

Provenance for SPARQL queries

C. V. Damásio¹ and A. Analyti² and G. Antoniou³

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 [8], 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

⁰ Extended version of the ISWC 2012 paper including proofs, published in Lecture Notes on Computer Science 7649, pp. 625-640, ISBN 978-3-642-35175-4, Springer Berlin Heidelberg, 2012. The original publication is available at www.springerlink.com.

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 [8]). This work has been extended to the case of full relational algebra in [7] by considering the notion of semirings with a monus operation (m -semirings [1]) and constant annotations, and the factorization property is proved for the special m -semiring that we denote by \mathcal{K}_{dprov} .

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 [8,7], 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 \mathcal{K}_{dprov} -relation (with the obvious ab-

breviations), 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 $t : U \rightarrow \mathbb{D}$ 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 $U\text{-Tup}$ and usual relations are subsets of $U\text{-Tup}$. A \mathcal{K} -relation over U is a function $R : U\text{-Tup} \rightarrow \mathbb{K}$, and its support is $\text{supp}(R) = \{t \mid R(t) \neq 0\}$.

In order to cover the full relational operators, the authors in [7] 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 language⁴ $\mathcal{RA}_{\mathcal{K}}^+(-, \delta)$ has the following operators [7]:

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

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

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

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

natural join If $R_i : U_i\text{-Tup} \rightarrow \mathbb{K}$, 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 $R : U\text{-Tup} \rightarrow \mathbb{K}$ and $\beta : U \rightarrow U'$ 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 $R_1, R_2 : U\text{-Tup} \rightarrow \mathbb{K}$ then $R_1 - R_2 : U\text{-Tup} \rightarrow \mathbb{K}$ is defined by:
 $(R_1 - R_2)(t) = R_1(t) \ominus R_2(t)$.

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

One major result of [7] 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}_{dprov} . \mathcal{K}_{dprov} 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}_{dprov} 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}_{dprov} 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 [7] 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 [9] 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

⁴ The authors use instead the notation $\mathcal{RA}_{\mathcal{K}}^+(\setminus, \delta)$.

semantics of the first version of SPARQL. The aim of our paper is on the treatment of non-monotonic constructs⁵ 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 [9]. For more details the reader is referred to sections 17 and 18 of the current SPARQL 1.1 W3C recommendation. 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 triple⁶ $\tau = (s, p, o)$ 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 $D = \{G, (<u_1 >, G_1), (<u_2 >, G_2), \dots, (<u_n >, G_n)\}$, where G is the default graph, and each pair $(<u_i >, G_i)$ is called a named graph, with each IRI 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 pattern⁷;
- If P_1 and P_2 are graph patterns then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, as well as $(P_1 \text{ MINUS } P_2)$, and $(P_1 \text{ OPTIONAL } P_2)$ are graph patterns;
- If P_1 is a graph pattern and R is a filter SPARQL expression⁸ then the construction $(P_1 \text{ FILTER } R)$ is a graph pattern;
- If P_1 is a graph pattern and $term$ is a variable or an IRI then $(\text{GRAPH } term \ P_1)$ 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, \dots, P_n is

⁵ OPTIONAL is non-monotonic when combined with some FILTER expressions, or if we understand unbound variables incomparable to bounded ones like we assume in the translation.

⁶ Literals in the subject of triples are allowed, since this generalization is expected to be adopted in the near future.

⁷ For simplicity, we do not allow blank nodes in triple patterns.

⁸ For the full syntax of filter expressions, see the W3C recommendation [9].

encoded as the graph pattern $((() \text{ AND } (P_1 \text{ AND } (P_2 \dots \text{ AND } P_n)) \dots))$. 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 $\mu : \mathbf{V} \rightarrow \mathbf{T}$. The 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 $\text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ 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 $v \rightarrow t$. 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 [9]). *Let Ω_1 and Ω_2 be multisets of solution mappings, and R a filter expression. Define:*

Join: $\Omega_1 \bowtie \Omega_2 = \{\{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \text{ and } \mu_2 \in \Omega_2 \text{ such that } \mu_1 \text{ and } \mu_2 \text{ are compatible}\}\}$

Union: $\Omega_1 \cup \Omega_2 = \{\{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}\}$

Minus: $\Omega_1 - \Omega_2 = \{\{\mu_1 \mid \mu_1 \in \Omega_1 \text{ such that } \forall \mu_2 \in \Omega_2 \text{ either } \mu_1 \text{ and } \mu_2 \text{ are not compatible or } \text{dom}(\mu_1) \cap \text{dom}(\mu_2) = \emptyset\}\}$

Diff: $\Omega_1 \setminus_R^{D(G)} \Omega_2 = \{\{\mu_1 \mid \mu_1 \in \Omega_1 \text{ such that } \forall \mu_2 \in \Omega_2 \text{ either } \mu_1 \text{ and } \mu_2 \text{ are not compatible, or } \mu_1 \text{ and } \mu_2 \text{ are compatible and } \mu_1 \cup \mu_2 \not\models_{D(G)} R\}\}$

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

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 $\llbracket \cdot \rrbracket_{D(G)}$ is defined recursively as follows:*

1. $\llbracket () \rrbracket_{D(G)} = \{\mu_0\}$;
2. $\llbracket t \rrbracket_{D(G)} = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G \}$, where $\text{var}(t)$ is the set of variables occurring in the triple pattern t ;
3. $\llbracket (P_1 \text{ AND } P_2) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G)} \bowtie \llbracket P_2 \rrbracket_{D(G)}$;
4. $\llbracket (P_1 \text{ UNION } P_2) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G)} \cup \llbracket P_2 \rrbracket_{D(G)}$;
5. $\llbracket (P_1 \text{ MINUS } P_2) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G)} - \llbracket P_2 \rrbracket_{D(G)}$;
6. $\llbracket (P_1 \text{ OPTIONAL } P_2) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G)} \bowtie_{\text{true}}^{D(G)} \llbracket P_2 \rrbracket_{D(G)}$, where P_2 is not a FILTER pattern;
7. $\llbracket (P_1 \text{ OPTIONAL } (P_2 \text{ FILTER } R)) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G)} \bowtie_R^{D(G)} \llbracket P_2 \rrbracket_{D(G)}$;
8. $\llbracket (P_1 \text{ FILTER } R) \rrbracket_{D(G)} = \{ \mu \in \llbracket P_1 \rrbracket_{D(G)} \mid \mu \models_{D(G)} R \}$;
9. Evaluation of $\llbracket (\text{GRAPH term } P_1) \rrbracket_{D(G)}$ depends on the form of term:
 - If term is an IRI corresponding to a graph name u_i in $D(G)$ then $\llbracket (\text{GRAPH term } P_1) \rrbracket_{D(G)} = \llbracket P_1 \rrbracket_{D(G_i)}$;
 - If term is an IRI that does not correspond to any graph in $D(G)$ then $\llbracket (\text{GRAPH term } P_1) \rrbracket_{D(G)} = \{\}$;
 - If term is a variable v then $\llbracket (\text{GRAPH term } P_1) \rrbracket_{D(G)} =$

$$= (\llbracket P_1 \rrbracket_{D(G_1)} \bowtie \{v \rightarrow \langle u_1 \rangle\}) \cup \dots \cup (\llbracket P_1 \rrbracket_{D(G_n)} \bowtie \{v \rightarrow \langle u_n \rangle\})$$

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 $\mu \models_{D(G)} \text{EXISTS}(P)$ (resp. $\mu \models_{D(G)} \text{NOT EXISTS}(P)$) iff $\llbracket \mu(P) \rrbracket_{D(G)}$ is a non-empty (resp. empty) multiset.*

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

$$Q = ((?who, \langle \text{foaf} : \text{account} \rangle, ?acc) \text{ OPTIONAL } \\ (?acc, \langle \text{foaf} : \text{accountServiceHomepage} \rangle, ?home) \\)$$

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:

$$\begin{aligned}
\llbracket Q \rrbracket_{D(G)} &= \\
&\llbracket (?who, < foaf : account >, ?acc) \rrbracket_{D(G)} \bowtie_{true}^{D(G)} \\
&\llbracket (?acc, < foaf : accountServiceHomepage >, ?home) \rrbracket_{D(G)} \\
&= \{ \{ (?who, < http : // people / david >), (?acc, < http : // bank >), \\
&\quad \{ (?who, < http : // people / felix >), (?acc, < http : // games >), \} \} \bowtie_{true}^{D(G)} \\
&\quad \{ \{ (?acc, < http : // bank >), (?home, < http : // bank / yourmoney >), \} \} \\
&= \{ \{ (?who, < http : // people / david >), (?acc, < http : // bank >, \\
&\quad (?home, < http : // bank / yourmoney >), \\
&\quad \{ (?who, < http : // people / felix >), (?acc, < http : // games >), \} \\
&\quad \} \}
\end{aligned}$$

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 [5], and into Datalog in [11]. Here, we follow a similar strategy but for simplicity of presentation we assume that a given RDF dataset $D = \{G_0, (< u_1 >, G_1), (< u_2 >, G_2), \dots, (< u_n >, G_n)\}$ 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 $(i, < u_i >)$ for each named graph $(< u_i >, G_i)$, 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 $(s, p, o) \in G_i$ ⁹. 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 [9].

⁹ 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.

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 [\rho_{G \leftarrow \text{gid}}(\mathbf{Graphs})]$.

Example 3. Consider the following triple patterns:

$$\begin{aligned} t_1 &= (?who, \langle \text{http://xmlns.com/foaf/0.1/account} \rangle, ?acc) \\ t_2 &= (?who, \langle \text{http://xmlns.com/foaf/0.1/known} \rangle, ?who) \\ t_3 &= (\langle \text{http://cd} \rangle, \langle \text{http://xmlns.com/foaf/0.1/name} \rangle, \text{"Carlos"@pt}) \end{aligned}$$

The corresponding translations into relational algebra are:

$$\begin{aligned} [t_1]_{\mathcal{R}}^G &= \Pi_{G, acc, who} \left[\rho_{\substack{G \leftarrow \text{gid} \\ acc \leftarrow \text{obj} \\ who \leftarrow \text{sub}}} (\sigma_{\text{pred} = \langle \text{http://xmlns.com/foaf/0.1/account} \rangle}(\mathbf{Quads})) \right] \\ [t_2]_{\mathcal{R}}^G &= \Pi_{G, who} \left[\rho_{\substack{G \leftarrow \text{gid} \\ who \leftarrow \text{sub}}} \left(\sigma_{\substack{\text{pred} = \langle \text{http://xmlns.com/foaf/0.1/known} \rangle \\ \wedge \\ \text{sub} = \text{obj}}}(\mathbf{Quads}) \right) \right] \\ [t_3]_{\mathcal{R}}^G &= \Pi_G \left[\rho_{G \leftarrow \text{gid}} \left(\sigma_{\substack{\text{sub} = \langle \text{http://cd} \rangle \wedge \\ \text{pred} = \langle \text{http://xmlns.com/foaf/0.1/name} \rangle \wedge \\ \text{obj} = \text{"Carlos"@pt}}}(\mathbf{Quads}) \right) \right] \end{aligned}$$

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 $[(\mathbf{GRAPH term } P_1)]_{\mathcal{R}}^G$ is

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

– If term is a variable v then $[(\mathbf{GRAPH term } P_1)]_{\mathcal{R}}^G$ is

$$[(\)]_{\mathcal{R}}^G \bowtie \Pi_{\{v\} \cup \text{var}(P_1)} \left[\rho_{G' \leftarrow \text{gid}, v \leftarrow \text{IRI}} (\sigma_{\text{gid} > 0}(\mathbf{Graphs})) \bowtie [P_1]_{\mathcal{R}}^{G'} \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 $(P_1 \text{ UNION } P_2)$. The relation algebra expression $[(P_1 \text{ UNION } P_2)]_{\mathcal{R}}^G$ is:

$$\begin{aligned} & \Pi_{G, \text{var}(P_1) \cup \{v \leftarrow \text{unb} \mid v \in \text{var}(P_2) \setminus \text{var}(P_1)\}} \left([P_1]_{\mathcal{R}}^G \right) \\ & \quad \cup \\ & \Pi_{G, \text{var}(P_2) \cup \{v \leftarrow \text{unb} \mid v \in \text{var}(P_1) \setminus \text{var}(P_2)\}} \left([P_2]_{\mathcal{R}}^G \right) \end{aligned}$$

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 $(P_1 \text{ AND } P_2)$ and let $\text{var}(P_1) \cap \text{var}(P_2) = \{v_1, \dots, v_n\}$ (which may be empty). The relational algebra expression $[(P_1 \text{ AND } P_2)]_{\mathcal{R}}^G$ is

$$\Pi_{G, \begin{matrix} \text{var}(P_1) - \text{var}(P_2), \\ \text{var}(P_2) - \text{var}(P_1), \\ v_1 \leftarrow \text{first}(v'_1, v''_1), \dots, \\ v_n \leftarrow \text{first}(v'_n, v''_n) \end{matrix}} \left[\sigma_{\text{comp}} \left(\begin{matrix} \rho_{v'_1 \leftarrow v_1} \left([P_1]_{\mathcal{R}}^G \right) \bowtie \rho_{v''_1 \leftarrow v_1} \left([P_2]_{\mathcal{R}}^G \right) \\ \vdots \\ \rho_{v'_n \leftarrow v_n} \left([P_1]_{\mathcal{R}}^G \right) \bowtie \rho_{v''_n \leftarrow v_n} \left([P_2]_{\mathcal{R}}^G \right) \end{matrix} \right) \right]$$

where comp is a conjunction of conditions $v'_i = \text{unb} \vee v''_i = \text{unb} \vee v'_i = v''_i$ for each variable $v_i (1 \leq i \leq n)$. 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:

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

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 [5,11]. 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 $(P_1 \text{ MINUS } P_2)$ and let $\text{var}(P_1) \cap \text{var}(P_2) = \{v_1, \dots, v_n\}$ (which may be empty). The relational algebra expression $[(P_1 \text{ MINUS } P_2)]_{\mathcal{R}}^G$ is

$$[P_1]_{\mathcal{R}}^G \bowtie \left[\delta \left([P_1]_{\mathcal{R}}^G \right) - \Pi_{G, \text{var}(P_1)} \left[\sigma_{\text{comp} \wedge \neg \text{disj}} \left([P_1]_{\mathcal{R}}^G \bowtie \rho_{v'_1 \leftarrow v_1} \left([P_2]_{\mathcal{R}}^G \right) \right) \right] \right]$$

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

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 $\neg \text{disj}$). 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 $[P_1]_{\mathcal{R}}^G$ joins with at most one tuple (itself) resulting from the difference operation.

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

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

where filter is a condition obtained from R where each occurrence of $\text{EXISTS}(P_i)$ (resp. $\text{NOT EXISTS}(P_i)$) is substituted by condition $ex_i \langle \rangle 0$ (resp. $ex_i = 0$),

where ex_i is a new attribute name. Expression $E_i(1 \leq i \leq m)$ is:

$$\begin{aligned} & \Pi_{G, \text{var}(P), ex_i \leftarrow 0} \left[\delta(P') - \Pi_{G, \text{var}(P)} \left(\sigma_{\text{subst}} \left(P' \bowtie \rho_{v'_1 \leftarrow v_1} (P'_i) \right) \right) \right) \right] \\ & \cup \\ & \Pi_{G, \text{var}(P), ex_i \leftarrow 1} \left[\delta(P') - \left[\delta(P') - \Pi_{G, \text{var}(P)} \left(\sigma_{\text{subst}} \left(P' \bowtie \rho_{v'_1 \leftarrow v_1} (P'_i) \right) \right) \right) \right] \right] \end{aligned}$$

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

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 [5].

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 $(P_1 \text{ OPTIONAL } (P_2 \text{ FILTER } R))$.

The relational algebra expression $[(P_1 \text{ OPTIONAL } (P_2 \text{ FILTER } R))]_{\mathcal{R}}^G$ is

$$\begin{aligned} & [(P_1 \text{ AND } P_2)]_{\mathcal{R}}^G \\ & \cup \\ & \Pi_{G, \text{var}(P_1) \cup \{v \leftarrow \text{unb} \mid v \in \text{var}(P_2) \setminus \text{var}(P_1)\}} \left[\begin{aligned} & \delta([P_1]_{\mathcal{R}}^G) \\ & - \\ & \Pi_{G, \text{var}(P_1)} \left([(P_1 \text{ AND } P_2) \text{ FILTER } R]_{\mathcal{R}}^G \right) \end{aligned} \right] \end{aligned}$$

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

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

Moreover, the tuples of relational algebra query (2) are in one-to-one correspondence with the solution mappings of $\llbracket P \rrbracket_{D(G)}$ when $V = \text{var}(P)$, 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 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}_{dprov} for language $\mathcal{RA}_{\mathcal{K}}^{\pm}(-, \delta)$. 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}_{dprov} -relations by annotating each tuple in **Graphs** and **Quads** with a new identifier;
- Construct an annotated query $SPARQL(P, D(G), V)_{\mathcal{K}_{dprov}}$ 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}_{dprov} .

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

By the factorization property of \mathcal{K}_{dprov} 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}_{dprov} -relations where the annotation tags are shown in the last column. The IRIs have been abbreviated:

Graphs	
gid	IRI
0	< > \parallel_{g_0}

Quads				
gid	sub	pred	obj	
0	< david >	< account >	< bank >	\parallel_{t_1}
0	< felix >	< account >	< games >	\parallel_{t_2}
0	< bank >	< accountServiceHomepage >	< bank/yourmoney >	\parallel_{t_3}

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

$$[Q_1]_{\mathcal{R}}^G = \Pi_{G,w,a} \left[\begin{array}{l} \rho_{G \leftarrow \text{gid}} (\sigma_{\text{pred}=\langle \text{account} \rangle}(\text{Quads})) \\ w \leftarrow \text{sub} \\ a \leftarrow \text{obj} \end{array} \right]$$

$$[Q_2]_{\mathcal{R}}^G = \Pi_{G,a,h} \left[\begin{array}{l} \rho_{G \leftarrow \text{gid}} (\sigma_{\text{pred}=\langle \text{accountServiceHomepage} \rangle}(\text{Quads})) \\ a \leftarrow \text{sub} \\ h \leftarrow \text{obj} \end{array} \right]$$

returning the annotated relations:

$$[Q_1]_{\mathcal{R}}^G = \begin{array}{c} G \quad w \quad a \\ \hline 0 \quad \langle \text{david} \rangle \quad \langle \text{bank} \rangle \\ \hline 0 \quad \langle \text{felix} \rangle \quad \langle \text{games} \rangle \end{array} \parallel_{t_1, t_2}$$

$$[Q_2]_{\mathcal{R}}^G = \begin{array}{c} G \quad a \quad h \\ \hline 0 \quad \langle \text{bank} \rangle \quad \langle \text{bank/yourmoney} \rangle \end{array} \parallel_{t_3}$$

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

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

obtaining the annotated relation:

$$[(Q_1 \text{ AND } Q_2)]_{\mathcal{R}}^G = \begin{array}{c} G \quad w \quad a \quad h \\ \hline 0 \quad \langle \text{david} \rangle \quad \langle \text{bank} \rangle \quad \langle \text{bank/yourmoney} \rangle \end{array} \parallel_{t_1 \times t_3}$$

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

$$\delta_1([Q_1]_{\mathcal{R}}^G) = \begin{array}{c} G \quad w \quad a \\ \hline 0 \quad \langle \text{david} \rangle \quad \langle \text{bank} \rangle \\ \hline 0 \quad \langle \text{felix} \rangle \quad \langle \text{games} \rangle \end{array} \parallel_{t_1} \begin{array}{c} 1 \\ 1 \end{array}$$

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

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

returning the annotated tuples:

$$\begin{array}{cccc|l} G & w & a & h & \\ \hline 0 & \langle \text{david} \rangle & \langle \text{bank} \rangle & \text{unb} & t_1 \times (1 - (t_1 \times t_3)) \\ 0 & \langle \text{felix} \rangle & \langle \text{games} \rangle & \text{unb} & t_2 \times (1 - 0) = t_2 \end{array}$$

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 $([Q_1 \text{ AND } Q_2]_{\mathcal{R}}^G)$ are:

$$\begin{array}{cccc|l} G & w & a & h & \\ \hline 0 & \langle \text{david} \rangle & \langle \text{bank} \rangle & \langle \text{bank/yourmoney} \rangle & t_1 \times t_3 \\ 0 & \langle \text{david} \rangle & \langle \text{bank} \rangle & \text{unb} & t_1 \times (1 - (t_1 \times t_3)) \\ 0 & \langle \text{felix} \rangle & \langle \text{games} \rangle & \text{unb} & t_2 \end{array}$$

and for our query, finally we get

$$\begin{array}{ccc|l} w & a & h & \\ \hline \langle \text{david} \rangle & \langle \text{bank} \rangle & \langle \text{bank/yourmoney} \rangle & g_0 \times t_1 \times t_3 \\ \langle \text{david} \rangle & \langle \text{bank} \rangle & \text{unb} & g_0 \times t_1 \times (1 - (t_1 \times t_3)) \\ \langle \text{felix} \rangle & \langle \text{games} \rangle & \text{unb} & g_0 \times t_2 \end{array}$$

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 \mathbf{t} and \mathbf{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 \mathbf{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}_{dprov} -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 [6,4,3,12,2]. A first major distinction is that we

extract how-provenance instead of only why-provenance¹⁰ of [6,4,3,2]. Both [6,3] address the problem of extracting data provenance for RDF(S) entailed triples, but do not support SPARQL. The theory developed in [4] 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 [2] 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 [6,4,3,2]. 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 [6,2] we do not address the RDF schema entailment rules, and therefore our work is only applicable to simple entailment.

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.

Acknowledgments C. V. Damásio was partially supported by FCT Project ERRO PTDC/EIACCO/121823/2010.

References

1. K. Amer. Equationally complete classes of commutative monoids with monus. *Algebra Universalis*, 18:129–131, 1984.
2. A. P. Antoine Zimmermann, Nuno Lopes and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Journal of Web Semantics*, 11:72–95, March 2012.

¹⁰ We use the terminology “how-” and “why-provenance” in the sense of [8].

3. P. Buneman and E. V. Kostylev. 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.
4. R. Dividino, S. Sizov, S. Staab, and B. Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *Web Semant.*, 7(3):204–219, 2009.
5. B. Elliott, E. Cheng, C. Thomas-Ogbuji, and Z. M. Ozsoyoglu. A complete translation from SPARQL into efficient SQL. In *Proc. of the 2009 Int. Database Engineering & Applications Symposium, IDEAS '09*, pages 31–42. ACM, 2009.
6. G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring RDF triples to capture provenance. In *Proc. of the 8th Int. Semantic Web Conf., ISWC '09*, pages 196–212, Berlin, Heidelberg, 2009. Springer-Verlag.
7. F. Geerts and A. Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2):173–185, 2010.
8. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. of PODS'07*, pages 31–40, New York, NY, USA, 2007. ACM.
9. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013 available at <http://www.w3.org/TR/sparql11-query/>.
10. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.
11. A. Polleres. From SPARQL to rules (and back). In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *Proc. of the 16th Int. Conf. on World Wide Web, WWW 2007*, pages 787–796. ACM, 2007.
12. Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, 2011.

A Proof of the main result

This appendix provides a detailed proof of Theorem 1 by structural induction on the graph patterns.

The induction proof will construct an expression containing as attributes the graph attribute and an attribute for each in-scope variable of the graph pattern. Assume that the active graph is given and has id j (0 for the default graph, or $1 \leq j \leq n$ for the case of a named graph). We show that the cardinality of solutions of each SPARQL algebra operator is respected for the active graph, which is the most difficult part. Notice that the graphs by definition do not have duplicate triples, and therefore there will not be duplicates in the original base relations **Graphs** and **Quads**.

Empty graph pattern Recall that the empty graph pattern is translated into $[(\cdot)]_{\mathcal{R}}^G = \Pi_G [\rho_{G \leftarrow \text{gid}}(\mathbf{Graphs})]$. This means that the result of the relational algebra query returns as many as tuples as graphs in the RDF dataset, and at least a tuple with identifier 0 for the initial default graph. Moreover, note that for each particular graph id (default or named) this relation has exactly 1 tuple with that id.

Triple pattern Since there are no duplicate triples in the graphs, then the number of possible solutions of a triple pattern are exactly the number of triples that match the pattern, one for each instance. We analyse the correctness of the translation of the triple pattern according to the number of variables that may occur on it: no variables, one variable, two variables and three variables.

0 vars: In this case the triple pattern $t = (t_1, t_2, t_3)$ contains only RDF terms which might be identical, or not. In this case we obtain the relational algebra expression $\Pi_G [\rho_{G \leftarrow \text{gid}} (\sigma_{\text{sub}=t_1 \wedge \text{pred}=t_2 \wedge \text{obj}=t_3}(\mathbf{Quads}))]$. The resulting relation has just the attribute (column) G . If the triple occurs in graph i then the resulting relation will contain a tuple having value i in attribute G ; otherwise no tuple will occur for graph i .

1 var: We have several cases to consider here: there is just one occurrence of a variable, two or three.

Consider that the variable occurs in the subject of the triple; for the remaining cases in predicate or object the reasoning is similar. So, let $t = (?v, t_2, t_3)$, obtaining the relational algebra expression

$$\Pi_{G,v} [\rho_{G \leftarrow \text{gid}, v \leftarrow \text{sub}} (\sigma_{\text{pred}=t_2 \wedge \text{obj}=t_3}(\mathbf{Quads}))]$$

The resulting relation has two columns, one for the graph G and one for collecting the bindings for v (i.e. the solution). If the triple occurs in graph i then the resulting relation will contain a tuple having value i in attribute G ; otherwise no tuple will occur for graph i .

Consider now that the variable occurs in the subject and predicate of the triple; for the remaining cases the reasoning is similar. So, let $t = (?v, ?v, t_3)$, obtaining the relational algebra expression

$$\Pi_{G,v} [\rho_{G \leftarrow \text{gid}, v \leftarrow \text{sub}} (\sigma_{\text{sub}=\text{pred} \wedge \text{obj}=t_3}(\mathbf{Quads}))]$$

The resulting relation has two columns, one for the graph G and one for collecting the bindings for v (i.e. the solution). The selection condition guarantees that the obtained instances match the triple pattern, obtaining for each graph i one tuple for each possible value of v .

If there are three occurrences of the variable $?v$ then $t = (?v, ?v, ?v)$, then one obtains the expression

$$\Pi_{G,v} [\rho_{G \leftarrow \text{gid}, v \leftarrow \text{sub}} (\sigma_{\text{sub}=\text{pred} \wedge \text{sub}=\text{obj} \wedge \text{pred}=\text{obj}}(\text{Quads}))]$$

Note that one of the equalities in the selection condition is redundant. As before, for each graph, we get a tuple for each possible value of v .

2 vars: There are two variant of patterns here $t = (?v1, ?v2, t_3)$ and $t = (?v1, ?v2, ?v2)$ and permutations. For the first pattern we get

$$\Pi_{G,v1,v2} [\rho_{G \leftarrow \text{gid}, v1 \leftarrow \text{sub}, v2 \leftarrow \text{pred}} (\sigma_{\text{obj}=t_3}(\text{Quads}))]$$

while for the second we get

$$\Pi_{G,v1,v2} [\rho_{G \leftarrow \text{gid}, v1 \leftarrow \text{sub}, v2 \leftarrow \text{pred}} (\sigma_{\text{pred}=\text{obj}}(\text{Quads}))]$$

It is easy to see that each solution (for each graph) corresponds exactly to one tuple in the evaluation of the translated relation.

3 vars: This case is immediate and being the triple pattern $t = (?v1, ?v2, ?v3)$. The translation is $\Pi_{G,v1,v2,v3} [\rho_{G \leftarrow \text{gid}, v1 \leftarrow \text{sub}, v2 \leftarrow \text{pred}, v3 \leftarrow \text{obj}}(\text{Quads})]$, obtaining a relation which is isomorphic to **Quads**, as expected. For each graph i , we obtain a tuple with value i in attribute G and remaining attributes corresponding exactly to one triple in graph i .

UNION pattern The translated relational algebra expression $[(P_1 \text{ UNION } P_2)]_{\mathcal{R}}^G$ is:

$$\begin{aligned} & \Pi_{G, \text{var}(P_1) \cup \{v \leftarrow \text{unb} \mid v \in \text{var}(P_2) \setminus \text{var}(P_1)\}} \left([P_1]_{\mathcal{R}}^G \right) \\ & \quad \cup \\ & \Pi_{G, \text{var}(P_2) \cup \{v \leftarrow \text{unb} \mid v \in \text{var}(P_1) \setminus \text{var}(P_2)\}} \left([P_2]_{\mathcal{R}}^G \right) \end{aligned}$$

The relational algebra expressions makes the union of two projections. Each projection will not remove any in-scope variable, and it is used to extend the columns with unbound values in order to obtain a relation with columns $G \cup \text{var}(P_1) \cup \text{var}(P_2)$, by making unbound the variables that do not occur in the sub-pattern $[P_1]_{\mathcal{R}}^G$ or $[P_2]_{\mathcal{R}}^G$, respectively. Therefore, each subexpression the projection operator will not remove any in-scope attributes of each sub-pattern $[P_1]_{\mathcal{R}}^G$ or $[P_2]_{\mathcal{R}}^G$, and thus it returns exactly as many as tuples as the number of solutions of each sub-pattern by induction hypothesis. The cardinality of the resulting expression is the sum of solutions of each sub-expression, according to the bag semantics of relational algebra.

AND pattern The translated relational algebra expression $[(P_1 \text{ AND } P_2)]_{\mathcal{R}}^G$ is:

$$\Pi_{G, \begin{matrix} var(P_1) - var(P_2), \\ var(P_2) - var(P_1), \\ v_1 \leftarrow first(v'_1, v''_1), \dots, \\ v_n \leftarrow first(v'_n, v''_n) \end{matrix}} \left[\sigma_{comp} \left(\begin{matrix} \rho_{v'_1 \leftarrow v_1} \left([P_1]_{\mathcal{R}}^G \right) \bowtie \rho_{v''_1 \leftarrow v_1} \left([P_2]_{\mathcal{R}}^G \right) \\ \vdots \\ \rho_{v'_n \leftarrow v_n} \left([P_1]_{\mathcal{R}}^G \right) \bowtie \rho_{v''_n \leftarrow v_n} \left([P_2]_{\mathcal{R}}^G \right) \end{matrix} \right) \right]$$

where $comp$ is a conjunction of conditions $v'_i = \text{unb} \vee v''_i = \text{unb} \vee v'_i = v''_i$ for each common variable $v_i (1 \leq i \leq n)$. The function $first$ returns the first argument which is not unb , or unb if both arguments are unb .

The joined subexpressions inside the selection have only the common attribute G due to the renaming of common variables. So, every tuple of the subexpression for $[P_1]_{\mathcal{R}}^G$ will join with every tuple of $[P_2]_{\mathcal{R}}^G$, for each graph. If there is a solution with cardinality c_1 of P_1 and solution with cardinality c_2 of P_2 , one will obtain $c_1 \times c_2$ tuples in the result of the joined expression. From these possibilities, the selection expression keeps the combinations of solutions which are compatible: for each pair v'_i and v''_i the condition guarantees that at least 1 variable is unbound or have the same value. So, we only keep the tuples for each possible merge (with cardinality $c_1 \times c_2$ since selection keeps duplicate tuples). The projection is necessary to obtain the relation on the original variables, besides G , obtaining the value from the first bound variable, if any. It is also necessary to recall that the bag semantics for the projection operator will obtain as many tuples as the contributing tuples, summing over the obtained solutions as required by the cardinality condition of the **AND** operator (see the definition of the \cup for K -relations) .

FILTER pattern The relational algebra expression $[(P \text{ FILTER } R)]_{\mathcal{R}}^G$ is

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

where $filter$ is a condition obtained from R where each occurrence of **EXISTS**(P_i) (resp. **NOT EXISTS**(P_i)) in R 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:

$$\Pi_{G, var(P), ex_i \leftarrow 0} \left[\delta(P') - \Pi_{G, var(P)} \left(\sigma_{subst} \left(P' \bowtie \rho_{v'_1 \leftarrow v_1} (P'_i) \right) \right) \right) \cup \left[\delta(P') - \Pi_{G, var(P), ex_i \leftarrow 1} \left(\delta(P') - \Pi_{G, var(P)} \left(\sigma_{subst} \left(P' \bowtie \rho_{v'_1 \leftarrow v_1} (P'_i) \right) \right) \right) \right] \right]$$

where $P' = [P]_{\mathcal{R}}^G$, $P'_i = [P_i]_{\mathcal{R}}^G$, and $subst$ is the conjunction of conditions $v_i = v'_i \vee v_i = \mathbf{unb}$ for each variable v_i in $var(P) \cap var(P_i) = \{v_1, \dots, v_n\}$.

The translation is complex due to the EXISTS expressions. Note that each E_i expression returns exactly one tuple for each solution of pattern P . Thus, the duplicate removal operations are there just to guarantee this and do not affect the cardinality of the solutions, obtaining one solution for each solution of P that obeys to the filter condition. The rest of the translation is more or less immediate, where the condition $subst$ does not correspond exactly to the compatibility condition used before, since according to the SPARQL semantics the variables of pattern P_1 are substituted in the EXISTS pattern (we discard the cases where a variable is bound by P_1 and not bound in P_i).

MINUS pattern The relational algebra expression $[(P_1 \text{ MINUS } P_2)]_{\mathcal{R}}^G$ is

$$[P_1]_{\mathcal{R}}^G \bowtie \left[\delta \left([P_1]_{\mathcal{R}}^G \right) - \Pi_{G, var(P_1)} \left[\sigma_{comp \wedge \neg disj} \left([P_1]_{\mathcal{R}}^G \bowtie \rho_{v'_1 \leftarrow v_1} \left([P_2]_{\mathcal{R}}^G \right) \right) \right] \right]$$

$$\left[\begin{array}{c} \vdots \\ v'_n \leftarrow v_n \end{array} \right]$$

where $comp$ is a conjunction of conditions $v_i = \mathbf{unb} \vee v'_i = \mathbf{unb} \vee v_i = v'_i$ for each variable common $v_i (1 \leq i \leq n)$, and $disj$ is the conjunction of conditions $v_i = \mathbf{unb} \vee v'_i = \mathbf{unb}$ for each variable $v_i (1 \leq i \leq n)$.

The difference expression will return either a tuple of $[P_1]_{\mathcal{R}}^G$ or not, without duplicates. The difference expression evaluation will return solutions of P_1 that do not belong to the expression in the right-hand side of the difference. Since the right-hand side of expression obtains the tuples corresponding to the solutions of P_1 for which there is at least one solution in P_2 that is compatible and not disjoint (no bound variable in common), then we obtain as result the tuples corresponding to solutions of P_1 such that for all solution P_2 the solutions are incompatible or are disjoint (the semantics of MINUS). Now, the difference expression will have no duplicates, and thus we keep only the solutions of P_1 that join (i.e. that obey to the condition), without increasing the cardinality of the result.

OPTIONAL pattern The relational algebra expression $[(P_1 \text{ OPTIONAL } (P_2 \text{ FILTER } R))]_{\mathcal{R}}^G$ is

$$\begin{array}{c} [(P_1 \text{ AND } P_2)]_{\mathcal{R}}^G \\ \cup \\ \Pi_{G, var(P_1) \cup \{v \leftarrow \mathbf{unb} \mid v \in var(P_2) \setminus var(P_1)\}} \end{array}$$

$$\left[[P_1]_{\mathcal{R}}^G \bowtie \left(\begin{array}{c} \delta \left([P_1]_{\mathcal{R}}^G \right) \\ - \\ \Pi_{G, var(P_1)} \left([(P_1 \text{ AND } P_2) \text{ FILTER } R]_{\mathcal{R}}^G \right) \end{array} \right) \right]$$

According to the semantics of SPARQL 1.1, the **OPTIONAL** pattern evaluation is performed by two operators: a join and a left join. This is particularly clear in the translation, where the join is the first expression and the left join the lower (big) expression below the union operator. The rationale of the translation of the left join operator is identical to the translation of the **MINUS** pattern, except now that we need to make unbound the variables in P_2 but not in P_1 . Again, the number of obtained tuples is according to the semantics of SPARQL 1.1: it is the sum of the tuples of the join with the sum of tuples of the left join (guaranteed by the bag semantics of \cup).

GRAPH pattern The translation of $(\text{GRAPH } term \ P_1)$ has two cases:

- If $term$ is an IRI then $[(\text{GRAPH } term \ P_1)]_{\mathcal{R}}^G$ is

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

The selection expression obtains a single tuple containing the identifier corresponding to the $term$, or obtains the empty relation if there is no named graph with that IRI. This tuple joins with $[P_1]_{\mathcal{R}}^{G'}$ to limit the results to the intended graph. Moreover it is important that the use of a renamed attribute for the graph is necessary in order to avoid clashes of attributes corresponding to the active graph. Moreover, the graph pattern will return the same results independently of the active graph and this is captured by the join with the empty graph pattern.

- If $term$ is a variable v then $[(\text{GRAPH } term \ P_1)]_{\mathcal{R}}^G$ is

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

This case is a little more complex, because now we consider all the named graphs ($\text{gid} > 0$), and bind the variable v with the corresponding IRIs. The rationale of the construction is the same of the previous case.

To conclude the proof, we just need to analyse the results of the expression corresponding to the full query $[\text{SPARQL}(P, D(G), V)]_{\mathcal{R}} = \Pi_V \left[\sigma_{G'=0} \left([()]_{\mathcal{R}}^{G'} \bowtie [P]_{\mathcal{R}}^{G'} \right) \right]$ with respect to the base relations **Graphs** and **Quads**, where G' is a new attribute name and $V \subseteq var(P)$. The selection starts the evaluation at the default graph ($G' = 0$) and projects in the selected variables. The correctness of the translation is now immediate due to the previous induction.