On Provenance of Queries on Semantic Web Data

Capturing trustworthiness, reputation, and reliability of Semantic Web data manipulated by SPARQL requires researchers to represent adequate provenance information, usually modeled as source data annotations and propagated to query results along with query evaluation. Alternatively, abstract provenance models can capture the relationship between query results and source data by taking into account the employed query operators. The authors argue the benefits of the latter for settings in which query results are materialized in several repositories and analyzed by multiple users. They also investigate how relational provenance models can be leveraged for SPARQL queries, and advocate for new provenance models.

The recent W3C Linking Open Data initiative boosts the publication and interlinkage of massive amounts of datasets on the Semantic Web as Resource Description Framework (RDF) data queried with the SPARQL query language (see www.linkeddata.org and www.w3.org/tr/rdf-sparql-query). Together with other Web 2.0 technologies (such as mashups), this initiative has essentially transformed the Web from a publishing-only environment into a vibrant place for information dissemination in which 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’s crucial to be able to assert the trustworthiness, reputation, and reliability of published information. This functionality essentially calls for representing and reasoning with the provenance of Semantic Web data manipulated by SPARQL queries. For instance, in the case of trust assessment (one of the key applications recognized by the W3C Provenance Incubator Group), query result trustworthiness is determined based on the trustworthiness of the data sources from which they’re derived. For simple Boolean trust assessment, we need to determine only which output data should be trusted. For ranked trust assessment, we need to choose the
most trusted among competing evidence from diverse sources. For uncertain and fuzzy data, we derive the probabilities of query results 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 source data annotations. If source annotations are static and common for all users, we can do this computation together with the query evaluation (as for annotated databases3-5). However, in general, different users have different beliefs (for example, about source data’s trustworthiness), and these beliefs may change over time, even when the relationship of query results with source data is unchanged. For this reason, the alternative approach that we propose is to use abstract provenance models to capture this relationship along with the query operators that combine source data to derive query results. This information can be recorded6 in the repository when the data is imported and then used to compute appropriate annotations for different applications and users at a later time.7

In this article, we focus on data provenance in the style of James Cheney and his colleagues,8 who use it in the results of declarative queries. This is different from workflow provenance9 (such as the open provenance model, or OPM; see http://twiki.ipaw.info/bin/view/Challenge/OPM), which typically describes procedural data processing and where operations are usually treated as black boxes10 because of their complexity. Consequently, workflow provenance is generally less fine-grained than data provenance. Moreover, we’re interested in implicit provenance11 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.2,12 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 might not reflect the structure and properties of query operators as needed to support alternative annotation computations.

It’s worth noting that previous work on modeling provenance in the Semantic Web mainly focused on representing and querying workflow provenance information for e-science using the RDF13 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 uniform resource identifiers, or URIs) the RDF graphs to which they belong. They’re an annotation mechanism in which graph identifiers are explicitly stored and queried along with the original triples. Renata Dividino and her colleagues15 studied implicit data provenance for a SPARQL fragment that’s closer to positive relational algebra. In this article, we take a first step toward designing an abstract provenance model that’s capable of recording provenance for all SPARQL operators. In particular, we

- identify the basic characteristics of abstract provenance models and argue the benefits of using them to compute annotations for various applications on Semantic Web data;
- review relational abstract provenance models that can be used to capture the provenance of relational queries over Semantic Web data; and
- explore the extent to which these models can be leveraged for SPARQL queries over Semantic Web data, identifying their limitations.

The main challenges of the third part stem from the SPARQL OPTIONAL operator, which is crucial for dealing with the incompleteness of Semantic Web data, but can’t be handled by relational provenance models. For this reason, we also advocate for the need of new provenance models for SPARQL queries.

Abstract Provenance Models

A few basic characteristics of abstract provenance models support a range of annotation computations required by different applications and users. In particular, we consider the benefits of recording abstract provenance information when data is materialized in a repository through queries.

In different application settings, we 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 data model’s
main constructs, such as sets of attributes, tuples, or relations 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, we can achieve this by defining a named graph per triple. However, for applications such as trust assessment, simply knowing the provenance tokens of source data might not be sufficient. Consider, for instance, queries combining data from different sources, some of which are trusted. Multiple sources could 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 a data item’s derivation, thereby storing information on how input data items were combined to produce the item in question.

Once we compute and materialize abstract provenance expressions along with the query results, we can evaluate them to compute the annotations for a particular application. This amounts to substituting 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, whereas 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 to Boolean expressions dealing with incompleteness or uncertainty to tuple multiplicities. 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’s a need to assess its quality afterward. More precisely,

- users typically want to compute annotations for a (possibly small) subset of the items in the repository; and
- source data imported into the repository might be unavailable when a user tries to assess some dimension of the data’s quality.

Ideally, it would be nice to have an abstract provenance model that can support all applications of interest. However, there’s often a trade-off between the expressiveness of provenance models and the cost for storing and manipulating the corresponding provenance expressions. Consequently, for systems that only need to support a subset of these applications, it might be desirable to rely on less-informative abstract provenance models if they can provide improved performance.

### Provenance Models for Relational Queries

As a first step toward capturing the provenance of Semantic Web data, let’s consider a case in which we query them using positive relational algebra (denoted by RA⁺). Indeed, because RDF is the basic data model for representing Semantic Web data as triples of the form (subject, predicate, object), we can store them in a relational table with three columns. Therefore, it’s possible to query them using RA⁺ queries. Then, we can take advantage of previous work on relational provenance models to capture the provenance of query results.

Table 1 shows an RDF triple set, denoted by \( T \), in relational form, where \( S, P, \) and \( O \) stand for a triple’s subject, predicate, and object.

<table>
<thead>
<tr>
<th>Triple set ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>( S )</strong></td>
</tr>
<tr>
<td>( a )</td>
</tr>
<tr>
<td>( d )</td>
</tr>
<tr>
<td>( f )</td>
</tr>
</tbody>
</table>

Table 1. RDF triple set, denoted by \( T \), in relational form, where \( S, P, \) and \( O \) stand for a triple’s subject, predicate, and object.

#### Queries on Semantic Web Data
from subquery \( \pi_{SP}(T) \times \pi_{PO}(T) \) and the other two as projections on the results of subquery \( \pi_{SO}(T) \times \pi_{PO}(T) \).

The lineage\(^{19} \) of a tuple in a query result is the set of source tuples 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 2).

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’re both represented by the same set \(\{c_3\}\), whereas the last one involves both \(c_1\) and \(c_2\). Therefore, the why-provenance of \((f, e)\) is \(\{c_1\}, \{c_2, c_3\}\) (see Table 2). Intuitively, each inner set represents one or more derivations that involve the same source data, while multiple tokens in an inner set, such as \(\{c_2, c_3\}\), indicate a join between the corresponding tuples.

The Perm model\(^{20} \) employs tuples, instead of provenance tokens, to encode source data provenance. To illustrate how Perm works, we consider the provenance expression of the last tuple in the result \(\{S_3f, O_3e\}\), where \(S\) and \(O\) represent attribute names of \(T\). Perm retains two tuples for \((S_3f, O_3e)\), one for every derivation. In particular, the first tuple is \(\{S_1f, P_1; g, O_1; e\}; S_2f, P_2; g, O_2; e\}\), where the first two attributes represent the result tuple \((f, e)\), while the two occurrences of \((f, g, e)\) encode the fact that \((f, g, e)\) has been used twice to derive the tuple \((f, e)\). On the other hand, tuple \(\{S_3f, O_3e; S_1f; P_1; g, O_1; e; S_2; d, P_2; b, O_2; e\}\) 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 it records separately the derivations that involve 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 \((f, e)\) in Table 2, we obtain \(\{\{c_1\}, \{c_3\}\}\), 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 2), tuple \((f, g, e)\), participates twice in the first two derivations of \((f, e)\) and thus each one has provenance \(\{c_3\}\). 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 \(\{c_3\}\). Thus, the provenance of \((f, e)\) is \(\{c_1\} \odot \{c_3\} \oplus \{c_3 \odot c_2\}\). Compared to lineage, why-provenance, and Trio-lineage, how-provenance is the most informative\(^{21} \) provenance model. More precisely, as Todd Green and his colleagues show,\(^{16} \) it’s universal for all provenance models that can be expressed as semirings.

As we previously explained, 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, whereas for others, it isn’t. With this in mind, let’s focus on the applications of Boolean and ranked trust assessment, to illustrate such differences in expressiveness requirements. However, we also want to note that we’re generally interested in abstract
provenance models that can be used for a wide range of applications.7,16

**Boolean Trust Assessment**

In the Boolean trust assessment 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’re trusted if at least one of the derivations is trusted. Based on these semantics, which other researchers follow in a relational context,6,16 the trusted result tuples can be computed by answering the query on the subsets of the input relations containing only the trusted tuples.

We can compute trusted result tuples through provenance by assigning the values true (or false) to provenance tokens of trusted (or untrusted) tuples. Consider, for instance, the why-provenance of \((f, e)\) in the output \(\{\{c_1\}, \{c_2, c_3\}\}\) and let \(c_1 = c_3 = \text{true}, c_2 = \text{false}\). Thus, \((f, e)\) is trusted because there exists a derivation (namely, \(\{c_3\}\)), for which all tokens represent trusted source tuples (by which we mean that they have the value true).

**Ranked Trust Assessment**

In the ranked trust assessment case,7 every source tuple is associated with a rank — that is, 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 of a union or projection operator in the query, the rank of the output tuple is the minimum rank among all derivations, such as 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, the result tuple has a higher rank, and therefore one that’s less trusted, than both of the input tuples.

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 \(\min(\min(c_1 + c_3, c_1 + c_2), c_2 + c_3) = \min(\min(3 + 3, 3 + 3), 2 + 3) = \min(6, 5) = 5\). Had we considered a less expressive model, we would have computed an incorrect rank for \((f, e)\). Consider, for instance, the 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. For example, the evaluation of the Trio-lineage expression for \((f, e)\) would produce \(\min(\min(c_5, c_3), c_2 + c_3) = \min(\min(3, 3), 2 + 3) = \min(3, 5) = 3\). We conclude that Trio, as well as the less expressive why-provenance and lineage, fail to compute the correct rank. Therefore, when we compare the ranked trust assessment with the Boolean trust assessment, we observe that the former requires a more expressive provenance model than the latter.

**Capturing SPARQL Queries’ Provenance**

Earlier, we explained how to use relational provenance models to capture the provenance of relational queries over Semantic Web data. However, because Semantic Web data are by default represented in RDF, we now focus on capturing the provenance of the SPARQL queries typically employed to manipulate them.

**SPARQL in a Nutshell**

We base our presentation of SPARQL on the algebra presented by Jorge Pérez and his colleagues.22 This algebra is based on triple patterns — that is, triples of the form \((x, y, z)\), where \(x, y, z\) can be constants or variables, the latter prefixed with a question mark. Triple patterns are used to bind variables to values in the dataset. A set of pairs (variable, value) — that is, the SPARQL analog of the relational valuation — is called a mapping.

For instance, the pattern \((?x, ?y, e)\) only matches triples whose object has the value \(e\), and the result of matching it to the first triple of \(T\) (Table 1), is the mapping \(\{(?x, a), (?y, b)\}\), indicating that variables \(?x, ?y\) are bound to values \(a, b\) respectively. The evaluation of a triple pattern on a set of triples is a bag of mappings, represented as a set of mappings along with a cardinality function that associates every mapping of the set with an integer.

To simplify the presentation, we use the tabular representation of the mapping bags shown in Figure 1, where each column corresponds to a variable in the mappings. The SPARQL algebra of Pérez and his colleagues22 defines

- the unary operators \(\sigma\) (filtering) and \(\pi\) (projection) that correspond to the SPARQL constructs FILTER and SELECT, respectively, and
Provenance in Web Applications

Filtering on the triple components is expressed by fixing one of them to a constant. For instance, let \( \Omega \) (Figure 1a) denote the evaluation of \( (?x, ?y, ?z) \) over \( T \). Then, \( \sigma_{?x=?}(\Omega) \) contains only mappings \{\(?x, a\), (?y, b), (?z, c)\}.

Projection specifies the subset of variables to be returned in the query result. For example, \( \Omega_1 = \pi_{?x, ?y}(\sigma_{?z=?}(\Omega)) \) is the bag of mappings obtained from projecting the variables \(?x, ?y\) of \( \sigma_{?z=?}(\Omega) \) in Figure 1b. Similarly, \( \Omega_2 \) in Figure 1c denotes the result of query \( \pi_{?x, ?z}(\sigma_{?y=?}(\Omega)) \). To simplify the presentation, we employ the symbol \( \mu_i \) in Figure 1 to identify individual mappings.

Unlike the relational union that’s defined on relations with the same attributes, the union (\( \cup \)) operation of 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 Figure 1f (in SQL, this would be a null value).

To define the semantics of the join (\( \bowtie \)) operator, Pérez and his colleagues introduce the notion of compatible mappings.\(^\text{22}\) 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 the corresponding input mappings. Unlike in relational algebra, where a null value in an attribute makes any join condition fail, unbound variables in SPARQL don’t affect the compatibility of mappings. Figure 1g shows the result of \( (\Omega_1 \cup \Omega_2) \bowtie \Omega_3 \), where \( \Omega_1 \cup \Omega_2 \) is shown in Figure 1f, while \( \Omega_3 \) is in Figure 1d. Note that although \( ?y \) is unbound — for example, 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 \( \bowtie \) between mapping bags \( \Omega_1 \) and \( \Omega_2 \) returns the mappings contained in the result of \( \Omega_1 \bowtie \Omega_2 \) as well as all mappings from \( \Omega_1 \) that are incompatible with any mapping in \( \Omega_2 \). In this manner, \( \bowtie \) is similar to the left outer join operator of the relational algebra. Figure 1h shows the result of \( \Omega_1 \bowtie \Omega_4 \), where \( \Omega_1 \) is shown in Figure 1b and \( \Omega_4 \) in Figure 1e. For instance, \( \mu_{19} \) appears 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 incompatible with \( \mu_{17} \). Following the SPARQL algebra presented by Pérez and his colleagues,\(^\text{22}\) we denote with \( \Omega_1 \bowtie \Omega_2 \) the mappings of \( \Omega_1 \) that are incompatible with any \( \Omega_2 \) mapping, such as \( \Omega_1 \bowtie \Omega_4 = \{\mu_3\} \). As Pérez and his colleagues\(^\text{22}\) show, the following equivalence holds:

\[
\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \bowtie \Omega_2). \quad (1)
\]

We should stress that some subtle differences exist between the “\( \setminus \)” operator of Pérez and his colleagues\(^\text{22}\) and the relational minus operator (denoted as “\( \setminus \)” below). The former checks mappings for compatibility, while the latter only compares tuples for equality. Compatibility between mappings is a \( 1 - n \) relationship — that is, a mapping of \( \Omega_1 \) may be compatible with many mappings of \( \Omega_2 \). On the contrary, equality between tuples is a \( 1 - 1 \) relationship. Consider

---

**Figure 1.** Sample SPARQL algebra operators. Operator \( \sigma \) filters out mappings that don’t match the filtering condition, while \( \pi \) projects out variables. Operator \( \cup \) outputs mappings that appear in at least one of the input mapping bags. The \( \bowtie \) operator outputs the result of joining compatible mappings, while the \( \setminus \) operator additionally outputs the mappings of the left-hand mapping bag that are incompatible with all mappings of the right-hand mapping bag. The main text provides information about the \((a)-(h)\) labels.
for instance the relational query \( R_1 - R_n \). A tuple of the \( R_1 \) relation can be equal to at most one tuple of \( R_n \). Consequently, the existence of multiple copies of a mapping in \( \Omega_1 \) and \( \Omega_2 \) doesn’t affect the cardinality of that mapping in the result: if a mapping \( \mu \) has cardinality \( m \) in \( \Omega_1 \) and there’s one compatible mapping with cardinality \( n \) in \( \Omega_2 \), \( \mu \) will have cardinality \( 0 \) in the result – which means that it won’t appear in it. On the contrary, in the relational context, if a tuple \( t \) has cardinality \( m \) in relation \( R_1 \) and \( n \) in \( R_n \), then the cardinality of \( t \) in \( R_1 - R_n \) is \( m - n \), if \( m > n \), and 0 otherwise.

**Provenance Models for Positive SPARQL**

From the previous presentation, there’s 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 of these operators as positive SPARQL (denoted by SPARQL\(^+\)) and investigate whether provenance models for RA\(^+\) queries can also be 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 \( \text{DISTINCT} \). Among the provenance models for relational queries, only how-provenance can be used to compute correct result multiplicities under bag semantics,\(^{16}\) whereas all models can handle set semantics. Finally, the differences between SPARQL and relational algebra for the \( \cup \) and \( \bowtie \) operators don’t affect the provenance of output mappings. Consequently, all the abstract provenance models for RA\(^+\) that we presented can be applied to SPARQL\(^+\) under set semantics, while how-provenance can be used when bag semantics are needed.

**Toward Provenance for SPARQL**

However, relational provenance models aren’t sufficient to capture provenance for the SPARQL algebra, essentially because of the \( \bowtie \) operator. This is because \( \bowtie \) involves a form of negation (see the use of \( \setminus \) in Equation 1), while most of the aforementioned models capture the provenance of positive queries. We illustrate the challenges posed by \( \bowtie \) through an example of Boolean trust assessment.

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 2 that are incompatible with any trusted mapping of \( \Omega_4 \) should appear in the query output as trusted. This semantics also coincides with the semantics of SPARQL\(^+\) if we apply the \text{EnsureTrust} operator to filter out untrusted mappings from input mapping sets (by setting the lower \( l \) and upper \( u \) bounds to true).

Suppose, for example, that mappings \( \mu_1 \) and \( \mu_2 \) of \( \Omega_1 \), and \( \mu_{17} \) of \( \Omega_4 \) are trusted (see Figures 2a and 2b). Figure 2c depicts the trusted mappings of \( \Omega_1 \bowtie \Omega_4 \). We can observe that \( \mu_{17} \) 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 incompatible with any trusted mapping of \( \Omega_4 \).

On the other hand, if \( \mu_1 \) and \( \mu_2 \) were trusted but \( \mu_{17} \) was not, \( \mu_1 \) would be incompatible with any trusted mapping of \( \Omega_4 \). Thus, mapping \( \mu_{21} \) should appear in the result as trusted (see Figure 2d). We can easily observe that, although \( \mu_{21} \) (respectively, \( \mu_{19} \)) doesn’t belong to the query result illustrated in Figure 2c (respectively, Figure 2d), an abstract provenance model that can be used for such trust computations would need to associate both those mappings with appropriate provenance expressions.

Existing provenance models for RA\(^+\) queries don’t support the semantics of SPARQL\(^+\) or \( \setminus \). Even Perm, which captures negation, doesn’t record sufficient information for enabling annotation computations such as the Boolean trust assessment (see Figure 2). More precisely, Perm records the reason why \( \mu_{20} \) exists in the result – for example, that \( \mu_2 \) is incompatible.
with $\mu_{17}$ — by keeping in the output the mapping \{(?x, f), (?y, g), (?y, h), (?z, c)\}. However, it doesn’t 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.

Similar to Perm, Dividino and her colleagues\textsuperscript{23} document that $\mu_{20}$ exists in the result because $\mu_2$ is incompatible with $\mu_{17}$. However, it doesn’t encode provenance information for $\mu_{21}$, and therefore it can’t infer that $\mu_{21}$ should be in the result as trusted if $\mu_{17}$ is untrusted.

$M$-semirings\textsuperscript{23} are 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 $m$-semiroring model would employ Equation 1 for $\Omega_1 \bowtie \Omega_4$. The provenance expressions for mappings in $\Omega_1 \bowtie \Omega_4$ are computed in the same manner as in the case of how-provenance — for example, the provenance of $\mu_{19}$ is $c_1 \ominus c_3$, where $c_1$ (respectively, $c_3$) is the provenance of $\mu_1$ (respectively, $\mu_{17}$) in $\Omega_1$ (respectively, $\Omega_4$). 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 $\Omega_4$, while $0$ denotes that $\mu_2$ doesn’t 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 $\mu_{22} = \{ (?y, b), (?z, e) \}$, which is compatible with $\mu_1$. Then, $\mu_{21}$ would appear in the result as trusted (as Figure 2d shows), only if both $\mu_{17}$ and $\mu_{22}$ were untrusted. However, the $m$-semiroring expression for $\mu_{21}$ could only encode a single mapping of $\Omega_4$.

We conclude that a new provenance model is needed to cope with the $\bowtie$ operator. In this model, provenance expressions should be recorded for some mappings that don’t appear in the result of a query involving the $\bowtie$ operator, such as $\mu_{21}$ in $\Omega_1 \bowtie \Omega_4$ in Figure 2c. It’s worth mentioning that this need also appears when provenance expressions should be computed for the relational left (or right) outer join.

Moreover, this model can’t be based on techniques used in relational provenance models to deal with relational minus because of the differences between the SPARQL algebra $\setminus$ 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 $\setminus$ operator should conform to SPARQL semantics for cardinalities of the corresponding mappings.

Unlike previous surveys,\textsuperscript{8,9} here 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. This approach, however, has limitations in capturing the semantics of the SPARQL $\text{OPTIONAL}$ operator because it implicitly introduces negation. We’re currently working on formalizing an abstract provenance model for SPARQL that supports a wide variety of applications involving annotation computations, as well as less expressive provenance models for less demanding applications.

References


Yannis Theoharis is a doctoral candidate in the computer science department at the University of Crete and a research assistant in the Information Systems and Software Technology Laboratory of the Foundation for Research and Technology Hellas, Institute of Computer Science (Forth-ICS). His research interests include Web data management, as well as database and information retrieval systems. Theoharis has an MSc in computer science from the University of Crete. Contact him at theohari@ics.forth.gr.

Iriini Fundulaki is an associate researcher at Forth-ICS and the head of the World Wide Web Conference’s (W3C) Greece office. Her research interests include databases and the Semantic Web, with a focus on access control for XML and RDF data, provenance for RDF data, XML personalization, and SPARQL query engines. Fundulaki has a PhD in computer science from the Conservatoire National des Arts et Metiers in Paris. Contact her at fundul@ics.forth.gr.

Grigoris Karvounarakis is a computer scientist at Logic-Blox and a postdoctoral researcher in the Information Systems and Software Technology Laboratory of Forth-ICS. His main research interests include data integration and exchange, data provenance, and Web query languages and models, including Semantic Web and peer-to-peer data management systems. Karvounarakis has a PhD in computer and information science from the University of Pennsylvania. Contact him at gregkar@ics.forth.gr.

Vassilis Christophides is a professor of computer science at the University of Crete and an affiliated researcher in the Information Systems and Software Technology Laboratory of Forth-ICS. His main research interests include databases and Web information systems, digital libraries, and preservation and scientific systems. Christophides has a PhD in structured documents and object bases from the Conservatoire National des Arts et Metiers of Paris. He received the 2004 SIGMOD Test of Time Award and the Best Paper Award at the International Semantic Web Conference in 2003 and 2007. Contact him at christop@ics.forth.gr.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.