Capturing Provenance of RDF Triples through Colors

G. Flouris
FORTH-ICS, Greece
fgeo@ics.forth.gr

I. Fundulaki
FORTH-ICS, Greece
fundul@ics.forth.gr

P. Pediaditis
FORTH-ICS, Greece
University of Crete, Greece
pped@csd.uoc.gr

Y. Theoharis
FORTH-ICS, Greece
University of Crete, Greece
theohari@ics.forth.gr

V. Christophides
FORTH-ICS, Greece
University of Crete, Greece
christop@ics.forth.gr

ABSTRACT

Recently, the W3C Linking Open Data effort has boosted the publication and inter-linkage of large amounts of RDF datasets on the Semantic Web. Various ontologies and knowledge bases with millions of RDF triples from Wikipedia and other sources, mostly in e-science, have been created and are publicly available. Recording provenance information of RDF triples aggregated from different heterogeneous sources is crucial in order to effectively support trust mechanisms, digital rights and privacy policies. Managing provenance becomes even more important when we consider not only explicitly stated but also implicit triples (through RDFS inference rules) in conjunction with declarative languages for querying and updating RDF graphs. In this paper we rely on *colored RDF triples* represented as quadruples to capture and manipulate explicit provenance information.

1. INTRODUCTION

Recently, the W3C Linking Open Data [29] effort has boosted the publication and interlinkage of large amounts of RDF datasets on the Semantic Web [1]. Various ontologies and knowledge bases with millions of RDF triples from Wikipedia [26] and other sources have been created and are available online [25]. In addition, numerous data sources in e-science are published nowadays as RDF graphs, most notably in the area of life sciences [27], to facilitate community annotation and interlinkage of both scientific and scholarly data of interest. Finally, Web 2.0 platforms are considering RDF and RDFS as non-proprietary exchange formats for the construction of information mashups [16, 28].

In this context, it is of paramount importance to be able to store the *provenance of a piece of data* in order to effectively support trust mechanisms, digital rights and privacy policies. *Provenance* means *origin* or *source* and refers to *from where* and *how* the piece of data was obtained [33]. In the context of scientific communities, provenance information can be used in the proof of the correctness of results and in general determines their quality. In some cases, provenance of data is considered more important than the result itself.

The popularity of the RDF data model [8] and RDF Schema language (RDFS) [2] is due to the flexible and extensible representation of information, independently of the existence or absence of a schema, under the form of *triples*. An RDF triple, (*subject*,*property*,*object*), asserts the fact that *subject* is associated with *object* through *property*. RDFS is used to add semantics to RDF triples, by imposing *inference rules* [15] (mainly related to the transitivity of subsumption relationships) which can be used to entail new *implicit* triples (i.e., facts) that are not explicitly asserted.

Currently, there is no adequate support for managing provenance information of implicit RDF triples affecting the semantics of query and update RDF languages. There are two different ways in which such implicit knowledge can be viewed and this affects the assignment of provenance information to implicit triples, as well as the semantics of the update operations. Under the *coherence* semantics [10], implicit knowledge does not depend on the explicit one but has a value on its own; therefore, there is no need for explicit "support" of some triple. Under this viewpoint, implicit triples are "first-class citizens", i.e., considered of equal value as explicit ones. On the other hand, under the *foundational* semantics [10], implicit knowledge is only valid as long as the supporting explicit knowledge is there. Therefore, each implicit triple depends on the existence of the explicit triple(s) that imply it.

In this paper we propose the use of *colors* (as in [4]) in order to capture the *provenance* of RDF *data* and *schema* triples. We record the color of an RDF triple as a *fourth column*, hence obtaining an *RDF quadruple* as in [7, 17].

To provide the intuition of the granularity levels of provenance for RDF datasets that we capture with this work, we compare it with provenance in the relational world. For instance, authors in [4] use colors to capture the provenance of relational tables, tuples and attributes. If we consider that a relational tuple of the form $[a_1{:}v_1, \ldots, a_k{:}v_k]$ with tuple identifier *tid*, corresponds to a set of triples $(tid,\ a_j,\ v_j)$, $j = 1, \ldots k$, then (see Figure 1): a color assigned to *(a)* a single triple captures provenance at the level of *an attribute of the relational tuple*; *(b)* a collection of triples sharing the same subject captures provenance at the level of *the relational tuple* and finally *(c)* a set of triples whose subjects are instances of the same RDFS class, captures *provenance* *of the relational table.* An RDF Named Graph [5] can be modeled by arbitrary sets of triples sharing the same color.

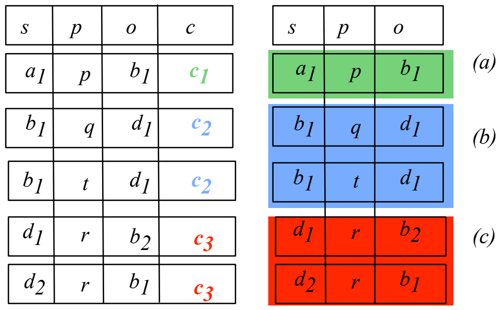


Figure 1: Granularity Levels of Provenance

The main contributions of this work are:

- We rely on the notion of *colors* to capture provenance information of explicit and implicit RDF triples. In particular, we employ an *abelian* group, defined by a *set of colors to record* and a *binary operation "+" to reason* over the provenance of RDF triples.
- We extend the RDFS inference rules [15] for computing the composite provenance of implicit RDF triples. In this respect, we devise an algorithm for determining *on the fly* the provenance of *non-materialized implicit* RDF triples.
- We study the semantics of *provenance propagation* and *querying* when colored RDF triples are represented as *quadruples*. In addition we discuss atomic update operations (i.e., inserts and deletes) for sets of RDF quadruples under the *coherence semantics* [10].

In this work, we consider coherence semantics, because we believe that implicit knowledge is equally important to the explicit one, i.e., the semantics of the RDF/S graph should be determined by its logical closure rather than its syntactic formulation (i.e., explicit triples).

The paper is structured as follows: in Section 2 we present the motivating example that we will use in this paper. Section 3 presents the basic RDF and RDFS notions. In Section 4 we introduce the notions of *color* and *quadruple* and discuss *inference* rules for sets of RDF quadruples. Section 5 discusses simple queries and atomic updates. We present related work in Section 6 and conclude in Section 7.

2. MOTIVATING EXAMPLE

We will use, for illustration purposes, an example taken from the News application domain. The RDFS schema of our News example captures information related to *newspapers* and *politicians* as well as the *relationships* between them and is contributed by different information sources.

Figure 2 shows relation $\mathcal{Q}(s,p,o,c)$ that stores both schema and data triples where s, p and o stand for the *subject*, *predicate* and *object* of an RDF triple. Column c is used to store the *provenance (color)* of a triple $(s,\ p,\ o)$. We will say that a triple $(s,\ p,\ o)$ is *colored* c iff (s,p,o,c) is in $\mathcal{Q}(s,p,o,c)$. The set of triples $(s,\ p,\ o)$ can be obtained by projecting on columns s, p and o of $\mathcal{Q}(s,p,o,c)$.

	s	p	o	c
q_1	*&NYT*	*endorses*	*&B. Obama*	c_2
q_2	*&NYT*	rdf:type	*Newspaper*	c_4
q_3	*Newspaper*	rdf:type	rdfs:Class	c_3
q_4	*Newspaper*	rdfs:subClassOf	*Mass Media*	c_3
q_5	*Mass Media*	rdfs:subClassOf	*Media*	c_5
q_6	*Candidate*	rdf:type	rdfs:Class	c_5
q_7	*&B. Obama*	rdf:type	*Candidate*	c_5
q_8	*endorses*	rdf:type	rdf:Property	c_1
q_9	*endorses*	rdfs:domain	*Newspaper*	c_1
q_{10}	*endorses*	rdfs:range	*Candidate*	c_1
q_{11}	*Candidate*	rdfs:subClassOf	*Person*	c_1
q_{12}	*supports*	rdf:type	rdf:Property	c_1
q_{13}	*supports*	rdfs:domain	*MassMedia*	c_1
q_{14}	*supports*	rdfs:range	*Person*	c_1
q_{15}	*endorses*	rdfs:subPropertyOf	*supports*	c_2
q_{16}	*&NYT*	*endorses*	*&B. Obama*	c_1
q_{17}	*Media*	rdf:type	rdfs:Class	c_5

Figure 2: Relation $\mathcal{Q}(s,p,o,c)$

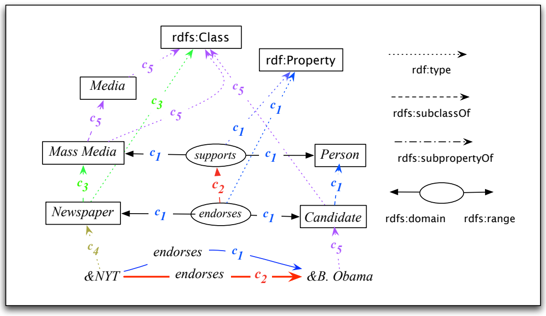


Figure 3: Graph representation for relation $\mathcal{Q}(s,p,o,c)$

Note that we can assign different colors to the same triple $(s,\ p,\ o)$ (quadruples q_1 and q_{16}). In this way, we can capture data integration scenarios in which the same piece of information originates from different sources.

Since RDF/S graphs can be seen as a kind of *node and edge labeled directed graphs*, we use the following graphical notation: *classes* and *properties* (binary relations between classes) of an RDFS vocabulary [2], are represented with boxes, and ovals respectively. *Instances* of classes contain their URI reference and to distinguish between individual resources and classes, we prefix the URI of the former with the "&" symbol. RDFS built-in properties [2] rdfs:subClassOf, rdf:type and rdfs:subPropertyOf are represented by dashed, dotted and dotted-dashed arrows respectively. Figure 3 shows the graph obtained from the quadruples in $\mathcal{Q}(s,p,o,c)$ of Figure 2. For (s,p,o,c), we color the edge with c and we draw $s \rightharpoonup p \xrightarrow{c} o$ (except if p is one of the built in RDF/S properties where we draw $s \xrightarrow{c} o$).

A great part of the information captured by a set of RDF triples can be inferred [15] by the *transitivity* of class and property subsumption relationships (rdfs:subClassOf and rdfs:subPropertyOf respectively) stated in the associated RDFS schemas. For instance, although not explicitly asserted in $\mathcal{Q}(s,p,o,c)$, we can infer that *Newspaper* is a subclass of *Media* because *(i) Newspaper* is a subclass of *Mass Media* (quadruple q_4) and *(ii) Mass Media* is a subclass of *Media* (quadruple q_5).

The question raised here is the following: *"what is the color of the implicit RDF triple?"*: the triple cannot be colored either c_3 or c_5 (colors of quadruples q_4 and q_5 resp.). In terms of provenance, we view the origin of this triple as *composite*: this triple should be colored $\underline{c_3 \text{ and } c_5}$. A possible solution is to add quadruples (*Newspaper*, rdfs:subClassOf, *Media*, c_3) *and* (*Newspaper*, rdfs:subClassOf, *Media*, c_5) in $\mathcal{Q}(s, p, o, c)$. But, in this case, query *"return the triples colored c_3"* would falsely return (*Newspaper*, rdfs:subClassOf, *Media*). Hence, *composite origin cannot be captured by associating with the implicit triple separately the colors of its implying triples.*

Instead, we capture the provenance of implicit triples using colors defined by applying the special operation "+" on the colors of its implying triples which creates a *new* color. This color can then be assigned to explicit and not just to implicit triples.

For instance, we color triple (*Newspaper*, rdfs:subClassOf, *Media*) with the color $c_{(3,5)} = c_3 + c_5$. In Section 4 we discuss the properties of operation "+" in detail.

As with annotated databases [4, 11, 12], we aim at supporting both *provenance propagation* and *provenance querying* for RDF data. In the case of *provenance propagation*, queries are targeted to the original data and provenance is propagated to the query results. The result of these queries are *explicit* and *implicit* colored RDF triples. For instance, one can ask the query *"return all instances of class Newspaper"*. The result of the query will be (&NYT, rdf:type, *Newspaper*, c_4). Query *"return all subclasses of Media"* will return (*Newspaper*, rdfs:subClassOf, *Media*, $c_{(3,5)}$) and (*Mass Media*, rdfs:subClassOf, *Media*, c_5). Note that (*Newspaper*, rdfs:subClassOf, *Media*, $c_{(3,5)}$) is an *implicit* quadruple obtained by applying the transitive rdfs:subClassOf subsumption relationship on quadruples (*Mass Media*, rdfs:subClassOf, *Media*, c_5) and (*Newspaper*, rdfs:subClassOf, *Mass Media*,c_3) as previously discussed.

On the other hand, in the case of *provenance querying*, queries are explicitly targeted at provenance information. Examples falling in this category are queries asking for the color of a given triple, or queries that filter triples given a color. For instance, the query *"return the color of (Newspaper,* rdfs:subClassOf*, Media)"*, will return $c_{(3,5)}$.

In this work we support atomic updates and more specifically *inserts* and *deletes*. One can *(a)* insert and *(b)* delete an explicit or an implicit quadruple. For our update operations we adhere to the coherence semantics [10] and we consider *redundant-free* graphs. According to the coherence semantics, we must explicitly retain all the implicit triples that will no longer be implied due to the deletion of an explicit triple. Redundant-free graphs have been chosen, because they offer a number of advantages in the case of transaction management for concurrent updates and queries.

Consider for instance, the deletion of quadruple (&B. *Obama*, rdf:type, *Candidate*, c_5). According to the coherence semantics we must retain the implicit information (&B. *Obama*, rdf:type, *Person*, $c_{(1,5)}$) implied by quadruples (&B. *Obama*, rdf:type, *Candidate*, c_5) and (*Candidate*, rdf:subClassOf, *Person*, c_1). Had we followed the foundational semantics [10], the implicit triple would be lost. Note that it would not be correct to remove the implicit knowledge in this example, because the removal of (&B. *Obama*, rdf:type, *Person*, $c_{(1,5)}$) was not dictated by the user's update itself, and should therefore be avoided. Furthermore, when redundancy elimination is applied, there is no way to know whether (&B. *Obama*, rdf:type, *Person*, $c_{(1,5)}$) was explicitly asserted or not and consequently, we don't know whether it should be removed. In this respect, coherence semantics is more adequate when considering redundant-free RDF/S graphs since they allow one to retain as much information as possible in the presence of updates.

3. PRELIMINARIES

In this work, we ignore non-universally identified resources, called *unnamed* or *blank nodes* [14]. In this respect, we consider two disjoint and infinite sets $\mathbf{U}$, $\mathbf{L}$, denoting the URIs and literals respectively.

DEFINITION 1. *An RDF triple* (*subject, predicate, object*) *is any element of the set* $\mathcal{T} = \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$.

The RDF Schema (RDFS) language [2] provides a built-in vocabulary for asserting user-defined schemas in the RDF data model. For instance, RDFS names rdfs:Resource (res), rdfs:Class (class) and rdf:Property (prop)[1] could be used as objects of triples describing *class* and *property* types. Furthermore, one can assert *instance of* relationships of resources with the RDF predicate rdf:type (type), while *subsumption* relationships among classes and properties are expressed with the RDFS rdfs:subClassOf (sc) and rdfs:subPropertyOf (sp) predicates respectively. In addition, RDFS rdfs:domain (domain) and rdfs:range (range) predicates allow one to specify the domain and range of the properties in a RDFS vocabulary. In the rest of this paper, we consider two disjoint and infinite sets of URIs of classes ($\mathbf{C} \subset \mathbf{U}$) and property types ($\mathbf{P} \subset \mathbf{U}$).

It should be finally stressed that RDFS schemas are essentially descriptive and not prescriptive, designed to represent data. We believe that this flexibility in representing schema-relaxable (or schema-less) information, is the main reason for RDF and RDFS popularity. Using the uniform formalism of RDF triples, we are able to represent in a flexible way both schema and instances in the form of RDF/S graphs. It should be noted that RDF/S graphs are not classical *directed labeled graphs*, because, for example, an RDFS predicate (e.g., rdfs:subPropertyOf) may relate other predicates (e.g., *endorses* and *supports*). Thus, the resulting structure is not a graph in the strict mathematical sense.

4. PROVENANCE FOR RDF/S DATA

In the same spirit as in [4] for relational databases, we assign a *color* to an RDF triple in order to capture its provenance.

DEFINITION 1. *Structure* ($\mathbf{I}$, *"+") is an* abelian group *where*

- $\mathbf{I} \subset \mathbf{U}$ *is the set of colors. We define the* neutral *color* $\varepsilon \in \mathbf{I}$*, and for each color* $c \in \mathbf{I}$ *we define its inverse denoted by* $-c$*;*
- *"+" is a binary operation with the following properties:*

$$
\begin{array}{llll}
c_1 + \varepsilon & = & c_1 & \textit{(Neutral)} \\
c_1 + (-c_1) & = & \varepsilon & \textit{(Inverse)} \\
c_1 + c_1 & = & c_1 & \textit{(Idempotence)} \\
c_1 + c_2 & = & c_2 + c_1 & \textit{(Commutativity)} \\
c_1 + (c_2 + c_3) & = & (c_1 + c_2) + c_3 & \textit{(Associativity)}
\end{array}
$$

[1] In parenthesis are the terms we will use in the paper to refer to the RDFS built-in classes and properties.

Binary operation "+" is defined to capture the provenance of *implicit* RDF triples. The intuition behind the properties of the "+" operation is the following: *(a)* an implicit RDF triple obtained from triples of the same color inherits the color of its implying triples (*idempotence*) *(b)* the color of an implicit triple is uniquely determined by the colors of its implying triples and not by the order of application of the inference rules (*commutativity* and *associativity*). For an implicit RDF triple, its implying triples are those used to obtain it through the application of the inference rules that we will discuss in Section 4.1.

We should note here that the inverse color does not introduce negative knowledge: if a triple is colored $-c_1$ (the inverse of color c_1) the intuition is that the triple can have any color except c_1. The neutral color is a color that is "absorbed" by any other color.

We denote with $c_{(1,2)}$ the color obtained by applying operation "+" on colors c_1 and c_2: $c_{(1,2)} = c_1 + c_2$. We say that color c_k is a *defining color* of $c_{1,2,...,n}$ and write $c_k \lessdot c_{1,2,...,n}$ iff $k \in \{1, 2, ..., n\}$.

Definition 2. *An RDF quadruple (subject, predicate, object, color) is any element of the set* $\mathcal{D} = \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L}) \times \mathbf{I}$.

Using this definition, we can define the notion of an RDF Dataset featuring triples associated with their provenance information as follows:

Definition 3. *An RDF Dataset d is a finite set of quadruples in* $\mathcal{D}$ $(d \subseteq \mathcal{D})$.

Note that none of the existing approaches combine intentional and extensional assignment of triples to provenance information (colors). In [30] RDF Named Graphs, capturing the provenance of RDF triples, are defined intentionally through SPARQL [31] views and do not support the explicit assignment of triples to such graphs, whereas in [5] a purely extensional definition is followed. The notion of colors as introduced in this paper allows us to capture both the intentional and extensional aspects of RDF graphs that are useful to record and reason about provenance information in the presence of updates.

4.1 Inference

In a similar manner as for RDF graphs (i.e., sets of triples) we define a consequence operator that abstracts a set of inference rules which compute the *closure* of an RDF dataset. Let d be an RDF Dataset. We write $Cn(d)$ to refer to the closure of d. Entailment for RDF datasets is defined as for RDF graphs: an RDF dataset d_1 *entails* RDF dataset d_2 iff the closure of d_2 is a subset of the closure of d_1 modulo the colors. We say that a dataset d entails a quadruple q, and write $d \vdash q$, iff q belongs to the closure of d. We say that a triple $t = (s,\ p,\ o)$ is colored c iff there exists a quadruple (s, p, o, c) in $Cn(d)$.

The inference rules shown in Table 1 extend those specified in [15] in a straightforward manner to take into account colors. They compute the color of the implicit triple, using the colors of its implying triples.

For instance, quadruple (*Newspaper*, rdfs:subClassOf, *Media*, $c_{(3,5)}$) is obtained by applying the *Transitivity of* sc $I_d^{(2)}$ rule on quadruples (*Newspaper*, rdfs:subClassOf, *Mass Media*, c_3) and (*Mass Media*, rdfs:subClassOf, *Media*, c_5). We overload the closure operator Cn in order to capture the *closure* of an RDF Dataset, computed using the inference rules of Table 1.

Reflexivity of sc	$I_d^{(1)}$:	$\dfrac{(C, \mathsf{type}, \mathsf{class}, c_1)}{(C, \mathsf{sc}, C, c_1)}$
Transitivity of sc	$I_d^{(2)}$:	$\dfrac{(C_1, \mathsf{sc}, C_2, c_1), (C_2, \mathsf{sc}, C_3, c_2)}{(C_1, \mathsf{sc}, C_3, c_{(1,2)})}$
Reflexivity of sp	$I_d^{(3)}$:	$\dfrac{(P, \mathsf{type}, \mathsf{prop}, c_1)}{(P, \mathsf{sp}, P, c_1)}$
Transitivity of sp	$I_d^{(4)}$:	$\dfrac{(P_1, \mathsf{sp}, P_2, c_1), (P_2, \mathsf{sp}, P_3, c_2)}{(P_1, \mathsf{sp}, P_3, c_{(1,2)})}$
Transitivity of class instantiation	$I_d^{(5)}$:	$\dfrac{(x, \mathsf{type}, C_1, c_1), (C_1, \mathsf{sc}, C_2, c_2)}{(x, \mathsf{type}, C_2, c_{(1,2)})}$
Transitivity of property instantiation	$I_d^{(6)}$:	$\dfrac{(P_1, \mathsf{sp}, P_2, c_1), (x_1, P_1, x_2, c_2)}{(x_1, P_2, x_2, c_{(1,2)})}$

Table 1: Inference Rules for RDF Datasets

4.2 Redundancy Elimination

In our work we consider that the RDF datasets are *redundant free*. An RDF dataset is redundant free if there does not exist a quadruple that can be implied by others when applying inference rules $I_d^{(1)}$ – $I_d^{(6)}$ of Table 1. The detection and removal of redundancies is straightforward using those rules.

In the sequel, we assume that queries and updates are performed upon redundant-free RDF datasets. In effect, this means that redundancies are detected (and removed) at update time rather than at query time. This choice was made because we believe that in real scale Semantic Web systems, query performance should prevail over update performance. Redundant-free RDF datasets were chosen because they offer a number of advantages in the case of transaction management for concurrent updates and queries.

5. QUERYING AND UPDATING RDF DATASETS

5.1 Querying RDF Datasets

In this section we discuss a simple class of queries that allow one to express queries on the *subclass*, *subproperty* and *type* hierarchies of an RDF dataset. We consider $\mathcal{V}$, $\mathcal{C}$ to be two sets of variables for resources and colors respectively; $\mathcal{V}$, $\mathcal{C}$, $\mathbf{U}$ (URIs) and $\mathbf{L}$ (literals) are mutually disjoint sets. We define a simple form of a quadruple pattern, called *q-pattern* which is an element from $(\mathbf{U} \cup \mathcal{V}) \times \{\mathsf{type} \cup \mathsf{sc} \cup \mathsf{sp}\} \times (\mathbf{U} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathcal{C})$. In our context, a query is of the form (H, B, C) where H (head) is a q-pattern, B (body) is an expression defined after the antecedent of the inference rules presented in Section 4.1, and C (constraints) is a conjunction of *atomic predicates*. Each atomic predicate has the form:

1. $v\ = const$ for $v \in \mathcal{V}$;
2. $v\ \lessdot\ v'$ for $v, v' \in \mathcal{C}$;

$$
\begin{array}{lcl}
[[\mathsf{type}]]_d & = & \{(x,\ y,\ c) \mid d \vdash_{\{I_d^{(2)}, I_d^{(5)}\}} (x, \mathsf{type}, y, c)\} \\
[[\mathsf{sc}]]_d & = & \{(x,\ y,\ c) \mid d \vdash_{\{I_d^{(1)}, I_d^{(2)}\}} (x, \mathsf{sc}, y, c)\} \\
[[\mathsf{sp}]]_d & = & \{(x,\ y,\ c) \mid d \vdash_{\{I_d^{(3)}, I_d^{(4)}\}} (x, \mathsf{sp}, y, c)\} \\
[[\mathsf{domain}]]_d & = & \{(x,\ y,\ c) \mid d \vdash (x, \mathsf{domain}, y, c)\} \\
[[\mathsf{range}]]_d & = & \{(x,\ y,\ c) \mid d \vdash (x, \mathsf{range}, y, c)\} \\
[[p]]_d & = & \{(x,\ y,\ c) \mid d \vdash_{\{I_d^{(4)}, I_d^{(6)}\}} (x, p, y, c)\}
\end{array}
$$

Table 2: $[[r]]_d$ for properties type, sc, sp, domain, range and p

$$
\begin{array}{lcl}
[[(a, exp, ?y, ?c)]]_d & = & \{\mu \mid dom(\mu) = \{?y, ?c\} \ and \ (a,\ \mu(?y),\ \mu(?c)) \in [[exp]]_d\} \\
[[(?x, exp, a, ?c)]]_d & = & \{\mu \mid dom(\mu) = \{?x, ?c\} \ and \ (\mu(?x),\ a,\ \mu(?c)) \in [[exp]]_d\} \\
[[(?x, exp, ?y, c)]]_d & = & \{\mu \mid dom(\mu) = \{?x, ?y\} \ and \ (\mu(?x),\ \mu(?y),\ c) \in [[exp]]_d\} \\
[[(a, exp, b, ?c)]]_d & = & \{\mu \mid dom(\mu) = \{?c\} \ and \ (a,\ b,\ \mu(?c)) \in [[exp]]_d\} \\
[[(a, exp, b, c)]]_d & = & \{\mu \mid dom(\mu) = \emptyset \ and \ (a,\ b,\ c) \in [[exp]]_d\}
\end{array}
$$

Table 3: Semantics of *q-patterns*

3. $c = c_1 + c_2 \ldots + c_k$ where $c \in \mathcal{C}$ and $c_i \in \mathbf{I}$.

According to the above definition, one can express constraints on resources *(1)*, on colors *(2)*, as well as to specify that a color considered in the query is defined by a set of other colors *(3)*. In addition, we require that all variables that appear in the head of the query (H) appear in the query's body (B). This restriction is imposed in order to have computationally desirable properties.

We denote variables with $?x$, $?y$, … for resources and $?c_1$, $?c_2$, … for colors. To define the query semantics, we use the notion of mapping in the same spirit as in [21] as follows: a *mapping* μ is a partial function $\mu : (\mathcal{V} \cup \mathcal{C}) \rightarrow \mathbf{U} \cup \mathbf{I}$. The domain of μ ($dom(\mu)$) is the subset of $\mathcal{V} \cup \mathcal{C}$ where μ is defined. Let μ denote a mapping and $?v$ a variable, then $\mu(?v)$ denotes the resource or color to which $?v$ is mapped through μ.

To define the semantics of a *q-pattern* we must define first the *semantics of a property r over an RDF Dataset d*, denoted by $[[r]]_d$. Given an RDF Dataset d, $[[r]]_d$ for properties type, sc, sp, domain, range and p is given in Table 2. We write $d \vdash_S q$ to denote that dataset d entails quadruple q when the inference rules in S are applied on d.

We write $\langle r \rangle_d$ to denote the semantics of property r when no inference rule is used. We can now define the semantics of a *q-pattern*. Consider an RDF dataset d and $q = (?X, exp, ?Y, ?c)$ a *q-pattern*, where exp is one of sc, sp, type. Then the evaluation of q over d is defined as follows:

$$
\begin{array}{lcl}
[[q]]_d & = & \{\mu \mid dom(\mu) = \{?X, ?Y, ?c\} \ and \\
 & & (\mu(?X),\ \mu(?Y),\ \mu(?c)) \in [[exp]]_d\}.
\end{array}
$$

In Table 3 we give the semantics of some *q-patterns* when URIs and colors are considered (a and b are constant URIs and c is a color identifier in $\mathbf{I}$).

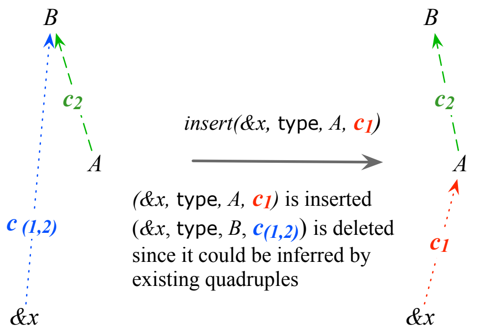
Finally, given a mapping μ we say that μ satisfies an atomic predicate C, denoted by $\mu \vdash C$, per the following conditions:

$$
\begin{array}{lll}
\mu \vdash (?x = const) & iff & \mu(?x) = const, const \in \mathbf{U}, \\
 & & ?x \in dom(\mu) \\
\mu \vdash (?x = ?y) & iff & \mu(?x) = \mu(?y), \\
 & & ?x, ?y \in dom(\mu) \\
\mu \vdash (?c = ?c') & iff & \mu(?c) = \mu(?c'), \\
 & & ?c, ?c' \in dom(\mu) \\
\mu \vdash (?c \lessdot ?c') & iff & \mu(?c) \lessdot \mu(?c') \\
 & & ?c, ?c' \in dom(\mu) \\
\mu \vdash (?c = c_1 + ... + c_k) & iff & \mu(?c) = c_1 + \ldots + c_k, \\
 & & ?c \in dom(\mu)
\end{array}
$$

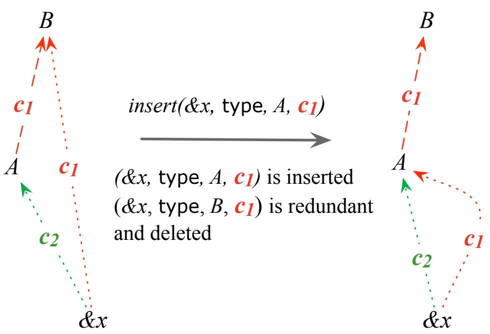
For a query $Q = (H, B, C)$ an RDF dataset d and mapping μ, such that $\mu \in [[B]]_d$, and $\mu \vdash C$, $\mu(H)$ is the quadruple obtained by replacing every variable $?x$ in $dom(\mu)$ with $\mu(?x)$. The color variable (if any) is replaced by the color obtained by applying the "+" operator as specified by the inference rules $I_d^{(1)}$ to $I_d^{(6)}$ of Table 1. The answer to Q is the union of the quadruples $\mu(H)$ for each such mapping μ.

5.2 Updating RDF Datasets

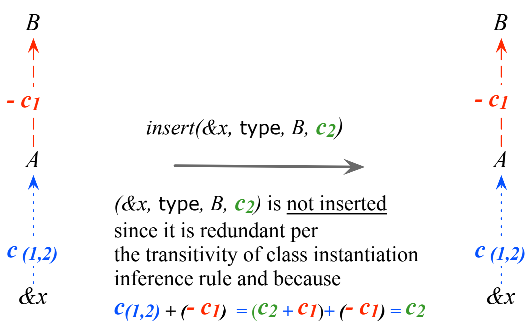
In this section we discuss *atomic* update operations (*inserts* and *deletes*). Recall that in our work we follow the *coherence semantics* [10] according to which the *implicit* triples are considered of equal value as the explicit ones and hence we need to retain information that would be lost in the case of a triple deletion. Moreover, we should enforce that the resulting RDF datasets will remain valid with respect to the employed RDFS schema. The notion of validity has been described in various fragments of the RDFS language ([18, 32]), and is used to overrule certain triple combinations. In the context of RDF datasets, the validity constraints are applied (and defined) at the level of the RDF dataset, but the color-related part of the quadruple is not considered. The validity constraints that we consider in this work concern the *disjointness* between class, property and color names and the acyclicity of rdfs:subClassOf and rdfs:subPropertyOf subsumption relationships. An additional validity constraint that we consider in our work is that the subject and object of the instance of some property should be correctly classified under the domain and range of the property respectively. For a full list of the related validity constraints, see [19].



(a) Class Instance Insertion (1)



(b) Class Instance Insertion (2)



(c) Class Instance Insertion (3)

Figure 4: Examples of Class Instance Insertions

The semantics of each atomic update is specified by its corresponding *effects* and *side-effects*. The effect of an insert or delete operation consists of the straightforward insertion/deletion of the requested quadruples. The side-effects ensure that the resulting RDF dataset continues to be valid and non-redundant as discussed in [35]. Update semantics adhere to the principle of *minimal change* [9], per which a minimal number of insertions and deletions should be performed in order to restore a valid and non-redundant state of an RDF dataset. The effects and side-effects of insertions and deletions are determined by the kind of triple involved, i.e., whether it is a *class instance* or *property instance* insertion or deletion. Due to space restrictions we only describe *class instance* insertions and deletions. The algorithms presented here differ from the algorithms that handle inserts and deletes in the absence of provenance information: when computing the triples to be inserted (or deleted) as a side-effect of the update operation, we must compute the colors of the non-materialized implicit triples, whereas in the original algorithms no such computation is necessary.

5.2.1 INSERT *Operation*

A primitive insert operation is of the form: $\mathsf{insert}(s, p, o, i)$ where $s, p \in \mathbf{U}$, $o \in \mathbf{U} \cup \mathbf{L}$, $i \in \mathbf{I}$.

Algorithm 1: Class Instance Insertion Algorithm

Data: $\mathsf{insert}(x, \mathsf{type}, y, i)$, RDF dataset d
Result: Updated RDF dataset d

1 **if** $(\exists\ (x,\ y,\ i) \in [[\mathsf{type}]]_d)$ **then return** d;
2 **if** $(y \notin \mathbf{C})$ **then**
3 | **return** d;
4 **forall** $((x,\ z,\ i') \in \langle\mathsf{type}\rangle_d$ *s.t.* $\exists(y,\ z,\ i'') \in [[\mathsf{sc}]]_d$ *and* $i' = i + i'')$ **do**
5 | $d = d \setminus \{(x, \mathsf{type}, z, i')\}$;
6 **end**
7 $d = d \cup \{(x, \mathsf{type}, y, i)\}$;
8 **return** d;

A formal description of the insertion of a quadruple (x, type, y, i) in an RDF dataset d along with its side-effects can be found in Algorithm 1. At line 1 we examine if the quadruple already belongs to the semantics of property type. If not, then we ensure that y is a class (lines 2–3). If it is, then we remove all class instantiation quadruples from the RDF dataset which can be implied through the quadruple to be inserted and the class subsumption relationships (lines 4–6). Finally, the quadruple is inserted (line 7). Examples of class instance insertions are shown in Figure 4.

5.2.2 DELETE *Operation*

A primitive delete operation is of the form: $\mathsf{delete}(s, p, o, i)$ where $s, p \in \mathbf{U}$, $o \in \mathbf{U} \cup \mathbf{L}$, $i \in \mathbf{I}$.

A formal description of the deletion of a quadruple (x, type, y, i) is given in Algorithm 2. At line 1 we examine if the quadruple belongs to the semantics of property type. If this is the case, then we must (1) insert all the quadruples that are implied by the quadruple that we wish to delete (recall that we follow the coherence semantics) and (2) delete the quadruples that if retained would imply the quadruple we wish to delete (lines 2–7).

In order to ensure that the RDF dataset is still valid after the updates, we must remove all properties originating from

Algorithm 2: Class Instance Deletion Algorithm

Data: delete(x, type, y, i), RDF dataset d
Result: Updated RDF dataset d

1 **if** $(\nexists\ (x,\ y,\ i) \in [[\mathsf{type}]]_d)$ **then return** d;
2 **forall** $((x,\ y',\ i') \in [[\mathsf{type}]]_d, (y',\ y,\ i'') \in [[\mathsf{sc}]]_d$ *s.t.* $i = i' + i'')$ **do**
3 | **forall** $(y',\ z,\ k) \in \langle \mathsf{sc} \rangle_d$ *s.t.* $y'! = z$ **do**
4 | | **if** $\nexists(z,\ y,\ h) \in [[\mathsf{sc}]]_d$ **then**
  | | $d = d \cup \{(x, \mathsf{type}, z, i' + k)\}$
5 | **end**
6 | $d = d \setminus \{(x, \mathsf{type}, y', i')\}$
7 **end**
8 **forall** $(x,\ o,\ h) \in \langle q \rangle_d$, *s.t.* $(q,\ c,\ i) \in \langle \mathsf{domain} \rangle_d$ **do**
9 | **if** $\nexists\ (x,\ c,\ j) \in [[\mathsf{type}]]_d$ **then**
10 | | $d = d \setminus \{(x, q, o, h)\}$;
11 | | **forall** q' *s.t.* $\exists\ (q,\ q',\ h'') \in [[\mathsf{sp}]]_d$ **do**
12 | | | **if** $\exists\ (x,\ e,\ k) \in [[\mathsf{type}]]_d$ *s.t.* $\exists\ (q,\ e,\ k') \in \langle \mathsf{domain} \rangle_d$ **then**
  | | | $d = d \cup \{(x, q', o, h + h'')\}$
13 | | **end**
14 |
15 **end**
16 **forall** $(o,\ x,\ h) \in \langle q \rangle_d$, *s.t.* $(q,\ c,\ i) \in \langle \mathsf{range} \rangle_d$ **do**
17 | **if** $\nexists\ (x,\ c,\ j) \in [[\mathsf{type}]]_d$ **then**
18 | | $d = d \setminus \{(o, q, x, h)\}$;
19 | | **forall** q' *s.t.* $\exists\ (q,\ q',\ h'') \in [[\mathsf{sp}]]_d$ **do**
20 | | | **if** $\exists\ (x,\ e,\ k) \in [[\mathsf{type}]]_d$ *s.t.* $\exists\ (q,\ e,\ k') \in \langle \mathsf{range} \rangle_d$ **then**
  | | | $d = d \cup \{(o, q', x, h + h'')\}$
21 | | **end**
22 |
23 **end**
24 **return** d;

(or reaching resp.) x whose domain (or range resp.) is a class that x is no longer an instance of (lines 8–23). Examples of class instance deletions can be found in Figures 5 and 6. Figure 6 shows a class instance deletion with the necessary property instance deletions to ensure a valid RDF dataset.

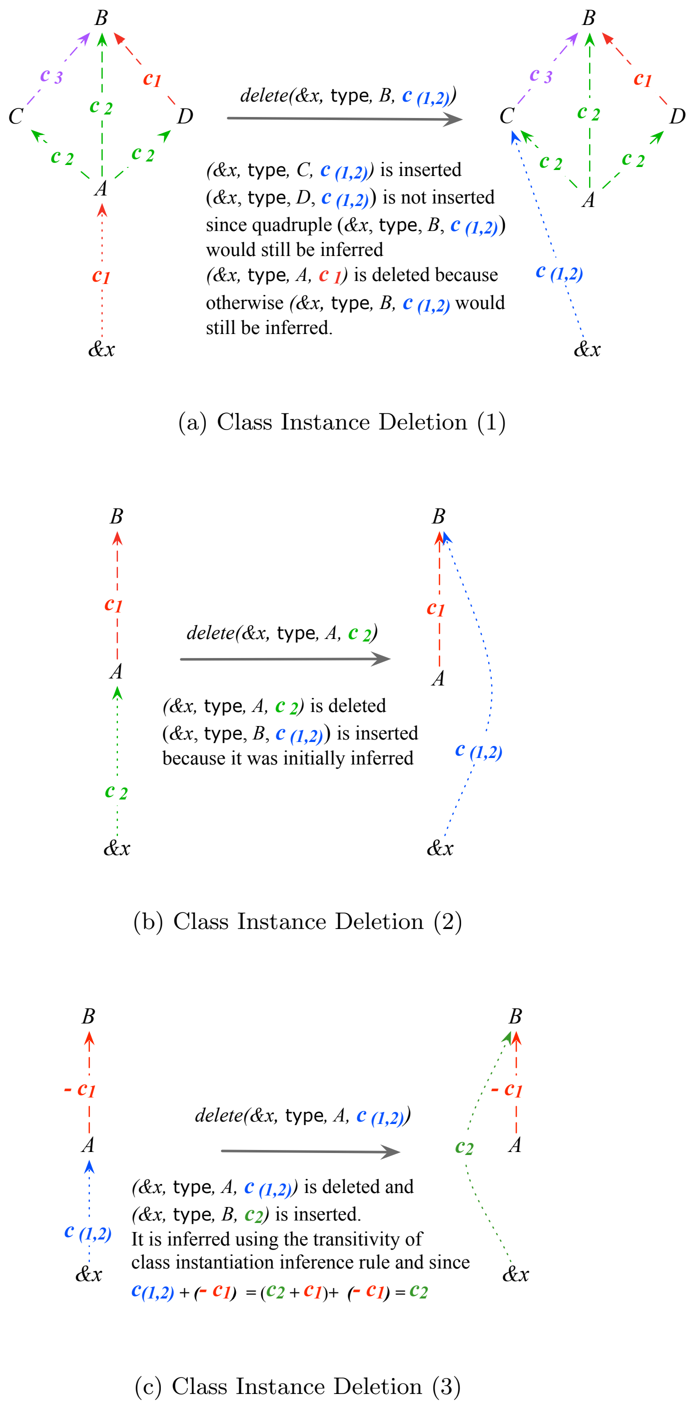
5.3 Complexity Analysis

When working with colored RDF triples, the basic kinds of queries that we need to answer are: *"what is the color of a triple"* and *"which triples have a given color"*. Both kinds of queries are classified as *provenance querying* problems. In order to answer them over an RDF dataset d we consider that a triple $t = (s,\ p,\ o)$ is colored c iff there exists a quadruple (s, p, o, c) in $Cn(d)$.

In our work we consider *redundant-free* RDF datasets: implicit triples are not materialized. In this section we discuss an algorithm that computes the color of *non-materialized implicit* RDF triples of an RDF dataset d *without* materializing the closure $Cn(d)$ of d and discuss its complexity.

For this analysis, we consider d to be an RDF dataset. We denote by M_t the set of triples of d: $M_t = \{(s, p, o) \mid \exists\ c\ \in \mathbf{I},\ (s, p, o, c)$ *a quadruple in* $d\}$ and with M_c the set of colors in d.

In order to determine whether a triple t is colored with a color c in an RDF dataset d, we must consider all possible combinations of triples in M_t that involve at least one triple that is colored by one of the defining colors of c. For



(a) Class Instance Deletion (1)

(b) Class Instance Deletion (2)

(c) Class Instance Deletion (3)

Figure 5: Examples of Class Instance Deletions

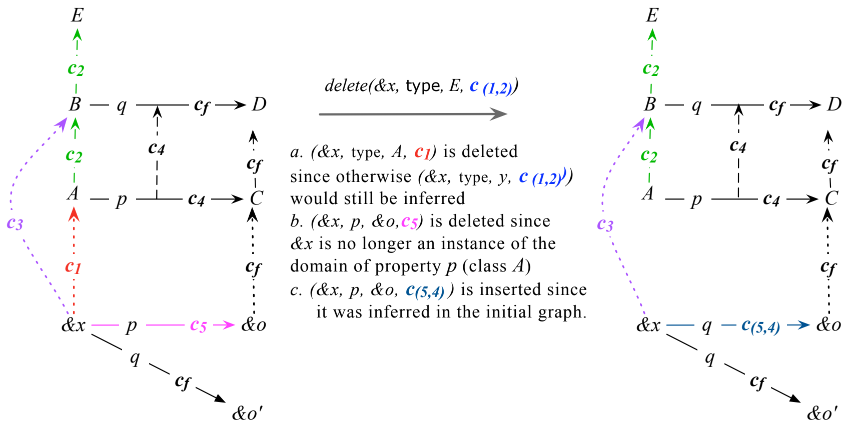


Figure 6: Class Instance and Property Instance Deletion

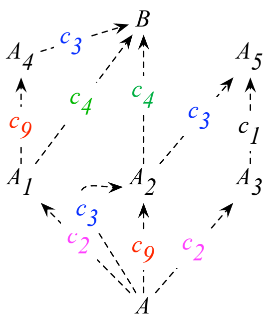


Figure 7: sc Hierarchy

instance, consider the class subsumption hierarchy shown in Figure 7. To compute the set of triples colored $c_{(2,3,9)} = c_2 + c_3 + c_9$, we need to consider the *combination of all triples colored with at least one* of colors c_2, c_3 and c_9.

A brute force approach would be to compute $Cn(d)$ in order to compute the color of a given triple. However, there is a much faster approach which avoids the computation of $Cn(d)$, but, instead, computes directly the set of triples from which t can be implied. Following this approach, we can determine whether a triple t is colored c in $O(|M_t|\log(|M_t|))$ time.

Determining the color of a triple not implied by others is trivial: these are all the triples that *do not involve* the RDFS type, sc and sp relationships (the inference rules in Table 1 consider only those). This is achieved in $O(\log(|M_t|))$ by a simple search for t in d.

To determine the color of implicit triples that involve the aforementioned built-in RDFS properties, we view the sc and sp hierarchies as *directed acyclic graphs.* Nodes in the graph are classes (properties resp.), and there exists an edge between nodes x and y iff x is a subclass (subproperty resp.) of y. The problem of obtaining the sequence of triples from which triple (A, sc, B), for instance, can be implied is equivalent to discovering the path(s) (i.e., sequences of edges) in the DAG from node A to node B. For each of these sequences of triples, we are going to keep the colors of each triple. Recall that a triple can be colored with different colors. The algorithm uses a depth-first search and terminates when B is reached. At each step of the algorithm, we *(i)* find the superclasses of the context node (i.e., the node that we are currently looking at) that are also subclasses of the target node (e.g. B) and *(ii)* store the set of colors of the triples (i.e., edges) that we have already examined.

Algorithm 3: Traverse_Subsumption_Graph

Data: Classes *source* and *target*, RDF Dataset d, Set of colors S;
Result: Set of colors S;
1 **if** *source=target* **then**
2 | **return** S;
3 $S \leftarrow \emptyset$;
4 **foreach** *class x, where x is a superclass of source and subclass of target* **do**
5 | **Let** $(source, \mathsf{sc}, x, c)$;
6 | **Let** $S_x \leftarrow \emptyset$;
7 | **forall** *colors c_i in S* **do**
8 | | $S_x \leftarrow S_x \cup \{c_i + c\}$;
9 | **end**
10 | Traverse_Subsumption_Graph(x,*target*,d,S_x);
11 | $S \leftarrow S \cup S_x$
12 **end**

In Algorithm 3 in order to obtain all the classes that are superclasses of *source* and subclasses of *target* (line 4), we use the labeling scheme introduced in [6] that captures the subsumption relationships between classes and properties and allows us to determine whether a class (or property) is a subclass (or subproperty) of another in *constant time.* This is achieved by simply comparing the labels of the classes/properties. The labeling scheme is *solely used to prune irrelevant subclass and subproperty paths*, thereby limiting our search space significantly, without taking into consideration colors of triples.

For instance, consider the subclass hierarchy shown in Figure 7. Suppose that we need to discover the color of triple (A, sc, B). We start with class A and in the first step we will keep classes A_1,A_2 and sets of colors $S_{A_1} = \{c_2\}$, $S_{A_2} = \{c_3, c_9\}$ since the triples that determine that A is a subclass of A_1, A_2 are determined by triples colored $\{c_2\}$, $\{c_3, c_9\}$

respectively. For each of the superclasses of A, we perform exactly the same process and we will be extending the sets of colors that we have obtained. The result of this process is the set of colors: $\{c_{(3,4)}, c_{(9,4)}, c_{(2,4)}, c_{(2,9,3)}\}$ where a color $c_{(1,\ldots,n)} = c_1 + \ldots + c_n$.

Given the fact that we prune subsumption paths that are not used to imply the triple in question using the labeling scheme, and that no cycles are allowed, the described process will, in the worst case, consider each triple in the RDF dataset d once. Given that each triple access requires a search in d, the cost of each access is $O(\log(|M_t|))$ (using an adequate indexing system). Therefore, the total complexity is $O(|M_t|\log(|M_t|))$.

For the other triples that may appear as implicit ones, the algorithm is similar. In particular, if t is of the form $(P,\ \mathsf{sp},\ Q)$, then the process is identical to the above, except that properties and subproperty relationships are considered instead of class and subclass relationships. If t is of the form $(A,\ \mathsf{type},\ B)$ then the process is almost identical, except that, in the first step of the recursion, we search for all classes whose explicit instance is A and are also (implicit or explicit) subclasses of B; the rest is the same. Finally, if t is of the form $(x,\ P,\ y)$, then, again, the process is identical, except that, in the first step of the recursion, we search for all explicit triples of the form $(x,\ Q,\ y)$ such that Q is an explicit or implicit subproperty of P. The rest of the recursion steps are as in the above cases.

6. RELATED WORK

So far, research on recording provenance for RDF data has focused on either associating triples with an *RDF named graph* [5] or by *extending* an RDF triple to a *quadruple* where the fourth element is a URI, a blank node or an identifier [7, 17]. These works vary in the semantics of the fourth element which is used to represent *provenance*, *context* and *access control* information. *RDF Named graphs* have been proposed in [5, 34] to capture *explicit provenance* information by allowing users to refer to specific parts of RDF/S graphs in order to decide *"how credible is"*, or *"how evolves"* a piece of information. An RDF named graph is a set of triples to which a URI has been assigned and can be referenced by other graphs as a normal resource; in this manner, one can assign explicit provenance information to this collection of triples.

Currently, there is no adequate support on how to manage provenance of *implicit* and explicit triples in the presence of *queries* and *updates*. Authors in [5] do not discuss RDFS inference, queries and updates in the presence of RDF named graphs. Unfortunately, existing declarative languages for querying and updating RDF triples have been extended either with RDF named graphs (such as SPARQL [23] and SPARQL Update [31]) or with RDFS inference support [22, 24], but not with both. In this paper, we attempt to fill this gap by proposing a framework based on the use of colors to capture provenance of RDF triples and reason about the provenance of implicit triples for simple queries and atomic updates. In a previous work [20], we have introduced the notion of *RDF/S graphsets* which builds upon and extends the notion of RDF named graphs. In that paper we showed that the mechanism of RDF named graphs cannot capture the provenance of implicit RDF triples, and proposed RDF graphsets as a solution to this problem. In this paper, we use colors as an elegant and uniform way to capture the provenance of both explicit and implicit RDF triples. Colors are a generalization of RDF named graphs [5]: a set of triples colored with a single color can be considered as belonging to the RDF named graph whose URI is the color (recall that colors are URIs). Colors obtained by applying the "+" operation on other colors simulate graphsets [20].

Note that none of the existing approaches combine intentional and extensional assignment of triples to provenance information (colors). In [30] RDF Named Graphs, capturing the provenance of RDF triples, are defined intentionally through SPARQL [31] views and do not support the explicit assignment of triples to such graphs, whereas in [5] a purely extensional definition is followed. The notion of colors as introduced in this paper allows us to capture both the intentional and extensional aspects of RDF graphs that are useful to record and reason about provenance information in the presence of updates.

On the other side of the spectrum, a significant amount of work on the issue has been done for relational and tree-structured databases [3, 4, 13, 12]. In [3, 4] authors discuss explicit provenance recording under copy-paste semantics where all operations are a sequence of delete-insert-copy-paste operations. In that work, new identifiers are introduced in the case in which the same object is deleted and then re-inserted, whereas in our case we are able to recognize the corresponding triple, and consequently preserve provenance information. In [13], fine-grained *where* and *how* provenance for relational databases is captured; however, updates are not considered in that work. Finally, in [12] authors consider a colored algebra to annotate columns and rows of relational tables at a coarse grained level which bares similarities to our color-based approach to record provenance of RDF triples.

7. CONCLUSION

In this paper we propose the use of colors to capture the provenance of RDF data and schema triples. We use the logical representation of quadruples to store the color of an RDF triple. The use of colors allows us to capture provenance at several granularity levels and can be considered as a generalization of RDF Named Graphs. One of the main contributions of the paper is the extension of RDFS inference rules to determine the provenance of *implicit* RDF triples, an extension which is not possible under the RDF named graphs approach. This problem has been overlooked in the majority of approaches that deal with managing provenance information for RDF graphs. As a future work we will study a more general algebraic structure as in [13] to capture the provenance of triples obtained by the SPARQL operators [23].

8. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American-American Edition, 2001.

[2] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `www.w3.org/TR/2004/REC-rdf-schema-20040210`, 2004.

[3] P. Buneman, A. P. Chapman, and J. Cheney. Provenance Management in Curated Databases. In *SIGMOD*, 2006.

[4] P. Buneman, J. Cheney, and S. Vansummeren. On the Expressiveness of Implicit Provenance in Query and Update Languages. In *ICDT*, 2007.

[5] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, Provenance and Trust. In *WWW*, 2005.

[6] V. Christophides, D. Plexousakis, M. Scholl, and S. Tourtounis. On labeling schemes for the semantic web. In *WWW*, 2003.

[7] E. Dumbill. Tracking Provenance of RDF Data. Technical report, ISO/IEC, 2003.

[8] B. McBride F. Manola, E. Miller. RDF Primer. `www.w3.org/TR/rdf-primer`, February 2004.

[9] P. Gardenfors. Belief Revision: An Introduction. *Belief Revision*, (29):1–28, 1992.

[10] P. Gardenfors. The dynamics of belief systems: Foundations versus coherence theories. *Revue Internationale de Philosophie*, 44:24–46, 1992.

[11] F. Geerts and J. V. den Bussche. Relational Completeness of Query Languages for Annotated Databases. In *DBPL*, 2007.

[12] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. In *ICDE*, 2006.

[13] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[14] C. Gutierrez, C. A. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *PODS*, 2004.

[15] P. Hayes. RDF Semantics. `www.w3.org/TR/rdf-mt`, February 2004. W3C Recommendation.

[16] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni. Rapid Prototyping of Semantic Mash-ups through Semantic Web Pipes. In *WWW*, 2009.

[17] R. MacGregor and I.-Y. Ko. Representing Contextualized Data using Semantic Web Tools. In *Practical and Scalable Semantic Systems*, 2003. In conjunction with ISWC.

[18] S. Munoz, J. Perez, and C. Gutierrez. Minimal deductive systems for RDF. In *ESWC*, 2007.

[19] P. Pediaditis. Querying and Updating RDF/S Named Graphs. Master's thesis, Computer Science Department, University of Crete, 2008.

[20] P. Pediaditis, G. Flouris, I. Fundulaki, and V. Christophides. On Explicit Provenance Management in RDF/S Graphs. In *TAPP*, 2009.

[21] J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *ISWC*, 2006.

[22] J. Perez, M. Arenas, and C. Gutierrez. nSPARQL: A Navigational Language for RDF. In *ISWC*, 2008.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. `www.w3.org/TR/rdf-sparql-query`, January 2008.

[24] PSPARQL. `psparql.inrialpes.fr`.

[25] DBLP Computer Science Bibliography. `www.informatik.uni-trier.de/~ley/db`.

[26] DBPedia. `www.dbpedia.org`.

[27] Gene Ontology. `www.geneontology.org`.

[28] RDFizers. `simile.mit.edu/wiki/RDFizers`.

[29] W3C Linking Open Data. `esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`.

[30] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the Web. In *WWW*, 2008.

[31] A. Seaborne and G. Manjunath. SPARQL/Update: A language for updating RDF graphs. `jena.hpl.hp.com/~afs/SPARQL-Update.html`, April 2008.

[32] G. Serfiotis, I. Koffina, V. Christophides, and V. Tannen. Containment and Minimization of RDF/S Query Patterns. In *ISWC*, 2005.

[33] W.-C. Tan. Provenance in databases: Past, current, and future. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2007.

[34] E. Watkins and D. Nicole. Named Graphs as a Mechanism for Reasoning About Provenance. In *Frontiers of WWW Research and Development - APWeb*, 2006.

[35] D. Zeginis, Y. Tzitzikas, and V. Christophides. On the foundations of computing deltas between rdf models. In *ISWC/ASWC*, 2007.