIssues/Diff
8. Bertossi L, Chomicki J (2003) Query answering in inconsistent databases. Semant Interoper Integr 35(2):1–41
9. Bohannon P, Fan W, Flaster M, Rastogi R (2005) A cost-based model and effective heuristic for repairing constraints by value modification. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, pp 143–154
10. Brickley D, Guha R (2004) df vocabulary description language 1.0: Rdf schema. www.w3.org/TR/2004/REC-rdf-schema-20040210
11. Cali A, Gottlob G, Lukasiewicz T (2009) Datalog±: a unified approach to ontologies and integrity constraints. In: Proceedings of the international conference on database theory (ICDT-09)
12. Calì A, Gottlob G, Pieris A (2010) Advanced processing for ontological queries. Proc VLDB Endow 3:554–565
13. Chirkova R, Fletcher G (2009) Towards well-behaved schema evolution. In: 12th International workshop on the web and databases, WebDB
14. Chomicki J, Marcinkowski J (2005) Minimal-change integrity maintenance using tuple deletions. Inf Comput 197(1/2):90–121
15. Chomicki J, Marcinkowski J (2005) On the computational complexity of minimal-change integrity maintenance in relational databases. In: Bertossi L, Hunter A, Schaub T (eds) Inconsistency tolerance. LNCS, vol 3300. Springer, Heidelberg, pp 119–150
16. Cong G, Fan W, Geerts F, Jia X, Ma S (2007) Improving data quality: consistency and accuracy. In: Proceedings of the 33rd international conference on very large data bases (VLDB-07)
17. Dayal U, Bernstein P (1978) On the updatability of relational views. In: Proceedings of the 4th international conference on very large data bases, pp 368–377
18. Dayal U, Bernstein P (1982) On the correct translation of update operations on relational views. ACM Trans Database Syst (TODS) 7(3):381–416
19. De Giacomo G, Lenzerini M, Poggi A, Rosati R (2009) On instance-level update and erasure in description logic ontologies. J Logic Comput 19(5):745–770
20. Deutsch A (2009) FOL modeling of integrity constraints (dependencies). In: Liu L, Özsu Tamer M (eds) Encyclopedia of database systems, vol 2009, Part 6. Springer, New York, pp 1155–1161
21. Ding L, Finin T, Peng Y, Da Silva P, McGuinness D (2005) Tracking rdf graph provenance using rdf molecules. In: Proceedings of the 4th international semantic web conference (poster)
22. Drabent W, Maluszynski J (2010) Hybrid rules with well-founded semantics. Knowl Inf Syst (KAIS) 25:137–168
23. Embury S, Brandt S, Robinson J, Sutherland I, Bisby F, Gray W, Jones A, White R (2001) Adapting integrity enforcement techniques for data reconciliation. Inf Syst 26(8):657–689
24. Fan W, Geerts F, Jia X (2008) A revival of integrity constraints for data cleaning. Proc VLDB Endow (PVLDB) 1(2):1522–1523
25. Flouris G (2006) On belief change and ontology evolution. PhD thesis, University of Crete, Greece
26. Flouris G (2007) On the evolution of ontological signatures. In: Proceedings of the workshop on ontology evolution
27. Flouris G, Huang Z, Pan J, Plexousakis D, Wache H (2006) Inconsistencies, negations and changes in ontologies. In: Proceedings of the 21st national conference on artificial intelligence (AAAI-06), pp 1295–1300
28. Flouris G, Manakanatas D, Kondylakis H, Plexousakis D, Antoniou G (2008) Ontology change: classification and survey. Knowl Eng Rev 23(2):117–152
29. Flouris G, Plexousakis D, Antoniou G (2005) On applying the agm theory to dls and owl. In: Proceedings of the 4th international semantic web conference (ISWC-05), pp 216–231
30. Gabel T, Sure Y, Voelker J (2004) D3.1.1.a: Kaon-ontology management infrastructure. SEKT informal deliverable

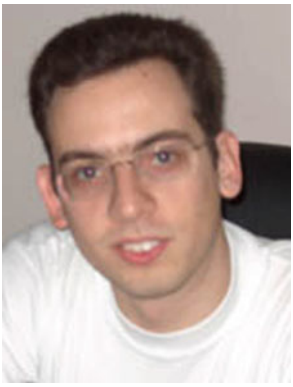
Author's personal copy

31. Gärdenfors P (1992) Belief revision. Cambridge University Press, chapter belief revision: an introduction, pp 1–28
32. Groza T, Grimnes G, Handschuh S, Decker S (2011) From raw publications to linked data. Knowl Inf Syst 1–21. doi:10.1007/s10115-011-0473-6
33. Gutierrez C, Hurtado C, Mendelzon A (2004) Foundations of semantic web databases. In: Proceedings of the 2004 ACM symposium on principles of database systems (PODS-04), pp 95–106
34. Gutierrez C, Vaisman A, Hurtado C (2011) Rdfs update: from theory to practice. In: Proceedings of the extended semantic web conference, ESWC-11
35. Haase P, Qi G (2007) An analysis of approaches to resolving inconsistencies in dl-based ontologies In: Proceedings of the international workshop on ontology dynamics (IWOD-07), pp 97–109
36. Haase P, van Harmelen F, Huang Z, Stuckenschmidt H, Sure Y (2005) A framework for handling inconsistency in changing ontologies. In: Proceedings of the 4th international semantic web conference (ISWC-05), pp 353–367
37. Keller A (1985) Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In: PODS, Vol 85, pp 154–163
38. Klein M, Noy N (2003) A component-based framework for ontology evolution. In: Proceedings of the workshop on ontologies and distributed systems
39. Konstantinidis G (2008) Belief change in semantic web environments. Master's thesis, University of Crete
40. Konstantinidis G, Flouris G, Antoniou G, Christophides V (2008) A formal approach for rdf/s ontology evolution. In: Proceedings of the 18th European conference on artificial intelligence
41. Kotidis Y, Srivastava D, Velegrakis Y (2006) Updates through views: a new hope. In: Proceedings of the 22nd international conference on data engineering (ICDE-06)
42. Kotis K, Vouros A (2006) Human-centered ontology engineering: the hcome methodology. Knowl Inf Syst 10:109–131
43. Lausen G, Meier M, Schmidt M (2008) Sparqling constraints for rdf. In: Proceedings of 11th international conference on extending database technology (EDBT-08), pp 499–509
44. Ludscher B, May W, Lausen G (1997) Referential actions as logical rules. In: Proceedings of the 16th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS-97)
45. Lusch U, Rudolph S, Vrandecic D (2009) Tempus fugit—towards an ontology update language. In: Proceedings of the 6th European semantic web conference (ESWC-09), pp 278–292
46. Magiridou M, Sahtouris S, Christophides V, Koubarakis M (2005) Rul: a declarative update language for rdf. In: Proceedings of the 4th international semantic web conference (ISWC-05)
47. Manola F, Miller E, McBride B (2004) Rdf primer. www.w3.org/TR/rdf-primer
48. Medeiros C, Tompa F (1986) Understanding the implications of view update policies. Algorithmica 1(1):337–360
49. Motik B, Horrocks I, Sattler U (2007) Bridging the gap between owl and relational databases. In: Proceedings of 17th international World Wide Web conference (WWW-07), pp 807–816
50. Noy N, Fergerson R, Musen M (2000) The knowledge model of protégé-2000: combining interoperability and flexibility. In: Proceedings of the 12th international conference on knowledge engineering and knowledge management (EKAW-00)
51. On B-W, Lee I, Lee D (2011) Scalable clustering methods for the name disambiguation problem. Knowl Inf Syst 31:1–23
52. Pan J, Horrocks I (2007) Rdfs(fa): connecting rdf(s) and owl dl. IEEE Trans Knowl Data Eng 19(2):192–206
53. Papavassiliou V, Flouris G, Fundulaki I, Kotzinos D, Christophides V (2009) On detecting high-level changes in rdf/s kbs. In: Proceedings of the 8th international semantic web conference (ISWC-09)
54. Pérez J, Arenas M, Gutierrez C (2006) Semantics and complexity of sparql. In: Proceedings of the 5th international semantic web conference (ISWC-06), pp 30–43
55. Serfiotis G, Koffina I, Christophides V, Tannen V (2005) Containment and minimization of rdf/s query patterns. In: Proceedings of the 4th international semantic web conference (ISWC-05)
56. Shu H (2000) Using constraint satisfaction for view update. J Intell Inf Syst 15:147–173
57. Stojanovic L, Maedche A, Motik B, Stojanovic N (2002) User-driven ontology evolution management. In: Proceedings of the 13th European conference on knowledge engineering and knowledge management (EKAW-02)
58. Stojanovic L, Motik B (2002) Ontology evolution within ontology editors. In: Proceedings of ontoWeb-SIG3 workshop
59. Sure Y, Angele J, Staab S (2003) Ontoedit: multifaceted inferencing for ontology engineering. J Data Semant 1(1):128–152
60. Tao J, Sirin E, Bao J, McGuinness D (2010) Extending owl with integrity constraints. In: Proceedings of the 23rd international workshop on description logics (DL-10), CEUR-WS 573

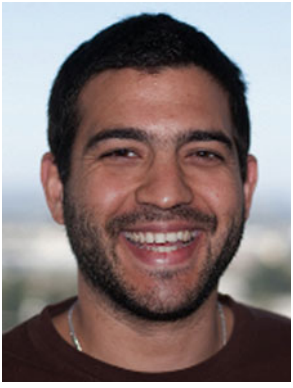
Author's personal copy

61. Umbrich J, Hausenblas M, Hogan A, Polleres A, Decker S (2010) Towards dataset dynamics: change frequency of linked open data sources. In: Proceedings of the WWW2010 workshop on linked data on the web (LDOW2010)
62. Volkel M, Winkler W, Sure Y, Kruk S, Synak M (2005) Semversion: a versioning system for rdf and ontologies. In: Proceedings of the 2nd European semantic web conference (ESWC-05)
63. Wang T (2006) Gauging ontologies and schemas by numbers. In: Proceedings of the 4th international EON workshop
64. Zeginis D, Tzitzikas Y, Christophides V (2011) On computing deltas of RDF/S knowledge bases. ACM Trans Web 5(3). doi:10.1145/1993053.1993056

Author Biographies



Giorgos Flouris is a Research and Development Engineer at FORTH-ICS. His research interests lie mainly in the areas of knowledge representation and reasoning, belief revision and the semantic web. He has a bachelor in Mathematics from the University of Athens and an M.Sc. and Ph.D. in Computer Science from the University of Crete. He was a post-doctoral research fellow at CNR, Italy. Giorgos has published several papers in peer-reviewed journals and conferences, and has received a number of fellowships and awards for his work. Giorgos has been involved in the EU projects PlanetData, APARSEN, KP-Lab and CASPAR, is a member of the steering committee of the International Workshop on Ontology Dynamics series and has co-edited the Special Issue on Recent Advances in Ontology Dynamics of the Journal of Logic and Computation.



George Konstantinidis is a Ph.D. Candidate at the Computer Science Department at the University of Southern California (USC) and a Research Assistant at the Information Sciences Institute (ISI) at USC. He studied Computer Science at the University of Crete and holds a M.Sc. degree on Computer Science from the University of Crete and (in collaboration with) the Foundation for Research and Technology-Hellas (FORTH-ICS). His research interests are in the intersection of Databases, Information Integration and the Semantic Web, and in particular Data Integration and Ontologies, Description Logics, Ontology Evolution, Ontology-Based Data Answering and Integration. He has been a USC Graduate Fellow for the year 2011–2012, during which his work has been supported by the (USC Graduate School Endowed) Myronis Scholarship. He has been a reviewer for IJCAI and TKDE.

Author's personal copy

Grigoris Antoniou is Professor of Computing at the University of Huddersfield, UK. Before that, he held professorial positions at Griffith University, Australia, University of Bremen, Germany, and the University of Crete, Greece. From 2004 to 2011 he was Head of the Information Systems Laboratory at FORTH. His research interests lie in knowledge and semantic technologies, and their application to web information systems and ambient intelligence. He has published over 200 technical papers in scientific journals and conferences, and is author of three books with international publishers (MIT Press, Addison-Wesley). He was General Chair of ESWC 2011 and PC Chair of ESWC 2010, ICEBE 2009 and SETN 06. Since 2008 he is member of the ECCAI Board. He has participated in a number of national and European projects, among them PlanetData and APARSEN. In recognition of his work, he was elected an ECCAI Fellow in 2006.

Vassilis Christophides is a Professor of Computer Science at the University of Crete and affiliated researcher at FORTH-ICS. His main research interests include Databases and Web Information Systems, Digital Libraries and Scientific Systems. He has extensive experience in the scientific coordination of national and European research projects at FORTH-ICS. He has published over 100 articles in high-quality international conferences, workshops and journals. He has received the 2004 SIGMOD Test of Time Award and the Best Paper Award at the 2nd and 6th International Semantic Web Conference in 2003 and 2007.