Formal aspects of querying RDF databases

Claudio Gutierrez[1], Carlos Hurtado[1], and Alberto Mendelzon[2]

[1] Department of Computer Science, Universidad de Chile
{cgutierr,churtado}@dcc.uchile.cl
[2] Department of Computer Science, University of Toronto
mendel@cs.toronto.edu

Abstract. We study formal aspects of querying databases containing RDF data. We present a formal definition of a query language for RDF and compare it with other proposals. Our language is intended to make it easy to formalize and prove results about its properties. We study novel features of query languages derived from the presence of blank nodes and reification. Finally we provide complexity results for query processing, static optimization of queries, and redundancy elimination in answers.

1 Introduction

The *Resource Description Framework (RDF)* [10] is the proposal of the W3C for a standard metadata model and language. RDF follows the W3C design principles of interoperability, evolution and decentralization. The RDF model is simple: the universe to be modeled is a set of *resources* (essentially anything that can have a *universal resource identifier*, URI); the language to describe them is a set of *properties* (technically binary predicates); descriptions are *statements* very much in the subject-predicate-object structure, where predicate and object are resources or strings. Both subject and object can be undetermined objects, known as *blank nodes.* The subject or object of an RDF statement can be another statement, a feature known as *reification.* A vocabulary of *properties* for this language can be defined along the lines given in the RDF Schema language [12].

Languages for querying RDF have been developed in parallel with RDF itself. We can mention rdfDB [3], an influential simple graph-matching query language from which several other query languages evolved. Among them, SquishQL [5] is a graph-navigation query language that was designed to test some of the functionalities of an RDF query language. It adds constraints on the variables and returns as results a table. SquishQL has several implementations like RDQL and Inkling [5]. RQL [4] has a very different syntax based on OQL, but can perform similar sorts of queries. It is a typed language following a functional approach and supports generalized path expressions. Other languages are Triple [9], a query and transformation language, QEL [7], a query-exchange language designed to work across heterogeneous repositories, and DQL [15], a language and protocol for querying DAML+OIL knowledge bases. Good surveys are [16, 17].

1.1 Problem Statement

There is very little research so far on foundational aspects of these languages, such as query semantics and the complexity of query processing. Such research is made necessary by the new features that arise in querying RDF graphs as opposed to standard databases; in particular, two main differences that deserve formal study are blank nodes and reification.

The presence of blank nodes in RDF graphs introduces redundancy. Furthermore, queries themselves, as we will show later, can create redundancy. Central issues in RDF query processing are how to keep RDF graphs as concise as possible, and what is the computational cost of obtaining such representations.

In order to support reification, the language needs expressiveness beyond what is encountered in classical databases. Another open issue is whether this extra expressiveness boosts the complexity of query processing and size of answers compared to classical databases.

1.2 Contributions

We study formal aspects of querying databases containing RDF data. We view RDF specifications as data (although we keep its knowledge base semantics), and study how to efficiently retrieve information from them.

This paper presents:

– A formal definition of a query language for RDF and comparison with other proposals (such as DQL). We present the language in a streamlined form that is not intended for practical use, but to make it easy to formalize and prove results about its properties.
– A formal study of novel features of query languages derived from the presence of blank nodes and reification, and the differences with standard languages studied in the database community.
– Complexity results for query processing, static optimization of queries, and redundancy elimination.

1.3 Related Work

One point of view has considered RDF metadata as a knowledge base and applied knowledge representation and reasoning techniques to RDF metadata. An example of this approach is DQL, a query language for the Semantic Web proposed in [15]. We discuss the database aspects of DQL in Section 3.4.

Another point of view follows the SQL/XQL approach, which views RDF metadata as a relational or XML database. We already mentioned several proposals and working implementation of such languages. They mainly concentrate on expressiveness and implementation issues. Although very rich from these points of view, none of them address formal issues. There are studies comparing features of these languages, such as syntax (body, head and variables in the query), serialization (XML, N3, ASCII), implementation aspects, etc. See for example [16, 17].

2 Preliminaries

In this section we present the RDF model following the W3C documents [10–12] and discuss different variants of some notions.

2.1 RDF graphs

Assume there is an infinite set U (RDF URI references); an infinite set $B = \{b_j : j \in \mathbb{N}\}$ (Blank nodes); and an infinite set L (RDF literals). A triple $(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple.* We often denote by UBL the union of the sets U, B and L.

Definition 1. *An* RDF graph *(just graph from now on) is a set of RDF triples. A* subgraph *is a subset of a graph.*

A graph is *ground* if it has no blank nodes.

Two graphs G_1, G_2 are *isomorphic*[3] if G_2 is obtained from G_1 by renaming its blank nodes by blank nodes in a consistent manner (i.e. avoiding name clashes). (Note that if G_2 can be obtained from G_1 in this way, then so can G_1 from G_2.)

The *merge* of two graphs G_1, G_2 is defined as the union of the set of triples of G_1 and G_2', where G_2' is an isomorphic copy of G_2 whose set of blank nodes is disjoint with that of G_1. (Note that the merge is unique up to isomorphism).

2.2 RDF graphs and standard graphs

RDF graphs are not quite classical graphs. They resemble labelled graphs with the particularity that edge labels are chosen from the set of nodes of the graph.

Definition 2. *1. A pseudograph is a triple (V, E, f) where V is a finite set of nodes, E is a finite set of edge names, and f is a function from E to $V \times V$. (That is, a directed graph that allows self-loops and multiple edges between pairs of nodes).*

An edge-labeled pseudograph *(just pseudograph from now on) is a pseudograph with an additional labeling function $\ell : E \to V$.*

2. Two pseudographs (V_i, E_i, ℓ_i) are isomorphic if (1) There is a (directed) graph isomorphism $\phi : (V_1, E_1) \to (V_2, E_2)$, and (2) $\phi \circ \ell_1 = \ell_2 \circ \phi$.

With the previous definition, an RDF graph is a pseudograph where V is a disjoint union of three sets: the URI References, Blank nodes, and Literals, respectively, that appear in any triple in the RDF graph, and with the following restrictions: (1) source nodes cannot be literals; (2) The image of the labeling function ℓ is contained in the set of URI references. The next theorem follows directly from the definitions.

Theorem 1. *Two RDF graphs G_1, G_2 are isomorphic if and only if there is an isomorphism $\phi : G_1 \to G_2$ of pseudo-graphs such that ϕ preserves URI references and literals.*

[3] In the RDF Concepts document [13] this notion is called "equality" of graphs.

Note 1 (Encoding of Standard Graphs). Note that standard graphs can be encoded by RDF graphs as follows. Choose a distinguished URI reference e. For a graph $G = (V, E)$, choose a set S of distinct blank nodes of the same size as V, and a bijection $c : V \to S$. The graph G is encoded by the RDF graph $\{(c(u), e, c(v)) : (u, v) \in E\}$.

2.3 Semantics of RDF graphs

In this section we deal with *simple* RDF Graphs, i.e. those that do not use a pre-defined vocabulary (class, subject, object, etc.) defined in RDF Schemas; in Section 3.3 we deal with a fragment of RDF(S) vocabulary. We implicitly use the same model-theoretic semantics as the W3C RDF Semantics document [11], although all we need for our purposes is to define entailment, which is characterized by Theorem 2 below, and corresponds roughly to logical consequence between the logical specifications defined by both graphs. The RDF Semantics document [11] describes entailment as follows: "Entailment is the key idea which connects model-theoretic semantics to real-world applications. If A entails B, then any interpretation that makes A true also makes B true, so that an assertion of A already contains the same "meaning" as an assertion of B; we could say that the meaning of B is somehow contained in, or subsumed by, that of A."

A *mapping* is a function μ : UBL $\to$ UBL preserving URI references and literals (i.e., $\mu(u) = u$ and $\mu(l) = l$ for all $u \in U$ and $l \in L$). Given a graph G, we define $\mu(G)$ as the set of all $(\mu(s), \mu(p), \mu(o))$ such that $(s, p, o) \in G$. A mapping μ is *consistent* with G if $\mu(G)$ is an RDF graph (i.e., if s is a subject of a triple, $\mu(s) \in UB$). In this case, we say that the graph $\mu(G)$ is an *instance* of the graph G. An instance of G is *proper* if $\mu(G)$ has fewer blank nodes than G. (This means that either μ instantiates a blank node or identifies two blank nodes of G.)[4]

The following theorem characterizes entailment among RDF graphs. For the purposes of this paper, we can think of it as the definition of entailment.

Theorem 2 (cf. RDF Semantics [11], Interpolation Lemma). *Let G_1, G_2 be RDF graphs. Then G_1 entails G_2 (denoted $G_1 \models G_2$) if and only if an instance of G_2 is a subgraph of G_1.*

We say that two graphs are *equivalent* (denoted $G_1 \equiv G_2$) if $G_1 \models G_2$ and $G_2 \models G_1$.

Example 1. A graph G entails any of its subgraphs.

Example 2. Consider the RDF graphs $G_1 = \{(a, b, c), (X, b, c), (a, b, Y)\}$ and $G_2 = \{(U, b, V), (V, b, c)\}$, and $G_3 = \{(a, b, c), (X, b, Y)\}$, where capital letters indicate blank nodes. Then $G_1 \models G_3$ but $G_1 \not\models G_2$.

[4] In the RDF Semantics document [11] "proper instance" refers only to the case where a blank node is instantiated.

Note 2. Yang and Kifer, in [14], present an F-logic version of RDF. They define two notions of entailment: $\models$ and $|\!\approx$. The first corresponds to the standard notion defined in the RDF Semantics document of the W3C as defined in Theorem 2. The second corresponds to a notion obtained using the same characterization of Theorem 2, but considering only non-proper mappings in the definition of instance.

2.4 Minimal representations: lean graphs

Conciseness in RDF is bound to the notion of *lean* graphs. In this section we study lean graphs.

Definition 3. *A graph G is* lean *if no proper instance of G is a subgraph of G. (That is, there is no mapping μ such that $\mu(G)$ is a proper subgraph of G.)*

Note 3. There is a significant difference between this notion of "lean" and the one stated in the RDF Semantics document [11]. The document reads: "a graph is *lean* if none of its triples is an instance of any other." Our definition captures more precisely the notion of "no redundancy" in an RDF graph which is the idea behind the concept of "lean". While every graph that is lean in the sense of that document is lean in our sense, the converse is not true. See for example the graph G_1 in Example 3, which under the RDF Semantics document definition is not lean.

With our definition we can still prove Anonymity Lemmas 1 and 2 in [11]:
Anonymity Lemma 1: A lean graph does not entail any of its proper instances.
Anonymity Lemma 2: If E' is obtained from a lean graph E by identifying two distinct blank nodes, then E does not entail E'.

Our notion has other desirable properties, e.g. Theorem 3 below.

Example 3. Let $G_1 = \{(X,b,d),(X,b,c),(Y,b,c),(Y,b,e)\}$ and $G_2 = \{(a,b,c),$ $(X,b,c),(Y,b,c)\}$. Then G_1 is lean, but G_2 is not lean.

Fundamental issues regarding lean graphs have not been yet studied. The first fundamental question that arises is whether there is a unique lean graph equivalent to a given one. The following theorem answers this question:

Theorem 3. *Each RDF graph is equivalent to a unique (up to isomorphism) lean graph.*

Proof. Define in the set of RDF graphs, the relation $G \Rightarrow \mu(G)$, if μ is a mapping and $\mu(G)$ is a proper subgraph of G. The relation $\Rightarrow$ has the property: if $B \Leftarrow A \Rightarrow C$, then there is D such that $B \Rightarrow^* D$ and $C \Rightarrow^* D$ (where $\Rightarrow^*$ is the transitive closure of $\Rightarrow$). The proof of this goes as follows: Let $B = \mu_1(A)$ and $C = \mu_2(A)$, and consider the mapping $\mu_2\mu_1$. Then, because $(\mu_2\mu_1)(\mu_2\mu_1)^j(A)$ is a subgraph of $(\mu_2\mu_1)^j(A)$, for some finite $k \geq 1$ it holds that $(\mu_2\mu_1)^k(A)$ is isomorphic to $(\mu_2\mu_1)^{k+1}(A)$.

From its definition, it follows that the relation $\Rightarrow$ clearly cannot have infinite chains $A_1 \Rightarrow A_2 \Rightarrow \cdots$. From the above argument, it also follows that $\Rightarrow$ is confluent. (For rewriting concepts, see [2].) Hence for each G there is a unique G_* such that $G \Rightarrow^* G_*$ and G_* is irreducible with respect to $\Rightarrow$ (i.e. is lean). This is the desired unique lean graph.

The following results show that it is hard to compute lean graphs.

Theorem 4. *Deciding if a graph is lean is coNP-complete.*

Proof. Recall that RDF graphs can encode standard graphs. Hence the proof is an encoding of the problem CORE:
Instance: A graph G
Question: Is there a homomorphism of G to a proper subgraph?
This problem was shown to be NP-complete by Hell and Nesetril [6].

From the above theorem it follows that finding minimal representations for graphs is hard.

3 Querying RDF databases

An RDF graph can be considered a standard relational database: a relation of triples with the attributes Subject, Predicate, and Object. The difference with standard relational databases is the presence of *blank nodes*, which stand for *anonymous* elements.

Thus, for us, an RDF *database* will be simply an RDF graph.

3.1 Query language

Let V be a set of variables (disjoint from UBL).

As query language, we will use the notion of *tableau* borrowed from the database literature (see for example [1]) but slightly extended to allow also a set of tuples in the head. A *tableau* is a pair (H, B) where H, B are RDF graphs over $V \cup \mathrm{UBL}$ and all variables of H occur also in B. We often write a tableau in the form $H \leftarrow B$ to indicate the similarity with logic programming and Datalog.

For example, a tableau such as

$$(?Dept, \text{pays}, ?Instr) \leftarrow (?Instr, \text{lectures}, ?Course), (?Dept, \text{offers}, ?Course),$$

where identifiers preceded by ? are variables, intuitively defines the instructors paid by a department to be those who teach courses offered by the department.

Definition 4. *A* query *is a tableau (H, B) plus a set of constraints C, which is a subset of the variables occurring in H. In other words, a query is a triple (H, B, C) such that:*

1. *H is an RDF graph over $UBL \cup V$, with $var(H) \subseteq var(B)$.*
2. *B is an RDF graph over $UL \cup V$. (i.e. has no blank nodes).*
3. *$C \subseteq var(H)$. (Constraints).*

For example, the tableau above is a query with no constraints. We can add to it the constraint $\{?Instr\}$; intuitively, as we will formalize in the next subsection, this means that the $?Instr$ variable must be bound to a non-blank element in each answer to the query.

Note 4. The condition $var(H) \subseteq var(B)$ avoids the presence of free variables in the head of the query. The presence of blank nodes in the body of the query is unnecessary, because –as we will see– a variable plays exactly the same role in this position. However, we do allow blank nodes in the head of the query to support reification at the query level. (See Examples 7 and 8 in Section 3.3.) Finally, as shown in the example above, $C \subseteq var(H)$ will represent the set of variables in the query that we are forcing to be instantiated by constants.

Note 5. Blank nodes in the head of the query are technically free terms of the form $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_k$ are variables occurring in the query, and f is a function symbol. Hence in our language, there is an arbitrary set of uninterpreted function symbols of different arities. We follow here the same approach as [8].

3.2 Answers to a query

Let D be a database, and V a set of variables.

A *valuation* is a function $v : V \to \mathrm{UBL}$. For a set C of variables, the valuation v satisfies the constraint C (denoted $v \models C$) if for all $x \in C$, $v(x)$ is not blank.[5]

A *matching* of the graph B in database D is a valuation v such that an instance of $v(B)$ is a subgraph of D, i.e. such that $D \models v(B)$.

The matchings that interest us are those that satisfy the constraints C.

Definition 5. *Let $q = (H, B, C)$ be a query and D a database. A* pre-answer *to q over D is the set*

$$\mathrm{preans}(q, D) = \{v(H) : v(B) \text{ is a matching in } D \text{ and } v \models C\}.$$

A graph $v(H)$ is called a single *answer of the query q over D.*

We can combine single answers in two different ways to obtain the answers to a query, leading to two main query semantics:

1. $\mathrm{ans}_\cup(q, D)$ is the set-theoretic union of all single answers. With this approach, queries properly capture the information carried by blank nodes inside D (in particular when blank nodes play the role of bridges between two single answers).
2. An alternative approach, $\mathrm{ans}_m(q, D)$, is to *merge* all single answers, which means to rename blank nodes if necessary to avoid name clashes.

Note that if there are no blank nodes in D, both approaches are the same and we are in the realm of classical databases.

[5] This constraint is called a *must-bind variable* in DQL [15].

Proposition 1. *Let D, D' be databases, and q a query. Then for both semantics, if $D \models D'$ then* $\mathrm{ans}(q, D) \models \mathrm{ans}(q, D')$.

Proof. By hypothesis, there is a mapping μ such that $\mu(D')$ is a subgraph of D. It is enough to prove that every graph $G \in \mathrm{preans}(q, D')$ is a subgraph of a graph in $\mathrm{preans}(q, D)$. Let H the head and B the body of the query. Then $G = v(H)$ for a valuation v, with $v(B)$ a subgraph of D'. Then $\mu(v(B))$ is a subgraph of D, hence $\mu(G) = \mu(v(H)) \in \mathrm{preans}(q, D)$.

Proposition 2. *For all queries q and databases D,* $\mathrm{ans}_{\cup}(q, D) \models \mathrm{ans}_m(q, D)$.

Proof. The statement follows from the fact that $G_1 \cup G_2 \models (G_1 \text{ merge } G_2)$. To check this last fact just consider the mapping from $(G_1 \text{ merge } G_2) \to G_1 \cup G_2$ that reverses the renaming of variables done in the merge.

Note 6. The converse of Proposition 2 does not hold. Consider the identity query q and the database $D = \{(X, b, c), (X, b, d)\}$. Then $\mathrm{ans}_{\cup}(q, D) = D$ and $\mathrm{ans}_m(q, D) = \{(X, b, c), (Y, b, d)\}$. Clearly there is no mapping from D to $\mathrm{ans}_m(q, D)$.

In the sequel, unless stated otherwise, we will assume the union-semantics.

Example 4. Consider a database D which has a blank node B with several properties, i.e., there are in D several triples $(B, p_1, z_1), (B, p_2, z_3), \ldots$. If we follow the merge-semantics, we cannot retrieve the properties of B with a data independent query. On the other hand, if we follow the union-semantics, the query $H = (X, \mathit{feature}, Y)$, $B = (X, Y, Z)$, $C = \emptyset$ will do it.

Example 5 (Identity query). With merge-semantics, there is no data-independent query that retrieves the whole database for all D's. With the union-semantics, the query q defined by $H = (X, Y, Z)$, $B = (X, Y, Z)$ and $C = \emptyset$ returns all triples in D, i.e. $\mathrm{ans}(q, D) = D$.

Example 6. Consider an RDF database consisting of tuples of the following sort:

(Course, name, CourseName)
(Lecturer, lectures, Course)
(Lecturer, name, LecturerName)
(Department, offers, Course)
(Department, belongs, University)
(University, located, Country)

In this database *name, lectures, offers, belongs, located* are RDF predicates defined in some ontology. Courses, Lecturers, Departments and Universities are URLs. CourseName is of type literal.

The predicates *belongs, worksAt* and *teachIn* which appear in the answers belong to another ontology.

1. Database courses in Canada. First consider the query defined by

   $$\begin{aligned} H &= (?Department, \text{belongs}, ?University) \\ B &= (?Course, \text{name}, \text{“Database”}), (?Department, \text{offers}, ?Course), \\ &\quad (?Department, \text{belongs}, ?University), (?University, \text{located}, \text{“Canada”}) \\ C &= \{?Department, ?University\}. \end{aligned}$$

   This query returns all triples (Department, belongs, University), where Department belongs to University and offers a Database course, University is located in Canada, and enforcing that Department and University are not blank.
2. In what universities (and if known, what departments) does John Bassi lecture?

   $$\begin{aligned} H &= (?Lecturer, \text{worksAt}, ?University), (?Lecturer, \text{teachIn}, ?Department) \\ B &= (?Lecturer, \text{name}, \text{“John Bassi”}), (?Lecturer, \text{lectures}, ?Course), \\ &\quad (?Department, \text{offers}, ?Course), (?Department, \text{belongs}, ?University) \\ C &= \{?Lecturer, ?University\}. \end{aligned}$$

   This query will return the name of all Universities at which John Bassi teaches. If the database contains information about the particular Department where Bassi teaches, it will be included. Otherwise, the Department will appear as a blank node in the answer.

Note 7 (Redundancy). We give some observations on redundancies in queries, databases and set of answers.

1. It is desirable to have queries with lean heads. Otherwise, the answer generated will have redundancies which could have been avoided.
2. It is not always possible to have lean graphs in body of queries. For example, consider the query $q = (H, B, \emptyset)$, where $H = (?Course, \text{related}, \text{“Database”})$ and $B = (?Department, \text{offers}, \text{“Database”}), (?Department, \text{offers}, ?Course)$. Clearly B is not lean and is equivalent to the lean graph $B' = (?Department,$ offers, “Database”$)$. It turns out that the query q cannot be reduced to one with body B' (see Note 9).
3. Even having lean databases and queries with lean heads and bodies does not avoid redundancies in the answer set. Consider the lean graph G_1 in Example 3, and the query $(Z, b, c) \leftarrow (Z, b, c)$. The answer set is $\{(X, b, c), (Y, b, c)\}$ which is not lean.

3.3 Reification

Now we will explore the previous concepts when some constant vocabulary is introduced. In this section we will restrict ourselves to a small subset of RDF's vocabulary description language, RDF Schema, which is an extension of RDF. It provides mechanisms for describing groups of related resources and the relationships among these resources. We will be particularly interested in the vocabulary needed to do reification.

Consider the following statement:

$$\text{A triple } (a, b, c) \text{ is a } \textit{statement}. \tag{1}$$

To state this inside RDF one can say: "There is an object B that is a statement, whose subject is a, predicate is b, and object is c." In order to write this down as a set of RDF statements, we need a vocabulary. We will use the following predicate constants:

`rdf:statement`
`rdf:subject`
`rfd:predicate`
`rdf:object`
`rdf:type`

They have a more or less self-explanatory semantics. For example, the triple $(a, \texttt{rdf}:\texttt{type}, c)$ means a is an instance of the class c. Using this vocabulary (whose formal definitions can be found in [12]), the sentence in (1) can be expressed as the following set of triples (when not needed, we will avoid the use of the namespace prefix `rdf`):

$$(B, \texttt{type}, \texttt{statement}), (B, \texttt{subject}, a), (B, \texttt{predicate}, b), (B, \texttt{object}, c)$$

This process is called *reification* of the statement. Our goal in this section is to study our query language extended with the vocabulary of reification in RDFSchema [12].

Note 8. In RDF semantics, statements are referred to by names, i.e. they are not by themselves objects. An implication of this is that from the existence of a triple (a, b, c) it does not follow that its reification also exists. Observe –using Theorem 2– that a triple does not entail its reification and its reification does not entail the triple. RDF follows the conservative approach that a statement is referred to, and there can be several such references, all distinct. Another alternative is to assume that the triple itself is an object of the universe. This is the approach of [14], where the advantages of such an approach are argued. From a database point of view, the current approach of RDF seems more adequate. With the current RDF semantics, an RDF specification, i.e. an RDF graph (database) is a *finite* set of objects, and answers to queries (as defined in this paper) are finite set of objects. However, if the triple itself is an object i_1, then having (a, b, c) in a database D would imply that $(i_1, \texttt{subject}, a)$ is also a valid statement (and hence an object i_2), hence $(i_2, \texttt{subject}, i_1)$ is a valid statement, and so on.

Example 7. The query that reifies a triple (a, b, c) (and creates a blank node $f(a, b, c)$ to refer to it) is:

$$\begin{aligned}(f(a, b, c), \texttt{type}, \texttt{statement}), (f(a, b, c), \texttt{subject}, a),& \\ (f(a, b, c), \texttt{predicate}, b), (f(a, b, c), \texttt{object}, c) \leftarrow (a, b, c).&\end{aligned}$$

The answer to this query applied to a database D is exactly a reification of the triple (a, b, c) if it exists in the database D. Note that we need a Skolem function f to give a different blank node to each triple.

Example 8. A generalization of the previous example: the reification of all triples in the database D.

$$(f(X,Y,Z), \texttt{type}, \texttt{statement}), (f(X,Y,Z), \texttt{subject}, X),$$
$$(f(X,Y,Z), \texttt{predicate}, Y), (f(X,Y,Z), \texttt{object}, Z) \leftarrow (X,Y,Z).$$

Example 9. All properties of an object b (in the database to be queried):
$(X, \texttt{type}, \texttt{property}) \leftarrow (b, X, Y)$
It is interesting to note that now we can test Leibniz's identity on a database by just comparing the results of two queries. Recall Leibniz's identity Law: $a \equiv b$ if and only if for all properties $P(\cdot)$, $P(a)$ iff $P(b)$.

Example 10. (Following an example in Yang and Kifer [14]). All statements made by Encyclopedia Britannica are true. Note that we will need different queries depending on the structure of the database containing the information.

1. If we assume that Encyclopedia Britannica is a database containing all its statements (triples), the following query of would do the work:

$$(f(X,Y,Z), \texttt{veracity}, \texttt{true}), (f(X,Y,Z), \texttt{type}, \texttt{statement}),$$
$$(f(X,Y,Z), \texttt{subject}, X), (f(X,Y,Z), \texttt{predicate}, Y),$$
$$(f(X,Y,Z), \texttt{object}, Z) \leftarrow (X,Y,Z).$$

2. If we assume that the statements of Encyclopedia Britannica are mixed in with several other statements from other sources (but already referred to as belonging to the E. Britannica) we need a query like:
$(X, \texttt{veracity}, \texttt{true}) \leftarrow (X, \texttt{type}, \texttt{statement}), (X, \texttt{made}, \texttt{EncycB})$.

3.4 The language DQL

We will discuss here aspects of DQL that are relevant from a database perspective and compare them with our approach.

1. DQL has a query pattern (our B), an answer pattern (our H) and a set of constraints very similar to our set of constraints C.
2. DQL has three sets of variables, "must-bind", "don't-bind" and "may-bind", which are a partition of the set of variables occurring in the query. Must-bind variables are those that must be bound in each answer. Don't-bind variables are those that must be not bound. In DQL this schema is oriented towards the type of answers the user is asking.
In our setting, "must-bind" variables correspond to the set of constraints C, "don't-bind" correspond to the the set $var(B) \setminus var(H)$ (they do not occur in the answer), and "may-bind" correspond to $var(H) \setminus C$.

3. In DQL the answer set is defined as the largest set of single answers that are entailed by the database such that no answer in the set is entailed by any other answer in the set.

   We do not enforce this condition, due to complexity issues shown in Theorem 4. We think that redundancy cleaning should be an option, and in the next section study the implications of avoiding such redundancies.

4 Complexity issues

In this section we focus on the complexity of query answering. This process has three main components: computing matchings, minimization of queries, and redundancy elimination in answers.

4.1 Computing matchings

In order to understand the complexity of computing the set of matchings for a query over a database, we consider the simpler problem of testing emptiness of the query answer set. Following the usual database theory practice, we distinguish between *query complexity*, that is, evaluation time as a function of query size for a fixed database, and *data complexity*, evaluation time as a function of database size for a fixed query.

1. Query complexity version: For a fixed database D, given a query q, is $q(D)$ non-empty?
2. Data complexity version: For a fixed query q, given a database D, is $q(D)$ non-empty?

Theorem 5. *The evaluation problem is NP-complete for the query complexity version, and polynomial for the database complexity version.*

Proof. Reduction of 3SAT to the problem of evaluating a conjunctive query over a database. Membership in NP follows immediately.

Data complexity version: This follows from the fact that the number of potential matchings of the body of q in D is bounded by the number of subgraphs of D of size q.

From the proof it also follows that the size of the set of answers of a query q issued against a database D is bounded by $|D|^{|q|}$, where $|D|$ is the size of the database (number of triples) and $|q|$ is the number of symbols in the query.

Also note that reification does not play any relevant role in this, that is, even with reification the query language preserves the tractability of answers.

4.2 Minimization of queries

Since Theorem 5 implies that query evaluation is likely to be exponential in query size, static optimization of queries is an important goal. To perform this analysis, we apply techniques similar to classical tableau analysis [1].

A homomorphism $h : q' \rightarrow q$ is a substitution (of variables and blank nodes) θ such that $\theta(B') \subseteq B$ and $\theta(H') = H$ and $C' \subseteq C$. As usual, we define $q \subseteq q'$ if $\mathrm{ans}(q, D) \subseteq \mathrm{ans}(q', D)$ for all databases D.

Theorem 6. *$q \subseteq q'$ if and only if there exists a homomorphism $h : q' \rightarrow q$.*

Proof. Assume there exists a homomorphism $h : q' \rightarrow q$. Then $C' \subseteq C$ and using Theorem 2, for each valuation v, we have $v(B) \models v(\theta(B'))$. Then, for each database D, if $D \models v(B)$, then $D \models v(\theta(B'))$. Hence, each answer $v(H)$ of q will also be an answer $v(H')$ of q' (recall $\theta(H') = H$ and $C' \subseteq C$.)

Conversely, assume $q \subseteq q'$. Consider the database D_B obtained from B by replacing each variable x by a constant a_x. Let v the valuation assigning x to a_x. Then $v(H) \in \mathrm{ans}(q, D_B) \subseteq \mathrm{ans}(q', D_B)$. So, there is a valuation v' such that $D \models v'(B)$ and $v'(H) = v(H)$. Clearly $v = v'$ on the variables of H. Consider the substitution $\theta = v' \circ v^{-1}$. The condition $C' \subseteq C$ follows from $q \subseteq q'$.

We say that a query $q = (H, B, C)$ is *minimal* if there is no query $q' = (H', B', C')$ equivalent to q such that $|B'| < |B|$ (where $|X|$ means the size of the set).

Theorem 7. *For each query $q = (H, B, C)$ there is a minimal query $q_m = (H, B_m, C_m)$ equivalent to q and $B_m \subseteq B$ and $C_m \subseteq C$.*

Proof. Let $q' = (H', B', C')$ be a minimal query equivalent to q. Then there are homomorphisms $\theta_1 : q \rightarrow q'$ and $\theta_2 : q' \rightarrow q$. Consider $q_m = \theta_2\theta_1(q)$.

Note 9. Observe that this minimization does not coincide exactly with the leanization of the body of the query, because the homomorphism that reduces query q to q_m poses another condition besides a mapping from body to body, namely that it must preserve heads. This is the rason why in Note 7, the query in item 2 cannot be further reduced.

Theorem 8. *Let q, q' be two queries. The following problems are NP-complete.*

1. *Is $q \subseteq q'$?*
2. *Is $q \equiv q'$?*

Proof. NP-hardness: coding of classical tableau. (Note that one relation with 3 attributes suffices.)

Membership in NP follows from noting that a witness is the homomorphism.

4.3 Redundancy elimination

Answers to queries in RDF usually have redundancies. Ideally, the answer set $\mathrm{ans}(q, D)$ should reduce these redundancies to the minimum, i.e. to an equivalent lean graph.

Now we will apply these results to redundance elimination in queries.

The naive approach to eliminate redundancy in answers is (1) to compute $\mathrm{ans}(q, D)$, and (2) to compute a lean equivalent to $\mathrm{ans}(q, D)$. Next theorem shows that in the worst case there is no better approach.

Theorem 9. *Given a lean database D and a query q, to decide whether* $\mathrm{ans}_\cup(q, D)$ *is lean is coNP-complete (in the size of D).*

The Theorem directly follows the fact that there is a query that computes the identity and from Theorem 4.

For merge-semantics redundancy elimination can be done much more efficiently:

Theorem 10. *Given a lean database D and a query q, deciding whether* $\mathrm{ans}_m(q, D)$ *is lean can be done in polynomial time in the size of D.*

Proof. Let $A = \mathrm{ans}_m(q, D)$ and let us refer to mappings from single answers to A as *single mappings.*

The key observation is that, because single answers do not share variables in merge-semantics, mappings $\mu : A \to A$ are exactly unions of single mappings $\mu_j : G_j \to A$ for each G_j single answer. (Note that in the case of union-semantics the union of the μ_j would not be a function.)

Thus an algorithm for finding proper mapping $\mu : A \to A$ only needs to compute single mappings and check whether (1) at least a single mapping is proper, or (2) two of them share a blank node in their range. This can be done in time polynomial on the set of single mappings, which size is polynomial on the size of D. Thus the complete test can be done in polytime.

5 Conclusions and Future Work

RDF databases pose new challenges to query languages, which arise due to particularities of the RDF model, such as reification and blank nodes. This paper intends to provide conceptual insight into the problem of dealing with these new features in query languages. Our work also establishes theoretical foundations for further research in this area.

Blank nodes play a crucial role in the semantics of query answering, although they do not affect complexity bounds dramatically. In fact, the behaviour of blank nodes is at the heart of different interpretations, both in query languages and in the formal semantics of RDF itself. For examle, the notions of *lean* and *proper instance* deserve further development.

The expressive power of reification in query languages needs further study. In particular, the proper fragment of logic in which RDF query languages must operate is still not well understood.

Our study brought forth the need to formalize richer properties and mechanisms of current working query languages for RDF. This formalization would establish a solid base to compare functionalities, features and limitations of these languages. For example, features like connectedness, reachability, paths, recursion, extended constraints, aggregation and views.

Acknowledgements C. Gutiérrez was partially funded by FONDECYT No 1030810. C. Hurtado was funded by Millenium Nucleus, Center for Web Research (P01-029-F), Mideplan.

References

1. S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley Publishing Co., 1995.
2. F. Baader, T. Nipkow, *Term Rewriting and All That*, Cambridge Univ. Press, 1998.
3. R. V. Guha, *rdfDB Query Language*, in `http://www.guha.com/rdfdb/query.html`
4. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, *RQL: A Declarative Query Language for RDF*, Proceedings WWW2002, Hawaii, USA, 2002.
5. L. Miller, A. Seaborne, A. Reggiori, *Three Implementations of SquishQL, a Simple RDF Query Language*, Proc. 1st. International Semantic Web Conference ISWC2002, Sardinia, Italy, 2002.
6. P. Hell, J. Nesetril, *The core of a graph*, Discrete Math. 109 (1992), 117-126.
7. *RDF Query Exchange Language (QEL)*, Edit. M. Nilsson, W. Siberski, `http://edutella.jxta.org/spec/qel.html`
8. Y. Papakonstantinou, V. Vassalos, *Query Rewriting for Semistructured Data*, Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, June 1999
9. M. Sintek, S. Deker, *TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web*, Proc. International Semantic Web Conference (ISWC), Sardinia, June 2002.
10. *Resource description framework (RDF) model and syntax specification*, Edit. O. Lassila, R. Swick, Working draft, W3C, 1998.
11. *RDF Semantics, W3C Working Draft, 23 January 2003* Edit. Patrick Hayes
12. *RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003*, Edit. Dan Brickley, R.V. Guha.
13. *RDF Concepts and Abstract Syntax*, Edit. G. Klyne, J. J. Carroll. W3C Working Draft 23 January 2003.
14. G. Yang, M. Kifer, *On the Semantics of Anonymous Identity and Reification*
15. *DAML Query Language (DQL), April 2003, Abstract Specification. DAML Joint Committee, R. Fikes, P. Hayes, I. Horrocks, Ed.*
16. *E. Prud'hommeaux, B. Grosof,* RDF Query and Rules: A Framework and Survey*,* `http://www.w3.org/2001/11/13-RDF-Query-Rules/`
17. *A. Magkanaraki et al.* Ontology Storage and Querying*, Technical Report No. 308, April 2002, Foundation for Research and Technology Hellas, Institute of Computer Science, Information System Laboratory.*