On Provenance of Queries on Linked Web Data

Yannis Theoharis[1,2], Irini Fundulaki[2], Grigoris Karvounarakis[3,2], and Vassilis Christophides[1,2]

[1]**Computer Science Department, University of Crete, Greece**

[2]**Information Systems Laboratory, Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH)**

[3]**LogicBox, USA**

Email: {theohari, fundul, gregkar, christop}@ics.forth.gr

Technical Report FORTH-ICS/TR-406 - June 2010

Abstract

Assessing the quality of information published on the Web according to the Linked Data paradigm emerges as a crucial need of various applications. Capturing trustworthiness, reputation and reliability of Linked Web Data manipulated by SPARQL, requires representing adequate provenance information to capture the relationship between query results and the source data. This relationship can either be materialized along with the query result or lost after the query evaluation. In the former case, query result provenance consists of abstract expressions on provenance tokens that uniquely identify source data items, while in the latter, it consists of annotations that are computed based on source data annotations along with query evaluation. We argue about the benefits of the former, 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.

Contents

List of Figures

Chapter 1

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 [51]. Various knowledge bases with billions of RDF triples from Wikipedia[2], U.S. Census[3], CIA World Factbook[4], open government sites in the US and the UK[5], news and entertainment sources[6], as well as various ontologies[7] have been created and made available online. In addition, numerous data and vocabularies in e-science are published nowadays as RDF graphs [39], most notably in life sciences[8], environmental or earth sciences[9] and astronomy[10], in order to facilitate community annotation and interlinkage of both scientific and scholarly data[11] of interest.

[1] linkeddata.org
[2] www.dbpedia.org
[3] www.rdfabout.com/demo/census
[4] www.aktors.org/interns/2005/cia/CIA World Factbook to RDF.htm
[5] www.data.gov,data.gov.uk
[6] backstage.bbc.co.uk,wiki.musicbrainz.org/RDF
[7] www.w3.org/TR/wordnet-rdf, www.mpi-inf.mpg.de/yago-naga/yago, http://www.daml.org/2003/09/factbook
[8] www.geneontology.org,www.biopax.org,dev.isb-sib.ch/projects/uniprot-rdf, swan.mindinformatics.org,www.nlm.nih.gov/research/umls, www.code.org/galen
[9] sweet.jpl.nasa.gov/ontology
[10] archive.astro.umd.edu/ont
[11] knoesis.wright.edu/library/ontologies/swetodblp

Compared to the traditional Web of documents featuring *untyped hyperlinks*, Linked Data design principles and implementation practices aim to realize the vision of a *Web of Data*. In this context, data is published on the Web in such way that its meaning is *explicitly defined*, and is connected to other data that reside in heterogeneous sources [5] through *typed* links. Linked Data is represented in the RDF data model [39] and queried using the SPARQL query language [48]. Linked Data and Web 2.0 technologies 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: Web users are no longer plain data consumers but have become active data producers and data dissemination agents. In Figure 1.1 we present a scenario in which Linked Data originates from heterogeneous sources (e.g. RDFa documents, ontologies and databases) and is processed by Web programs (e.g. mashups [47, 42]) that employ SPARQL queries.

In this context, assessing the quality of published Linked Data emerges as a crucial need. Being able to assert *trustworthiness*, *reputation* and *reliability* of Linked Data are just few of the challenges recognized by the W3C Provenance Incubator Group [56], which essentially require to capture and represent the *provenance* of Web data. For instance, in the case of applications requiring *trust assessment* [3, 2], in order to determine which data in the result of a query should be trusted, judgments regarding the trustworthiness of data sources must be taken into account. To this end, users assess their beliefs for the reliability of data sources. For the simplest case, i.e. *boolean trust* assessment, we just want to determine which output data should be trusted or untrusted. In other applications, *ranked trust* assessment is needed, e.g. to choose among competing evidence from diverse sources or to produce ranked results in keyword search [53, 54]. In data exchange or warehousing settings there is a need for *debugging* [13], e.g., to trace sources of wrongful information in query results, or

to identify schema mappings and views that may have lead to incorrect results. For *uncertain* and *fuzzy* data, the probabilities of query results are derived based on the probabilities associated with the original data [4, 33, 23, 34, 41]. In *access control* systems, provenance information is necessary in order to assign a confidentiality access level to query results using the access level of the input data [20, 52]. Finally, authors in [38, 29] have shown the benefits from using provenance information of data in a view to perform *view update* and *incremental view maintenance*, when updates occur in the view or the sources respectively, as it has been shown for collaborative data sharing systems [35].

In all these applications, 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. However, in general, different users may have different beliefs about which source data is trusted or untrusted, and these beliefs may change over time. For this reason, an alternative approach is to use *abstract provenance models* to capture the relationship of query results with source data along with the query operators that combined them. As depicted in Figure 1.1, this information can be recorded in the repository when the data is imported to compute appropriate annotations for different applications and users at a later time [36].

In this paper, we focus on *data provenance* in the style of [12], i.e. provenance of data in the result of *declarative queries*. This is different from *workflow* provenance [6, 16, 21, 44] which typically describes *procedural data processing*, in which operations are usually treated as black boxes 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 [9] of regular queries, i.e., ones that only manipulate data and are oblivious about the possible annotations of source data. Implicit provenance captures the abstract structure and properties of query operators

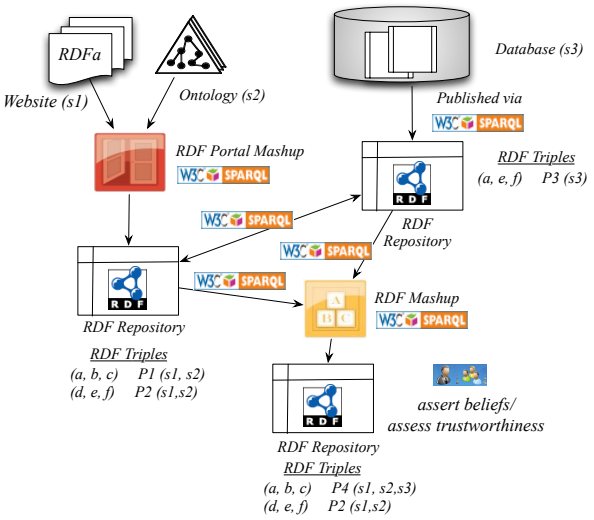

Figure 1.1: Motivating Example

and can, thus, be used for various annotation computations [4, 36]. This is in contrast with work on *explicit* provenance [9], where the query language is extended so that queries can also manipulate source annotations and specify explicitly the annotation of the results. Because of this additional flexibility, the resulting annotations can be arbitrary and may not reflect the structure and properties of query operators, as needed in order to support annotation computations.

It is worth noting that previous work on modeling provenance in the *Semantic Web* [28, 22, 40, 49, 58, 32, 43] mainly addresses workflow computations for e-science applications. The focus is given on representing and querying workflow-related provenance information using RDF or OWL. Earlier work on RDF data provenance includes named graphs [11] (or more fine-grained RDF molecules [17]), which have been proposed as a means to define *ownership* of RDF triples by objectifying (through URIs) the RDF (sub)graphs to which they belong [57, 50]. Named graphs are essentially an explicit provenance mechanism where graph identifiers are stored and queried along with the original triples. To this end, SPARQL had to be extended and users

should appropriately write their queries to specify which triples should belong to which named graph. The limitations of named graphs to reason over provenance information in the presence of RDFS [7] inference rules and updates have been shown in [19, 45]. Finally, authors in [18] have studied implicit data provenance for a fragment of SPARQL that is closer to the positive relational algebra.

In this paper, we take a first step towards designing an abstract provenance model for SPARQL. In particular, we make the following contributions:

- We identify the basic characteristics of abstract provenance models, as prescribed by the requirements of the aforementioned applications, and argue on the benefits of using abstract provenance to compute annotations for various forms of data quality (Section 2).

- We review relational abstract provenance models [4, 10, 14, 27, 30], that can be used to capture the provenance of relational queries over Linked Data (Section 3).

- We explore the extent to which these models can be leveraged for SPARQL queries over Linked Data, and identify their limitations to cope with *negation* associated with the SPARQL OPTIONAL operator (Section 4). In particular, we advocate the need for new provenance models to capture the full expressive power of SPARQL queries.

Chapter 2

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.

2.1 Basic Characteristics of Abstract Provenance Models

In all applications described in the previous section, there is a need to identify and refer to source data that is involved in the derivation of query results. To address this problem in the relational world, the most common approach is to *annotate* source data with appropriate abstract labels called *provenance tokens* [30], which are essentially unique IDs for source data items. The term "*colors*" was introduced by [25] to represent annotations for the relational model, and [24] discussed *color relations* as a general abstraction of annotated databases. In addition, [9] used colors as provenance tokens to capture explicit provenance in the context of relational databases. In this

paper, we use the term "provenance tokens" to refer to annotations on data sources. The granularity of the annotated data items typically depends on the main constructs of the data model, e.g. *tuples* [4, 30, 10, 14, 27] or *attributes* and *relations* [10] in the relational context.

Besides provenance tokens, provenance models define *abstract operations* on tokens. Then, the provenance of output data items in a query result is encoded as an *abstract expression* that combines provenance tokens with provenance operations. Instead of rich provenance expressions, one could consider as provenance of the output the set of provenance tokens of source data items. For Linked 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 an 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 resulting data item.

Beyond data provenance, for some applications it might be useful to encode provenance information of performed computations. For example, in the case of debugging of schema mappings, or (boolean or ranked) trust computations, one may need to track down particular mappings that may be introducing faulty data or specify that some mappings are more or less trustworthy than others [13, 36, 37]. In order to reason on such forms of provenance, it may be necessary to name particular queries (or mappings) and record these names in the provenance of results whose derivation involves the corresponding queries. To simplify the presentation, in this paper we focus on models that do not encode mapping names.

Once abstract provenance expressions have been computed and materialized along

with the query results, one can *evaluate* them to compute appropriate annotations for a particular application. The evaluation of a provenance expression requires 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 corresponds to a particular application. For instance, in the case of boolean trust assessment, provenance tokens can be substituted with $true$ or $false$, for trusted and untrusted source tuples, respectively, and combined through $\vee$ (disjunction) and $\wedge$ (conjunction). This substitution is only possible if the abstract operations of the provenance model generalizes the corresponding operations among annotations. For example, as we show in Section 3.1, an abstract provenance model that ignores multiple occurrences of the same source tuple in a derivation cannot be used for *ranked trust* assessment where the number of such occurrences matters.

2.2 Benefits from Abstract Provenance Information

Instead of materializing abstract provenance expressions, one could consider annotating source data with appropriate values and compute annotations for query results during query evaluation. This approach was followed by previous work on query answering on annotated databases, for several kinds of annotations ranging from probabilistic event expressions to boolean expressions dealing with incompleteness or uncertainty, or to tuple multiplicities. In this manner, the annotation computation is coupled with query evaluation. As a result, if the assignment of annotations on source data changes, the query should be evaluated again.

In the context of Linked Data, abstract provenance models are highly beneficial

compared to annotation systems, 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 about the appropriate source data annotations;
- a user's beliefs may change;
- users typically want to compute annotations for a (possibly small) subset of the items in the repository;
- source data imported to the repository may be unavailable at the time a user tries to assess some dimension of data quality [8].

Furthermore, the use of abstract provenance models is highly beneficial for applications that require incremental computations. For instance, *view maintenance* and *view update* problems can be incrementally solved only if information relating the input to the output data is available. This is due to the fact that there is no way to precompute appropriate annotations at query evaluation time, since it is impossible to know which updates will happen in the future when materializing the original view.

Ideally, one would use an abstract provenance model that could support all applications of interest. However, in many cases there is a tradeoff between the expressiveness of provenance models and the cost for storing and manipulating the corresponding provenance expressions, as we explain later in this paper. As a result, for systems that need to support only a subset of these applications it may be desirable to rely on less-informative abstract provenance models if they can provide improved performance.

Chapter 3

Provenance Models for Relational Queries over Linked Data

Linked Data published in RDF can be represented as *triples* (*subject*, *predicate*, *object*) and stored in relational databases in a unique table with three columns. Thus, it can be queried using positive relational algebra (RA^+) and its provenance can be captured by existing relational provenance models.

Table (a) of Figure 3.1 shows an RDF triple set in relational form. S, P and O stand for the *subject*, *predicate* and *object* of a triple. The fourth column represents its provenance token.

Consider now 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 3.1 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 $\pi_{SP}(T) \bowtie \pi_{PO}(T)$. The other two are returned as projections on the results of subquery $\pi_{SO}(T) \bowtie \pi_{PO}(T)$.

Lineage [14] encodes only the provenance tokens of the tuples that were used in

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**
$a\ c$	$\{c_1\}$	$\{\{c_1\}\}$	$\{\{c_1\},\{c_1\}\}$	$c_1 \odot c_1$
$a\ e$	$\{c_1, c_2\}$	$\{\{c_1, c_2\}\}$	$\{\{c_1, c_2\}\}$	$c_1 \odot c_2$
$d\ c$	$\{c_1, c_2\}$	$\{\{c_1, c_2\}\}$	$\{\{c_1, c_2\}\}$	$c_1 \odot c_2$
$d\ e$	$\{c_2, c_3\}$	$\{\{c_2\},\{c_2, c_3\}\}$	$\{\{c_2\},\{c_2\},\{c_2, c_3\}\}$	$c_2 \odot c_2 \oplus c_2 \odot c_2 \oplus c_2 \odot c_3$
$f\ e$	$\{c_2, c_3\}$	$\{\{c_3\},\{c_2, c_3\}\}$	$\{\{c_3\},\{c_3\},\{c_2, c_3\}\}$	$c_3 \odot c_3 \oplus c_3 \odot c_3 \oplus c_2 \odot c_3$

(b)

Figure 3.1: Example of *lineage*, *why*-provenance, *Trio-lineage* and *how*-provenance (b) for the query Q over T (a)

some derivation of the query result. The first and second derivations of (f, e) use only tuple (f, g, e), annotated with 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 3.1). On the other hand, the remaining models encode some information about the operators that were used for each derivation.

Why-provenance [10] encodes all the different derivations of a tuple in the query result by storing *sets of sets of provenance tokens*. In our example, the first and second derivation is annotated with $\{c_3\}$, whereas in the last the result is annotated with c_3 and c_2. Therefore, the *why*-provenance of (f, e) is $\{\{c_3\}, \{c_2, c_3\}\}$ (see Table (b) of Figure 3.1). That kind of nesting is used by *why*-provenance to encode the different operators. In particular, each inner set represents one or more derivations that have been influenced by the same source data, while multiple tokens in an inner set, e.g. $\{c_2, c_3\}$, denote a join between the tuples annotated with these tokens.

Perm [27] 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 (f, e). Perm retains two tuples for (f, e), one for every derivation, i.e. (f, e, f, g, e, f, g, e) and (f, e, f, g, e, d, b, e). The first two attributes represent the result tuple (f, e), while the remaining ones represent its provenance. In particular, the two occurrences of (f, g, e) in (f, e, f, g, e, f, g, e) encode the fact that it has been used twice to derive the tuple (f, e). On the other hand, tuple (f, e, f, g, e, d, b, 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 the case of *why*-provenance. Note that *Perm* cannot be used in contexts in which the query input carries provenance information (i.e., query composition).

Compared to *why*-provenance, *Trio*-lineage [4] additionally records how many times each derivation appears in the query result. If we follow the set representation of *why*-provenance to express *Trio*-lineage, each provenance expression would be a *bag* (instead of a set for *why*-provenance) of sets of tokens (see Table (b) of Figure 3.1). Hence, for the first two derivations we obtain $\{\{c_3\}, \{c_3\}\}$ whereas for the last we have $\{\{c_2, c_3\}\}$.

Finally, *how*-provenance [30] 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 3.1), the same tuple, i.e. (f, g, e), participates twice in the first two derivations of (f, e), 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) are annotated with c_3 and c_2 respectively. Therefore, the provenance of that derivation is $c_2 \odot c_3$. The total provenance expression of (f, e) is $c_3 \odot c_3 \oplus \ c_3 \odot c_3 \oplus c_2 \odot c_3$.

3.1 Expressiveness of Provenance Models

As we discussed in the previous Section, 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, we illustrate such differences in expressiveness requirements for applications involving boolean or ranked trust assessment. In particular, we first describe the semantics of these applications for positive queries and then examine which of the provenance models of the previous section meet the expressiveness requirements of each application.

3.1.1 Boolean Trust Semantics

In the case of boolean trust assessment, trusted (resp. untrusted) tuples are annotated with *true* (resp. *false*). Given a query, the goal is to find which tuples of the output are trusted. Boolean trust semantics is equivalent with the set semantics for query evalution on the subset of input that is trusted. In this manner, untrusted tuples are treated as missing from a (input / output) relation and vice versa, all possible tuples that are missing from a relation are treated as untrusted.

Trustworthiness is propagated through query operators according to boolean trust semantics. In the case of union or projection, where multiple derivations of an output tuple may occur, a trusted derivation suffices to infer that the tuple is trusted. In the case of join, both joined tuples should be trusted in order to consider their result as trusted.

3.1.2 Ranked Trust Semantics

In the context of *ranked trust* assessment, 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 ∞ indicates tuples that are completely untrusted. More specifically, tuples annotated with ∞ are treated as missing.

In the case of multiple derivations of a tuple in the result of a union or projection, the derivation of the minimum rank, i.e. the most trusted, is considered in the output. In the case of join, the resulting tuple is annotated with the summation of the ranks of the two input tuples. In this respect, it has a higher rank, i.e. is less trusted, than both of them.

3.1.3 Provenance Models for Boolean and Ranked Trust Assessment

In the case of *boolean trust assessment*, we associate the provenance token of every source tuple in Table (a) of Figure 3.1 with a boolean value. Additionally, we consider that $\oplus$, $\odot$ represent the boolean operators $\vee$ and $\wedge$ respectively. For example, suppose that the first and third tuple are trusted i.e. $c_1 = c_3 = true$, while the second one is untrusted, i.e. $c_2 = false$. Then, *lineage* is not expressive enough to support *boolean trust* management, since it can not distinguish between (d, e) (that is untrusted) and (f, e) (that is trusted), because they have the same provenance, i.e. $\{c_2, c_3\}$. The remaining models are able to support *boolean trust* management. In particular, *why*-provenance and *Trio*-lineage consider as trusted those tuples whose provenance has at least one element (i.e. inner set) whose all tokens are true. For instance, (a, c) is deemed trusted due to the appearance of $\{c_1\}$ and (f, e) due to the appearance of $\{c_3\}$, while the remaining tuples are deemed untrusted due to the appearance of c_2 in every inner set of the provenance expressions. The same result is also achieved by *how*-provenance since $c_1 \odot c_1$ for (a, c) evaluates to $true \wedge true = true$ and $c_3 \odot c_3 \oplus c_3 \odot c_3 \oplus c_2 \odot c_3$, for (f, e) to $(true \wedge true) \vee (true \wedge true) \vee (false \wedge true) = true$. The expressions of the remaining tuples evaluate to *false*.

Ranked trust assessment requires a more expressive provenance model than boolean trust assessment. In this case, $\oplus$, $\odot$ correspond to the binary operators *min* and + respectively. For instance, let $c_1 = 1$, $c_2 = 2$, $c_3 = 3$. Then, if we consider the most detailed provenance expression that is derived by *how*-provenance, the rank of (f, e) in the output is computed as $min(min(c_3, c_3) + c_3, c_2 + c_3) = min(min(3, 3) + 3, 2 + 3) = min(6, 5) = 5$. Had we considered a less expressive model, we would compute 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 result to $min(min(c_3, c_3), c_2 + c_3) = min(min(3, 3), 2 + 3) = min(3, 5) = 3$. We conclude that *Trio*, as well as the less expressive *why*-provenance and *lineage*, fail to compute the correct rank.

Chapter 4

Capturing the provenance of SPARQL queries

In this Section, we consider Linked Data queried by SPARQL, rather than relational queries. We first leverage the relational provenance models for a fragment of SPARQL that is closer to RA^+. As a next step, we show their limitations for modeling abstract provenance information for SPARQL.

4.1 SPARQL in a Nutshell

We base our presentation of SPARQL on the algebra presented in [46]. A core construct of this algebra is a *triple pattern*, of the form (x, y, z), where x, y, z can be constants or variables prefixed with "?". Triple patterns are used to bound variables to values in the dataset. A set of pairs $(variable, value)$, i.e. the 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, 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

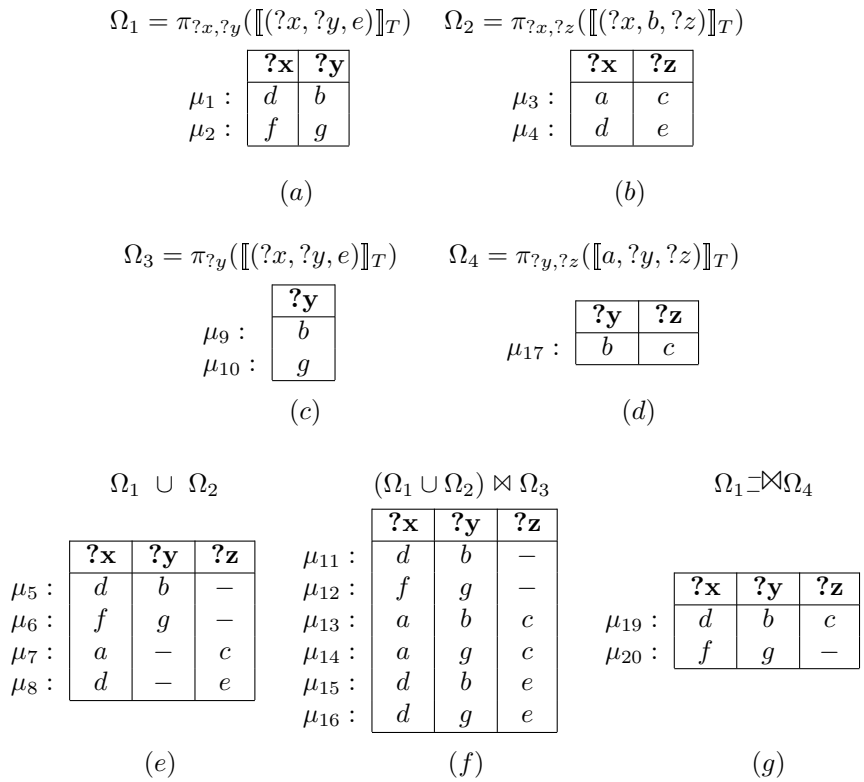

Figure 4.1: Example of SPARQL Algebra Operators

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. Hereafter, we denote with $[\![tp]\!]_T$ the evaluation of the triple pattern tp over a triple set T. To simplify the presentation, we will use the tabular representation of the mapping bags shown in Figure 4.1, where each column corresponds to a variable in the mappings.

The SPARQL algebra in [46] defines: a) the unary operators σ (filtering) and π (projection) that correspond to the SPARQL constructs FILTER and SELECT, respectively and b) the binary operators, $\cup$, $\bowtie$, $\overline{\bowtie}$ 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 Ω denote the evaluation $[\![(?x, ?y, ?z)]\!]_T$. Then $\sigma_{?x=a}(\Omega)$ contains only the mapping $\{(?x, a), (?y, b), (?z, c)\}$.

Projection specifies the subset of variables to be returned in the query result. For example, $\Omega_1 = \pi_{?x,?y}([\![(?x, ?y, e)]\!]_T)$ is the bag of mappings obtained from projecting the variables $?x$, $?y$ of $[\![(?x, ?y, e)]\!]_T$ (Table (a) of Figure 4.1). Similarly, Ω_2 in Table (b) of Figure 4.1 denotes the result of query $\pi_{?x,?z}([\![(?x, b, ?z)]\!]_T)$. To simplify the presentation, we employ symbols μ_i in Figure 4.1 to identify individual mappings.

Unlike relational union that is defined on relations of the same schema, the input mapping bags of union ($\cup$) may contain mappings with different variables. For instance, the result of $\Omega_1 \cup \Omega_2$ shown in Table (e) of Figure 4.1, includes mappings with unbound variables, denoted by "-" (in SQL that would be represented as a *null* value).

In order to define the semantics of the join ($\bowtie$) operator, [46] 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 contains 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 4.1 shows the result of $(\Omega_1 \cup \Omega_2) \bowtie \Omega_3$ in Table (f), where $\Omega_1 \cup \Omega_2$ is shown in Table (e), while Ω_3 in Table (c). Note that, although $?y$ is unbound in μ_7, SPARQL considers it to be compatible with μ_9 and μ_{10}.

Finally, the application of the optional (⟕) operator between mapping bags Ω_l and Ω_r returns the mappings contained in the result of $\Omega_l \bowtie \Omega_r$, as well as all mappings from Ω_l that are not compatible with any mapping in Ω_r. In this manner, ⟕ is similar to the left outer join operator of the relational algebra. Figure 4.1 shows the result of $\Omega_1 \text{⟕} \Omega_4$ in

Table (g), where Ω_1 is shown in Table (a), while $\Omega_4 = \pi_{?y,?z}([\![a, ?y, ?z)]\!]_T)$, in Table (d). For instance, μ_{19} is in the result because of the join between μ_1 and μ_{17}, while μ_{20} appears in the result because μ_2 belongs to Ω_1 and is not compatible with μ_{17}. Following [46], we denote with $\Omega_l \setminus \Omega_r$ the mappings of Ω_l that are not compatible with any Ω_r mapping, e.g. $\Omega_1 \setminus \Omega_4 = \{\mu_2\}$. As shown in [46], the following equivalence holds between $\overline{\bowtie}$, $\bowtie$, $\cup$ and $\setminus$:

$$\Omega_l \overline{\bowtie} \Omega_r = (\Omega_l \bowtie \Omega_r) \cup (\Omega_l \setminus \Omega_r) \qquad (1)$$

4.2 Provenance Models for Positive SPARQL

From the previous presentation, there is a clear analogy of the SPARQL algebraic operators of projection (π), filter (σ), join ($\bowtie$) and union ($\cup$) with the corresponding operators of RA^+. For this reason, we refer to the fragment of SPARQL consisting only of the above operators as *positive* SPARQL (SPARQL^+) and investigate whether provenance models for RA^+ queries presented in Section 3, 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, while all models can handle set semantics. Finally, the differences between SPARQL and relational algebra for the $\cup$ (SPARQL $\cup$ resembles the relational outer union) and $\bowtie$ (unbound variables in SPARQL do not affect compatibility, unlike null values that make any relational join condition to fail)

operators do not affect the provenance of output mappings (see also Section 4.1). As a consequence, all abstract provenance models for RA^+ presented in Section 3, can be applied to $SPARQL^+$ under set semantics, while how-provenance can be used when bag semantics is needed.

We should also mention *why*-provenance model for
$SPARQL^+$ presented in [18]. Although it is inspired by the semiring model [30], the two defined abstract operators are idempotent and thus multiple contributions of the same tuple in a join, or in a union are ignored.

4.3 Towards Provenance for Full SPARQL

However, relational provenance models are not sufficient to capture provenance for the full SPARQL algebra, essentially due to the use of the optional ($\sqsupset\!\bowtie$) operator. This is because $\sqsupset\!\bowtie$ involves negation (see the use of "$\setminus$" in expression (1)), while most of the aforementioned models capture the provenance of positive queries. To illustrate the challenges posed by $\sqsupset\!\bowtie$, we will consider the *boolean trust* assessment, whose semantics for positive queries were presented in Section 3.1.1. Below, we describe the semantics of that application with respect to optional.

4.3.1 Boolean Trust Semantics for Optional

As presented in Section 4.1, $\Omega_l \sqsupset\!\bowtie \Omega_r$ retains in the output as much information of Ω_l as possible, i.e. some mappings of Ω_l are joined with some mappings of Ω_r, while the remaining Ω_l mappings appear in the result as such. Recall from Section 3.1.1, that boolean trust semantics is equivalent with the set semantics for query evalution on the subset of input that is trusted. In this respect, *trusted* mappings of Ω_l that are not compatible with any *trusted* mapping of Ω_r should appear in the output as trusted mappings. This also coincides with the semantics for $\sqsupset\!\bowtie$ of *EnTrust* operator

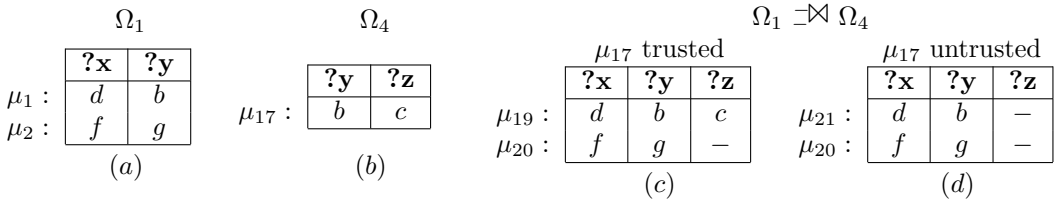

Figure 4.2: Example of Boolean Trust

in *tSPARQL* [31].

Suppose, for example, that mappings μ_1, μ_2 of Ω_1 and μ_{17} of Ω_4 are trusted, as shown in Tables (a) and (b) of Figure 4.2. Table (c) shows the trusted mappings of $\Omega_1 \text{ }⟕⋈ \Omega_4$. In

particular, μ_{19} is the result of the join between two trusted mappings, i.e. μ_1 and μ_{17}, while μ_{20} is trusted because μ_2 is trusted in Ω_1 and is not compatible with any trusted mapping of Ω_4.

On the other hand, suppose that mappings μ_1, μ_2 are trusted but μ_{17} is untrusted. In that case, μ_1 is no longer compatible with any trusted mapping of Ω_4. As a result, mapping $\mu_{21} : \{(?x, d), (?y, b)\}$ should be trusted in the result (see also Table (d) of Figure 4.2). The challenge here is caused by the *non-monotonicity* of the $⟕⋈$ operator. For instance, the change of μ_{17} from trusted to untrusted, has two effects in different directions: μ_{19} becomes untrusted, while μ_{21} becomes trusted.

4.3.2 Limitations of RA$^+$ Provenance Models

Lineage, *why*-provenance, *Trio* and *how*-provenance do not support any operators resembling $⟕⋈$ or $\setminus$. *Perm* is the only model that captures the left outer join operator. Unfortunately, it does not record sufficient information to support annotation computations such as the boolean trust assessment example described above. In particular, *Perm* documents the reason why μ_{20} exists in the result, i.e. that μ_2 is

not compatible with μ_{17}. To this end, it keeps in the output a mapping (one per Ω_4 mapping), i.e. $\{(?x, f), (?y, g),$
$(?y_{17}, b), (?z_{17}, c)\}$. In addition, *Perm* keeps the mapping $\{(?x, d), (?y, b),$
$(?z, c), (?x_1, d), (?y_1, b), (?y_{17}, b), (?z_{17}, c)\}$ to encode that μ_{19} was derived by joining the two compatible mappings μ_1 and μ_{17}. As a result, in the case that μ_{17} is untrusted, *Perm* can infer that μ_{19} becomes untrusted due to the appearance of $\{(?y_{17}, b), (?z_{17}, c)\}$ in the extended mapping. However, it has no way to infer that μ_{21} should become trusted and appear in the result.

Similar to *Perm*, [18] documents that μ_{20} exists in the result because μ_2 is not compatible with μ_{17}. In particular, assume that μ_1, μ_2, μ_{17} are annotated with c_1, c_2, c_3 respectively. The provenance expression that [18] computes for μ_{20} is $c_2 \wedge \neg c_3$, where $\wedge$ is an idempotent, abstract provenance operator defined for $\bowtie$ and $\neg$ is a unary provenance operator defined for $\setminus$. Like *Perm*, [18] does not encode provenance information for μ_{21}, and therefore it can not infer that μ_{21} should become trusted and appear in the result.

M-semirings [26] is a recent extension of *how*-provenance for capturing the relational minus operator. To this end, it defines an additional, third provenance operator, denoted by $\ominus$. For instance, the provenance expression that [26] computes for μ_{20} is $c_2 \ominus 0$, where c_2 is the provenance of μ_2 in Ω_1, while 0 denotes that μ_2 does not belong to Ω_4. According to $\ominus$ properties [26], $c_2 \ominus 0 = c_2$. In this manner, *m-semirings* keep even less information than *Perm* with respect to negation, since they do not even document the reason, why μ_{20} appears in the result of $\Omega_1 \overline{\bowtie} \Omega_4$. Consequently, in the case that μ_{17} is untrusted, *m-semirings* cannot infer that μ_{21} should become trusted and appear in the result (like *Perm*).

Beyond *Perm*, the remaining, relational provenance models presented in Section 3, do not consider negation. To cope with $\overline{\bowtie}$ semantics, one could consider the following naive extension. Assume for instance, that μ_1, μ_2, μ_{17} are annotated with c_1, c_2, c_3

respectively. Then, one could annotate μ_{19} with $c_1 \odot c_3$, i.e. in the same manner as if it was derived by $\Omega_1 \bowtie \Omega_4$. Additionally, μ_{20} in $\Omega_1 \overline{\bowtie} \Omega_4$ could be annotated with c_2, i.e. the same annotation as μ_2 in Ω_1 from which it was derived. Suppose now that μ_{17} is untrusted, i.e. $c_3 = false$. Then, such a naive extension of *how*-provenance suffices to infer that μ_{19} should be untrusted, because $c_1 \odot c_3$ evaluates to $true \wedge false = false$. However, it cannot infer that μ_{21} should become trusted, because μ_{21} would not even appear in the result of $\Omega_1 \overline{\bowtie} \Omega_4$.

4.3.3 Challenges for a SPARQL Provenance Model

We conclude that a new provenance model is needed to cope with the *non-monotonicity* of $\overline{\bowtie}$. In particular, provenance expressions should be computed and stored for some mappings that do not appear in the result of a query. For instance, μ_{21} does not appear in the result of $\Omega_1 \overline{\bowtie} \Omega_4$ in Table (c) of Figure 4.2, because it is compatible with μ_{17}. However, the provenance expression of μ_{21} needs to encode the fact that this mapping is not trusted due to the existence of the *compatible* and trusted mapping μ_{17} in Ω_4, but should become trusted if μ_{17} becomes untrusted.

To achieve this, provenance expressions for these additional mappings need to record information about the *compatibility* between pairs of mappings, beyond the usual expressions of provenance tokens combined with abstract operators. To record such information, we need to identify mappings that may refer to triples from the source dataset or to intermediate results of other SPARQL expressions. As explained earlier, every triple is matched to a unique mapping. Since provenance tokens can be used to identify source triples, it is reasonable to consider using provenance tokens as the basis for identifying mappings. The caveat with this approach is that multiple different mappings may be obtained from the same source triple or other mapping, through the use of projection (π) operator. Such mappings may be compatible, even if the mappings from which they were derived were not compatible. For instance,

μ_5 and μ_7 in Table 4.1 are not compatible because the variable $?x$ is bound to d in the former, while to a in the latter. However, if we project them on $?y, ?z$, then the resulting mappings are compatible. As a result, the provenance tokens of the triples from which they were derived do not suffice as mapping identifiers. Instead, some information about the subset of variables of the mapping, should also be encoded.

4.4 Provenance for Alternative SPARQL Result Constructs

In the previous paragraphs of this section we considered SPARQL queries with a SELECT clause specifying what should be returned. However, SPARQL offers additional constructs that can change the type of the query result, such as ASK, CONSTRUCT and DESCRIBE. In this paragraph, we investigate whether these constructs imply any additional requirement for a SPARQL provenance model.

In particular, an ASK query returns yes, if the pattern matching part has a non empty result. Conceptually, ASK is a projection on an empty set of variables, and consequently it can be handled as *projection*. CONSTRUCT and DESCRIBE are the only SPARQL operators whose output is a set of triples. In the case of CONSTRUCT, values from mappings are used to create one or more triples according to a pattern specified in the CONSTRUCT clause. Therefore, we annotate each triple with the provenance of the mapping that produced it. If multiple mappings result in the same triple, the triple is assigned to the sum of the provenance expressions, in the same spirit as in *projection*. In the case of DESCRIBE, for every valuation of a query variable, the output contains all the triples that have that value as a subject, predicate or object. Conceptually, DESCRIBE can be expressed as the output of three CONSTRUCT queries. Therefore, DESCRIBE can be handled similarly to

CONSTRUCT, with the difference that triple with multiple derivations should be annotated with the sum of the provenance of every derivation.

Chapter 5

Conclusions

Unlike previous surveys [12, 6, 16, 21], in this paper we focused on data provenance models for RDF data, and more specifically on how implicit provenance information of SPARQL query results can be used to compute on demand annotations reflecting various dimensions of data quality. Our work is motivated by the significance of maintaining provenance information for a particular instance of RDF data, i.e. Linked Data, which is typically fetched from multiple sources, integrated and then materialized in multiple repositories. In particular, 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 $\overline{\supset}\bowtie$ operator, that implicitly introduces negation.

We are currently working on the formalization of an abstract provenance model for full SPARQL. We pay particular attention to the tradeoff between expressiveness and efficiency of abstract provenance models for Linked Data. Due to the complexity of dealing with the non-monotonicity of negation, a provenance model for the full expressive power of SPARQL is expected to be more demanding than relational ones in terms of storage space or annotation computation time. Therefore, the exploration

of a potential hierarchy of SPARQL provenance models of increasing expressiveness is important. In addition, we plan to explore previous work on relational provenance models enhanced with schema mappings [36, 37] in order to capture provenance of various computations considered by web programs such as mashups. In this direction, we also plan to explore pioneering work on a unified model of workflow and data provenance such as proposed in [1]. Finally, we plan to investigate the challenges that are posed for a provenance model, in order to capture *recursive* computations, such as those implied by RDFS [45, 19] or OWL [55] inference rules, or those that can be expressed by *Proof Markup Language* [15].

Bibliography

[1] U. Acar, P. Buneman, J. Cheney, J. V. den Bussche, N. Kwasnikowska, and S. Vansummeren, *A Graph Model of Data and Workflow Provenance*, TAPP, 2010.

[2] O. Hartig, *Querying Trust in RDF Data with tSPARQL*, ESWC, 2009.

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

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

[5] C. Bizer, T. Heath, and T. Berners-Lee, *Linked data - the Story so far*, IJSWIS (2009).

[6] Rajendra Bose and James Frew, *Lineage Retrieval for Scientific Data Processing: a Survey*, ACM Computing Surveys **37** (2005), no. 1, 1–28.

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

[8] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, *Curated Databases*, PODS, 2008.

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

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

[11] J. J. Carroll, C. Bizer, P. J. Hayes, and P. Stickler, *Named Graphs*, J. Web Semantics **3** (2005), no. 4.

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

[13] Laura Chiticariu and Wang-Chiew Tan, *Debugging schema mappings with routes*, VLDB, 2006.

[14] Yingwei Cui and Jennifer Widom, *Lineage tracing for general data warehouse transformations*, VLDB, 2001.

[15] P. P. da Silva, D. L. McGuinness, and R. Fikes, *A Proof Markup Language for Semantic Web Services*, Information Systems **31** (2006), no. 4.

[16] Susan B. Davidson and Juliana Freire, *Provenance and scientific workflows: challenges and opportunities*, SIGMOD, 2008.

[17] L. Ding, T. Finin, Y. Pen, P. Pinheiro da Silva, and D. L. McGuinness, *Tracking RDF Graph Provenance using RDF Molecules*, ISWC, 2005.

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

[19] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides, *Coloring RDF Triples to Capture Provenance*, ISWC, 2009.

[20] J. Nathan Foster, Todd J. Green, and Val Tannen, *Annotated XML: queries and provenance*, PODS, 2008.

[21] J. Freire, D. Koop, E. Santos, and C. T. Silva, *Provenance for Computational Tasks: A Survey*, Computing in Science and Engg. **10** (2008), no. 3.

[22] J. Frey, D. D. Roure, K. Taylor, J. Essex, H. Mills, and E. Zaluska, *Combechem: A case study in provenance and annotation using the semantic web*, IPAW, 2006.

[23] N. Fuhr and T. Rölleke, *A probabilistic relational algebra for the integration of information retrieval and database systems*, ACM Transactions on Information Systems **14** (1997), no. 1.

[24] F. Geerts, J. V. den Bussche, M. Arenas, and C. Gutierrez, *Completeness of Query Languages for Annotated Databases*, Journal of Computer and System Sciences (2010).

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

[26] F. Geerts and A. Poggi, *On Database Query Languages for K-Relations*, Journal of Applied Logic **8** (2010), no. 2.

[27] Boris Glavic and Gustavo Alonso, *Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting*, ICDE, 2009.

[28] Jennifer Golbeck and James Hendler, *A Semantic Web Approach to the Provenance Challenge*, Concurrency and Computation: Practice and Experience **20** (2008), no. 5.

[29] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen, *Update exchange with mappings and provenance*, VLDB, 2007, Amended version available as Univ. of Pennsylvania report MS-CIS-07-26.

[30] Todd J. Green, Grigoris Karvounarakis, and Val Tannen, *Provenance semirings*, PODS, 2007.

[31] O. Hartig, *Specification for tSPARQL*, `http://trdf.sourceforge.net/documents/tsparql.pdf`.

[32] O. Hartig, *Provenance Information in the Web of Data*, Linked Data on the Web Workshop at WWW, 2009.

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

[34] Tomasz Imielinski and Witold Lipski, *Incomplete information in relational databases*, JACM **31** (1984), no. 4.

[35] Zachary G. Ives, Todd J. Green, Grigoris Karvounarakis, Nicholas E. Taylor, Val Tannen, Partha Pratim Talukdar, Marie Jacob, and Fernando Pereira, *The* ORCHESTRA *collaborative data sharing system*, SIGMOD Rec. (2008).

[36] G. Karvounarakis, Z. G. Ives, and V. Tannen, *Querying Data Provenance*, SIGMOD, 2010, to appear.

[37] Grigoris Karvounarakis, *Provenance for collaborative data sharing*, Ph.D. thesis, University of Pennsylvania, July 2009.

[38] Grigoris Karvounarakis and Zachary G. Ives, *Bidirectional mappings for data and update exchange*, WebDB, 2008.

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

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

[41] Inderpal Singh Mumick and Oded Shmueli, *Finiteness properties of database queries*, Fourth Australian Database Conference, February 1993.

[42] C. Mungall, *A SPARQL endpoint for a database of annotated gene expression*, `http://www.bioontology.org/wiki/index.php/OBD:SPARQL-InSitu`.

[43] J. W. Murdock, D. McGuinness, P. P. da Silva, C. Welty, and D. Ferrucci, *Explaining Conclusions from Diverse Knowledge Sources*, ISWC, 2006.

[44] *Open provenance model*, http://twiki.ipaw.info/bin/view/Challenge/OPM, 2008.

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

[46] J. Pérez, M. Arenas, and C. Gutierrez, *Semantics and complexity of SPARQL*, ACM Trans. Database Systems **34** (2009), no. 3.

[47] D. Le Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni, *Rapid prototyping of semantic mash-ups through semantic web pipes*, WWW, 2009.

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

[49] S. S. Sahoo, R. S. Barga, A. P. Sheth, K. Thirunarayan, and P. Hitzler, *PrOM: A Semantic Web Framework for Provenance Management in Science*, Technical Report, Kno.e.sis Center, 2009.

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

[51] N. Shadbolt, T. Berners-Lee, and W. Hall, *The Semantic Web Revisited*, IEEE Intelligent Systems (2006).

[52] Nikhil Swamy, Brian J. Corcoran, and Michael Hicks, *Fable: A language for enforcing user-defined security policies*, S&P, May 2008.

[53] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary G. Ives, Fernando Pereira, and Sudipto Guha, *Learning to create data-integrating queries*, VLDB, 2008.

[54] T. Tran, Haofen Wang, S. Rudolph, and P. Cimiano, *Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data*, ICDE, 2009.

[55] O. Udrea, D. R. Recupero, and V. S. Subrahmanian, *Annotated RDF*, ACM Trans. Comput. Logic **11** (2010), no. 2.

[56] *W3C Provenance Incubator Group*, `http://www.w3.org/2005/-Incubator/-prov/wiki/W3C_Provenance_Incubator-_Group_Wiki`.

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

[58] J. Zhao, C. Wroe, Goble C, R. Stevens, D. Quan, and M. Greenwood, *Using Semantic Web Technologies for Representing e-Science Provenance*, ISWC, 2004.