Provenance for SPARQL Queries

Carlos Viegas Damásio[1], Anastasia Analyti[2], and Grigoris Antoniou[3]

[1] CENTRIA, Departamento de Informática Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
cd@fct.unl.pt

[2] Institute of Computer Science, FORTH-ICS, Crete, Greece
analyti@ics.forth.gr

[3] Institute of Computer Science, FORTH-ICS, and
Department of Computer Science, University of Crete, Crete, Greece
antoniou@ics.forth.gr

Abstract. Determining trust of data available in the Semantic Web is fundamental for applications and users, in particular for linked open data obtained from SPARQL endpoints. There exist several proposals in the literature to annotate SPARQL query results with values from abstract models, adapting the seminal works on provenance for annotated relational databases. We provide an approach capable of providing provenance information for a large and significant fragment of SPARQL 1.1, including for the first time the major non-monotonic constructs under multiset semantics. The approach is based on the translation of SPARQL into relational queries over annotated relations with values of the most general m-semiring, and in this way also refuting a claim in the literature that the `OPTIONAL` construct of SPARQL cannot be captured appropriately with the known abstract models.

Keywords: How-provenance, SPARQL queries, m-semirings, difference.

1 Introduction

A general data model for annotated relations has been introduced in [9], for positive relational algebra (i.e. excluding the difference operator). These annotations can be used to check derivability of a tuple, lineage, and provenance, perform query evaluation of incomplete database, etc. The main concept is the notion of $\mathcal{K}$-relations where tuples are annotated with values (tags) of a commutative semiring $\mathcal{K}$, while positive relational algebra operators semantics are extended and captured by corresponding compositional operations over $\mathcal{K}$. The obtained algebra on $\mathcal{K}$-relations is expressive enough to capture different kinds of annotations with set or bag semantics, and the authors show that the semiring of polynomials with integer coefficients is the most general semiring. This means that to evaluate queries for any positive algebra query on an arbitrary semiring, one can evaluate the query in the semiring of polynomials (factorization property of [9]). This work has been extended to the case of full relational algebra in [8] by considering the notion of semirings with a monus operation (m-semirings [2]) and constant annotations, and the factorization property is proved for the special m-semiring that we denote by $\mathcal{K}_{dprovd}$.

The use of these abstract models based on $\mathcal{K}$-relations to express provenance in the Semantic Web has been advocated in [12]. However, the authors claim that the existing m-semirings are not capable to capture the appropriate provenance information for SPARQL queries. This claim is supported by the authors using a simple example, which we have adapted to motivate our work:

Example 1. Consider the following RDF graph expressing information about users' accounts and homepages, resorting to the FOAF vocabulary:

```
@prefix people: <http://people/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
people:david foaf:account <http://bank> .
people:felix foaf:account <http://games> .
<http://bank> foaf:accountServiceHomepage <http://bank/yourmoney>.
```

The SPARQL query

```
PREFIX foaf <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE { ?who foaf:account ?acc .
        OPTIONAL { ?acc foaf:accountServiceHomepage ?home }
}
```

returns the solutions (mappings of variables):

?who	?acc	?home
<http://people/david>	<http://bank>	<http://bank/yourmoney>
<http://people/felix>	<http://games>	

However, if the last triple is absent from the graph then the solutions are instead:

?who	?acc	?home
<http://people/david>	<http://bank>	
<http://people/felix>	<http://games>	

In order to track provenance of data, each tuple of data can be tagged with an annotation of a semiring. This annotation can be a boolean, e.g. to annotate that the tuple is trusted or not, a set of identifiers of tuples returning lineage of the tuple, or more complex annotations like the polynomials semiring to track full how-provenance [9,8], i.e. how a tuple is generated in the result under bag semantics.

Returning to the introductory example, assume that we represent the 3 triples in the input RDF graph as the ternary K_{dprovd}-relation (with the obvious abbreviations), where the last column contains the triple identifier (annotation):

Triples

sub	pred	obj	
< david >	< account >	< bank >	t_1
< felix >	< account >	< games >	t_2
< bank >	< accountServiceHomepage >	< bank/yourmoney >	t_3

The expected annotation of the first solution of the SPARQL query is $t_1 \times t_3$, meaning that the solution was obtained by joining triples identified by t_1 and t_3, while for the second solution the corresponding annotation is simply t_2. However, if we remove the last tuple we obtain a different solution for `david` with annotation just t_1. The authors in [12] explain why the existing approaches to provenance for the Semantic Web cannot handle the situation of Example 1, basically because there are different bindings of variables depending on the absence/presence of triples, and it is claimed that the m-semiring K_{dprovd} also cannot handle it. The rest of our paper shows that this last claim is wrong, but that requires some hard work and long definitions since the method proposed relies on the translation of SPARQL queries into relational algebra. The result is the first approach that provides adequate provenance information for `OPTIONAL`, `MINUS` and `NOT EXISTS` constructs under the multiset (bag) semantics of SPARQL.

The organization of the paper is the following. We review briefly in the next section the basics of K-relations. The SPARQL semantics is introduced in Section 3, and its translation into relational algebra is the core of the paper and can be found in Section 4. Using the relational algebra translation of SPARQL, we use K_{dprovd} to annotate SPARQL queries and show in Section 5 that Example 1 is properly handled. We finish with some comparisons and conclusions.

2 Provenance for *K*-Relations

A commutative semiring is an algebraic structure $\mathcal{K} = (\mathbb{K}, \oplus, \otimes, 0, 1)$ where $(\mathbb{K}, \oplus, 0)$ is a commutative monoid ($\oplus$ is associative and commutative) with identity element 0, $(\mathbb{K}, \otimes, 1)$ is a commutative monoid with identity element 1, the operation $\otimes$ distributes over $\oplus$, and 0 is the annihilating element of $\otimes$. In general, a tuple is a function $t : U \to \mathbb{D}$ where U is a finite set of attributes and $\mathbb{D}$ is the domain of values, which is assumed to be fixed. The set of all such tuples is U-`Tup` and usual relations are subsets of U-`Tup`. A $\mathcal{K}$-relation over U is a function $R : U\text{-}\mathtt{Tup} \to \mathbb{K}$, and its support is $supp(R) = \{t \mid R(t) \neq 0\}$.

In order to cover the full relational operators, the authors in [8] assume that the $\mathcal{K}$ semiring is naturally ordered (i.e. binary relation $x \preceq y$ is a partial order, where $x \preceq y$ iff there exists $z \in \mathbb{K}$ such that $x \oplus z = y$), and require additionally that for every pair x and y there is a least z such that $x \preceq y \oplus z$, defining in this way $x \ominus y$ to be such smallest z. A $\mathcal{K}$ semiring with such a monus operator is designated by m-semiring. Moreover, in order to capture duplicate elimination, the authors assume that the m-semiring is finitely generated. The query language[1] $\mathcal{RA}^+_{\mathcal{K}}(-, \delta)$ has the following operators [8]:

empty relation: For any set of attributes U, we have $\emptyset : U\text{-}\mathtt{Tup} \to \mathbb{K}$ such that $\emptyset(t) = 0$ for any t.

union: If $R_1, R_2 : U\text{-}\mathtt{Tup} \to \mathbb{K}$ then $R_1 \cup R_2 : U\text{-}\mathtt{Tup} \to \mathbb{K}$ is defined by: $(R_1 \cup R_2)(t) = R_1(t) \oplus R_2(t)$.

[1] The authors use instead the notation $\mathcal{RA}^+_{\mathcal{K}}(\setminus, \delta)$.

projection: If $R : U\text{-}\texttt{Tup} \to \mathbb{K}$ and $V \subseteq U$ then $\Pi_V(R) : V\text{-}\texttt{Tup} \to \mathbb{K}$ is defined by $(\Pi_V(R))\,(t) = \bigoplus_{t=t' \text{ on } V \text{ and } R(t') \neq 0} R(t')$.

selection: If $R : U\text{-}\texttt{Tup} \to \mathbb{K}$ and the selection predicate P maps each U-tuple to either 0 or 1 depending on the (in-)equality of attribute values, then $\sigma_P(R) : U\text{-}\texttt{Tup} \to \mathbb{K}$ is defined by $(\sigma_P(R))\,(t) = R(t) \otimes P(t)$.

natural join: If $R_i : U_i\text{-}\texttt{Tup} \to \mathbb{K}$, for $i = 1, 2$, then $R_1 \bowtie R_2$ is the $\mathcal{K}$-relation over $U_1 \cup U_2$ defined by $(R_1 \bowtie R_2)(t) = R_1(t) \otimes R_2(t)$.

renaming: If $R : U\text{-}\texttt{Tup} \to \mathbb{K}$ and $\beta : U \to U'$ is a bijection then $\rho_\beta(R)$ is the $\mathcal{K}$-relation over U' defined by $(\rho_\beta(R))(t) = R(t \circ \beta^{-1})$.

difference: If $R_1, R_2 : U\text{-}\texttt{Tup} \to \mathbb{K}$ then $R_1 - R_2 : U\text{-}\texttt{Tup} \to \mathbb{K}$ is defined by: $(R_1 - R_2)(t) = R_1(t) \ominus R_2(t)$.

constant annotation: If $R : U\text{-}\texttt{Tup} \to \mathbb{K}$ and k_i is a generator of $\mathbb{K}$ then $\delta_{k_i} : U\text{-}\texttt{Tup} \to \mathbb{K}$ is defined by $(\delta_{k_i}(R))(t) = k_i$ for each $t \in \mathit{supp}(R)$ and $(\delta_{k_i}(R))(t) = 0$ otherwise.

One major result of [8] is that the factorization property can be obtained for $\mathcal{RA}^+_{\mathcal{K}}(-, \delta)$ by using a special m-semiring with constant annotations that we designate by $\mathcal{K}_{dprovd}$. $\mathcal{K}_{dprovd}$ is the free m-semiring over the set of source tuple ids X, which is a free algebra generated by the set of (tuple) identifiers in the equational variety of m-semirings. Elements of $\mathcal{K}_{dprovd}$ are therefore terms defined inductively as: identifiers in X, 0, and 1 are terms; if s and t are terms then $(s + t)$, $(s \times t)$, $(s - t)$, and $\delta_{k_i}(t)$ are terms, and nothing else is a term. In fact annotations of $\mathcal{K}_{dprovd}$ are elements of the quotient structure of the free terms with respect to the congruence relation induced by the axiomatization of the m-semirings, in order to guarantee the factorization property (see [8] for more details). In our approach, X will be the set of graph and tuple identifiers.

We slightly extend the projection operator, by introducing new attributes whose value can be computed from the other attributes. In our approach, this is simply syntactic sugar since the functions we use are either constants or return one of the values in the arguments.

3 SPARQL Semantics

The current draft of SPARQL 1.1 [1] defines the semantics of SPARQL queries via a translation into SPARQL algebra operators, which are then evaluated with respect to a given RDF dataset. In this section, we overview an important fragment corresponding to an extension of the work in [10] that presents the formal semantics of the first version of SPARQL. The aim of our paper is on the treatment of non-monotonic constructs of SPARQL, namely OPTIONAL, MINUS and NOT EXISTS, and thus we focus in the SELECT query form, ignoring property paths, GROUP graph patterns and aggregations, as well as solution modifiers. The extension of our work to consider all the graph patterns is direct from the results presented. Regarding FILTER expressions, we analyse with detail the EXISTS and NOT EXISTS constructs, requiring special treatment. We assume the reader has basic knowledge of RDF and we follow closely the presentation of [1]. For more details the reader is referred to sections 17 and 18 of the current SPARQL 1.1 W3C working draft. We also make some simplifying assumptions that do not affect the results of our paper.

3.1 Basics

Consider disjoint sets of IRI (absolute) references $\mathbf{I}$, blank nodes $\mathbf{B}$, and literals $\mathbf{L}$ including plain literals and typed literals, and an infinite set of variables $\mathbf{V}$. The set of RDF terms is $\mathbf{T} = \mathbf{IBL} = \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. A triple[2] $\tau = (s, p, o)$ is an element of $\mathbf{IBL} \times \mathbf{I} \times \mathbf{IBL}$ and a graph is a set of triples. Queries are evaluated with respect to a given RDF Dataset $D = \{G, (<\mathtt{u_1}>, G_1), (<\mathtt{u_2}>, G_2), \ldots, (<\mathtt{u_n}>, G_n)\}$, where G is the default graph, and each pair $(<\mathtt{u_i}>, G_i)$ is called a named graph, with each IRI $\mathtt{u_i}$ distinct in the RDF dataset, and G_i being a graph.

3.2 Graph Patterns

SPARQL queries are defined by graph patterns, which are obtained by combining triple patterns with operators. SPARQL graph patterns are defined recursively as follows:

– The empty graph pattern ().
– A tuple $(\mathbf{IL} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{IL} \cup \mathbf{V})$ is a graph pattern called triple pattern[3];
– If P_1 and P_2 are graph patterns then (P_1 `AND` P_2), (P_1 `UNION` P_2), as well as (P_1 `MINUS` P_2), and (P_1 `OPTIONAL` P_2) are graph patterns;
– If P_1 is a graph pattern and R is a filter SPARQL expression[4] then the construction (P_1 `FILTER` R) is a graph pattern;
– If P_1 is a graph pattern and *term* is a variable or an IRI then (`GRAPH` *term* P_1) is a graph pattern.

The SPARQL 1.1 Working Draft also defines Basic Graph Patterns (BGPs), which correspond to sets of triple patterns. A Basic Graph Pattern $P_1, \ldots, P_n$ is encoded as the graph pattern (() `AND` (P_1 `AND` ($P_2 \ldots$ `AND` P_n)) ...). We ignore in this presentation the semantics of `FILTER` expressions, whose syntax is rather complex. For the purposes of this paper it is enough to consider that these expressions after evaluation return a boolean value, and therefore we also ignore errors. However, we show how to treat the `EXISTS` and `NOT EXISTS` patterns in `FILTER` expressions since these require querying graph data, and therefore provenance information should be associated to these patterns.

3.3 SPARQL Algebra

Evaluation of SPARQL patterns return multisets (bags) of solution mappings. A solution mapping, abbreviated solution, is a partial function $\mu : \mathbf{V} \to \mathbf{T}$. The

[2] Literals in the subject of triples are allowed, since this generalization is expected to be adopted in the near future.
[3] For simplicity, we do not allow blank nodes in triple patterns.
[4] For the full syntax of filter expressions, see the W3C Working Draft [1].

domain of μ is the subset of variables of $\mathbf{V}$ where μ is defined. Two mappings μ_1 and μ_2 are compatible if for every variable v in $dom(\mu_1) \cap dom(\mu_2)$ it is the case that $\mu_1(v) = \mu_2(v)$. It is important to understand that any mappings with disjoint domain are compatible, and in particular the solution mapping μ_0 with empty domain is compatible with every solution. If two solutions μ_1 and μ_2 are compatible then their union $\mu_1 \cup \mu_2$ is also a solution mapping. We represent extensionally a solution mapping as a set of pairs of the form (v, t); in the case of a solution mapping with a singleton domain we use the abbreviation $v \to t$. Additionally, if P is an arbitrary pattern we denote by $\mu(P)$ the result of substituting the variables in P defined in μ by their assigned values.

We denote that solution mapping μ satisfies the filter expression R with respect to the active graph G of dataset D by $\mu \models_{D(G)} R$. Including the parameter $D(G)$ in the evaluation of filter expressions is necessary in order to evaluate `EXISTS`(P) and `NOT EXISTS`(P) filter expressions, where P is an arbitrary graph pattern. If these constructs are removed from the language, then one only needs to consider the current solution mapping to evaluate expressions (as done in [10]).

Definition 1 (SPARQL algebra operators [1]). *Let Ω_1 and Ω_2 be multisets of solution mappings, and R a filter expression. Define:*

Join: $\Omega_1 \bowtie \Omega_2 = \{\!|\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1$ *and* $\mu_2 \in \Omega_2$ *such that* μ_1 *and* μ_2 *are compatible* $|\!\}$

Union: $\Omega_1 \cup \Omega_2 = \{\!|\mu \mid \mu \in \Omega_1$ *or* $\mu \in \Omega_2|\!\}$

Minus: $\Omega_1 - \Omega_2 = \{\!|\mu_1 \mid \mu_1 \in \Omega_1$ *such that* $\forall_{\mu_2 \in \Omega_2}$ *either* μ_1 *and* μ_2 *are not compatible or* $dom(\mu_1) \cap dom(\mu_2) = \emptyset|\!\}$

Diff: $\Omega_1 \setminus_R^{D(G)} \Omega_2 = \{\!|\mu_1 \mid \mu_1 \in \Omega_1$ *such that* $\forall_{\mu_2 \in \Omega_2}$ *either* μ_1 *and* μ_2 *are not compatible, or* μ_1 *and* μ_2 *are compatible and* $\mu_1 \cup \mu_2 \not\models_{D(G)} R|\!\}$

LeftJoin: $\Omega_1 ⟕_R^{D(G)} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus_R^{D(G)} \Omega_2)$

The **Diff** operator is auxiliary to the definition of **LeftJoin**. The SPARQL 1.1 Working Draft also introduces the notion of sequence to provide semantics to modifiers like `ORDER BY`. The semantics of the extra syntax is formalized by several more operators, namely aggregates and sequence modifiers (e.g. ordering), as well as property path expressions; we briefly discuss their treatment later on. Since lists can be seen as multisets with order and, without loss of generality regarding provenance information, we just consider multisets.

Definition 2 (SPARQL graph pattern evaluation). *Let $D(G)$ be an RDF dataset with active graph G, initially the default graph in $D(G)$. Let P, P_1 and P_2 be arbitrary graph patterns, and t a triple pattern. The evaluation of a graph pattern over $D(G)$, denoted by $[\![.]\!]_{D(G)}$ is defined recursively as follows:*

1. $[\![()]\!]_{D(G)} = \{\!|\mu_0|\!\}$;
2. $[\![t]\!]_{D(G)} = \{\!|\ \mu \mid dom(\mu) = var(t)$ *and* $\mu(t) \in G|\!\}$, *where* $var(t)$ *is the set of variables occurring in the triple pattern* t;
3. $[\![(P_1 \texttt{ AND } P_2)]\!]_{D(G)} = [\![P_1]\!]_{D(G)} \bowtie [\![P_2]\!]_{D(G)}$;
4. $[\![(P_1 \texttt{ UNION } P_2)]\!]_{D(G)} = [\![P_1]\!]_{D(G)} \cup [\![P_2]\!]_{D(G)}$;

5. $[\![(P_1\ \texttt{MINUS}\ P_2)]\!]_{D(G)} = [\![P_1]\!]_{D(G)} - [\![P_2]\!]_{D(G)}$;
6. $[\![(P_1\ \texttt{OPTIONAL}\ P_2)]\!]_{D(G)} = [\![P_1]\!]_{D(G)} ⟕^{D(G)}_{true} [\![P_2]\!]_{D(G)}$, *where* P_2 *is not a* FILTER *pattern;*
7. $[\![(P_1\ \texttt{OPTIONAL}\ (P_2\ \texttt{FILTER}\ R))]\!]_{D(G)} = [\![P_1]\!]_{D(G)} ⟕^{D(G)}_{R} [\![P_2]\!]_{D(G)}$;
8. $[\![(P_1\ \texttt{FILTER}\ R)]\!]_{D(G)} = \{\!| \mu \in [\![P_1]\!]_{D(G)} \mid \mu \models_{D(G)} R\ |\!\}$;
9. *Evaluation of* $[\![(\texttt{GRAPH}\ term\ P_1)]\!]_{D(G)}$ *depends on the form of term:*
   – *If term is an IRI corresponding to a graph name* $\texttt{u}_\texttt{i}$ *in* $D(G)$ *then* $[\![(\texttt{GRAPH}\ term\ P_1)]\!]_{D(G)} = [\![P_1]\!]_{D(G_i)}$;
   – *If term is an IRI that does not correspond to any graph in* $D(G)$ *then* $[\![(\texttt{GRAPH}\ term\ P_1)]\!]_{D(G)} = \{\!|\,|\!\}$;
   – *If term is a variable* v *then* $[\![(\texttt{GRAPH}\ term\ P_1)]\!]_{D(G)} =$

$$= ([\![P_1]\!]_{D(G_1)} \bowtie \{\!| v \rightarrow <\texttt{u}_1> |\!\}) \cup \ldots \cup ([\![P_1]\!]_{D(G_n)} \bowtie \{\!| v \rightarrow <\texttt{u}_\texttt{n}> |\!\})$$

The evaluation of EXISTS and NOT EXISTS is performed in the satisfies relation of filter expressions.

Definition 3. *Given a solution mapping* μ *and a graph pattern* P *over an RDF dataset* $D(G)$ *then* $\mu \models_{D(G)}$ EXISTS(P) *(resp.* $\mu \models_{D(G)}$ NOT EXISTS(P)*) iff* $[\![\mu(P)]\!]_{D(G)}$ *is a non-empty (resp. empty) multiset.*

Example 2. The SPARQL query of Example 1 corresponds to the following graph pattern :

$$\begin{aligned} Q = (\ & (?who, <\texttt{foaf:account}>, ?acc)\ \texttt{OPTIONAL} \\ & (?acc, <\texttt{foaf:accountServiceHomepage}>, ?home) \\ ) & \end{aligned}$$

The evaluation of the query result with respect to the RDF dataset $D = \{G\}$, just containing the default graph G, specified in the example is:

$$\begin{aligned} & [\![Q]\!]_{D(G)} = \\ & [\![(?who, <\texttt{foaf:account}>, ?acc)]\!]_{D(G)} ⟕^{D(G)}_{true} \\ & [\![(?acc, <\texttt{foaf:accountServiceHomepage}>, ?home)]\!]_{D(G)} \\ & = \{\!| \{(?who, <\texttt{http://people/david}>), (?acc, <\texttt{http://bank}>)\}, \\ & \quad \{(?who, <\texttt{http://people/felix}>), (?acc, <\texttt{http://games}>)\} |\!\} ⟕^{D(G)}_{true} \\ & \quad \{\!| \{(?acc, <\texttt{http://bank}>), (?home, <\texttt{http://bank/yourmoney}>)\}\ |\!\} \\ & = \{\!| \{(?who, <\texttt{http://people/david}>), (?acc, <\texttt{http://bank}>), \\ & \quad (?home, <\texttt{http://bank/yourmoney}>)\}, \\ & \quad \{(?who, <\texttt{http://people/felix}>), (?acc, <\texttt{http://games}>)\} \\ & |\!\} \end{aligned}$$

The evaluation of query Q returns, as expected, two solution mappings.

4 Translating SPARQL Algebra into Relational Algebra

The rationale for obtaining how-provenance for SPARQL is to represent each solution mapping as a tuple of a relational algebra query constructed from the original SPARQL graph pattern. The construction is intricate and fully specified, and is inspired from the translation of full SPARQL 1.0 queries into SQL, as detailed in [6], and into Datalog in [11]. Here, we follow a similar strategy but for simplicity of presentation we assume that a given RDF dataset $D = \{G_0, (<\mathtt{u_1}>, G_1), (<\mathtt{u_2}>, G2), \ldots, (<\mathtt{u_n}>, G_n)\}$ is represented by the two relations: `Graphs(gid,IRI)` and `Quads(gid,sub,pred,obj)`. The former stores information about the graphs in the dataset D where `gid` is a numeric graph identifier, and `IRI` an IRI reference. The relation `Quads` stores the triples of every graph in the RDF dataset. Different implementations may immediately adapt the translation provided here in this section to their own schema.

Relation `Graphs(gid,IRI)` contains a tuple $(i, <\mathtt{u_i}>)$ for each named graph $(<\mathtt{u_i}>, G_i)$, and the tuple $(0, <\ >)$ for the default graph, while relation `Quads(gid,sub,pred,obj)` stores a tuple of the form (i, s, p, o) for each triple $(s, p, o) \in G_i$[5]. With this encoding, the default graph always has identifier 0, and all the graph identifiers are consecutive integers.

It is also assumed the existence of a special value `unb`, distinct from the encoding of any RDF term, to represent that a particular variable is unbound in the solution mapping. This is required in order to be able to represent solution mappings as tuples with fixed and known arity. Moreover, we assume that the variables are totally ordered (e.g. lexicographically). The translation requires the full power of relational algebra, and notice that bag semantics is assumed (duplicates are allowed) in order to obey to the cardinality restrictions of SPARQL algebra operators [1].

Definition 4 (Translation of triple patterns). *Let $t = (s, p, o)$ be a triple pattern and G an attribute. Its translation $[(s, p, o)]_{\mathcal{R}}^{G}$ into relational algebra is constructed from relation* `Quads` *as follows:*

1. *Select the tuples with the conjunction obtained from the triple pattern by letting* `Quads.sub` $= s$ *(resp.* `Quads.pred` $= p$, `Quads.obj` $= o$*) if s (resp. p, o) are RDF terms; if a variable occurs more than once in t, then add an equality condition among the corresponding columns of* `Quads`*;*
2. *Rename* `Quads.gid` *as G; rename as many as* `Quads` *columns as distinct variables that exist in t, such that there is exactly one renamed column per variable;*
3. *Project in G and variables occurring in t;*

The empty graph pattern is translated as $[()]_{\mathcal{R}}^{G} = \Pi_G \left[\rho_{G \leftarrow \mathtt{gid}}(\mathtt{Graphs})\right]$.

Example 3. Consider the following triple patterns:

$t_1 = (?who, <\mathtt{http://xmlns.com/foaf/0.1/account}>, ?acc)$
$t_2 = (?who, <\mathtt{http://xmlns.com/foaf/0.1/knows}>, ?who)$
$t_3 = (<\mathtt{http://cd}>, <\mathtt{http://xmlns.com/foaf/0.1/name}>, \text{"Carlos"@pt})$

[5] For simplicity `sub`, `pred`, and `obj` are text attributes storing lexical forms of the triples' components. We assume that datatype literals have been normalized, and blank nodes are distinct in each graph. The only constraint is that different RDF terms must be represented by different strings; this can be easily guaranteed.

The corresponding translations into relational algebra are:

$$[t_1]^G_{\mathcal{R}} = \Pi_{G,acc,who}\left[\rho_{\substack{G \leftarrow \texttt{gid}\\ acc \leftarrow \texttt{obj}\\ who \leftarrow \texttt{sub}}}\left(\sigma_{\texttt{pred=<http://xmlns.com/foaf/0.1/account>}}(\texttt{Quads})\right)\right]$$

$$[t_2]^G_{\mathcal{R}} = \Pi_{G,who}\left[\rho_{\substack{G \leftarrow \texttt{gid}\\ who \leftarrow \texttt{sub}}}\left(\sigma_{\substack{\texttt{pred =< http : //xmlns.com/foaf/0.1/knows>}\\ \wedge\\ \texttt{sub = obj}}}(\texttt{Quads})\right)\right]$$

$$[t_3]^G_{\mathcal{R}} = \Pi_{G}\left[\rho_{G \leftarrow \texttt{gid}}\left(\sigma_{\substack{\texttt{sub =< http : //cd>} \wedge\\ \texttt{pred =< http : //xmlns.com/foaf/0.1/name>} \wedge\\ \texttt{obj = "Carlos"@pt}}}(\texttt{Quads})\right)\right]$$

The remaining pattern that requires querying base relations is GRAPH:

Definition 5 (Translation of GRAPH pattern). *Consider the graph pattern* (GRAPH *term* P_1) *and let* G' *be a new attribute name.*

– *If term is an IRI then* $[(\texttt{GRAPH}\ term\ P_1)]^G_{\mathcal{R}}$ *is*

$$[()]^G_{\mathcal{R}} \bowtie \Pi_{var(P_1)}\left[\Pi_{G'}\left(\rho_{G' \leftarrow \texttt{gid}}\left(\sigma_{term=\texttt{IRI}}(\texttt{Graphs})\right)\right) \bowtie [P_1]^{G'}_{\mathcal{R}}\right]$$

– *If term is a variable* v *then* $[(\texttt{GRAPH}\ term\ P_1)]^G_{\mathcal{R}}$ *is*

$$[()]^G_{\mathcal{R}} \bowtie \Pi_{\{v\} \cup var(P_1)}\left[\rho_{G' \leftarrow \texttt{gid}, v \leftarrow \texttt{IRI}}\left(\sigma_{\texttt{gid}>0}(\texttt{Graphs})\right) \bowtie [P_1]^{G'}_{\mathcal{R}}\right]$$

Notice that the relational algebra query resulting from the translation of the pattern graph P_1 renames and hides the graph attribute. The join of the empty pattern is included in order to guarantee that each query returns the graph identifier in the first "column".

Definition 6 (Translation of the UNION pattern). *Consider the graph pattern* (P_1 UNION P_2). *The relation algebra expression* $[(P_1\ \texttt{UNION}\ P_2)]^G_{\mathcal{R}}$ *is:*

$$\begin{array}{c}\Pi_{G,var(P_1) \cup \{v \leftarrow \texttt{unb} \mid v \in var(P_2) \setminus var(P_1)\}}\left([P_1]^G_{\mathcal{R}}\right)\\ \bigcup\\ \Pi_{G,var(P_2) \cup \{v \leftarrow \texttt{unb} \mid v \in var(P_1) \setminus var(P_2)\}}\left([P_2]^G_{\mathcal{R}}\right)\end{array}$$

The union operator requires the use of an extended projection in order to make unbound variables which are present in one pattern but not in the other. The ordering of the variables in the projection must respect the total order imposed in the variables. This guarantees that the attributes are the same and by the same order in the resulting argument expressions of the union operator.

Definition 7 (Translation of the `AND` pattern). *Consider the graph pattern* $(P_1 \texttt{ AND } P_2)$ *and let* $var(P_1) \cap var(P_2) = \{v_1, \ldots, v_n\}$ *(which may be empty). The relational algebra expression* $[(P_1 \texttt{ AND } P_2)]^G_{\mathcal{R}}$ *is*

$$\Pi_{\substack{G,\\ var(P_1)-var(P_2),\\ var(P_2)-var(P_1),\\ v_1 \leftarrow first(v_1', v_1''),\ldots,\\ v_n \leftarrow first(v_n', v_n'')}} \left[\sigma_{comp} \left(\rho_{\substack{v_1' \leftarrow v_1\\ \vdots \\ v_n' \leftarrow v_n}} \left([P_1]^G_{\mathcal{R}}\right) \bowtie \rho_{\substack{v_1'' \leftarrow v_1\\ \vdots \\ v_n'' \leftarrow v_n}} \left([P_2]^G_{\mathcal{R}}\right) \right) \right]$$

where comp is a conjunction of conditions $v_i' = \texttt{unb} \vee v_i'' = \texttt{unb} \vee v_i' = v_i''$ *for each variable* $v_i (1 \le i \le n)$*. The function* $first$ *returns the first argument which is not* `unb`*, or* `unb` *if both arguments are* `unb`*. Note that if the set of common variables is empty then the relational algebra expression simplifies to:*

$$\Pi_{G, var(P_1)\cup var(P_2)} \left[[P_1]^G_{\mathcal{R}} \bowtie [P_2]^G_{\mathcal{R}} \right]$$

We need to rename common variables in both arguments, since an unbound variable is compatible with any bound or unbound value in order to be able to check compatibility using a selection (it is well-known that the semantics of `unb` is different from semantics of NULLs in relational algebra). The use of the $first$ function in the extended projection is used to obtain in the solution the bound value of the variable, whenever it exists. This technique is the same with that used in [6,11]. The use of the extended projection is not essential, since it can be translated into a more complex relational algebra query by using an auxiliary relation containing a tuple for each pair of compatible pairs of variables.

Definition 8 (Translation of the `MINUS` pattern). *Consider the graph pattern* $(P_1 \texttt{ MINUS } P_2)$ *and let* $var(P_1) \cap var(P_2) = \{v_1, \ldots, v_n\}$ *(which may be empty). The relational algebra expression* $[(P_1 \texttt{ MINUS } P_2)]^G_{\mathcal{R}}$ *is*

$$[P_1]^G_{\mathcal{R}} \bowtie \left[\delta\left([P_1]^G_{\mathcal{R}}\right) - \Pi_{G, var(P_1)} \left[\sigma_{comp \wedge \neg disj} \left([P_1]^G_{\mathcal{R}} \bowtie \rho_{\substack{v_1' \leftarrow v_1\\ \vdots \\ v_n' \leftarrow v_n}} \left([P_2]^G_{\mathcal{R}}\right) \right) \right] \right]$$

where comp is a conjunction of conditions $v_i = \texttt{unb} \vee v_i' = \texttt{unb} \vee v_i = v_i'$ *for each variable* $v_i (1 \le i \le n)$*, and disj is the conjunction of conditions* $v_i = \texttt{unb} \vee v_i' = \texttt{unb}$ *for each variable* $v_i (1 \le i \le n)$*. Note that if the set of common variables is empty then the above expression reduces to* $[P_1]^G_{\mathcal{R}}$ *since* $disj = true$*.*

This is the first of the non-monotonic SPARQL patterns, and deserves some extra explanation. We need to check dynamically if the domains of variables are disjoint since we do not know at translation time what are the unbound variables in the solution mappings, except when trivially the arguments of MINUS do not share any variable. The expression on the right hand side of the difference operator returns a tuple corresponding to a solution mapping μ_1 of P_1 whenever it is possible to find a solution mapping μ_2 of P_2 that it is compatible with μ_1 (condition *comp*) and the mappings do not have disjoint domains (condition $\neg disj$). By deleting these tuples (solutions) from solutions of P_1 we negate the condition, and capture the semantics of the MINUS operator. The use of the duplicate elimination δ ensures that only one tuple is obtained for each solution mapping, in order to guarantee that the cardinality of the result is as what is specified by SPARQL semantics: each tuple in $[P_1]^G_{\mathcal{R}}$ joins with at most one tuple (itself) resulting from the difference operation.

Definition 9 (Translation of FILTER pattern). *Consider the graph pattern* (P FILTER R)*, and let* [NOT] EXISTS(P_1), *…,* [NOT] EXISTS(P_m) *the* EXISTS *or* NOT EXISTS *filter expressions occurring in R (which might not occur). The relational algebra expression* $[(P \texttt{ FILTER } R)]^G_{\mathcal{R}}$ *is*

$$\Pi_{G,var(P)}\left[\sigma_{filter}\left([P]^G_{\mathcal{R}} \bowtie E_1 \bowtie \ldots \bowtie E_m\right)\right]$$

where $filter$ is a condition obtained from R where each occurrence of EXISTS(P_i) *(resp.* NOT EXISTS(P_i)*) is substituted by condition $ex_i <> 0$ (resp. $ex_i = 0$), where ex_i is a new attribute name. Expression $E_i (1 \leq i \leq m)$ is:*

$$\Pi_{G,var(P),ex_i \leftarrow 0}\left[\delta(P') - \Pi_{G,var(P)}\left(\sigma_{subst}\left(P' \bowtie \rho_{\substack{v'_1 \leftarrow v_1 \\ \vdots \\ v'_n \leftarrow v_n}}(P'_i)\right)\right)\right]$$

$$\cup$$

$$\Pi_{G,var(P),ex_i \leftarrow 1}\left[\delta(P') - \left[\delta(P') - \Pi_{G,var(P)}\left(\sigma_{subst}\left(P' \bowtie \rho_{\substack{v'_1 \leftarrow v_1 \\ \vdots \\ v'_n \leftarrow v_n}}(P'_i)\right)\right)\right]\right]$$

where $P' = [P]^G_{\mathcal{R}}$, $P'_i = [P_i]^G_{\mathcal{R}}$, and subst is the conjunction of conditions $v_i = v'_i \vee v_i =$ unb *for each variable v_i in $var(P) \cap var(P_i) = \{v_1, \ldots, v_n\}$. Note that if there are no occurrences of* EXISTS *patterns, then* $[(P \texttt{ FILTER } R)]^G_{\mathcal{R}}$ *is* $\sigma_R\left([P]^G_{\mathcal{R}}\right)$.

The translation of FILTER expressions turns out to be very complex due to the EXISTS patterns. For each exists expression we need to introduce an auxiliary expression returning a unique tuple for each solution mapping of P, the top expression when the pattern P_i does not return any solution, and the bottom expression when it does. We need the double negation in order to not affect the cardinality of the results of the filter operation when pattern P returns more than one solution. Obviously, our translation depends on the capability of expressing arbitrary SPARQL conditions as relational algebra conditions; this is not immediate but assumed possible due to the translation provided in [6].

We can now conclude our translation by taking care of the `OPTIONAL` graph pattern, since it depends on the translation of filter patterns:

Definition 10 (Translation of `OPTIONAL` pattern). *Consider the graph pattern* $(P_1 \texttt{ OPTIONAL } (P_2 \texttt{ FILTER } R))$.
The relational algebra expression $[(P_1 \texttt{ OPTIONAL } (P_2 \texttt{ FILTER } R))]^G_{\mathcal{R}}$ *is*

$$
\begin{gathered}
[(P_1 \texttt{ AND } P_2)]^G_{\mathcal{R}} \\
\bigcup \\
\Pi_{G, var(P_1) \cup \{v \leftarrow \texttt{unb} \mid v \in var(P_2) \setminus var(P_1)\}} \\
\left[[P_1]^G_{\mathcal{R}} \bowtie \left(\begin{array}{c} \delta\left([P_1]^G_{\mathcal{R}}\right) \\ - \\ \Pi_{G, var(P_1)}\left([(P_1 \texttt{ AND } P_2) \texttt{ FILTER } R]^G_{\mathcal{R}}\right) \end{array} \right) \right]
\end{gathered}
$$

The translation of $(P_1 \texttt{ OPTIONAL } P_2)$ *is obtained from the translation of the graph pattern* $(P_1 \texttt{ OPTIONAL } (P_2 \texttt{ FILTER } true))$.

The translation of the `OPTIONAL` pattern has two parts, one corresponding to the `JOIN` operator (top expression) and one corresponding to the **Diff** operator. The translation of the **Diff** operator uses the same technique as the `MINUS` operator but now we remove from solutions of P_1 those solution mappings of P_1 that are compatible with a mapping of P_2 and that satisfy the filter expression.

Theorem 1 (Correctness of translation). *Given a graph pattern P and a RDF dataset $D(G)$ the process of evaluating the query is performed as follows:*

1. *Construct the base relations* `Graphs` *and* `Quads` *from $D(G)$;*
2. *Evaluate* $[SPARQL(P, D(G), V)]_{\mathcal{R}} = \Pi_V \left[\sigma_{G'=0}\left([()]^{G'}_{\mathcal{R}} \bowtie [P]^{G'}_{\mathcal{R}}\right)\right]$ *with respect to the base relations* `Graphs` *and* `Quads`, *where G' is a new attribute name and $V \subseteq var(P)$.*

Moreover, the tuples of relational algebra query (2) are in one-to-one correspondence with the solution mappings of $[\![P]\!]_{D(G)}$ when $V = var(P)$, and where an attribute mapped to `unb` *represents that the corresponding variable does not belong to the domain of the solution mapping.*

Proof. The proof is by by structural induction on the graph patterns and can be found in the extended version of this paper available at `http://arxiv.org/abs/1209.0378`.

The constructed translation will be used to extract how-provenance information for SPARQL queries, addressing the problems identified in [12].

5 Provenance for SPARQL Queries

The crux of the method has been specified in the previous section, and relies on the properties of the extended provenance m-semiring $\mathcal{K}_{dprovd}$ for language $\mathcal{RA}^{+}_{\mathcal{K}}(-, \delta)$. We just need a definition before we illustrate the approach.

Definition 11 (Provenance for SPARQL). *Given a graph pattern P and a RDF dataset $D(G)$ the provenance for P is obtained as follows:*

- *Construct the base $\mathcal{K}_{dprovd}$-relations by annotating each tuple in* `Graphs` *and* `Quads` *with a new identifier;*
- *Construct an annotated query $SPARQL(P, D(G), V)_{\mathcal{K}_{dprovd}}$ from relational algebra $[SPARQL(P, D(G), V)]_{\mathcal{R}}$ expression by substituting the duplicate elimination operator by δ_1 where 1 is the identity element of $\mathcal{K}_{dprovd}$.*

The provenance information for P is the annotated relation obtained from evaluating $SPARQL(P, D(G), V)_{\mathcal{K}_{dprovd}}$ with respect to the annotated translation of the dataset $D(G)$.

By the factorization property of $\mathcal{K}_{dprovd}$ we know that this is the most general m-semiring, and thus the provenance obtained according to Definition 11 is the most informative one. We just need to illustrate the approach with Example 1 in order to completely justify its appropriateness.

Example 4. First, we represent the RDF dataset by $\mathcal{K}_{dprovd}$-relations where the annotation tags are shown in the last column. The IRIs have been abbreviated:

`Graphs`

gid	IRI	
0	< >	g_0

`Quads`

gid	sub	pred	obj	
0	<david>	<account>	<bank>	t_1
0	<felix>	<account>	<games>	t_2
0	<bank>	<accountServiceHomepage>	<bank/yourmoney>	t_3

Returning to query $Q = (Q_1$ `OPTIONAL` $Q_2)$ of Example 2 with (sub)patterns $Q_1 = (?w, <\texttt{account}>, ?a)$ and $Q_2 = (?a, <\texttt{accountServiceHomepage}>, ?h)$, we obtain the following expressions for Q_1 and Q_2:

$$[Q_1]^G_{\mathcal{R}} = \Pi_{G,w,a} \left[\rho_{\substack{G \leftarrow \texttt{gid} \\ w \leftarrow \texttt{sub} \\ a \leftarrow \texttt{obj}}} (\sigma_{\texttt{pred}=<\texttt{account}>}(\texttt{Quads})) \right]$$

$$[Q_2]^G_{\mathcal{R}} = \Pi_{G,a,h} \left[\rho_{\substack{G \leftarrow \texttt{gid} \\ a \leftarrow \texttt{sub} \\ h \leftarrow \texttt{obj}}} (\sigma_{\texttt{pred}=<\texttt{accountServiceHomepage}>}(\texttt{Quads})) \right]$$

returning the annotated relations:

$$[Q_1]^G_{\mathcal{R}} = \begin{array}{lll||l} G & w & a & \\ \hline 0 & \texttt{<david>} & \texttt{<bank>} & t_1 \\ 0 & \texttt{<felix>} & \texttt{<games>} & t_2 \end{array}$$

$$[Q_2]^G_{\mathcal{R}} = \begin{array}{lll||l} G & a & h & \\ \hline 0 & \texttt{<bank>} & \texttt{<bank/yourmoney>} & t_3 \end{array}$$

The expression $[(Q_1 \texttt{ AND } Q_2)]^G_{\mathcal{R}}$ used in the construction of the expression for the `OPTIONAL` pattern is:

$$\Pi_{G,w,a\leftarrow first(a',a''),h} \left[\sigma_{a'=a''\vee a'=\texttt{unb}\vee a''=\texttt{unb}} \left(\rho_{a''\leftarrow a}\left([Q_1]^G_{\mathcal{R}}\right) \bowtie \rho_{a'\leftarrow a}\left([Q_2]^G_{\mathcal{R}}\right)\right)\right]$$

obtaining the annotated relation:

$$[(Q_1 \texttt{ AND } Q_2)]^G_{\mathcal{R}} = \begin{array}{llll||l} G & w & a & h & \\ \hline 0 & \texttt{<david>} & \texttt{<bank>} & \texttt{<bank/yourmoney>} & t_1 \times t_3 \end{array}$$

We also need to determine the value of $\delta_1([Q_1]^G_{\mathcal{R}})$ which is simply:

$$\delta_1([Q_1)]^G_{\mathcal{R}}) = \begin{array}{lll||l} G & w & a & \\ \hline 0 & \texttt{<david>} & \texttt{<bank>} & 1 \\ 0 & \texttt{<felix>} & \texttt{<games>} & 1 \end{array}$$

We can now construct the expression corresponding to the **Diff** operator of SPARQL algebra, namely:

$$\Pi_{G,w,a,h\leftarrow\texttt{unb}} \left[[Q_1]^G_{\mathcal{R}} \bowtie \left(\begin{array}{c} \delta_1\left([Q_1]^G_{\mathcal{R}}\right) \\ - \\ \Pi_{G,w,a}\left([(Q_1 \texttt{ AND } Q_2)]^G_{\mathcal{R}}\right) \end{array} \right) \right]$$

returning the annotated tuples:

G	w	a	h	
0	`<david>`	`<bank>`	`unb`	$t_1 \times (1 - (t_1 \times t_3))$
0	`<felix>`	`<games>`	`unb`	$t_2 \times (1 - 0) = t_2$

This is the important step, since K-relations assign an annotation to every possible tuple in the domain. If it is not in the support of the relation, then it is tagged with 0. Therefore, the solutions for $([(Q_1 \texttt{ AND } Q_2)]^G_{\mathcal{R}}$ are:

G	w	a	h	
0	`<david>`	`<bank>`	`<bank/yourmoney>`	$t_1 \times t_3$
0	`<david>`	`<bank>`	`unb`	$t_1 \times (1 - (t_1 \times t_3))$
0	`<felix>`	`<games>`	`unb`	t_2

and for our query, finally we get

w	a	h	
`<david>`	`<bank>`	`<bank/yourmoney>`	$g_0 \times t_1 \times t_3$
`<david>`	`<bank>`	`unb`	$g_0 \times t_1 \times (1 - (t_1 \times t_3))$
`<felix>`	`<games>`	`unb`	$g_0 \times t_2$

The interpretation of the results is the expected and intuitive one. Suppose that (i) we use the boolean m-semiring, with just the two values `t` and `f`, meaning that we trust or not trust a triple, (ii) product corresponds to conjunction, (iii) sum corresponds to disjunction, and (iv) difference is defined as $x - y = x \wedge \neg y$. So, if we trust g_0 and t_1, t_2 and t_3 we are able to conclude that we trust the first and third solutions (substitute 1 and the identifiers of trusted triples by `t` in the annotations, and then evaluate the resulting boolean expression). If we do not trust t_3 but trust the other triples then we trust the second and third solutions. Also mark how the graph provenance is also annotated in our solutions. Accordingly, if we don't trust the default graph then we will not trust any of the solutions. Therefore, our method was capable of keeping in the same annotated $\mathcal{K}_{dprovd}$-relation the several possible alternative solutions, one in each distinct tuple. This was claimed to not be possible in [12].

6 Discussion and Conclusions

The literature describes several approaches to extract data provenance/annotated information from RDF(S) data [7,5,4,12,3]. A first major distinction is that we extract how-provenance instead of only why-provenance[6] of [7,5,4,3]. Both [7,4] address the problem of extracting data provenance for RDF(S) entailed triples, but do not support SPARQL. The theory developed in [5] implements the difference operator using a negation, but it does not handle duplicate solutions according to the semantics of SPARQL because of idempotence of sum; additionally, the proposed difference operator to handle why-provenance discards the information in the right hand argument. The most complete work is [3] which develops a framework for annotated Semantic Web data, supporting RDFS entailment and providing a query language extending many of the SPARQL features in order to deal with annotated data, exposing annotations at query level via annotation variables, and including aggregates and subqueries (but not property path patterns). However, the sum operator is idempotent in order to support RDFS entailment, and by design the `UNION` operator is not interpreted in the annotation domain. Moreover, the `OPTIONAL` graph pattern discards in some situations the information in the second argument, and thus cannot extract full provenance information.

The capability of extracting full data how-provenance for SPARQL semantics as prescribed in [12] has been shown possible with our work, refuting their claim that existing algebras could not be used for SPARQL. Our approach, like [12], rests on a translation of SPARQL into annotated relational algebra contrasting with the abstract approach of [7,5,4,3]. The authors in [12] argue that this translation process does not affect the output provenance information for the case of (positive) SPARQL. In this way, the major constructs of SPARQL 1.1 are taken care respecting their bag semantics. However, contrary to the works of [7,3] we do not address the RDF schema entailment rules, and therefore our work is only applicable to simple entailment.

[6] We use the terminology "how-" and "why-provenance" in the sense of [9].

We plan to address the complete semantics of SPARQL. In particular, aggregates can be handled by summing ($\oplus$) tuples for each group, while property path patterns can generate annotation corresponding to products ($\otimes$) of the involved triples in each solution. This extension is enough to be able to capture data provenance for RDFS entailment. We also want to explore additional applications in order to assess fully the potential of the proposed method.

References

1. SPARQL 1.1 query language, 2012. W3C Working Draft (January 05, 2012), `http://www.w3.org/TR/2012/WD-sparql11-query-20120105/`
2. Amer, K.: Equationally complete classes of commutative monoids with monus. Algebra Universalis 18, 129–131 (1984)
3. Antoine Zimmermann, A.P., Lopes, N., Straccia, U.: A general framework for representing, reasoning and querying with annotated semantic web data. Journal of Web Semantics 11, 72–95 (2012)
4. Buneman, P., Kostylev, E.V.: Annotation algebras for RDFS. In: Proc. of the 2nd Int. Ws. on the Role of Semantic Web in Provenance Management (SWPM 2010). CEUR Workshop Proceedings (2010)
5. Dividino, R., Sizov, S., Staab, S., Schueler, B.: Querying for provenance, trust, uncertainty and other meta knowledge in RDF. Web Semant. 7(3), 204–219 (2009)
6. Elliott, B., Cheng, E., Thomas-Ogbuji, C., Ozsoyoglu, Z.M.: A complete translation from SPARQL into efficient SQL. In: Proc. of the 2009 Int. Database Engineering & Applications Symposium, IDEAS 2009, pp. 31–42. ACM (2009)
7. Flouris, G., Fundulaki, I., Pediaditis, P., Theoharis, Y., Christophides, V.: Coloring RDF Triples to Capture Provenance. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 196–212. Springer, Heidelberg (2009)
8. Geerts, F., Poggi, A.: On database query languages for K-relations. J. Applied Logic 8(2), 173–185 (2010)
9. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proc. of PODS 2007, pp. 31–40. ACM, New York (2007)
10. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3), 16:1–16:45 (2009)
11. Polleres, A.: From SPARQL to rules (and back). In: Williamson, C.L., Zurko, M.E., Patel-Schneider, P.F., Shenoy, P.J. (eds.) Proc. of the 16th Int. Conf. on World Wide Web, WWW 2007, pp. 787–796. ACM (2007)
12. Theoharis, Y., Fundulaki, I., Karvounarakis, G., Christophides, V.: On provenance of queries on semantic web data. IEEE Internet Computing 15(1), 31–39 (2011)