# On Provenance of Queries on Semantic Web Data

Yannis Theoharis, Irini Fundulaki, Grigoris Karvounarakis, and Vassilis Christophides

*Abstract*— **Assessing the quality of data currently published on the Semantic Web emerges as a crucial need of various applications. Capturing trustworthiness, reputation and reliability of Semantic Web data manipulated by SPARQL, requires to represent adequate provenance information usually modeled as annotations on source data and propagated to query results along with query evaluation. Alternatively, one can use abstract provenance models to capture the relationship between query results and the source data by taking into account the employed query operators. We argue the benefits of the latter for settings in which the query results are materialized in several repositories and analyzed by multiple users. We investigate the extent to which relational provenance models can be leveraged for SPARQL queries and identify their limitations. Finally, we advocate the need for new provenance models capturing the full expressive power of SPARQL.**

**H.2.1 [Database Management]: Data Models**

## I. Introduction

Recently, the W3C Linking Open Data initiative[1] has boosted the publication and interlinkage of massive amounts of data sets on the Semantic Web as RDF data, queried with the SPARQL query language[2]. Together with Web 2.0 technologies (e.g. mashups) they have essentially transformed the Web from a publishing-only environment into a vibrant place for information dissemination where data is *exchanged*, *integrated*, and *materialized* in distributed repositories behind SPARQL endpoints. In this open environment where Semantic Web data is represented by incomplete or replicated sets of RDF triples, it is crucial to be able to assert the *trustworthiness*, *reputation* and *reliability* of the published information. This functionality essentially calls for representing and reasoning on the *provenance* of Semantic Web data manipulated by SPARQL queries. For instance, in the case of *trust assessment* [1] (one of the key applications recognized by the W3C Provenance Incubator Group[3]), the trustworthiness of query results is determined based on the trustworthiness of data sources from which they were derived. For simple *boolean trust* assessment we only need to determine which output data should be trusted. For *ranked trust* assessment we need to choose the most trusted among competing evidence from diverse sources. Additionally, for *uncertain* and *fuzzy* data, the probabilities of query results are derived based on the probabilities associated with the original data [2].

In all these cases, the goal is to compute appropriate *annotations* for query results that reflect data quality, based on the annotations of source data. If source annotations were static and common for all users, this computation could be done together with the query evaluation (as for annotated databases [3], [4], [5]). However, in general, different users have different beliefs about e.g., the trustworthiness of source data, and these beliefs may change over time, even when the relationship of query results with source data is unchanged. For this reason, an alternative approach is to use *abstract provenance models* to capture this relationship along with the query operators that combined source data to derive query results. This information can be recorded [6] in the repository when the data is imported, and used to compute appropriate annotations for different applications and users at a later time [7].

In this paper, we focus on *data provenance* in the style of [8], i.e. provenance of data in the result of *declarative queries*. This is different from *workflow* provenance [9] (e.g. OPM[4]), which typically describes procedural data processing, and where operations are usually treated as black boxes [10] due to their complexity. As a result, workflow provenance is in general less *fine-grained* than data provenance. Moreover, we are interested in *implicit* provenance [11] of queries that only manipulate data and are oblivious about the possible annotations thereof. Implicit provenance captures the abstract structure and properties of query operators and can, thus, be used for various annotation computations [7], [12]. This is in contrast with work on *explicit* provenance [11], where queries can also manipulate source annotations and specify explicitly the annotation of the results. Consequently, the resulting annotations can be arbitrary and may not reflect the structure and properties of query operators, as needed to support alternative annotation computations.

It is worth noting that previous work on modeling provenance in the *Semantic Web* mainly focuses on representing and querying workflow provenance information for e-science using the RDF [13] data model. Earlier work on RDF data provenance includes named graphs [14], which have been proposed as a means to define *ownership* of RDF triples by objectifying (through URIs) the RDF graphs to which they belong. They are an annotation mechanism where graph identifiers are explicitly stored and queried along with the original triples. Finally [15] has studied implicit data provenance for a SPARQL fragment that is closer to the positive relational algebra. In this vision paper, we take a first step towards designing an abstract provenance model for SPARQL. In particular, we:

- identify the basic characteristics of abstract provenance models and argue on the benefits of using those to compute annotations for various applications on Semantic Web data (Section II).
- review relational abstract provenance models, that can be used to capture the provenance of relational queries over Semantic Web data (Section III).
- explore the extent to which these models can be leveraged for SPARQL queries over Semantic Web data, and identify

Y. Theoharis, I. Fundulaki, G. Karvounarakis and V. Christophides are with the Institute of Computer Science, FORTH. Y. Theoharis and V. Christophides are also with the Department of Computer Science, University of Crete, Greece. G. Karvounarakis is also with LogicBox, USA (Email: {theohari, fundul, gregkar, christop}@ics.forth.gr).

[1] linkeddata.org

[2] SPARQL: www.w3.org/TR/rdf-sparql-query

[3] http://www.w3.org/2005/-Incubator/-prov/wiki/W3C_Provenance_Incubator-_Group_Wiki

[4] http://twiki.ipaw.info/bin/view/Challenge/OPM

their limitations (Section IV). The main challenges stem from the *SPARQL* OPTIONAL operator, which is crucial for dealing with the incompleteness of Semantic Web data but – as we explain later – cannot be handled by relational provenance models. For this reason, we advocate the need for new provenance models for SPARQL queries.

## II. Abstract Provenance Models

In this section we study the basic characteristics of abstract provenance models to support a range of annotation computations required by different applications and users. We pay particular attention to the benefits of recording abstract provenance information when data is materialized in a repository through queries.

In different application settings there is a need to identify and refer to source data involved in the derivation of query results. To this end, the most common approach in the relational world is to *annotate* source data with appropriate, unique, abstract labels called *provenance tokens* [16]. The granularity of the annotated data items typically depends on the main constructs of the data model, e.g. *sets of attributes* [17], [18], *tuples* [12], [16], [17], [19], [20] or *relations* [17] for the relational data model.

Then, we can abstractly describe the provenance of output data in a query result as a set of provenance tokens of source data. For Semantic Web data, this can be achieved by defining a named graph per triple. However, for applications such as trust assessment, simply knowing the provenance tokens of source data may not be sufficient. Consider for instance, queries combining data from different sources, some of which are trusted. Multiple sources may be involved in alternative derivations of a data item in the query result. Thus, to make trust judgments, we need more detailed *provenance expressions*, that in addition to provenance tokens also record the *query operators* involved in the derivation of a data item, thereby storing information on *how* input data items were combined to produce the item in question.

Once abstract provenance expressions have been computed and materialized along with the query results, one can *evaluate* them to compute the annotations for a particular application. This amounts to the substitution of the provenance tokens and abstract operations with a concrete set of values and operations on them, respectively. The former reflects user beliefs about how source data should be annotated, while the latter reflects the particular application needs.

An alternative approach is to annotate source data with appropriate values and compute annotations for query results during query evaluation. This approach was followed in previous work on query answering for annotated databases, for several kinds of annotations ranging from probabilistic event expressions [3] to boolean expressions dealing with incompleteness or uncertainty [4], or to tuple multiplicities [5].

In the context of Semantic Web data, abstract provenance models are highly beneficial compared to annotated databases, because data is materialized in repositories from various sources and there is a need to assess its quality afterwards. More precisely:

- an application may require the evaluation of multiple new or existing dimensions of data quality;
- for a particular application, different users may have different perceptions that may change over the time about the appropriate source data annotations;
- users typically want to compute annotations for a (possibly small) subset of the items in the repository;
- source data imported into the repository may be unavailable when a user tries to assess some dimension of data quality.

Ideally, one would like to have an abstract provenance model that can support all applications of interest. However, there is often a tradeoff between the expressiveness of provenance models and the cost for storing and manipulating the corresponding provenance expressions. As a result, for systems that only need to support a subset of these applications it may be desirable to rely on less-informative abstract provenance models if they can provide improved performance.

## III. Provenance models for relational queries over Semantic Web Data

As a first step towards capturing the provenance of Semantic Web data, we consider the case in which we query them using positive relational algebra (denoted by $\text{RA}^+$). Indeed, since RDF is the basic data model for representing Semantic Web data as *triples* of the form (*subject*, *predicate*, *object*), they can be stored in a relational table with three columns. Therefore, it is possible to query them using $\text{RA}^+$ queries. Then, we can take advantage of previous work on relational provenance models to capture the provenance of query results.

Table (a) of Figure I shows an RDF triple set, denoted by $T$, in relational form. $S$, $P$ and $O$ stand for the *subject*, *predicate* and *object* of a triple. The fourth column represents the triple's provenance token.

Consider the query $Q(T) = \pi_{SO}(\pi_{SP}(T) \bowtie \pi_{PO}(T) \cup \pi_{SO}(T) \bowtie \pi_{PO}(T))$. The first column of Table (b) of Figure I shows the result of $Q(T)$. To illustrate the main characteristics and differences of the various relational provenance models, we will focus on the provenance of the last tuple $(f, e)$ in the result of $Q(T)$. This tuple has three derivations. One is obtained as a projection of tuple $(f, g, e)$ from subquery $\pi_{SP}(T) \bowtie \pi_{PO}(T)$ and the other two as projections on the results of subquery $\pi_{SO}(T) \bowtie \pi_{PO}(T)$.

The *lineage* [19] of a tuple in the result of a query is the set of (provenance tokens of) source tuples that were involved in some derivation of that result tuple. The first and second derivations of $(f, e)$ only use tuple $(f, g, e)$, annotated with the provenance token $c_3$. The third derivation uses both $(f, g, e)$ and $(d, b, e)$, the latter annotated with $c_2$. Consequently, we obtain the provenance expression $\{c_2, c_3\}$ (see Table (b) of Figure I).

Other relational provenance models also encode some information about the operators that were used in each derivation. For instance, *why*-provenance [17] encodes all the different derivations of a tuple in the query result by storing a set of provenance tokens *for each derivation*.In our example, the first and second derivations of $(f, e)$ only involve $c_3$, so they are both represented by the same set ($\{c_3\}$), whereas the last one involves both $c_3$ and $c_2$. Therefore, the *why*-provenance of $(f, e)$ is $\{\{c_3\}, \{c_2, c_3\}\}$ (see Table (b) of Figure I). Intuitively, each inner set represents one or more derivations that involve the same source data, while multiple tokens in an inner set, e.g. $\{c_2, c_3\}$, indicate a join between the corresponding tuples.

*Perm* [20] employs the tuples, instead of provenance tokens, to encode provenance of source data. To illustrate how Perm works, we consider the provenance expression of the last tuple in the result ($\mathbf{S}$:$f, \mathbf{O}$:$e$), where $\mathbf{S}$, $\mathbf{O}$ represent attribute names of $T$. Perm retains two tuples for ($\mathbf{S}$:$f, \mathbf{O}$:$e$), one for every derivation. In particular, the first tuple is

| Triple Set $T$ | | | |
|---|---|---|---|
| **S** | **P** | **O** | **Prov.** |
| $a$ | $b$ | $c$ | $c_1$ |
| $d$ | $b$ | $e$ | $c_2$ |
| $f$ | $g$ | $e$ | $c_3$ |

$(a)$

| $Q(T) = \pi_{SO}(\pi_{SP}(T) \bowtie \pi_{PO}(T) \cup \pi_{SO}(T) \bowtie \pi_{PO}(T))$ | | | | |
|---|---|---|---|---|
| **S O** | **Lineage** | **Why** | **Trio-lineage** | **How** |
| $a$ $c$ | $\{c_1\}$ | $\{\{c_1\}\}$ | $\{\{c_1\},\{c_1\}\}$ | $(c_1 \odot c_1) \oplus (c_1 \odot c_1)$ |
| $a$ $e$ | $\{c_1, c_2\}$ | $\{\{c_1, c_2\}\}$ | $\{\{c_1, c_2\}\}$ | $c_1 \odot c_2$ |
| $d$ $c$ | $\{c_1, c_2\}$ | $\{\{c_1, c_2\}\}$ | $\{\{c_1, c_2\}\}$ | $c_1 \odot c_2$ |
| $d$ $e$ | $\{c_2, c_3\}$ | $\{\{c_2\},\{c_2, c_3\}\}$ | $\{\{c_2\},\{c_2\},\{c_2, c_3\}\}$ | $(c_2 \odot c_2) \oplus (c_2 \odot c_2) \oplus (c_2 \odot c_3)$ |
| $f$ $e$ | $\{c_2, c_3\}$ | $\{\{c_3\},\{c_2, c_3\}\}$ | $\{\{c_3\},\{c_3\},\{c_2, c_3\}\}$ | $(c_3 \odot c_3) \oplus (c_3 \odot c_3) \oplus (c_2 \odot c_3)$ |

$(b)$

FIGURE I

EXAMPLE OF *lineage*, *why*-PROVENANCE, *Trio-lineage* AND *how*-PROVENANCE $(b)$ FOR THE QUERY $Q$ OVER $T$ $(a)$

$(\mathbf{S}{:}f, \mathbf{O}{:}e, \mathbf{S_1}{:}f, \mathbf{P_1}{:}g, \mathbf{O_1}{:}e, \mathbf{S_2}{:}f, \mathbf{P_2}{:}g, \mathbf{O_2}{:}e)$, where the first two attributes represent the result tuple $(f, e)$, while the two occurrences of $(f, g, e)$ encode the fact that it has been used twice to derive the tuple $(f, e)$. On the other hand, tuple $(\mathbf{S}{:}f, \mathbf{O}{:}e, \mathbf{S_1}{:}f, \mathbf{P_1}{:}g, \mathbf{O_1}{:}e, \mathbf{S_2}{:}d, \mathbf{P_2}{:}b, \mathbf{O_2}{:}e)$, encodes that $(f, g, e)$ and $(d, b, e)$ were used to derive $(f, e)$. In this manner, the provenance information that *Perm* encodes is similar to *why*-provenance.

*Trio*-lineage [12] is similar to *why*-provenance, but additionally records separately even derivations involving the same set of source tuples. A *Trio*-lineage expression is a *bag* of sets of tokens, each of which corresponds to one derivation. Hence, for the first two derivations of $(f, e)$ (see Table (b) of Figure I) we obtain $\{\{c_3\}, \{c_3\}\}$ whereas for the last we have $\{\{c_2, c_3\}\}$.

Finally, *how*-provenance [16] encodes not only the union and join operators, but also the number of times a tuple participates in a join. To this end, it employs the abstract binary operator $\oplus$ to encode union and projection and $\odot$ to encode join. In our example (see Table (b) of Figure I), tuple $(f, g, e)$, participates twice in the first two derivations of $(f, e)$ and, thus, each one has provenance $c_3 \odot c_3$. The remaining derivation results from a join between $(f, g, e)$ and $(d, b, e)$, annotated with $c_3$ and $c_2$ respectively, resulting in the provenance expression $c_2 \odot c_3$. Thus, the provenance of $(f, e)$ is $(c_3 \odot c_3) \oplus (c_3 \odot c_3) \oplus (c_2 \odot c_3)$. Compared to *lineage*, *why*-provenance and *Trio*-lineage, how-provenance is the most informative [21] provenance model. More precisely, as shown in [16], it is *universal* for all provenance models (such as the aforementioned ones) that can be expressed as semirings.

### A. Expressiveness of Provenance Models

As we explained above, some provenance models capture more information than others, at the expense of producing more complex provenance expressions. For some applications the additional information is necessary while for others it is not. In this section, and the rest of this paper, we focus on the applications of boolean and ranked trust assessment, to illustrate such differences in expressiveness requirements, but we are generally interested in abstract provenance models that can also be used for a wide range of applications [7], [16].

*1) Boolean Trust Assessment:* In this case, given a query, the goal is to find which result tuples are trusted, based on the trustworthiness of the input tuples. More specifically, a derivation is trusted only if all contributing tuples are trusted. For tuples with multiple derivations, they are trusted if at least one of the derivations is trusted. Based on this semantics, which is also followed in the relational context [6], [16], one can compute the trusted result tuples by answering the query on the subsets of the input relations containing only the trusted tuples.

Trusted result tuples can be computed through provenance, by assigning the values *true* (resp. *false*) to provenance tokens of trusted (resp. untrusted) tuples. Consider for instance the *why*-provenance of $(f, e)$ in the output, i.e. $\{\{c_3\}, \{c_2, c_3\}\}$ and let $c_1 = c_3 = true$, $c_2 = false$. Then, $(f, e)$ is trusted, because there exists a derivation (namely $\{c_3\}$), for which all tokens represent trusted source tuples (i.e., have the value *true*).

*2) Ranked Trust Assessment:* In *ranked trust* assessment [7], every source tuple is associated with a *rank*, i.e., a natural number that denotes how trusted it is. In particular, 0 is the rank of the most trusted tuples, while $\infty$ indicates tuples that are completely untrusted.

If a tuple has multiple derivations, as a result a union or projection operator in the query, the rank of the output tuple is the minimum rank among all derivations, i.e. that of the most trusted derivation. In the case of a join, the rank of the resulting tuple is the sum of the ranks of the input tuples. In this respect, it has a higher rank, i.e. is less trusted, than both of them.

For instance, let $c_1 = 1$, $c_2 = 2$, $c_3 = 3$. Then, if we consider the most detailed provenance expression derived by *how*-provenance, the rank of $(f, e)$ in the output is computed as $min(min(c_3 + c_3, c_3 + c_3), c_2 + c_3) = min(min(3+3, 3+3), 2+3) = min(6, 5) = 5$. Had we considered a less expressive model, we would have computed an incorrect rank for $(f, e)$. Consider for instance *Trio*-lineage. The operator "+" applies on annotations included in an inner set and the results (one sum per inner set) are then combined with $min$, i.e. the evaluation of the *Trio*-lineage expression for $(f, e)$ would produce $min(min(c_3, c_3), c_2 + c_3) = min(min(3, 3), 2 + 3) = min(3, 5) = 3$. We conclude that *Trio*, as well as the less expressive *why*-provenance and *lineage*, fail to compute the correct rank. Therefore, comparing ranked with boolean trust assessment, we observe that the former requires a more expressive provenance model than the latter.

## IV. CAPTURING THE PROVENANCE OF SPARQL QUERIES

In the previous section we explained how relational provenance models can be used to capture the provenance of relational queries over Semantic Web data. However, since Semantic Web data are by default represented in RDF, in this section we focus on capturing the provenance of SPARQL queries typically employed to manipulate them.

### A. SPARQL in a Nutshell

We base our presentation of SPARQL on the algebra presented in [22]. This algebra is based on triple patterns, i.e. triples of the form $(x, y, z)$, where $x$, $y$, $z$ can be constants or variables, the latter prefixed with "?". Triple patterns are used to bind variables to values in the dataset. A set of pairs $(variable, value)$, i.e. the

$\Omega$ = evaluation of $(?x, ?y, ?z)$ over $T$

| ?x | ?y | ?z |
|---|---|---|
| $a$ | $b$ | $c$ |
| $d$ | $b$ | $e$ |
| $f$ | $g$ | $e$ |

$(a)$

$\Omega_1 = \pi_{?x,?y}(\sigma_{?z=e}(\Omega))$

| | ?x | ?y |
|---|---|---|
| $\mu_1$ : | $d$ | $b$ |
| $\mu_2$ : | $f$ | $g$ |

$(b)$

$\Omega_2 = \pi_{?x,?z}(\sigma_{?y=b}(\Omega))$

| | ?x | ?z |
|---|---|---|
| $\mu_3$ : | $a$ | $c$ |
| $\mu_4$ : | $d$ | $e$ |

$(c)$

$\Omega_3 = \pi_{?y}(\sigma_{?z=e}(\Omega))$

| | ?y |
|---|---|
| $\mu_9$ : | $b$ |
| $\mu_{10}$ : | $g$ |

$(d)$

$\Omega_4 = \pi_{?y,?z}(\sigma_{?x=a}(\Omega))$

| | ?y | ?z |
|---|---|---|
| $\mu_{17}$ : | $b$ | $c$ |

$(e)$

$\Omega_1 \cup \Omega_2$

| | ?x | ?y | ?z |
|---|---|---|---|
| $\mu_5$ : | $d$ | $b$ | – |
| $\mu_6$ : | $f$ | $g$ | – |
| $\mu_7$ : | $a$ | – | $c$ |
| $\mu_8$ : | $d$ | – | $e$ |

$(f)$

$(\Omega_1 \cup \Omega_2) \bowtie \Omega_3$

| | ?x | ?y | ?z |
|---|---|---|---|
| $\mu_{11}$ : | $d$ | $b$ | – |
| $\mu_{12}$ : | $f$ | $g$ | – |
| $\mu_{13}$ : | $a$ | $b$ | $c$ |
| $\mu_{14}$ : | $a$ | $g$ | $c$ |
| $\mu_{15}$ : | $d$ | $b$ | $e$ |
| $\mu_{16}$ : | $d$ | $g$ | $e$ |

$(g)$

$\Omega_1 \text{⟕} \Omega_4$

| | ?x | ?y | ?z |
|---|---|---|---|
| $\mu_{19}$ : | $d$ | $b$ | $c$ |
| $\mu_{20}$ : | $f$ | $g$ | – |

$(h)$

FIGURE II
EXAMPLE OF SPARQL ALGEBRA OPERATORS

SPARQL analog of the relational *valuation*, is called a *mapping*. For instance, the pattern $(?x, ?y, c)$ only matches triples whose *object* has the value $c$, and the result of matching it to the first triple of $T$ (Table (a) in Figure I), is the mapping $\{(?x, a), (?y, b)\}$ indicating that variables $?x$, $?y$ are bound to values $a$ and $b$, respectively. The evaluation of a triple pattern on a set of triples is a bag of *mappings*, i.e. a set of *mappings* along with a cardinality function, that associates every mapping of the set with an integer. To simplify the presentation, we will use the tabular representation of the mapping bags shown in Figure II, where each column corresponds to a variable in the mappings.

The SPARQL algebra in [22] defines: a) the unary operators $\sigma$ (filtering) and $\pi$ (projection) that correspond to the SPARQL constructs FILTER and SELECT, respectively and b) the binary operators, $\cup$, $\bowtie$, $\text{⟕}$ for the SPARQL constructs UNION, AND, and OPTIONAL respectively.

Filtering on the triple components is expressed by fixing one of them to a constant. For instance, let $\Omega$ (Table (a) of Figure II) denote the evaluation of $(?x, ?y, ?z)$ over $T$. Then $\sigma_{?x=a}(\Omega)$ contains only mapping $\{(?x, a), (?y, b), (?z, c)\}$.

Projection specifies the subset of variables to be returned in the query result. For example, $\Omega_1 = \pi_{?x,?y}(\sigma_{?z=e}(\Omega))$ is the bag of mappings obtained from projecting the variables $?x$, $?y$ of $\sigma_{?z=e}(\Omega)$ (Table (b) of Figure II). Similarly, $\Omega_2$ in Table (c) of Figure II denotes the result of query $\pi_{?x,?z}(\sigma_{?y=b}(\Omega))$. To simplify the presentation, we employ symbols $\mu_i$ in Figure II to identify individual mappings.

Unlike relational union that is defined on relations with the same attributes, the union ($\cup$) operation of the SPARQL algebra can be applied on bags of mappings containing different variables. In such cases, the result may include mappings with unbound variables, denoted by "-" in Table (f) of Figure II (in SQL that would be a *null* value).

In order to define the semantics of the join ($\bowtie$) operator, [22] introduces the notion of *compatible mappings*. Two mappings are *compatible* if they agree on their *common* variables. The output of $\bowtie$ for two compatible input mappings is a mapping whose set of variables is the union of their bound variables. For each variable in the output, its value is the same as in the corresponding input mapping(s). Unlike in relational algebra, where a null value in an attribute makes any join condition fail, unbound variables in SPARQL do not affect the compatibility of mappings. Figure II shows the result of $(\Omega_1 \cup \Omega_2) \bowtie \Omega_3$ in Table (g), where $\Omega_1 \cup \Omega_2$ is shown in Table (f), while $\Omega_3$ in Table (d). Note that, although $?y$ is unbound e.g., in $\mu_7$, SPARQL considers $\mu_7$ to be compatible with $\mu_9$ and $\mu_{10}$, for which $?y$ is bound.

Finally, the application of the operator $\text{⟕}$between mapping bags $\Omega_l$ and $\Omega_r$ returns the mappings contained in the result of $\Omega_l \bowtie \Omega_r$, as well as all mappings from $\Omega_l$ that are not compatible with any mapping in $\Omega_r$. In this manner, $\text{⟕}$ is similar to the left outer join operator of the relational algebra. Figure II shows the result of $\Omega_1 \text{⟕} \Omega_4$ in Table (h), where $\Omega_1$ is shown in Table (b) and $\Omega_4$ in Table (e). For instance, $\mu_{19}$ is in the result because of the join between $\mu_1$ and $\mu_{17}$, while $\mu_{20}$ appears in the result because $\mu_2$ belongs to $\Omega_1$ and is not compatible with $\mu_{17}$. Following [22], we denote with $\Omega_l \setminus \Omega_r$ the mappings of $\Omega_l$ that are not compatible with any $\Omega_r$ mapping, e.g. $\Omega_1 \setminus \Omega_4 = \{\mu_2\}$. As shown in [22], the following equivalence holds:

$$\Omega_l \text{⟕} \Omega_r = (\Omega_l \bowtie \Omega_r) \cup (\Omega_l \setminus \Omega_r) \quad (1)$$

We should stress that there are some subtle differences between the "$\setminus$" operator of [22] and the relational minus operator (denoted "–" below). The former checks mappings for compatibility, while the latter only compares tuples for equality. It should be mentioned that compatibility between mappings is an $1 - n$ relationship, i.e. a mapping of $\Omega_l$ may be compatible with many mappings of $\Omega_r$. On the contrary, equality between tuples is an $1 - 1$ relationship. Consider for instance, the relational query $R_l - R_r$. A tuple of $R_l$ relation can be equal to at most one tuple of $R_r$. As a consequence, the existence of multiple copies of a mapping in $\Omega_l$ and $\Omega_r$ does not affect the cardinality of that mapping in the result: if a mapping $\mu$ has cardinality $m$ in $\Omega_l$ and there is one compatible mapping with cardinality $n$ in $\Omega_r$, $\mu$ will have cardinality 0 in the result, i.e., it will not appear in it. On the contrary, in the relational context, if a tuple $t$ has cardinality $m$ in relation $R_l$ and $n$ in $R_r$, then the cardinality of $t$ in $R_l - R_r$, is $m - n$, if $m > n$, and 0, otherwise.

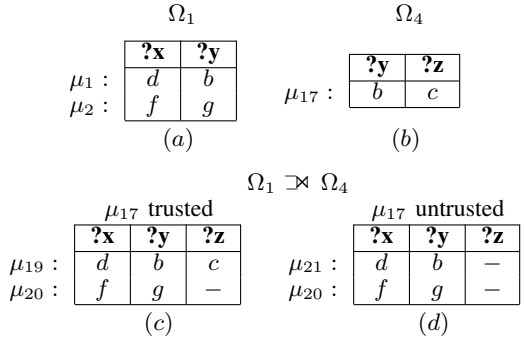


FIGURE III
EXAMPLE OF BOOLEAN TRUST

### B. Provenance Models for Positive SPARQL

From the previous presentation, there is a clear analogy of the SPARQL algebra operators of projection ($\pi$), filter ($\sigma$), join ($\bowtie$) and union ($\cup$) with the corresponding operators of positive relational algebra (RA$^+$). For this reason, we refer to the fragment of SPARQL consisting only the above operators as *positive* SPARQL (denoted by SPARQL$^+$) and investigate whether provenance models for RA$^+$ queries presented in Section III, can be also applied to SPARQL$^+$ queries, despite their subtle differences.

One difference lies in the fact that relational algebra operates on tuples, while SPARQL algebra operates on mappings. However, this is easily handled by associating mappings that are returned by triple patterns with the provenance tokens of the triples they matched. Moreover, SPARQL algebra adopts bag semantics by default, although set semantics can be enforced through the use of the operator DISTINCT. Among the provenance models for relational queries, only *how*-provenance can be used to compute correct result multiplicities under bag semantics [16], while all models can handle set semantics. Finally, the differences (mentioned in Section IV-A) between SPARQL and relational algebra for the $\cup$ and $\bowtie$ operators do not affect the provenance of output mappings. As a consequence, all abstract provenance models for RA$^+$ presented in Section III, can be applied to SPARQL$^+$ under set semantics, while *how*-provenance can be used when bag semantics is needed.

### C. Towards Provenance for SPARQL

However, relational provenance models are not sufficient to capture provenance for the SPARQL algebra, essentially due to the use of the $\overline{\bowtie}$ operator. This is because $\overline{\bowtie}$ involves a form of negation (see the use of $\setminus$ in expression (1)), while most of the aforementioned models capture the provenance of positive queries. We illustrate the challenges posed by $\overline{\bowtie}$ through an example of boolean trust assessment (Section III-A).

To compute the set of trusted mappings in the result of a SPARQL query, we can evaluate the SPARQL query on the subsets of input mapping sets that include only the trusted mappings. Hence, trusted mappings of $\Omega_1$ of Figure III that are not compatible with any trusted mapping of $\Omega_4$ should appear in the query output as trusted. This semantics also coincides with *tSPARQL*[5], if we apply the EnsureTrust operator to filter out untrusted mappings from input mapping sets (by setting the lower ($l$) and upper ($u$) bounds to $true$).

[5]http://trdf.sourceforge.net/ documents/tsparql.pdf

Suppose, for example, that mappings $\mu_1$, $\mu_2$ of $\Omega_1$ and $\mu_{17}$ of $\Omega_4$ are trusted (see Tables (a) and (b) of Figure III). The trusted mappings of $\Omega_1 \overline{\bowtie} \Omega_4$ are depicted in Table (c). One can observe that $\mu_{19}$ belongs to the result as a derivation of two compatible and trusted mappings, $\mu_1$ and $\mu_{17}$, while $\mu_{20}$ is trusted because $\mu_2$ is trusted in $\Omega_1$ and is not compatible with any trusted mapping of $\Omega_4$.

On the other hand, if $\mu_1$, $\mu_2$ were trusted but $\mu_{17}$ was untrusted, $\mu_1$ would not be compatible with any trusted mapping of $\Omega_4$. Thus, mapping $\mu_{21}$ should appear in the result as trusted (see Table (d) of Figure III). One can easily observe that, although $\mu_{21}$ (resp. $\mu_{19}$) does not belong to the query result illustrated in Table III (c) (resp. (d)), an abstract provenance model that can be used for such trust computations would need to associate for both those mappings with appropriate provenance expressions.

Existing provenance models for *RA*$^+$ queries (see Section III) do not support the semantics of SPARQL $\overline{\bowtie}$ or $\setminus$. Even *Perm*, which captures negation, does not record sufficient information for enabling annotation computations such as the boolean trust assessment (see Figure III). More precisely, *Perm* records the reason why $\mu_{20}$ exists in the result, i.e. that $\mu_2$ is not compatible with $\mu_{17}$, by keeping in the output the mapping {*(?x, f), (?y, g), (?y*$_{17}$*, b), (?z*$_{17}$*, c)*}. However, it does not encode any provenance expression for $\mu_{21}$. Thus, when $\mu_{17}$ is untrusted, it has no way to infer that $\mu_{21}$ should appear in the result as trusted.

Similarly to *Perm*, [15] documents that $\mu_{20}$ exists in the result because $\mu_2$ is not compatible with $\mu_{17}$. However, it does not encode provenance information for $\mu_{21}$, and therefore it can not infer that $\mu_{21}$ should be in the result as trusted, if $\mu_{17}$ is untrusted.

*M-semirings* [23] is a recent extension of *how*-provenance for capturing the relational minus operator. To this end, it defines an additional abstract operator, denoted by $\ominus$. To compute provenance expressions for our running example, the m-semiring model would employ equation (1) for $\Omega_1 \overline{\bowtie} \Omega_4$. The provenance expressions for mappings in $\Omega_1 \bowtie \Omega_4$ are computed in the same manner as in the case of *how*-provenance, e.g. the provenance of $\mu_{19}$ is $c_1 \odot c_3$, where $c_1$ (resp. $c_3$) is the provenance of $\mu_1$ (resp. $\mu_{17}$) in $\Omega_1$ (resp. $\Omega_4$). The $\ominus$ operator is employed to compute the provenance of mappings in $\Omega_1 \setminus \Omega_4$. In particular, the provenance of $\mu_{20}$ is $c_2 \ominus 0$, where $c_2$ is the provenance of $\mu_2$ in $\Omega_1$, while 0 denotes that $\mu_2$ does not belong to $\Omega_4$. According to the formal properties of $\ominus$, $c_2 \ominus 0 = c_2$. Moreover, the provenance of $\mu_{21}$ is $c_1 \ominus c_3$. Consequently, in the case that $\mu_{17}$ is untrusted, m-semirings infer that $\mu_{21}$ should appear in the result as trusted. However, m-semirings follow the semantics of relational minus, which differs from the semantics of $\setminus$ in SPARQL algebra. Consider for instance, that $\Omega_4$ had an additional mapping $\mu_{22} = \{(?y, b), (?z, e)\}$, that is compatible with $\mu_1$. Then $\mu_{21}$ would appear in the result as trusted (as shown in Table (d) of Figure III), only if *both* $\mu_{17}$ and $\mu_{22}$ were untrusted. However, the m-semiring expression for $\mu_{21}$ could only encode a single mapping of $\Omega_4$.

We conclude that a new provenance model is needed in order to cope with the $\overline{\bowtie}$ operator. In this model, provenance expressions should be recorded for some mappings that do not appear in the result of a query involving the $\overline{\bowtie}$ operator, e.g. for $\mu_{21}$ in $\Omega_1 \overline{\bowtie} \Omega_4$ in Table (c) of Figure III. It is worth mentioning that this need also appears in the case that provenance expressions should be computed for the relational left (or right) outer join.

Moreover, this model cannot be based on techniques used in

relational provenance models to deal with relational minus, due to the differences (see also section IV-A) between the SPARQL algebra \ operator and the relational minus. In particular, the provenance expression of a mapping should encode some information about *all* the compatible mappings of the right-hand mapping set, instead of encoding information of a single tuple in the right-hand relation. Finally, the provenance expression for the \ operator should conform to SPARQL semantics for cardinalities of the corresponding mappings that, as explained in Section IV-A.

## V. Conclusions

Unlike previous surveys [8], [9], in this paper we focused on data provenance models for Semantic Web data. More specifically, we discussed how implicit provenance information of SPARQL query results can be used to compute annotations reflecting various dimensions of data quality. We reviewed existing abstract provenance models for the relational data model, and showed that they can be leveraged for positive SPARQL queries over RDF data. Finally, we identified the limitations of these models in capturing the semantics of the SPARQL OPTIONAL operator, that implicitly introduces negation. We are currently working on the formalization of an abstract provenance model for SPARQL that supports a wide variety of applications involving annotation computations, as well as of less expressive provenance models for less demanding applications.

## References


[1] D. Artz and Y. Gil, "A Survey of Trust in Computer Science and the Semantic Web," *Web Semantics*, vol. 5, no. 2, 2007.

[2] H. Huang and C. Liu, "Query Evaluation on Probabilistic RDF Databases," in *WISE*, 2009.

[3] N. Fuhr and T. Rölleke, "A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems," *ACM TOIS*, vol. 14, no. 1, 1997.

[4] T. Imielinski and W. Lipski, "Incomplete Information in Relational Databases," *JACM*, vol. 31, no. 4, 1984.

[5] I. S. Mumick and O. Shmueli, "Finiteness Properties of Database Queries," in *ADC*, 1993.

[6] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen, "Update exchange with mappings and provenance," in *VLDB*, 2007.

[7] G. Karvounarakis, Z. G. Ives, and V. Tannen, "Querying Data Provenance," in *SIGMOD*, 2010.

[8] J. Cheney, L. Chiticariu, and W. C. Tan, "Provenance in databases: Why, where and how," *Foundations and Trends in Databases*, vol. 1, no. 4, 2009.

[9] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for Computational Tasks: A Survey," *CiSE*, vol. 10, no. 3, 2008.

[10] D. Srivastava and Y. Velegrakis, "Intensional Associations Between Data and Metadata," in *SIGMOD*, 2007.

[11] P. Buneman, J. Cheney, and S. Vansummeren, "On the Expressiveness of Implicit Provenance in Query and Update Languages," *ACM TODS*, vol. 33, no. 4, 2008.

[12] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," in *VLDB*, 2006.

[13] S. Miles, S. C. Wong, W. Fang, P. Groth, K. P. Zauner, and L. Moreau, "Provenance-based Validation of E-science Experiments," *Web Semantics*, vol. 5, no. 1, 2007.

[14] J. J. Carroll, C. Bizer, P. J. Hayes, and P. Stickler, "Named Graphs," *Web Semantics*, vol. 3, no. 4, 2005.

[15] R. Dividino, S. Sizov, S. Staab, and B. Schueler, "Querying for provenance, trust, uncertainty and other meta knowledge in RDF," *Web Semantics*, vol. 7, no. 3, 2009.

[16] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007.

[17] P. Buneman, S. Khanna, and W. Tan, "Why and Where: A Characterization of Data Provenance," in *ICDT*, 2001.

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

[19] Y. Cui and J. Widom, "Lineage Tracing for General Data Warehouse Transformations," in *VLDB*, 2001.

[20] B. Glavic and G. Alonso, "Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting," in *ICDE*, 2009.

[21] T. J. Green, "Containment of conjunctive queries on annotated relations," in *ICDT*, 2009.

[22] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," *ACM TODS*, vol. 34, no. 3, 2009.

[23] F. Geerts and A. Poggi, "On Database Query Languages for K-Relations," *Applied Logic*, vol. 8, no. 2, 2010.