

Management of Meta Knowledge for RDF Repositories

Bernhard Schueler, Sergej Sizov, Steffen Staab

University of Koblenz-Landau, Computer Science
ISWeb - Information Systems and Semantic Web
Koblenz, Germany
{bernie,sizov,staab}@uni-koblenz.de

Abstract

The integration of knowledge from heterogeneous information sources and applications does not only require the conceptual mapping of information structures, but it also requires the treatment of semantic meta knowledge (i.e. knowledge about knowledge) in a generic manner. We describe a generic mechanism for (i) modeling and (ii) querying semantic meta knowledge in the context of RDF repositories. We substantiate our approach by use cases from a project about multi-modal information integration from different information sources.

1 Introduction

1.1 Motivation

Meta data and meta knowledge is the description of additional information about pieces of data stored in a database or knowledge base. Typical representatives of meta knowledge are provenance (the description of the origins of a piece of data and the process by which it arrived in a database), reliability (e.g. accuracy of the algorithm that produced the given record), or timestamp (e.g. timepoint of the last update of the record). Understanding and utilization of meta knowledge is a complex issue that includes a) general methodology of meta knowledge organization and b) domain-specific customization and interpretation of particular dimensions.

As a motivation example, we consider the analysis of complex technical problems using various data sources in large-scale corporate networks. Usually, the relevant pieces of problem-related information are scattered across several systems, data repositories, and media types (e.g. textual documents, photographic images and X-ray images of dam-

aged parts, raw sensor data, results of laboratory tests, etc.). Consequently, the systematic problem analysis requires expert knowledge and experience in different areas of the product development and maintenance. Furthermore, the complex problem analysis typically requires a flexible and integrated view on all aspects of the product development and maintenance. In doing so, a variety of different document types can be considered:

- problem reports (Word documents) containing text, images, 2D plots and tables,
- spreadsheets containing tables and images,
- Powerpoint slides with short problem descriptions, containing text, images, tables,
- numerical measurements capturing temperature, noise and vibration of jet engines.

In our application scenario the key interest of the users is to retrieve and integrate data and knowledge across teams and departments of a company. To this reason, we assume that information from particular documents/collections is automatically extracted (using corresponding media-specific knowledge extraction algorithms) and shared via semantic knowledge bases (e.g. as a large-scale collection of facts represented as RDF triples and stored in an RDF repository). Particular applications, such as the problem analysis/resolution framework, retrieve relevant information from the knowledge base using common querying mechanisms, e.g. the SPARQL query language (Figure 1). To select relevant facts and correctly interpret them, it is necessary for the engineer to clearly understand when and how the fact was compiled from which sources, with which reliability and by what extraction algorithm.

In order to provide advanced representation/ranking/browsing capabilities, applications may

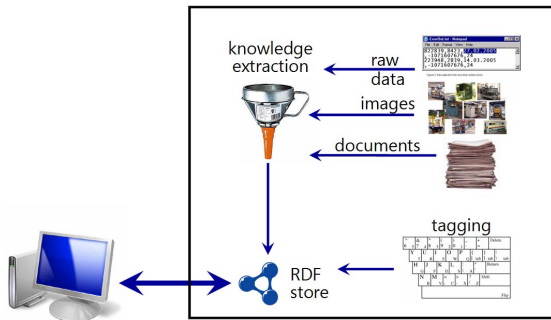


Figure 1. General Application Scenario

exploit various kinds of *meta knowledge* (i.e. knowledge about knowledge pieces stored in the repository), such as:

- storage location of extraction sources (e.g. document URI, segment within the document)
- author of the document that was used for extracting facts
- manually added annotations (tagging) to source documents
- timestamps of document creation and knowledge extraction
- estimated accuracy of automatic knowledge extraction method (i.e. certainty/reliability of extracted facts)

Besides meta knowledge we assume that the retrieval process is supported by necessary domain-specific ontologies (e.g. representing relationships between products and their components, organizational structure of the corporation, and the like).

1.2 Sample Scenario

Our hypothetical application scenario aims to analyze possible causes of damage of the aircraft jet engine. We assume that the damaged part (e.g. jet blades with *burned edges*) has been observed and reported during the maintenance of the engine type *T5678*. Potentially, the observed problem may have a number of possible indications, including *overheating*, *vibration*, or *wrong airflow*. The goal of the problem resolution team is to identify problems with similar appearance with engines of type *T5678* reported in the past and to establish the workplan for repair.

Using the TriG syntax for Named Graphs (see section 2.2), meta knowledge for our application example can be expressed as shown in Figure 2. This example contains four graphs (groups of statements). The statements within each graph are enclosed by brackets. The

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
@baseURI http://www.problem.com/sampleDocument

:G1 {
  T5678 hasProblem overheating.
  overheating causedBy vibration.
}

:G2 {
  T5678 hasProblem burnedEdges.
  burnedEdges causesProblem wrongAirflow.
  overheating causesProblem vibration
}

:G3 {
  :G1 source <http://example/report01.doc> .
  :G1 agent Bob.
  :G1 extractor textAnalyzer.
  :G1 certainty "0.9".
  :G1 timestamp "5/5/2007"
}

:G4 {
  :G2 source <http://example/image01.jpg> .
  :G2 agent Mary.
  :G2 extractor imageAnalyzer.
  :G2 certainty "0.6" .
  :G2 timestamp "6/6/2006"
}

```

Figure 2. Meta Knowledge: Example with Named Graphs

first graph in the example named `http://www.example.org/sampleDocument\#G1` contains facts automatically extracted from a problem resolution report (e.g. a WinWord document). The second graph contains facts automatically extracted by image analysis tools from pictures of the damaged part. The third and fourth graphs contain meta knowledge about statements within G1 and G2: location of source documents, name of document creator, media-specific algorithm that extracted facts from these documents, result certainty, and timestamps.

The SPARQL query language for RDF allows querying across multiple named graphs. To this reason, the problem resolution application can directly exploit meta knowledge in a variety of ways. For instance, it can restrict the scope by explicitly selecting sources of facts (e.g. `source = "http://example/report01.doc"`), or by specifying the required degree of reliability (e.g. `certainty > 0.8`).

In fact, the majority of recent proposals on meta knowledge management have been concerned by designing methods of direct meta knowledge exploitation for particular applications, i.e. capturing, representing, and querying specific dimensions of meta information. Such application-driven exploitation of meta knowledge has been shown very beneficial for a variety of scenarios, e.g. addressing the following issues:

1. how to capture provenance and/or trust information

2. how to use provenance to evaluate trustworthiness of a semantic association, or
3. how to use provenance knowledge for pruning search on the Semantic Web

However, isolated applications typically do not provide the general notion of meta knowledge associated with results of an arbitrary query. The lack of generalization mechanisms makes the re-use in new application domains all but impossible. For instance, the problem resolution framework of a *car manufacturer* that uses different evidence collection mechanisms (e.g. custom human annotations instead of automatically acquired facts, with completely different meta attributes) would require re-designing queries related to meta knowledge on the application level.

1.3 Our Objective

Our proposal aims to abstract from particular application domains and to generalize the notion of meta knowledge for arbitrary queries and results in the knowledge management framework. In doing so, we introduce the generalized notion of meta knowledge and its representation in a knowledge base (RDF repository). Furthermore, we analyze the formal relational model of RDF querying (using the SPARQL query language as an example) and identify relational algebra calculations that are necessary to acquire meta information for an arbitrary query result.

The remainder of this paper is organized as follows. Section 2 introduces technical basics of representing, querying, and organizing meta knowledge. Section 3 introduces the formal model of meta knowledge and its aggregation for annotating results of arbitrary queries to the knowledge base. Section 4 concludes the proposal and gives an outlook on future work directions.

2 Technical Basics

2.1 Provenance

Provenance is one of the major dimensions of meta knowledge. In the recent literature, the issue of provenance has been studied in the context of digital libraries, database systems, and knowledge management applications. In the database community, the term *provenance* makes usually reference to the description of the origins of a piece of data and of the process by which it arrives in a database [2, 3]. In the Semantic Web field provenance has been recently considered in applications for assessing the trustworthiness of information [4, 6].

By examining provenance in a general setting, we adopt the established terminology [5, 3, 6] and draw a distinction

between where-provenance, why-provenance, and how-provenance:

1. *where* - where the given fact/statement is physically serialized in one or more RDF statements ('where does a given piece of data come from')
2. *why* - the collection of facts/statements that contributed to produce the query answer, e.g. a composed statement ('which facts contributed to this answer')
3. *how* - how the query result was produced ('how facts contributed to the answer')

2.2 Reification

The issue of reification (i.e. the possibility of making assertions about statements) has been extensively discussed in the recent literature. A number of approaches, including Triple [14] and ConceptBase [10], have been proposed to address the reification problem in the A-box. This paper adopts the notion of general and simple variation on RDF, called Named RDF Graphs [4]. A Named Graph is an RDF graph which is assigned a name in the form of a URIref. The name of a graph may occur either in the graph itself, in other graphs, or not at all. Graphs may share URIrefs but not blank nodes. To avoid unnecessary triple bloat, recent formalisms for reification are based on quads that consist of an RDF triple and a further URIref or blank node or ID. The fourth element is used to refer to information sources, to model IDs or statement IDs, or more generally to "contexts". Named Graphs can be seen as a reformulation of quads in which the fourth elements distinct syntactic and semantic properties are clearly distinguished, and the relationship to RDFs triples, abstract syntax and semantics is clearly defined.

2.3 Annotated Relations

In the relational data model, meta data is usually considered from the point of view of *annotated relations*. Annotated relations are relations of a database where each tuple is annotated with an element of a set K . An example for K is the set of Boolean formulas build from the set containing all tuple identifiers and a distinct element NULL ($\mathbf{0}$). Tuples which are not in a relation are associated with $\mathbf{0}$. Regarding the identifiers as Boolean variables, $\mathbf{0}$ is assigned the value 0 and all other variables the value 1. Formally, an annotated relation is a function $R : Tup(A) \rightarrow K$, where $Tup(A)$ signifies the set of all tuples over a set of attributes A .

With this kind of annotated relations we can calculate the results of relational algebra operations as following: the output relation is an annotated relation as well and each tuple is annotated with a Boolean formula built from the annotations of the corresponding input tuples. The evaluation

$$\begin{array}{ccc}
Rel_1 = \begin{array}{|c|c|c|c|} \hline \text{id} & A & B & C \\ \hline t_1 & a & b & c \\ \hline \end{array}, & Rel_2 = \begin{array}{|c|c|c|} \hline \text{id} & C & D \\ \hline t_2 & c & d \\ \hline \end{array} & \\
\{(a, b, c, d)\} & \{(a, b, c, d)[t_1 \wedge t_2], \\ & (a, b, c, e)[\mathbf{0}], \dots\} & \\
\uparrow & \iff & \uparrow \\
\{(a, b, c)\} \bowtie \{(c, d)\} & & \{(a, b, c)[t_1], \dots\} \bowtie \{(c, d)[t_2], \dots\}
\end{array}$$

Table 1. Evaluation using annotations.

of the Boolean formula returns 1 if the a tuple is in the derived relation and 0 if not. The key question is how to build Boolean formulas in such way that they represent the operations of relational algebra. Consider the following definition of natural join and Figure 1:

natural join $R_3 = R_1 \bowtie R_2$ is defined by

$$R_3(t) = R_1(t[\text{dom}(R_1)]) \wedge R_2(t[\text{dom}(R_2)]),$$

where $\text{dom}(R_3) = \text{dom}(R_1) \cup \text{dom}(R_2)$.

The join of Rel_1 and Rel_2 is performed in two different ways. Either by the traditional definition of join or by using annotations which are build using tuple identifiers. Using the traditional definitions relations contain only those tuples which are in a relation. Using annotations the tuples which are not in a relation, such as (a, b, c, e) , are mapped to $\mathbf{0}$. They need not (and cannot, given infinite domains) be represented explicitly. Consider selection with annotated relations as another example:

selection $R_2 = \sigma_{c(x)}(R_1)$, is defined by

$$R_2(t) = R_1(t) \wedge c(t),$$

where $c(x)$ is a function mapping tuples x to $\{0,1\}$.

Regarding probabilistic interpretation of annotations complete definitions for all RA operations can be found in [7, 9]. We are especially interested in using Boolean formulas as abstract representations which subsume the evaluation according to different semantics.

In [9] Imielinsky and Lipski present *c-tables* which are used to represent Boolean conditions on tuples. In [7] Fuhr and Rölleke present a probabilistic evaluation of Boolean formulas which they call *event expressions*. In their model tuple identifiers are not Boolean variables but independent binary random variables. If one identifier has the value 1 this represents the event consisting of the union of all states of the database where this tuple is in the database. An identifier is a mapping to $[0,1]$ rather than a variable mapped to $\{0,1\}$. The authors show that the probabilistic relational algebra which they present has ordinary relational algebra

as special case. We can conclude that once the Boolean algebra expressions have been constructed for the tuples of a derived relation different interpretations are possible.

For the algebra RA+ which is the relational algebra without set minus an abstraction for annotations of tuples is presented in [8]. This abstraction subsumes other expressions beyond Boolean formulas by requiring the annotation expressions to form a commutative semiring $(K, +, \cdot, 0, 1)$. Hence $+$ and \cdot are commutative binary operations and $(K, +, 0)$ as well as $(K, \cdot, 1)$ are commutative monoids. Further, \cdot is distributive over $+$ and $\forall a, 0 \cdot a = a \cdot 0 = 0$.

This algebraic structure subsumes Boolean algebra without negation, $(\{0, 1\}, \vee, \wedge, 0, 1)$, as well as an algebra on positive Boolean formulas over a set of variables B themselves, $(\text{PosBool}(B), \vee, \wedge, (0), (1))$, as well as the algebra of probabilistic events Ω without complement, $(2^\Omega, \cup, \cap, \emptyset, \Omega)$. In order to permit simplifications of Boolean formulas $\text{PosBool}(B)$ has to be defined in such way that it only contains one representative for equivalent Boolean formulas. For RA+ the semantics of the annotation expressions just mentioned are subsumed. Beyond these semantics commutative semirings also subsume bag semantics, i.e. each tuple can occur several times and the annotation of a tuple contains the information how many times it occurs. For this semantics the definition of set minus poses a problem since there should not be negative amounts of tuples.

Given that RA+ operations are evaluated using annotated relations with annotations forming a commutative semiring some but not all identities of RA+ expressions hold according to [8]. This means the values of the annotation are identical for tuples resulting from equivalent expressions in RA. Identities which do not hold are idempotence of union and (self-)join. As pointed out in [7] all equivalences from RA hold for evaluation via Boolean formulas as annotations.

As general representation of annotations the authors of [8] present the semiring of polynomials (without minus) with variables ranging over tuples and coefficients from \mathbb{N} . The authors show that for any semiring defined on a set K the semantics of RA+ queries can be reconstructed using this semiring of polynomials. For our purposes in this paper we will stick to Boolean formulas as annotations since we need a form of set minus as we will point out below.

2.4 SPARQL Semantics

In the remainder of this paper, we assume that the knowledge base is accessed by SPARQL queries [13]. The algebraic formalization of this query language is provided by [11, 12]. The formalization is based on set-theoretic operations for sets of variable assignments which are partial functions $\mu : V \rightarrow T$. Here V is a set of variables and T is the set consisting of IRIs, blank nodes and literals used in RDF tripels.

For such assignments the authors define the notion of compatibility as following: two mappings μ_1 and μ_2 are *compatible* if for all $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ holds $\mu_1(x) = \mu_2(x)$. For sets of assignments join, union, difference and left outer join are defined as following [11, 12]:

Definition 2.1 *Let Ω_1 and Ω_2 be sets of assignments then*

$$\begin{aligned} \Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are compatible mappings}\} \\ \Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\} \\ \Omega_1 \setminus \Omega_2 &= \{\mu \in \Omega_1 \mid \forall \mu' \in \Omega_2 : \mu \text{ and } \mu' \text{ are not compatible}\} \\ \Omega_1 \rhd \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2) \end{aligned}$$

These operations on variable assignments are used to define the evaluation of complex graph pattern stated using SPARQL based on the evaluation of basic graph patterns. We consider a single triple pattern to be a basic graph pattern. Its evaluation over an RDF dataset D is defined as set of variable assignments. For basic graph pattern and complex pattern build using AND, OPT, UNION and FILTER from the SPARQL language the semantics according to [11] are summarized in the following definition:

Definition 2.2 *Let D be an RDF dataset over T , t a triple pattern and P_1, P_2 graph patterns. The evaluation of a graph pattern over D , denoted by $\llbracket \cdot \rrbracket_D$, is defined recursively:*

- $\llbracket t \rrbracket_D = \{\mu : V \rightarrow T \mid \text{dom}(\mu) = \text{var}(t) \wedge \text{trip}(\mu) \in D\}$, where $\text{var}(t)$ is the set of variables occurring in t and $\text{trip}(\mu)$ the triple obtained by replacing the variables in t according to μ ,
- $\llbracket P_1 \text{ AND } P_2 \rrbracket_D = \llbracket P_1 \rrbracket_D \bowtie \llbracket P_2 \rrbracket_D$,
- $\llbracket P_1 \text{ OPT } P_2 \rrbracket_D = \llbracket P_1 \rrbracket_D \rhd \llbracket P_2 \rrbracket_D$,
- $\llbracket P_1 \text{ UNION } P_2 \rrbracket_D = \llbracket P_1 \rrbracket_D \cup \llbracket P_2 \rrbracket_D$,
- $\llbracket P_1 \text{ FILTER } R \rrbracket_D = \{\mu \in \llbracket P_1 \rrbracket_D \mid \mu \models R\}$, where R is a built-in condition and $\mu \models R$ denotes that μ satisfies R .

Examples for built-in conditions are $?X=c$, $?X=?Y$ and $\text{isIRI}(?X)$. Constraints are evaluated according to a three-valued logic with the truth values *true*, *false* and *error*. For more details see [11, 12, 13].

In [11] associativity and commutativity of AND and UNION is proved along with other algebraic properties of complex graph pattern. In [12] the definitions are extended to account for the *named graphs* extension of SPARQL [13].

3 Meta Knowledge

In the first part of this section, we introduce the general notion of *annotated tuples* and construct Boolean formulas for annotating arbitrary query results. Next, we adopt the model of annotated items to RDF variable assignments. Finally, we discuss how these annotations can be interpreted with arbitrary domains of meta knowledge in order to construct the meta annotation for a query answer.

3.1 Representational Framework

We assume that meta knowledge is represented by m -dimensional data records $d_j \in D \subseteq \text{dom}(A_1) \times \dots \times \text{dom}(A_m)$ with attributes A_i that correspond to some arbitrary domains $\text{dom}(A_i)$ (dimensions of meta knowledge, e.g. agent, timestamp, source, extraction source URI, etc.). Each $\text{dom}(A_i)$ is extended by a special symbol \otimes_i which is interpreted as the absence of the corresponding information for a given statement. Apart from this requirement, application-specific definition of domains $\text{dom}(A_i)$ is not restricted in any way.

For the sake of simplicity, we abstract from concrete representation formalisms (e.g. Named Graphs) and particular implementational issues and assume that

- each triple j in the RDF repository is associated with its unique record of meta data $d_j \in D$.
- for each triple j , the query processor can internally access the associated record of meta data d_j (e.g. by querying corresponding named graphs that represent meta knowledge associated with j)
- meta knowledge of a meta knowledge statement is an empty m -dimensional record $(\otimes_1, \dots, \otimes_m)$. In other words, the special case of *nested* reification ('meta knowledge on meta knowledge') is not supported

3.1.1 Annotated RDF variable assignments

Similarly to the evaluation of RA operations on annotated relations presented above we define the evaluation of SPARQL queries on RDF graphs via annotations of variable assignments for graph patterns. We use annotations from the set of Boolean formulas built from the set containing an identifier for each triple and a distinct element $\mathbf{0}$ (NULL). The identifiers can be regarded as Boolean variables. Triples which are not in the RDF dataset are associated with $\mathbf{0}$. $\mathbf{0}$ is assigned the value 0 and all triple identifiers are assigned the value 1.

A set of variable assignments $\mu : V \rightarrow T$ can be represented by a relation where the variables are the attributes and assignments the tuples of this relation. Therefore an

```

CONSTRUCT (?Y causeOf ?Z)
WHERE
{
  {
    {?X hasProblem ?Y .} AND
    {?Y causesProblem ?Z .}
  }
  FILTER {?X = "T5678"}
}

```

subject	predicate	object	A
T5678	hasProblem	burnedEdges	t1
overheating	causedBy	vibration	t2
burnedEdges	causesProblem	wrongAirflow	t3
overheating	causesProblem	vibration	t4
T5678	hasProblem	overheating	t5

Table 2. A query and annotated triples.

annotation of such a set of assignments can be regarded as function $\Omega : \text{Trip}(T^{|V|}) \rightarrow K$, where $T^{|V|}$ is the set of all tuples of the length $|V|$ over the domain T .

According to [11, 12] a single triple pattern can be used as basic graph pattern. We want to annotate triples with identifiers and propagate these identifiers to the sets of assignments for triple patterns. This is necessary e.g. to specify probabilities for single triples or to track the provenance of a query result back to the triples in the RDF dataset. Thus, we define the evaluation of a single triple pattern as following:

Definition 3.1 Let D be an RDF dataset and t a triple pattern. The evaluation of t over D , is defined by the annotated relation Ω , $\text{dom}(\Omega) = \text{var}(t)$,

$$\Omega(\mu) = \begin{cases} \text{id}(\text{trip}(\mu)) & \text{if } \text{trip}(\mu) \in D, \\ \mathbf{0} & \text{else} \end{cases}$$

where $\text{trip}(\mu)$ is the triple obtained by replacing the variables in t according to μ and $\text{id}(x)$ is the identifier of triple x .

As an example, table 2 shows a SPARQL query and a set of annotated triples according to our application scenario introduced in Section 1: finding problems (and their causes) related to the jet engine type T5678. The triples which are not in the dataset and thus annotated with $\mathbf{0}$ in our model are left out. In fact, they cannot be represented given there is an infinite domain for one of the attributes.

The evaluations of the two triple patterns $?X$ hasProblem $?Y$ and $?X$ causesProblem $?Z$ are depicted in the upper part of table 3. The evaluation of $?X$ hasProblem $?Y$ contains two variable assignments

$$T_1 =$$

?X	?Y	A
T5678	burnedEdges	t1
T5678	overheating	t5
x	burnedEdges	$\mathbf{0}$

$$T_2 =$$

?Y	?Z	A
burnedEdges	wrongAirflow	t3
overheating	vibration	t4
burnedEdges	z	$\mathbf{0}$

$$T_1 \bowtie T_2 =$$

?X	?Y	?Z	A
T5678	burnedEdges	wrongAirflow	t1 \wedge t3
T5678	overheating	vibration	t5 \wedge t4
x	burnedEdges	wrongAirflow	$\mathbf{0} \wedge \text{t3}$
T5678	burnedEdges	z	t1 \wedge $\mathbf{0}$

Table 3. Evaluation of AND.

which are based on the triples t1 and t5, respectively. For $?Y$ causesProblem $?Z$ there are two variable assignments as well. One is based on triple t3 and the other one on triple t4.

In definition 2.2 the semantics of the key constructs of the SPARQL language [13] have been defined. In this recursive definition the semantics are mostly based on operations defined on sets of variable assignments as defined in definition 2.1. Most of these operations have a direct equivalent in relational algebra. Therefore we provide an alternative definition of these operations based on annotated variable assignments.

Definition 3.2 Let Ω_1, Ω_2 be sets of annotated variable assignments. We define \bowtie, \cup, \setminus and \Rightarrow via operations on the annotations of the assignments as following:

- $(\Omega_1 \bowtie \Omega_2)(\mu) = \Omega_1(\mu_1) \wedge \Omega_2(\mu_2)$, where μ_1 and μ_2 are compatible and $\mu = \mu_1 \cup \mu_2$,
- $(\Omega_1 \cup \Omega_2)(\mu) = \Omega_1(\mu) \vee \Omega_2(\mu)$,
- $(\Omega_1 \setminus \Omega_2)(\mu) = \Omega_1(\mu) \wedge \neg \left(\bigvee_{\mu_2, \Omega_2(\mu_2) \neq \mathbf{0}} \Omega_2(\mu_2) \right)$, where μ_2 is compatible to μ ,
- $(\Omega_1 \Rightarrow \Omega_2)(\mu) = (\Omega_1 \bowtie \Omega_2)(\mu) \vee (\Omega_1 \setminus \Omega_2)(\mu)$.

As a consequence the semantics of the language parts defined in definition 2.2, except FILTER expressions, can be based on definition 3.2 instead of using definition 2.1. To illustrate this we continue the example from above:

For the evaluation of AND from the query in table 2 we have to calculate the join of the two relations in the upper part of table 3 according to definition 3.2. The resulting set

?X	?Y	?Z	A
T5678	burned Edges	wrong Airflow	$(t1 \wedge t3) \wedge R_{X=T5678}$ (T5678, burnedEdges, wrongAirflow)
T5678	vibra- tion	over- heating	$(t5 \wedge t4) \wedge R_{X=T5678}$ (T5678, vibration, overheating)

Table 4. Evaluation of FILTER.

of variable assignments is shown in the lower part of table 3. This relation contains assignments annotated with $t1 \wedge t3$ and $t5 \wedge t4$. Whether these annotations of derived variable assignments really identify the tuples which they annotate is an interesting question. In our opinion this question can be answered by looking at equivalences among SPARQL queries. For RA it has been shown in [7] that all equivalences from RA can be preserved evaluating RA expressions via Boolean expressions as annotations.

According to our model triples which are not in the RDF dataset and assignments which are not possible are annotated by $\mathbf{0}$. They need not (and cannot) be represented explicitly. The property of abstract annotations to subsume different semantics is illustrated by additional tuples which are shown in italics at the bottom of T_1 , T_2 , and $T_1 \bowtie T_2$ in Table 3. These 'fictional' tuples and variable assignments are not 'physically' contained in our relations. The fictional tuple $(x, \textit{burnedEdges}, \textit{wrongAirflow})$ in the evaluation of AND is based on the fictional tuple $(x, \textit{burnedEdges})$ in T_1 and the tuple $(\textit{burnedEdges}, z)$ in T_2 in Figure 3. The tuple $(x, \textit{burnedEdges}, \textit{wrongAirflow})$ is annotated with a formula which evaluates to 0 in Boolean algebra (since $\mathbf{0}$ is mapped to 0) and to $\mathbf{0}$ in the algebra of Boolean formulas. For the latter $\mathbf{0} \wedge x = x \wedge \mathbf{0} = \mathbf{0}$ holds by definition. As an explanation consider that $(\textit{PosBool}(B), \vee, \wedge, \mathbf{0}, \mathbf{1})$ forms a semiring for which this property holds and the above formula is contained in this semiring.

For FILTER expressions we propose the following definition:

Definition 3.3

$$(\Omega \text{ FILTER } R)(\mu) = \Omega(\mu) \wedge R_c(\mu),$$

where $R_c(\mu)$ denotes a function mapping μ to either 0 or 1 according the condition c .

In order to complete the evaluation of the SPARQL query from table 2 we need to apply the filtering condition to the variable assignment resulting from the join shown in table 3. Both variables of the result satisfy the condition. Thus, they are contained in the result which is listed in table 4. The Boolean formulas used as annotations are constructed according to definition 3.3

Similar to the probabilistic evaluation of annotations for relational algebra the annotations for variable assignments

for graph patterns can be used for a probabilistic evaluation of SPARQL queries. Given that triples are assigned a probability the annotation expressions presented here can be evaluated according to the evaluation of *event expressions* presented in [7]. Furthermore, the framework can also be used for uncertainty reasoning using fuzzy logic, if fuzzy logic operations for \vee , \wedge and \neg are applied to evaluate the annotations. Indeed, a Boolean expression build from triple identifiers as defined here does not only carry the information *which* triples have contributed to a variable assignment (why-provenance) but also *how* they contributed (how-provenance).

Conceptually, the method discussed so far would separately generate records of meta information d_j for each particular statement j in the result set. This degree of granularity is potentially too high for realistic applications that are rather interested in aggregated meta knowledge about the result set as a whole. To address this problem, we introduce for the group of annotated variable assignments Ω an additional *grouping* operator $\tau(\Omega)$ that combines symbolic Boolean expressions associated with particular variable assignments using the \vee operation.

3.2 Usage

Using annotations of variable assignments presented in Section 3, meta knowledge can be interpreted in order to construct the meta annotation for a query answer. By defining custom (possibly different) interpretations for algebraic operations \vee , \wedge , and \neg on particular dimensions $\text{dom}(A_1) \dots \text{dom}(A_m)$ of meta knowledge, we may obtain the m -dimensional record for the query result. Our application-specific custom definitions for the sample scenario (Section 1) can be defined as follows:

$$\begin{aligned}
\text{source}(A \wedge B) &= \text{source}(A) \cup \text{source}(B) \\
\text{source}(A \vee B) &= \text{source}(A) \cup \text{source}(B) \\
\text{source}(\neg A) &= \cup_{B \neq A} \text{source}(B) \\
\text{agent}(A \wedge B) &= \text{agent}(A) \cup \text{agent}(B) \\
\text{agent}(A \vee B) &= \text{agent}(A) \cup \text{agent}(B) \\
\text{agent}(\neg A) &= \cup_{B \neq A} \text{agent}(B) \\
\text{extractor}(A \wedge B) &= \text{extractor}(A) \cup \text{extractor}(B) \\
\text{extractor}(A \vee B) &= \text{extractor}(A) \cup \text{extractor}(B) \\
\text{extractor}(\neg A) &= \cup_{B \neq A} \text{extractor}(B) \\
\text{certainty}(A \wedge B) &= \text{certainty}(A) \cdot \text{certainty}(B) \\
\text{certainty}(A \vee B) &= \text{certainty}(A) + \text{certainty}(B) \\
&\quad - \text{certainty}(A \wedge B) \\
\text{certainty}(\neg A) &= 1 - \text{certainty}(A) \\
\text{timestamp}(A \wedge B) &= \min(\text{timestamp}(A), \text{timestamp}(B)) \\
\text{timestamp}(A \vee B) &= \min(\text{timestamp}(A), \text{timestamp}(B)) \\
\text{timestamp}(\neg A) &= \min_{B \neq A} \text{timestamp}(B)
\end{aligned}$$

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
@baseURI http://www.problem.com/sampleQueryAnswer
```

```
:G11 {burnedEdges causeOf wrongAirflow
      overheating causeOf vibration }
:G11Meta {
  :G11 source <http://example/report01.doc> ,
            <http://example/image01.jpg> .
  :G11 agent Bob , Mary .
  :G11 extractor textAnalyzer , imageAnalyzer .
  :G11 certainty "0.71" .
  :G11 timestamp "6/6/2006" .
  :G11 how-provenance "t1 ^ t3 v t5 ^ t4" }
```

Figure 3. Query Results

Note that since the set of triple identifiers is finite all shown operations including negation (e.g. $\text{source}(\neg A)$) will produce finite results.

The desired result set (using the TriG syntax for Named Graphs [1]) that could be generated by the query processor is shown in Figure 3. It consists of two parts. The first part (graph *G11*) contains facts constructed accordingly to the query specification. The second part (graph *G11Meta*) contains records of meta knowledge, aggregated by $\tau(\Omega)$ and constructed according to specified interpretations for particular dimensions. Additionally, *G11Meta* provides the record on *how-provenance* that explains how the result has been constructed. It also implies the *why-provenance* of the result (i.e. $\{t_1, t_3, t_4, t_5\}$). By replacing custom interpretations of algebraic operators for particular domains $\text{dom}(A_i)$ we may easily obtain alternate interpretations for meta knowledge. For example, by re-defining $\text{certainty}^*(A \wedge B) = \min(\text{certainty}(A), \text{certainty}(B))$ we change the probabilistic interpretation of certainty values to fuzzy interpretation.

4 Conclusion

In this proposal, we presented the approach for representing and interpreting meta knowledge for knowledge bases. Our aim was to abstract from particular application domains and implementations and to provide a general representational model. The resulting framework can be flexibly adapted to user's needs by customizing interpretations of *how-provenance* (represented by Boolean formulas) for particular meta knowledge dimensions.

In the future, we will conduct the formal analysis of the proposed framework and its algebraic properties. Furthermore, the framework will be justified in order to support the

notion of Named Graphs for the SPARQL query language, and the notion of *nested* meta knowledge.

Acknowledgements

This work was funded by the X-Media project (www.x-media-project.org) sponsored by the European Commission as part of the Information Society Technologies (IST) programme under EC grant number IST-FP6-026978.

References

- [1] C. Bizer and J. J. Carroll. Modelling context using named graphs. In *W3C Semantic Web Interest Group Meeting*, Cannes, France, 2004.
- [2] P. Buneman, S. Khanna, and W. C. Tan. Data Provenance: Some Basic Issues. *20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), New Delhi, India*, pages 87–93, 2000.
- [3] P. Buneman, S. Khanna, and W. C. Tan. Why and Where: A Characterization of Data Provenance. *8th International Conference on Database Theory (ICDT), London, UK*, pages 316–330, 2001.
- [4] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. *14th International World Wide Web Conference (WWW), Chiba, Japan*, pages 613–622, 2005.
- [5] Y. Cui and J. Widom. Practical Lineage Tracing in Data Warehouses. *16th International Conference on Data Engineering (ICDE), San Diego, USA*, pages 367–378, 2000.
- [6] L. Ding, P. Kolari, T. Finin, A. Joshi, Y. Peng, and Y. Yesha. On homeland security and the Semantic Web: A provenance and trust aware inference framework. In *Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security*, 2005.
- [7] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, January 1997.
- [8] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS'07*, Beijing, China, June 2007. ACM.
- [9] T. Imielinsky and W. Lipsky. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.
- [10] M. Jarke, R. Gellersdoerfer, M. Jeusfeld, and M. Staudt. ConceptBase - A Deductive Object Base for Meta Data Management. *J. Intell. Inf. Syst.*, 4(2):167–192, 1995.
- [11] J. Perez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, volume 4273 of *LNCS*, pages 30–43, 2006.
- [12] J. Perez, M. Arenas, and C. Gutierrez. Semantics of SPARQL. Technical Report TR/DCC-2006-17, Universidad de Chile, October 2006.
- [13] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Working draft, W3C, March 2007. <http://www.w3.org/TR/rdf-sparql-query/>.
- [14] M. Sintek and S. Decker. TRIPLE – an RDF Rule Language with Context and Use Cases. In *Proceedings of the W3C Workshop on Rule Languages for Interoperability*, Washington, DC, 2005.