Towards Well-Behaved Schema Evolution

Rada Chirkova
Department of Computer Science
North Carolina State University, Raleigh, USA
chirkova@csc.ncsu.edu

George H.L. Fletcher
School of Engineering and Computer Science
Washington State University, Vancouver, USA
fletcher@vancouver.wsu.edu

ABSTRACT

We study the problem of schema evolution in the RDF data model. RDF and the RDFS schema language are W3C standards for flexibly modeling and sharing data on the web. Although schema evolution has been intensively studied in the database and knowledge representation communities, only recently has progress been made on the study of RDFS schema evolution. Indeed, the flexible nature of RDF poses novel challenges. In particular, since the data model does not strictly distinguish data from metadata, schema evolution is intimately related to data updates. A major issue encountered during RDFS database updates is a certain type of "nondeterminism" exhibited during schema evolution. In current solutions, such nondeterminism is handled by extra-logical rules or heuristics. Is it possible to characterize the class of RDFS updates which are well-behaved, i.e., with a well-defined semantics avoiding ad-hoc solutions? In this paper, we present our first steps in a project to formally reason about such issues in RDF schema evolution. Specifically, we introduce an effective notion of determinism in RDF schema evolution, formally characterize a large class of well-behaved updates on RDFS graphs with respect to this definition, and show that computing such updates is tractable via a polynomial time algorithm.

1. INTRODUCTION

RDF is a mature W3C standard [23] for flexibly modeling graph-like data, which is proving to be a popular and effective format for creating and sharing data on the web. As such, large collections of RDF data are becoming more common. A key feature of RDF is the lack of a strict distinction between data and metadata, in contrast to traditional data models. This feature makes RDF naturally suited for the full spectrum from unstructured to structured data. Indeed, alongside semistructured data, it is often the case that traditional hierarchical and relational data are also encoded and shared in RDF [18]. (Please see the case study given in the Appendix, Section A.1.)

In this paper, we study the problem of schema evolution in the RDF data model. The W3C standards provide a basic vocabulary, the RDF schema language (RDFS) [25], for indicating how some data objects are to be interpreted as structural metadata (i.e., as schema elements). In this paper, we focus our attention on schema evolution in RDF data conforming to the RDFS standard. Although schema evolution has been intensively studied in the database [21] and knowledge engineering [12, 20] communities, only recently has progress been made on the study of RDFS schema evolution [14, 15]. The characteristics of RDF data pose novel challenges. In particular, since the data model blurs data and metadata, the issue of schema evolution in RDF is intimately related to data updates.

EXAMPLE 1.1. *An RDFS database, often called a "graph," is essentially a collection of assertions encoded as "triples" (we give formal definitions in Section 3 below). For example, consider the set of triples $G = \{t_1, t_2, t_2, t_4\}$, where*

$$\begin{aligned} t_1 &= (\texttt{loves}, \texttt{subproperty}, \texttt{isFondOf}) \\ t_2 &= (\texttt{jack}, \texttt{loves}, \texttt{jill}) \\ t_3 &= (\texttt{jack}, \texttt{isFondOf}, \texttt{jill}) \\ t_4 &= (\texttt{jill}, \texttt{detests}, \texttt{jack}). \end{aligned}$$

RDFS associates a special semantics with the atom `subproperty`. *In particular, from assertions t_1 and t_2, the semantics of this "keyword" permits us to* infer *triple t_3. To consider some updates of graph G, next suppose we desire to remove the assertion t_4 from G. Here, we face no problems; we simply remove t_4 from G. If, however, we want to remove t_3, we are faced with a choice: we must remove t_3 and one or both of t_1 and t_2. Which alternative do we choose? Is the choice arbitrary, or is there some way to systematically choose which set of triples to delete from G, so as to remove assertion t_3? It turns out that this "nondeterminism" during deletion is inherent in updating RDFS databases.*

The semantics of RDFS keywords imposes constraints on RDFS graphs. In this sense, we argue that schema evolution in RDFS is essentially data evolution under constraints. Some updates, such as removing t_4 in Example 1.1 above, are trivial. However, the "nondeterminism" exhibited during other updates, such as during the removal of t_3 in Example 1.1, does not admit a well-behaved semantics. Consequently, in RDFS schema evolution solutions, such nondeterminism must be handled by extra-logical rules or heuristics (e.g., [14, 15]). Is it possible to characterize a broad class of RDFS updates which *are* well-behaved, i.e., those updates with a well-defined semantics avoiding ad-hoc solutions?

In this paper, we present our first steps towards a framework to formally reason about such questions, to contribute to the design of principled RDFS schema evolution solutions. In the following sections, we discuss closely related research (Section 2), and introduce basic notation and definitions (Section 3). We then proceed as follows:

- we begin by introducing the theory of database dependencies as a tool for reasoning about RDFS schema evolution problems (Section 4);
- following this, we formally define a notion of determinism in schema evolution (Section 5.1);
- next, we use this notion to precisely characterize a broad class of well-behaved updates on RDFS graphs (Section 5.2); and
- finally, we show that computing such updates is tractable via a polynomial time algorithm (Section 6).

We conclude in Section 7 with a discussion of ongoing and future research directions in this project.

2. RELATED WORK

The investigation we initiate in this paper builds on and complements a rich literature on schema and ontology evolution. Indeed, RDF schema evolution is intimately related to schema evolution and updates in traditional data models [2, 21] as well as to ontology evolution in richer knowledge representation systems [12, 20].[1] Each of these topics is quite mature, and therefore we indicate here only selected recent references to most closely related research.

To our knowledge, the state of the art on systematic studies of RDFS evolution are [14] and [15]. The present study directly complements the results of these papers. In particular, whereas [15] takes a top-down, system-level approach to RDFS schema evolution and [14] studies formal foundations for handling nondeterminism in RDFS schema evolution, we present in this paper our initial results in a bottom-up investigation into the space of well-behaved deterministic evolution of RDFS graphs.

Recent closely related studies of instance and schema level evolution under richer ontology languages include [8, 31]. As illustrated in Example 1.1, already non-trivial issues arise during schema evolution for "small" languages such as RDFS, the focus of the present study

3. PRELIMINARIES

We now introduce our basic notation and definitions.

3.1 RDFS graphs: data model

We introduce an abstraction of the RDF data model, following [7, 13]. We assume an enumerable set of *atoms* $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ (e.g., URIs, unicode literals) and an enumerable set of *blank nodes* $\mathcal{B} = \{A, B, C, \ldots\}$, such that $\mathcal{A} \cap \mathcal{B} = \emptyset$. An *RDFS graph* is a finite set $G \subseteq (\mathcal{A} \cup \mathcal{B}) \times \mathcal{A} \times (\mathcal{A} \cup \mathcal{B})$. A *triple* $(s, p, o) \in G$ is interpreted as the statement "subject s stands in relationship p to object o." In particular, the predicate p is "meta" data in this triple, in the sense that the triple can be interpreted as the statement "$(s, o) \in p$," for binary predicate p. However, p itself can appear as a subject or object elsewhere in the graph.

[1]We briefly discuss relational schema evolution in the Appendix, Section A.2.

The W3C standards associate a semantics with graphs which reinforces this reading of triples [24, 25]. In particular, graphs are viewed as first order formulas, a notion of graph interpretation is developed, and a corresponding entailment relation $\models$ is defined on graphs (in the standard sense that $G \models H$ if every model of G is a model of H). For a finite set of reserved keywords in $\mathcal{A}$, the standards associate a semantics enforced via $\models$. For example, keyword $\mathtt{sp}$, interpreted as "subproperty," is a transitive relation over "class properties." If graph G contains the triples $(a, \mathtt{sp}, b)$ and $(b, \mathtt{sp}, c)$, then the semantics of $\models$ ensures that $G \models (a, \mathtt{sp}, c)$.

Continuing this interpretation of graphs as first-order formulas, with each RDFS graph G we associate a Boolean query q_G whose body is a conjunction of the (representations of the) triples in G. In this representation, we use a single ternary (database) relation symbol g so that each RDF triple (s, p, o) is translated into subgoal $g(s, p, o)$ of the query. Each atom of G is represented by a constant in q_G, whereas each blank node of G is represented by a variable in q_G. We call q_G the *associated query* of G.

EXAMPLE 3.1. *Let $G_1 = \{(a, \mathtt{sp}, b), (b, \mathtt{sp}, c)\}$. Then*

$$q_{G_1}() \ :- \ g(a, \mathtt{sp}, b), g(b, \mathtt{sp}, c)$$

is the associated query of RDFS graph G_1. □

3.2 RDFS graphs: formal semantics

The purpose of this subsection is to define the "meaning" of an RDFS graph. Specifically, we spell out conditions under which two RDFS graphs represent the "same" graph. Observe that these conditions are essential for our being able to test whether the result of an RDFS schema update is unique. Consider an example.

EXAMPLE 3.2. *Let RDFS graph G_1 be as in Example 3.1, and let $G_2 = \{(a, \mathtt{sp}, b), (b, \mathtt{sp}, c), (a, \mathtt{sp}, c)\}$. G_1 and G_2 "represent the same" RDFS graph in presence of the following (transitivity) RDFS rule for* $\mathtt{sp}$ *[13, 24, 25]:*

$$\frac{(a, \mathtt{sp}, b) \quad (b, \mathtt{sp}, c)}{(a, \mathtt{sp}, c)}$$

□

We now provide formal definitions regarding the semantics of RDFS graphs. We consider a *schema language* to be a pair $(\mathcal{V}, \Delta)$, where $\mathcal{V} \subseteq \mathcal{A}$ is a finite set of keywords and Δ is a finite set of derivation rules in which the only constants mentioned are those occurring in $\mathcal{V}$. In this paper, we specifically study the language $\mathcal{L}_{RDFS} = (\mathcal{V}_{RDFS}, \Delta_{RDFS})$ with the set of RDFS keywords $\mathcal{V}_{RDFS} = \{\mathtt{type}, \mathtt{prop}, \mathtt{sp}, \mathtt{class}, \mathtt{sc}, \mathtt{dom}, \mathtt{range}\}$, and derivation rules Δ_{RDFS}, as given in [13]. For example, the transitivity rule of Example 3.2 for keyword $\mathtt{sp}$ is a member of Δ_{RDFS}. We discuss the rest of Δ_{RDFS} in Section 4. Provability ($\vdash$) using derivation rules, for various fragments and extensions of $\mathcal{L}_{RDFS}$, has been shown to be sound and complete with respect to corresponding notions of graph entailment ($\models$), in the sense that $G \models H$ if and only if $G \vdash H$, for RDFS graphs G and H [24, 28, 13, 19]. In particular, it has been shown that the semantics of RDFS graphs is completely defined in terms of Δ_{RDFS}, viewed as a proof system, in the sense that it holds for arbitrary RDFS graphs G and H that $G \models_{\mathcal{L}_{RDFS}} H$ if and only if $G \vdash_{\mathcal{L}_{RDFS}} H$ [24, 28].

We make the notion of provability precise, as follows [13].

Definition 3.1. *Let G, H be RDFS graphs. Define $G \vdash_{\mathcal{L}_{RDFS}} H$ if and only if there exists a sequence of graphs $G_1, \ldots, G_n$, for some $n > 1$, with $G_1 = G$ and $G_n = H$, and for each $1 < i \leqslant n$, one of the following holds:*

- *there exists a homomorphism from G_i to G_{i-1}, i.e., there exists a mapping $h : (\mathcal{A} \cup \mathcal{B}) \to (\mathcal{A} \cup \mathcal{B})$ such that $h|_{\mathcal{A}}$ is the identity and, for each $(s, p, o) \in G_i$ it is the case that $(h(s), h(p), h(o)) \in G_{i-1}$;*
- *$G_i \subseteq G_{i-1}$; or*
- *there is an instantiation $t_1, \ldots, t_k \to t$ of one of the derivation rules in Δ_{RDFS}, such that $\{t_1, \ldots, t_k\} \subseteq G_{i-1}$ and $G_i = G_{i-1} \cup \{t\}$.*

We now have the tools to give an operational semantics for RDFS graphs, via provability in $\mathcal{L}_{RDFS}$.

Definition 3.2. *Given an RDFS graph G, the Δ_{RDFS}-closure of G is the graph $\mathsf{cl}(G)$ obtained by repeated application of the rules of Δ_{RDFS} to G, adding new triples to G until no new triples are generated.*

We collect the following useful observations, which follow immediately from the Definitions 3.1 and 3.2.

Proposition 3.1. *For an arbitrary RDFS graph G:*

1. *$\mathsf{cl}(G)$ is unique, finite, and can be computed in time polynomial in the size of G,*
2. *$\mathsf{cl}(G) \supseteq G$,*
3. *$\mathsf{cl}(G) \models_{\mathcal{L}_{RDFS}} G$, and*
4. *$G \models_{\mathcal{L}_{RDFS}} \mathsf{cl}(G)$.*

Proof. (1) Clearly $\mathsf{cl}(G)$ is finite, since the RDFS rules only introduce a finite set of new atoms, and therefore the set of all possible derived triples is finite.[2]

Next, suppose $\mathsf{cl}(G)$ were not unique, i.e., there exist two distinct closure graphs C_1 and C_2. Without loss of generality, we can then suppose that there exists a triple $t \in C_1$ such that $t \notin C_2$. By definition of $\mathsf{cl}(G)$, there exists a proof $G \vdash_{\mathcal{L}_{RDFS}} t$. Since the same proof holds for all closures of G, we conclude that C_2 can not be a closure of G, a contradiction. Hence, our assumption is in error, and we conclude that $\mathsf{cl}(G)$ must be unique.

Finally, that $\mathsf{cl}(G)$ can be computed in $\mathcal{O}(|G|^2)$ time has been observed in [19, 28].

(2) By Definition 3.2, we have that $\mathsf{cl}(G)$ contains G.

(3) By Definition 3.1 and (2), we have that $\mathsf{cl}(G) \vdash_{\mathcal{L}_{RDFS}} G$. Since $\vdash_{\mathcal{L}_{RDFS}}$ is sound with respect to $\models_{\mathcal{L}_{RDFS}}$, we have that $\mathsf{cl}(G) \models_{\mathcal{L}_{RDFS}} G$.[3]

(4) By Definitions 3.1 and 3.2, we have that $G \vdash_{\mathcal{L}_{RDFS}} \mathsf{cl}(G)$. Since $\vdash_{\mathcal{L}_{RDFS}}$ is sound with respect to $\models_{\mathcal{L}_{RDFS}}$, we have that $G \models_{\mathcal{L}_{RDFS}} \mathsf{cl}(G)$. □

A variety of other normal forms for RDFS graphs have been proposed (e.g., [7, 13]). Most reasoning tasks associated with these alternate RDFS graph representations, however, are intractable. Computing Δ_{RDFS}-closures, on the other hand, is tractable, as we observed in Proposition 3.1. Hence, we take the *semantics of an RDFS graph G* to be $\mathsf{cl}(G)$. Determining entailment $G \models_{\mathcal{L}_{RDFS}} H$, however, is still intractable in the presence of blank nodes.

Proposition 3.2. *Given RDFS graphs G and H, it is the case that $G \models_{\mathcal{L}_{RDFS}} H$ iff q_H contains $q_{\mathsf{cl}(G)}$.*

This result follows directly from Proposition 3.1 and Definition 3.1. As an immediate corollary, we have from standard results on complexity of query containment [2] that

Corollary 3.1. *$\models_{\mathcal{L}_{RDFS}}$ is NP-complete.*

Note that Propositions 3.1 and 3.2 and Corollary 3.1 are variations of standard results on RDFS (e.g., [13]). We give them here in this form for their utility in the sequel.

We say that two RDFS graphs G and H are *equivalent*, denoted $G \equiv_{\mathcal{L}_{RDFS}} H$, when it holds that $G \models_{\mathcal{L}_{RDFS}} H$ and $H \models_{\mathcal{L}_{RDFS}} G$. In the absence of blank nodes, determining if $G \equiv_{\mathcal{L}_{RDFS}} H$ is tractable, since this amounts to computing and comparing the Δ_{RDFS}-closures of G and H. As observed above, however, in the presence of blank nodes, determining if $G \equiv_{\mathcal{L}_{RDFS}} H$ is, in general, intractable.

3.3 The Schema Evolution Problem

As we saw in Section 1, schema evolution in RDFS is induced by graph updates. In this section, we develop a general formulation of the schema evolution problem.

Let an *update action* α be one of $t^{[+]}$ or $t^{[-]}$ (denoting addition or deletion, respectively), for some triple t.[4] Our concern here is defining the semantics of updating a graph G with action α, which we denote (G, α). In other words, (G, α) is an RDFS graph capturing the update α on G. As we observed in Section 1, the impact of update actions can be quite subtle [14, 15, 20]. A primary issue is understanding the interaction of actions and Δ_{RDFS}. Our interest here is in characterizing *RDFS-expressible* update graphs, i.e., update graphs expressible as unique well-defined RDFS graphs.

Problem statement: Given an RDFS graph G and an update action α, does there exist an RDFS-expressible update graph (G, α)?

As we saw in Example 1.1, such update graphs do not always exist.

4. RDFS SEMANTICS VIA DEPENDENCIES

We next show how to recast the semantics of RDFS graphs in terms of data dependencies and chase [2]. This perspective is instrumental for establishing our main results.

4.1 Dependencies and chase

We assume reader familiarity with the basic notions of dependencies and chase; please see [2] for details. Here, we define the notions used in the sequel.

Query equivalence under dependencies. Given a query Q, we denote by $Q(D)$ the answer to Q on database D. Further, given a set Σ of embedded dependencies [2], we say that $D \models \Sigma$ if D satisfies Σ. For queries Q and P, we say that Q is *equivalent to P under* Σ (denoted $Q \equiv_\Sigma P$) if for every database D such that $D \models \Sigma$ we have $Q(D) = P(D)$. $Q \equiv P$ is defined as $Q \equiv_\Sigma P$ for $\Sigma = \emptyset$.

[2]In particular, note that $|\mathsf{cl}(G)| \leqslant (|\mathcal{A}(G)| + |\mathcal{B}(G)| + |\mathcal{V}_{RDFS}|)^3$.

[3]This is also clear at the semantic level, since by (2) we have that G is contained in its closure, and therefore any model of the closure must also be a model of G itself.

[4]For a brief discussion of evolution of relational data encoded in RDF, please see the Appendix, Section A.2.

Chase. Assume a conjunctive query *(CQ query)* $Q() :- \xi(\bar{X})$ and an embedded tuple-generating dependency *(tgd)* [2] $\sigma : \phi(\bar{U}, \bar{W}) \to \exists \bar{V}\ \psi(\bar{U}, \bar{V})$, such that atoms in σ may involve constants. (See Section 4.2.) Assume w.l.o.g. that Q has none of the variables $\bar{V}$. The *chase of Q with σ is applicable* if there is a homomorphism h from ϕ to ξ such that h cannot be extended to a homomorphism h' from $\phi \wedge \psi$ to ξ. In that case, a *chase step* of Q with σ and h is a rewrite of Q into $Q'() :- \xi(\bar{X}) \wedge \psi(h(\bar{U}), \bar{V})$.

Given a set Σ of tgds as described above, a Σ-*chase sequence* C is a sequence of CQ queries $Q_0, Q_1, \ldots$ such that every query Q_{i+1} ($i \geq 0$) in C is obtained from Q_i by a chase step $Q_i \Rightarrow^{\sigma} Q_{i+1}$ using a dependency $\sigma \in \Sigma$. A chase sequence $Q = Q_0, Q_1, \ldots, Q_n$ is *terminating* if $D^{(Q_n)} \models \Sigma$, where $D^{(Q_n)}$ is the canonical database for Q_n. In this case we say that $(Q)_\Sigma = Q_n$ is the (terminal) *result* of the chase.

Chase of CQ queries under sets of "weakly acyclic" tgds (without constants) terminates in finite time [10]. All chase results using arbitrary embedded dependencies, for a given CQ query, are equivalent in the absence of dependencies [10].

The following result is immediate from [2, 9, 10].

Theorem 4.1. *Given CQ queries Q_1, Q_2 and set Σ of embedded dependencies without constants. Then $Q_1 \equiv_\Sigma Q_2$ iff $(Q_1)_\Sigma \equiv (Q_2)_\Sigma$, assuming both chase results exist.* □

4.2 Δ_{RDFS} as tuple-generating dependencies

We use the W3C standards for the semantics of RDFS graphs [24] to come up with the following eight embedded tgds with constants from $\mathcal{V}_{RDFS}$.

$\sigma_1 : \forall x :\ g(x, \texttt{type}, \texttt{prop}) \to g(x, \texttt{sp}, x)$
$\sigma_2 : \forall x, y, z :\ g(x, \texttt{sp}, y) \wedge g(y, \texttt{sp}, z) \to g(x, \texttt{sp}, z)$
$\sigma_3 : \forall w, x, y, z :\ g(x, \texttt{sp}, y) \wedge g(z, x, w) \to g(z, y, w)$
$\sigma_4 : \forall x :\ g(x, \texttt{type}, \texttt{class}) \to g(x, \texttt{sc}, x)$
$\sigma_5 : \forall x, y, z :\ g(x, \texttt{sc}, y) \wedge g(y, \texttt{sc}, z) \to g(x, \texttt{sc}, z)$
$\sigma_6 : \forall x, y, z :\ g(x, \texttt{sc}, y) \wedge g(z, \texttt{type}, x) \to g(z, \texttt{type}, y)$
$\sigma_7 : \forall w, x, y, z :\ g(x, \texttt{dom}, y) \wedge g(z, x, w) \to g(z, \texttt{type}, y)$
$\sigma_8 : \forall w, x, y, z :\ g(x, \texttt{range}, y) \wedge g(z, x, w) \to g(w, \texttt{type}, y)$

Here, w, x, y, z are variables, and each of the remaining arguments (e.g., sp) is a constant in $\mathcal{V}_{RDFS}$. Each of σ_1 through σ_8 is interpreted on an RDFS graph G as "if, for some consistent set of variable bindings, each clause of the left-hand side of σ_i holds in G, then the right-hand side also holds in G." We refer to $\sigma_1, \ldots, \sigma_8$ collectively as Σ^*.

Generally, given a $\mathcal{V}$ we say that an embedded tgd with constants from $\mathcal{V}$ is a *keyword tgd* (w.r.t. $\mathcal{V}$). All the elements of Σ^* above are keyword tgds, w.r.t. $\mathcal{V}_{RDFS}$, by definition. To our knowledge, keyword tgds have not been singled out for study in the literature. (Cf. [11] for a state of the art discussion of dependencies with constants.) We say that a keyword tgd is *deterministic* if its left-hand side consists of a single relational atom, and is *nondeterministic* otherwise.

We are now ready to establish the main result of this section, which is that for an arbitrary RDFS graph G, with associated query q_G, computing $\mathsf{cl}(G)$ amounts to computing the terminal chase result $(q_G)_{\Sigma^*}$ using the set $\Sigma^* = \{\sigma_1, \ldots, \sigma_8\}$. This result allows us to prove the main result of this paper (Theorem 5.3, below).

We begin by formally associating each of the eight RDFS derivation rules of [13] with a separate keyword tgd in Σ^*. The bijective mapping, which we call μ^*, is straightforward (for instance, the transitivity rule for sp is associated with keyword tgd σ_2) and is omitted due to the space limit.

Proposition 4.1. *For an RDFS graph G and its associated query q_G, consider rule $r \in \Delta_{RDFS}$ such that for the keyword tgd $\sigma = \mu^*(r)$, $\sigma \in \Sigma^*$, chase of q_G with σ is applicable. Let q' be the result of applying σ to q_G. Then q' is the associated query of RDFS graph G' such that G' can be obtained from G in one entailment step using the rule r.*

The converse of Proposition 4.1 (i.e., constructing q' from G') also holds and is omitted due to the space limit.

As an immediate corollary, we obtain our main result:

Theorem 4.2. *Given an RDFS graph G and its associated query q_G, $q_{\mathsf{cl}(G)}$ and $(q_G)_{\Sigma^*}$ are isomorphic.*

In the remainder of the paper, we will use Theorem 4.2 to compute $\mathsf{cl}(G)$ via $(q_G)_{\Sigma^*}$. Note that it is immediate from Theorem 4.2 that for an arbitrary RDFS graph G, the terminal chase result $(q_G)_{\Sigma^*}$ always exists and is unique (it is enough to recall from Proposition 3.1 that $\mathsf{cl}(G)$ is unique and finite). Interestingly, existence and uniqueness of $(q_G)_{\Sigma^*}$ can be obtained independently by reasoning on *arbitrary* (as opposed to just RDFS) graphs and on *arbitrary* schema languages $\mathcal{L} = (\mathcal{V}, \Delta)$, as long as Δ in $\mathcal{L}$ can be associated (analogously to our association μ^* between Δ_{RDFS} and Σ^*) with a set of only full (and possibly keyword) tgds and (possibly keyword) egds on a database schema with a single relational atom. The latter observation is the first example of our use of generic tools (specifically $\mathcal{L}$ and database chase) to solve our RDFS-specific problem.

5. WELL-BEHAVED SCHEMA UPDATES

Towards resolving the schema update problem introduced in Section 3.3, we next introduce a broad class of updates, and show that they are well-behaved in the sense that they result in unique and well defined RDFS graphs. The crucial notion is that of *determinism*.

5.1 Varieties of determinism

The following notion of a derivation will prove essential.

Definition 5.1. *Let G be an RDFS graph and $t \in \mathsf{cl}(G)$. A* derivation *of t is a finite sequence of Δ_{RDFS} rules, such that matches, via some fixed mapping μ, for all atoms of each rule in the sequence are in $\mathsf{cl}(G)$, with t being the final right hand side inference. We call the derivation* deterministic *if each rule applied is deterministic. Finally, t is said to be* non-derivable *if $(G - \{t\}) \not\models_{\mathcal{L}_{RDFS}} t$.*

We first introduce a strong notion of determinism.

Definition 5.2. *Let G be an RDFS graph and $t \in \mathsf{cl}(G)$. If every possible derivation of t in G is deterministic, we say t is* strictly deterministic *in G. Otherwise, we say t is* strictly nondeterministic *in G.*

EXAMPLE 5.1. *We give an example to illustrate strict determinism and its limitations. Let $G = \{t_1, t_2, t_3\}$ where*

$$t_1 = (\texttt{a}, \texttt{type}, \texttt{class})$$
$$t_2 = (\texttt{class}, \texttt{sc}, \texttt{myClass})$$
$$t_3 = (\texttt{myClass}, \texttt{sc}, \texttt{class})$$

Trivially, t_1 is strictly deterministic in G, since it is non-derivable. Next, consider $t = (\mathsf{a}, \mathsf{sc}, \mathsf{a})$. There is a deterministic derivation of t in G, namely $t_1 \xRightarrow{\sigma_4} t$. Note, however, that t is strictly non-deterministic. For example, t can be derived non-deterministically from G using two applications of the non-deterministic rule σ_6:

$$\begin{aligned} t_1, t_2 &\xRightarrow{\sigma_6} (\mathsf{a}, \mathtt{type}, \mathtt{myClass}) \\ (\mathsf{a}, \mathtt{type}, \mathtt{myClass}), t_3 &\xRightarrow{\sigma_6} t_1 \\ t_1 &\xRightarrow{\sigma_4} t \end{aligned}$$

Note that t is deterministically derivable from t_1, and, furthermore, all other derivations of t in G make essential (in a sense which we will make precise below) use of t_1.

Clearly, strict determinism is too rigid. In particular, strict determinism disallows derivations which are "essentially" deterministic, as illustrated in Example 5.1. Towards a more relaxed notion of determinism, we introduce a few more definitions. For graph G and $t \in \mathsf{cl}(G)$, we say $H \subseteq G$ is *t-minimal* if $H \models_{\mathcal{L}_{RDFS}} t$ and, for every $t' \in H$, it holds that $(H - \{t'\}) \not\models_{\mathcal{L}_{RDFS}} t$. Furthermore, for triple $t^* \in G$, let $\mathsf{detpaths}(t^*, t)$ denote the set of all $t' \in G$ such that t' participates in a deterministic derivation of t from t^*.

DEFINITION 5.3. *Let G be an RDFS graph and $t \in \mathsf{cl}(G)$. We say t is* deterministic *in G if for every t-minimal $H \subseteq G$, there exists a non-derived $t^* \in G$ such that*

$$(H - \mathsf{detpaths}(t^*, t)) \not\models_{\mathcal{L}_{RDFS}} t.$$

Otherwise, we say t is nondeterministic *in G.*

We limit our inspection to t-minimal subsets, since strict supersets of t-minimal sets may admit multiple unrelated derivations of t. Furthermore, we require that t^* be non-derived, since otherwise t^* might itself be nondeterministically derived. (Note that every deterministically derived triple must ultimately be derived from a non-derivable triple.)

Observe that every strictly deterministic triple is also deterministic. The converse, however, is not necessarily true. For example, triple t of Example 5.1 is deterministic in G but not strictly deterministic. Towards capturing a wide range of well-behaved updates, in the sequel we focus on schema evolution involving deterministic triples.

5.2 Deletion

We next introduce two notions of deletion. The first, a "conservative" notion of deletion, focuses on maximal preservation of the semantics of the original graph. The second notion of deletion focuses on "aggressive" removal of information during deletion.

5.2.1 Conservative deletion

We first present results on conservative deletion, where "conservative" means deletions which minimally impact the semantics of a graph.

DEFINITION 5.4. *Let G be an RDFS graph and t be a triple. A* candidate update graph *for $(G, t^{[-]})$ is a graph C_G over the atoms and blank nodes occurring in G and $\mathcal{V}_{RDFS}$, such that*

1. *$C_G \not\models_{\mathcal{L}_{RDFS}} t$, and*
2. *for any subset $H \subseteq \mathsf{cl}(G)$ such that $H \not\models_{\mathcal{L}_{RDFS}} t$, it is the case that $C_G \models_{\mathcal{L}_{RDFS}} H$.*

Note that the set of candidate update graphs is always finite, since $\mathcal{V}_{RDFS}$ and the set of atoms and blank nodes occurring in any RDFS graph is always finite. Recall that we want to choose as our actual update graph some unique, well-defined graph. Fortunately, if the set of candidate graphs is not empty, then there is always a unique maximal graph.

DEFINITION 5.5. *Let G be an RDFS graph and t be a triple. An* update graph *for $(G, t^{[-]})$ is a RDFS graph U_G where*

1. *U_G is a candidate update graph for $(G, t^{[-]})$, and*
2. *$|\mathsf{cl}(G) \ominus U_G| \leqslant |\mathsf{cl}(G) \ominus C_G|$, for any candidate update graph C_G for $(G, t^{[-]})$.*[5]

Note that an update graph does not necessarily exist.

EXAMPLE 5.2. *Consider again graph G_1 of Example 3.1 and $t = (a, \mathsf{sp}, c)$. By condition (2) of Definition 5.4, any candidate update graph for $(G_1, t^{[-]})$ must model both (a, sp, b) and (b, sp, c). However, any such graph would not satisfy condition (1) of the definition. We conclude that no update graph exists for $(G, t^{[-]})$.*

In fact, for any triple t that is nondeterministic in graph G, it is the case that no update graph exists for $(G, t^{[-]})$. However, when such an update graph does exist, it is unique.

THEOREM 5.1. *Let G be an RDFS graph and t be a triple. An update graph for $(G, t^{[-]})$ exists if and only if $t \notin \mathsf{cl}(G)$ or t is deterministic in G. Furthermore, when it exists, the update graph is unique.*

PROOF. ($\Leftarrow$) Suppose $t \notin \mathsf{cl}(G)$. We argue that $\mathsf{cl}(G)$ is an update graph for $(G, t^{[-]})$. Since $t \notin \mathsf{cl}(G)$, clearly $\mathsf{cl}(G)$ satisfies condition (1) of Definition 5.5. Furthermore, we have that trivially $\mathsf{cl}(G)$ uniquely satisfies condition (2) of Definition 5.5 .

Suppose t is deterministic in G. Let $Ancestors(t) = \bigcup_{t^* \in G} \mathsf{detpaths}(t^*, t)$ and let $U_G = \mathsf{cl}(G) - Ancestors(t)$. Clearly U_G satisfies condition (1) of Definition 5.5, since it fulfills condition (1) of Definition 5.4 by the removal of all ancestors of t, and fulfills condition (2) trivially since t is deterministic. We further argue that U_G is the *only* candidate update graph satisfying condition (2) of Definition 5.5. Indeed, suppose that C' is a candidate update graph different from U_G such that $|\mathsf{cl}(G) \ominus C'| \leqslant |\mathsf{cl}(G) \ominus U_G|$. The only way this is possible is if C' contains one of the ancestors of t. Since t is deterministic, this would imply that $C \models_{\mathcal{L}_{RDFS}} t$, a contradiction.

($\Rightarrow$) Suppose $t \in \mathsf{cl}(G)$ and t is nondeterministic in G. Furthermore, suppose for sake of contradiction that there exists a candidate update graph C for $(G, t^{[-]})$. By condition (1) of Definition 5.4, it must be the case that C does not model t. For this to hold, since t is nondeterministic, it must be the case that, in some derivation of t, there exists a nondeterministic rule in Δ_{RDFS} with instantiation $t_1, \ldots, t_m \to t'$, for $t_1, \ldots, t_m, t' \in \mathsf{cl}(G)$ and $m > 1$, such

[5]Where $\ominus$ is the symmetric difference operator, defined as $A \ominus B = (A - B) \cup (B - A)$.

that, C does not model some t_i, $1 \leqslant i \leqslant m$. Otherwise, C would indeed model t. By condition (2) of Definition 5.4, however, C must model any strict subset of $t_1, \ldots, t_m$, a contradiction, since one of these subsets is $\{t_i\}$. We conclude that our assumption that C exists is in error, and hence there is no deterministic candidate update graph (and hence no deterministic update graph) for $(G, t^{[-]})$. $\square$

To summarize, an RDFS-expressible update graph exists only when t is deterministic, which essentially means that it is possible to uniquely extract t from the graph. Furthermore, when an update graph does exist, it is unique. Note that the update graph may be quite large (since we are working with closures). Finding and maintaining a suitable reduction of the update graph is an interesting issue which we leave open for further study.

5.2.2 Aggressive deletion

We next present results on aggressive deletions. By aggressive, we mean deletions that potentially have more "impact" on the semantics of a graph, in the sense that assertions retained during conservative deletion may be lost. The benefit of this approach is that we no longer need to deal with full closures of graphs.

Define a *lossy candidate update graph* for $(G, t^{[-]})$ just as in Definition 5.4 except that now in condition (2) we only consider subsets $H \subseteq G$. Similarly, define a *lossy update graph* for $(G, t^{[-]})$ just as in Definition 5.5 except that now in condition (2) we only consider the symmetric differences of U_G and C_G with the original graph G.

We note that the set of lossy candidate update graphs is also finite. Furthermore, any candidate update graph is a lossy candidate update graph. We want to choose for our actual update graph some unique, well-defined graph. Fortunately, among the set of lossy candidate graphs, there is a unique maximal graph in the the original graph. Indeed, it is straightforward to extend the proof of Theorem 5.1 to establish that lossy update graphs are also well-behaved.

THEOREM 5.2. *Let G be an RDFS graph and t be a triple. A lossy update graph for $(G, t^{[-]})$ exists if and only if $t \notin \mathsf{cl}(G)$ or t is deterministic in G. Furthermore, when it exists, the lossy update graph is unique.*

In particular, when it exists, we have from the proof that the lossy update graph for $(G, t^{[-]})$ is $G - \bigcup_{t^* \in G} \mathsf{detpaths}(t^*, t)$.

We now provide an illustration of the difference between update and lossy update graphs.

EXAMPLE 5.3. *Consider RDFS graph $G = \{t_1, t_2\}$ where*

$$\begin{aligned} t_1 &= (\mathtt{Max}, \mathtt{type}, \mathtt{Dog}) \\ t_2 &= (\mathtt{Dog}, \mathtt{sc}, \mathtt{Mammal}) \end{aligned}$$

Then, $\mathsf{cl}(G) =$

$$\{(\mathtt{Max}, \mathtt{type}, \mathtt{Dog}), (\mathtt{Dog}, \mathtt{sc}, \mathtt{Mammal}), (\mathtt{Max}, \mathtt{type}, \mathtt{Mammal})\},$$

the update graph for $(G, (\mathtt{Max}, \mathtt{type}, \mathtt{Dog})^{[-]})$ *is*

$$\{(\mathtt{Dog}, \mathtt{sc}, \mathtt{Mammal}), (\mathtt{Max}, \mathtt{type}, \mathtt{Mammal})\},$$

and the lossy update graph for $(G, (\mathtt{Max}, \mathtt{type}, \mathtt{Dog})^{[-]})$ *is*

$$\{(\mathtt{Dog}, \mathtt{sc}, \mathtt{Mammal})\}.$$

5.3 Deterministic unchase

In this subsection, we establish a tight connection between "unchase" and computing update graphs. Unchase, which was introduced in [3], can be intuitively understood as "undoing" chase steps. Here, we restrict our scope to unchase using deterministic keyword tgds. Formally, given a CQ query Q and a deterministic keyword tgd σ as specified in Section 4, *deterministic unchase of Q using σ is applicable* if there exists a homomorphism h from *all* of σ to the body of Q. Let $s(\bar{X})$ be the subgoal of Q that is the image of the right-hand side of σ under h; then the *deterministic-unchase step of Q with σ* is a query Q' resulting from removing $s(\bar{X})$ from the body of Q.

Given a set of deterministic keyword tgds Σ, a Σ-unchase sequence U is a sequence of CQ queries $Q_0, Q_1, \ldots$ such that every query Q_{i+1} $(i \geq 0)$ in U is obtained from Q_i by a deterministic-unchase step using some $\sigma \in \Sigma$. Observe that such a sequence U always terminates in finite time, that is, there exists an $n \geq 0$ such that $U = Q_0, \ldots, Q_n$ and no deterministic-unchase steps using Σ are applicable to Q_n. We call the query Q_n the (terminal) *result of the deterministic unchase of Q_0 using Σ,* and denote Q_n by $(Q)^U_\Sigma$.

Consider a Σ-unchase sequence $U = Q_0, \ldots, Q_n$ with the following property. Assume for the moment $n \geq 2$ in U for ease of exposition. Consider an $i \in \{0, \ldots, n-2\}$, and suppose (I) the (deterministic) unchase step $Q_i \Rightarrow Q_{i+1}$ in U removes from Q_i a subgoal s_i that is the image of the right-hand side of some $\sigma \in \Sigma$ under a homomorphism h, such that h maps the left-hand side of σ to a subgoal s_{i+1} (of both Q_i and Q_{i+1}). We call this s_{i+1} the *source of s_i at unchase step i in U.* Let (II) the unchase step $Q_{i+1} \Rightarrow Q_{i+2}$ in U remove from Q_{i+1} the subgoal s_{i+1}. Now we say that $U = Q_0, \ldots, Q_n$, with $n \geq 0$, is an *underivation of some subgoal s_0 of Q_0* if (I) implies (II) in U for all $i \in \{0, \ldots, n-2\}$. Let Q_n in underivation U be the result of removing subgoal s_{n-1} from Q_{n-1} in U; then we call the source s_n of s_{n-1} (at unchase step $n-1$ in U) the *root* of U.

Intuitively, an underivation U of s_0 in Q_0, with root s_n, can be visualized as a "path" $s_0 \rightarrow s_1 \rightarrow s_{n-1} \rightarrow s_n$ in the query Q_0. Here, each s_i is the subgoal being removed from Q_i in the ith step of the underivation U, with $0 \leq i \leq n-1$.

We now relate specific triples in an RDFS graph G with those subgoals of q_G that are involved in underivation of a subgoal of q_G using the set Σ^*_{det} of deterministic keyword tgds for RDFS, introduced in Section 4.2. (Recall that q_G is the associated query of G.) Given a triple t_0 in G and its associated subgoal s_0 in q_G, let $U = Q_0, \ldots, Q_n$ be an underivation of s_0 in q_G, with root s_n, and with subgoal s_i being removed at the ith step of U, for $0 \leq i \leq n-1$. Let each of these s_i, for $i \in \{0, \ldots, n\}$, be associated with a triple t_i in the graph G. Denote by $\mathit{detUnchaseSequence}(G, t_0, \Sigma^*_{det})$ the set $\{t_0, \ldots, t_n\}$. We define $\mathit{detUnchase}(G, t_0, \Sigma^*_{det})$ as the union of all sets $\mathit{detUnchaseSequence}(G, t_0, \Sigma^*_{det})$. It is easy to see that given G, Σ^*_{det}, and triple $t_0 \in \mathsf{cl}(G)$, $\mathit{detUnchaseSequence}(G, t_0, \Sigma^*_{det})$ is always finite and unique. Then the following holds:

THEOREM 5.3. *Triple t is deterministic in RDFS graph G if and only if $t \in \mathsf{cl}(G)$ and $(G - \mathit{detUnchase}(G, t, \Sigma^*_{det})) \not\models t$.*

It is easy to see that for a triple t and RDFS graph G, the complexity of computing $\mathit{detUnchase}(G, t)$ is $\mathcal{O}(d \times n)$, where d is the number of dependencies in Σ^*_{det} and n is the number of triples in $\mathsf{cl}(G)$.

5.4 Addition

Unlike deletions, adding triples to a graph is always well-behaved. (In contrast, in richer schema systems this is not always the case, cf. [12, 20]). Hence, we have the following direct semantics for adding assertions to a graph.

Definition 5.6. *Let G be an RDFS graph and t be a triple. The* update graph *for $(G, t^{[+]})$ is $G \cup \{t\}$.*

Trivially, the update graph for a triple addition is always unique and finite.

6. COMPUTING UPDATE GRAPHS

In this section, we give a direct algorithm for computing update graphs for deletion. In the case of deleting ground triples from RDFS graphs (i.e., triples without blank nodes), the algorithm runs in PTIME. The algorithm requires computing entailment, and hence, the case of deleting a triple with blank nodes is intractable (recall Corollary 3.1). The algorithm makes essential use of Theorems 5.1 and 5.3.

Our method is given in Algorithm 6.1. The key step is computing the deterministic unchase of the triple to be deleted. The correctness of the algorithm follows directly from Theorems 5.1 and 5.3. The cost of line (1) is the cost of computing deterministic unchase, which requires $\mathcal{O}(|G|)^2)$ time to compute $\mathsf{cl}(G)$ (see Proposition 3.1 and [19, 28]), and then $\mathcal{O}(|\mathsf{cl}(G)|)$ time to unchase t (we consider the set of dependencies to be constant). The cost of line (2) is again the cost of computing $\mathsf{cl}(G)$. Finally, the cost of line (3) is the cost of computing entailment, which in the worst case is NP-complete (cf. Corollary 3.1). In the case of a ground t, however, the cost of checking entailment is dominated by the cost of computing the closure [19]. In summary, we have

Theorem 6.1. *For an RDFS graph G and a ground triple t, the cost of computing the update graph $(G, t^{[-]})$ is $\mathcal{O}(|G|^2)$.*

Of course, Algorithm 6.1 is rather naive, and quite open to optimization. For example, it might be possible to avoid computing the full closure of G. We leave such optimization issues open for further investigation.

We also note that the algorithm (and running time) for computing *lossy* update graphs is identical to Algorithm 6.1, with the exception that, in line (2) we set $G^* = G - T^*$. The correctness of this algorithm follows from Theorems 5.2 and 5.3, as for computing regular update graphs.

7. DISCUSSION

In this paper we have initiated a study of well-behaved RDFS schema evolution. We characterized a broad class of updates which are well-behaved, in the sense that update results are unique and well-defined. Furthermore, we gave a tractable algorithm for computing updates, when they exist.

There are several immediate directions for further investigation. We are currently studying variations of update graphs, which admit efficient maintenance. This study also includes an investigation into various implementations of closure graphs proposed in the literature, as well as efficient implementation of the unchase algorithm on these representations. In another direction, it is important to study system support for identifying and handling non-deterministic updates, e.g., in the framework of Konstantinidis et al. [15].

```
Input: RDFS graph $G$ and triple $t$
Output: Update graph for $(G, t^{[-]})$ if it exists, and
        "None" otherwise
begin
1   let $T^*$ denote the set of all triples participating in
    the deterministic unchase of $t$ under $\Delta_{RDFS}$
2   let $G^* = \mathsf{cl}(G) - T^*$
3   if $G^* \models_{\mathcal{L}_{RDFS}} t$ then
        output "None"
    else
        output $G^*$
end
```

Algorithm 6.1: Computing $(G, t^{[-]})$

8. REFERENCES

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. J. Hollenbach. SW-Store: A vertically partitioned DBMS for semantic web data management. *VLDB J*, in press.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, USA, 1995.

[3] R. Chirkova and M. R. Genesereth. Database reformulation with integrity constraints. In *Workshop on Logic and Computational Complexity*, 2005.

[4] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In *SIGMOD*, pages 268–279, Austin, Texas, 1985.

[5] C. Cunningham, C. A. Galindo-Legaria, and G. Graefe. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *VLDB*, pages 998–1009, Toronto, 2004.

[6] C. Curino, H. Moon, and C. Zaniolo. Graceful database schema evolution: the PRISM workbench. *Proc. of the VLDB Endowment*, 1(1):761–772, 2008.

[7] J. de Bruijn, E. Franconi, and S. Tessaris. Logical reconstruction of normative RDF. In *OWLED*, 2005.

[8] G. De Giacomo et al. On instance-level update and erasure in description logic ontologies. *J. Log. Comp.*, to appear.

[9] A. Deutsch. *XML Query Reformulation over Mixed and Redundant Storage.* PhD thesis, Univ. Pennsylvania, 2002.

[10] A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[11] W. Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[12] G. Flouris et al. Ontology change: classification and survey. *Knowledge Eng. Review*, 23(2):117–152, 2008.

[13] C. Gutiérrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. In *PODS*, 2004.

[14] C. Gutiérrez et al. The meaning of erasing in RDF under the Katsuno-Mendelzon approach. In *WebDB*, 2006.

[15] G. Konstantinidis et al. A formal approach for RDF/S ontology evolution. In *ECAI*, Patras, Greece, 2008.

[16] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Trans. Database*

Syst., 26(4):476–519, 2001.

[17] G. Lausen. Relational databases in RDF: Keys and foreign keys. In *SWDB-ODBIS*, pages 43–56, Vienna, 2007.

[18] G. Lausen, M. Meier, and M. Schmidt. SPARQLing constraints for RDF. In *EDBT*, 2008.

[19] S. Muñoz, J. Pérez, and C. Gutiérrez. Minimal deductive systems for RDF. In *ESWC*, 2007.

[20] N. F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4):428–440, 2004.

[21] E. Rahm and P. A. Bernstein. An online bibliography on schema evolution. *SIGMOD Record*, 35(4):30–31, 2006.

[22] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, Rome, 2001.

[23] Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recom., 2004. Edit. G. Klyne and J. Carroll (Eds).

[24] RDF Semantics. W3C Recom., 2004. P. Hayes (Ed).

[25] RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recom., 2004. D. Brickley and R.V. Guha (Eds).

[26] V. M. Sarathy, L. V. Saxton, and D. Van Gucht. Algebraic foundation and optimization for object based query languages. In *IEEE ICDE*, pages 81–90, Vienna, 1993.

[27] M. Sintek and M. Kiesel. RDFBroker: A signature-based high-performance RDF store. In *ESWC*, pages 363–377, Budva, Montenegro, 2006.

[28] H. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *J Web Sem*, 3(2-3):79–115, 2005.

[29] Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of RDF/S stores. In *ISWC*, pages 685–701, Galway, Ireland, 2005.

[30] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. II.* Computer Science Press, Rockville, MD, USA, 1989.

[31] Z. Wang, K. Wang, R. W. Topor, and J. Z. Pan. Forgetting concepts in DL-Lite. In *ESWC*, 2008.

[32] K. Wilkinson. Jena property table implementation. In *SSWS*, pages 35–46, Athens, Georgia, USA, 2006.

[33] C. M. Wyss and E. L. Robertson. Relational languages for metadata integration. *ACM Trans. Database Syst.*, 30(2):624–660, 2005.

APPENDIX

A.1 Encoding RDF in the relational model (and vice versa)

A variety of relational encodings have been proposed for RDF data.

EXAMPLE A.1. *Consider the graph*

$$\begin{aligned} G \;=\; \{&\langle \texttt{author1}, \texttt{name}, \texttt{Shimabukuro}\rangle,\\ &\langle \texttt{author1}, \texttt{age}, \texttt{38}\rangle,\\ &\langle \texttt{author1}, \texttt{homeTown}, \texttt{Naha}\rangle,\\ &\langle \texttt{author1}, \texttt{genre}, \texttt{poetry}\rangle,\\ &\langle \texttt{author1}, \texttt{type}, \texttt{Author}\rangle,\\ &\langle \texttt{author2}, \texttt{name}, \texttt{Bulgakov}\rangle,\\ &\langle \texttt{author2}, \texttt{age}, \texttt{44}\rangle,\\ &\langle \texttt{author2}, \texttt{genre}, \texttt{fantasy}\rangle,\\ &\langle \texttt{author2}, \texttt{type}, \texttt{Author}\rangle\}. \end{aligned}$$

Under the property table *encoding, we have one relation per class* `type` *[32]. In the case of G, we have one table named* `Author`*:*

subject	name	age	homeTown	genre
author1	Shimabukuro	38	Naha	poetry
author2	Bulgakov	44	⊥	fantasy

Note that there is missing information for `Bulgakov`*, represented by a null value. There are a range of variations on this approach [27, 29]. If we also have* `type` *as an attribute, and encode all triples in a single table, then the property table approach is essentially the classic* universal relation *concept [30]. Note that this encoding works only if subjects take at most one value on a given attribute, which is too restrictive for real-world RDF data. In such a case, we could have multiple tuples per subject, or model the data in the* nested relational model *[2]*

An alternative to property tables is the binary schema-aware *or* vertical partitioning *encoding [1, 29]. In this approach there is a binary table for each unique predicate. For our example graph, we have*

type

subject	*object*
author1	Author
author2	Author

name

subject	*object*
author1	Shimabukuro
author2	Bulgakov

age

subject	*object*
author1	38
author2	44

genre

subject	*object*
author1	poetry
author2	fantasy

homeTown

subject	*object*
author1	Naha

Note that null-values are not necessary in this encoding.

This is a fairly old idea – see, for example, work on the decomposition storage and Tarski models [4, 26]. Indeed, the vertical partitioning approach can be thought of as a decomposition of the (nested) universal relation corresponding to G.

Going the other direction, RDF encodings have also been proposed for relational data, e.g., [17, 18].

EXAMPLE A.2. *One natural encoding is the inverse of the property table approach: for each relation R and for each tuple t in R, assign a (globally) unique atom* t_{id}*, generate the triple* $\langle t_{id}, \texttt{type}, R\rangle$*, and then for each attribute-value pair* $A : V$ *of t, generate the triple* $\langle t_{id}, A, V\rangle$*. If R is empty, and* *it is desired to keep (static) information about R, then one can emit metadata about R, such as $\langle R, \texttt{type}, \texttt{class}\rangle$ and information about R's attributes.*

Recently, work has appeared on enforcing traditional constraints (functional dependencies and inclusion dependencies) on relational data in this RDF encoding [17, 18].

A.2 Relational schema evolution

Relational schema evolution has been intensively studied for the past few decades [21]. However, many issues remain open. Typically, systems select a class of schema transformations, based on real-world experience, for study. Alternatively, generic "metadata" extensions to standard relational query languages have been proposed and investigated (e.g., [16, 33])

EXAMPLE A.3. *A recent proposal, in the first class, is the PRISM system [6]. In PRISM, a set of* schema modification operators *(SMO) is proposed, consisting of:*

- *create/drop table R,*
- *rename table R as T,*
- *copy table R into T,*
- *merge (i.e., union) tables R and S into T,*
- *(horizontally) partition table R into S and $\bar{S}$ on condition θ,*
- *(vertically) decompose table $R(\bar{A}, \bar{B}, \bar{C})$ into tables $S(\bar{A}, \bar{B})$ and $T(\bar{A}, \bar{C})$,*
- *join tables R and S into T on condition θ*
- *add column C to table R, with constant value or some function of tuples values in R,*
- *rename column B to C in table R.*

SMOs provide a broad coverage of ways in which relational schemata may evolve; however, it misses some natural types of relational schema evolution (e.g, column pivot [5, 22, 33]).

Now, it is straightforward to translate each of these into a "program" of actions on the RDF encoding of the relations (see Section 3.3). For example, a drop table R is implemented by dropping each participating tuple $t \in R$, which in turn amounts to removing all triples mentioning atom t_{id} (and, if necessary, all triples with subject R).