# Foundations of Semantic Web Databases

Claudio Gutierrez[uc] Carlos Hurtado[uc] Alberto O. Mendelzon[ut]
[uc]Department of Computer Science, Universidad de Chile
{cgutierr,churtado}@dcc.uchile.cl
[ut]Department of Computer Science, University of Toronto
mendel@cs.toronto.edu

## ABSTRACT

The Semantic Web is based on the idea of adding more machine-readable semantics to web information via annotations written in a language called the Resource Description Framework (RDF). RDF resembles a subset of binary first-order logic including the ability to refer to anonymous objects. Its extended version, RDFS, supports reification, typing and inheritance. These features introduce new challenges into the formal study of sets of RDF/RDFS statements and languages for querying them. Although several such query languages have been proposed, there has been little work on foundational aspects. We investigate these, including computational aspects of testing entailment and redundancy. We propose a query language with well-defined semantics and study the complexity of query processing, query containment, and simplification of answers.

## 1. INTRODUCTION

The Web is a huge collection of interconnected data. Managing and processing such information is difficult due to the fact that the Web lacks *semantic* information. The Semantic Web is a proposal to build an infrastructure of machine-readable semantics for the data on the Web. In 1998 the W3C issued a recommendation of a metadata model and language to serve as the basis for such infrastructure, the *Resource Description Framework (RDF)* [22]. As RDF evolves, it is increasingly gaining attraction from both researchers and practitioners, and is being implemented in world-wide initiatives such as the Open Directory Project, Dublin Core, FOAF, and RSS.

RDF follows the W3C design principles of interoperability, extensibility, evolution and decentralization. Particularly, the RDF model was designed with the following goals: simple data model; formal semantics and provable inference; extensible URI-based vocabulary; allowing anyone to make statements about any resource. In the RDF model, 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 anonymous objects, known as *blank nodes.* The subject or object of an RDF statement can be another statement, a feature known as *reification.* In addition, the RDF specification includes a built-in vocabulary with a normative semantics (RDFS). This vocabulary deals with inheritance of classes and properties, as well as typing, among other features [24]. Good introductory references for the RDF model are [25] and [26]. Figure 1 shows an example of RDF data.


*PODS 2004* June 14-16, 2004, Paris, France.


### 1.1 Problem Statement

Languages for querying RDF have been developed in parallel with RDF itself, e.g. rdfDB [10], SquishQL [15], RQL [13], Triple [19] QEL [16], DQL [21], SeRQL [4]. However, there is very little research so far on foundational aspects of the RDF data model and RDF query languages. Research on formal aspects of RDF data and query languages is made necessary by the new features that arise in querying RDF graphs as opposed to standard databases: the presence of blank nodes, reification, premises in queries, and the RDFS vocabulary with predefined semantics. Formal aspects that need study are normal forms and redundancy elimination, entailment, semantics of query languages, query containment and complexity of query processing.

The RDF data model allows several representations for the same information. This raises the question about the existence of normal forms and testing of equivalence among them. Currently different RDF query languages implement different querying mechanisms and functionalities that have not been the subject of a systematic and integrated study. Traditional database notions of query containment do not translate directly to the RDF setting. They need to be reformulated to take into account the fact that RDF queries process logical specifications rather than plain data. Additionally, premises and constraints on queries add further com-

plexity to the problem. Regarding query processing, the presence of predefined semantics and blank nodes in RDF introduce new problems. These include testing entailment of databases and query conditions for keeping RDF databases and query outputs as concise as possible.

## 1.2 Contributions

In this paper, we study formal aspects of querying databases containing RDF data.[1] We study main features of RDF query languages derived from the presence of blank nodes, reification, premises in queries and the role of predefined RDF vocabulary in query processing.

We introduce a notion of normal form for RDF graphs, give an algorithm to compute it, and study its complexity. The normal form proposed in this paper combines the notions of *closure* and of *core* for RDF graphs, which by themselves do no lead to unique representations for RDF data. The normal form gives a unique representative for each class of equivalent RDF graphs.

We give a formal definition of a query language for RDF. 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. The language comprises all the main features encountered in current RDF query languages and gives them a well defined semantics. We study the differences with standard languages studied in the database community.

We investigate theoretical and complexity issues related to query containment, query processing, and redundancy elimination for queries. The normal form allows us to approach RDF queries with similar techniques to classical databases. This is achieved by reducing the processing of RDF queries, from computing knowledge base entailments, as defined for RDF query languages, to the search for mappings between RDF graphs. This approach permits to view RDF specifications as data while keeping their knowledge-base semantics.

## 1.3 Related Work

The RDF model was introduced five years ago as a W3C recommendation [22]. Its formal semantics, however, is still an ongoing work [23]. From a logical point of view, Yang and Kifer in [20], present an F-logic version of RDF. They define two notions of entailment for RDF graphs and concentrate mainly in the semantics of blank nodes and reification. Extensions of the model, adding expressiveness leading to the realm of descriptive logics, can be found in the *Web Ontology Language*, OWL [17].

Several ideas from the processing of semistructured data are of use in the RDF context, e.g. incomplete answers [6], and query rewriting [**?**]. Despite the apparent similarities of the models, aspects like blank nodes, graph-like structure, and semantics, make the problems studied in this paper somehow orthogonal to problems addressed in previous research on semistructured data.

The notion of core has appeared in various contexts, e.g. graphs [12], data exchange [7].

Languages for querying RDF have been developed in parallel with RDF itself. We can mention rdfDB [10], an influential simple graph-matching query language from which several other query languages evolved. Among them, SquishQL [15] 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 a table as result. SquishQL has several implementations like RDQL and Inkling [15]. RQL [13] 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. Its new version is [4]. Other languages are Triple [19], a query and transformation language, QEL [16], a query-exchange language designed to work across heterogeneous repositories, and DQL [21], a language for querying DAML+OIL knowledge bases, that consider RDF data as a knowledge base, applying reasoning techniques to RDF querying. Good surveys are [18, 14].

Queries with premises (see Section 4.3) have been studied in the logic programming community, e.g. [8]. Their complexity aspects from a database point of view are studied in [5]. Premises also appear in the context of query languages for knowledge bases, e.g. DQL [21]. In SQL-like RDF query languages, this feature appears as a specification of a schema to be used when processing the query [4, 15].

# 2. PRELIMINARIES

In this section we present the RDF model following the W3C documents [22, 23, 24]. We use the abstract representation of the model as graphs, and do not discuss any serialization of the model (e.g. its XML-based syntax).

## 2.1 RDF graphs

Assume there is an infinite set $U$ (RDF URI references); an infinite set $B = \{N_j : j \in \mathbb{N}\}$ (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*. In such a triple, $v_1$ is called the *subject*, $v_2$ the *predicate* and $v_3$ the *object*. 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. The* universe *of a graph $G$,* universe($G$)*, is the set of elements of UBL that occur in the triples of $G$. The* vocabulary *of $G$ is the set* universe($G$) $\cap$ ($U \cup L$)*.*

[1]Preliminary results for the case with no rdfs vocabulary were presented at the First International Workshop on Semantic Web and Databases co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003.

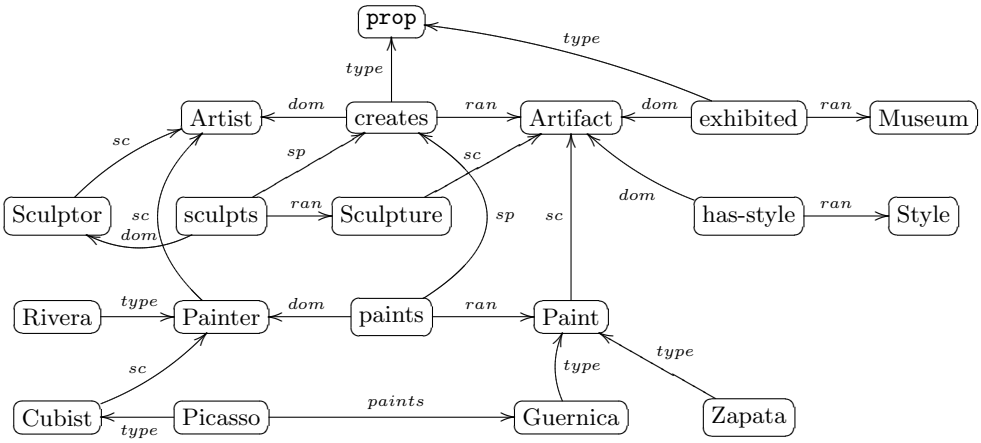



**Figure 1: An RDF graph specifying a schema to describe art resources. The relations *subclass (sc)*, *subproperty (sp)*, *type*, *domain* and *range* belong to the RDFS vocabulary. The triple *(Picasso, paints, Guernica)* shows that in RDF specifications, schemas and data can be described at the same level. Note that the set of arc labels and node labels may intersect, e.g. *paints* is both a node label and an arc label. There are arcs not depicted to avoid crowding the figure. Example taken from [4].**

*We will use letters $N, X, Y, \ldots$ to denote blank nodes, and $a, b, c, \ldots$ for URIs and literals.*

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

Graphically we represent RDF graphs as follows: each triple $(a, b, c)$ is represented by $a \xrightarrow{b} c$. Note that the set of arc labels can have non-empty intersection with the set of node labels.

We will need some technical definitions. A *map* is a function $\mu$ : UBL $\rightarrow$ UBL preserving URIs 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 map $\mu$ is *consistent* with $G$ if $\mu(G)$ is an RDF graph, *i.e.* , if $s$ is the subject of a triple, then $\mu(s) \in UB$, and if $p$ is the predicate of a triple, then $\mu(p) \in U$. 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 $\mu$ sends a blank node to a URI or a literal, or identifies two blank nodes of $G$. We will overload the meaning of map and speak of a *map* $\mu : G_1 \rightarrow G_2$ if there is a map $\mu$ such that $\mu(G_1)$ is a subgraph of $G_2$.

Two graphs $G_1, G_2$ are *isomorphic*, denoted $G_1 \cong G_2$, if there are maps $\mu_1, \mu_2$ such that $\mu_1(G_1) = G_2$ and $\mu_2(G_2) = G_1$.

We define two operations on graphs. The *union* of $G_1, G_2$, denoted $G_1 \cup G_2$, is the set theoretical union of their sets of triples. The *merge* of $G_1, G_2$, denoted $G_1 + G_2$, is the union $G_1 \cup G_2'$, where $G_2'$ is an isomorphic copy of $G_2$ whose set of blank nodes is disjoint with that of $G_1$. Note that $G_1 + G_2$ is unique up to isomorphism.

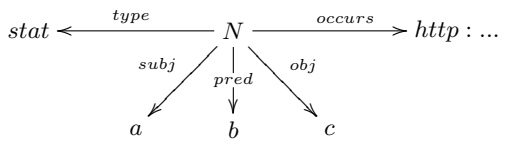
## 2.2 RDFS Vocabulary

There is a set of reserved words defined in the RDF vocabulary description language, RDF Schema [24], – just *rdfs-vocabulary* for us– that may be used to describe properties like attributes of resources (traditional attribute-value pairs), and also to represent relationships between resources. It defines classes and properties that may be used for describing groups of related resources and relationships between resources.

*Classes* are sets of resources. Elements of a class are known as *instances* of that class. To state that a resource is an instance of a class, the property rdf:type may be used. The following are the most important classes (in brackets the name we will use in this paper) rdfs:Resource [`res`], rdfs:Class [`class`], rdfs:Literal [`literal`], rdfs:Datatype [`datatype`], rdf:XMLLiteral [`xmlLit`], rdf:Property [`prop`].

*Properties* are binary relations between subject resources and object resources. The built-in properties are: rdfs: range [`range`], rdfs:domain [`dom`], rdf:type [`type`], rdfs: subClassOf [`sc`], rdfs:subPropertyOf [`sp`].

The *reification* vocabulary was designed to allow making statements about statements. It consists of rdf:Statement [`stat`], rdf:subject [`subj`], rdf:predicate [`pred`], and rdf: object [`obj`]. The following graph states that the triple $(a, b, c)$, occurs in page *http:....*



Logically this graph states: "There *exists* an anonymous resource of type statement which occurs in *http:...*, whose subject, predicate and object are, respectively, $a$, $b$ and $c$."

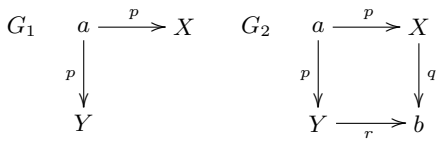
The formal semantics of rdfs vocabulary will be presented in Section 3.2 below.[2]

## 2.3 Lean graphs

We study in this section the notion of *core* for RDF graphs which is related to the notion of *lean* in [23]. Similar notions have been investigated in other contexts [12, 11, 7]. For RDF graphs, this notion is closely bound to minimal representations and normal forms of RDF graphs as will be shown in Section 3.

DEFINITION 2. *A graph $G$ is* lean *if there is no map $\mu$ such that $\mu(G)$ is a proper subgraph of $G$.*

EXAMPLE 1. *Let $p, q, r$ be different predicates and consider:*




*Then $G_1$ is not lean, but $G_2$ is lean because there is no proper map of $G_2$ into itself.*

The following theorem will be the basis of the applications of the notion of core in the context of RDF graphs.

THEOREM 1 (CORE). *Each RDF graph $G$ contains a unique (up to isomorphism) lean subgraph which is an instance of $G$. We will denote this unique subgraph by* core$(G)$.

Computing cores is hard:

THEOREM 2. *Let $G, G'$ be RDF graphs.*

1. *Deciding if $G$ is lean is coNP-complete.*

2. *Deciding if $G' \cong$ core$(G)$ is DP-complete.*

# 3. SEMANTICS OF RDF GRAPHS

The normative semantics for RDF graphs given in [23] follows standard classical treatment in logic with the notions of model, interpretation, entailment, and so on.

[2]We omit in this paper vocabulary intended to describe lists, collections, some variations on these, as well as vocabulary to help document and describe other functionalities for which there is no normative semantics. The complete vocabulary can be consulted in [24].

Roughly speaking an interpretation $I$ of an RDF graph $G$ consists of: (1) a non-empty set of resources $Res$; (2) a distinguished subset $Lit \subseteq Res$, the literals; (2) a set of binary properties $Prop \subseteq Res \times Res$; and (3) mappings from the vocabulary of $G$, $U \to Res \cup Prop$ and $L \to Lit$, plus some integrity restrictions. Given $I$, $I$ satisfies $G$ iff $G$ is true under this interpretation. Then, an RDF graph $G_1$ entails $G_2$, denoted $G_1 \models G_2$, iff every interpretation over the vocabulary of $G \cup G_2$ which satisfies $G_1$ also satisfies $G_2$. For further discussion on this model theoretical semantics we refer the reader to [23].

We say that two graphs are *equivalent*, denoted $G_1 \equiv G_2$, if $G_1 \models G_2$ and $G_2 \models G_1$. Note that isomorphism is a purely syntactic relation among graphs, but equivalence relies on the semantic notion of entailment.

## 3.1 Semantics of simple RDF graphs

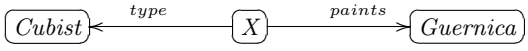
For our purposes, it is sufficient to have a working characterization of the notion of entailment between RDF graphs. We will proceed in two steps: In this section, we present a characterization of the notion of entailment between *simple RDF graphs*, i.e., those graphs that do not use vocabulary with a predefined semantics. Then, in Section 3.2, we will give a general characterization of entailment for graphs using rdfs vocabulary.

For the class of simple graphs entailment is characterized by the following theorem.

THEOREM 3 (CF. RDF SEMANTICS [23]). *Let $G_1, G_2$ be simple RDF graphs. Then $G_1$ entails $G_2$, denoted $G_1 \models G_2$, if and only if there is a map $G_2 \to G_1$.*

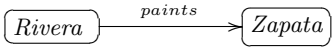
EXAMPLE 2. *A graph $G$ entails any of its subgraphs.*

EXAMPLE 3. *Refer to the graph in Figure 1 and call it $G$. Then $G$ entails the graph*




*This can be interpreted as: "From the graph in Figure 1 it follows that there is a cubist who painted Guernica",*

*But $G$ does not entail the graph*




*This can be interpreted as: "From the graph in Figure 1 it does not follow that Rivera painted Zapata."*

THEOREM 4 (FOLKLORE). *1. Deciding entailment of simple RDF graphs is NP-complete. 2. Deciding equivalence of simple RDF graphs is isomorphism-complete.*

NOTE 1. *It is interesting to observe that the complexity of testing entailment and equivalence depends heavily*

*on the set of blank nodes of the RDF graph. This is important because in most RDF data currently found in practice the set of blank nodes is small compared to the set of URIs or literals (ground nodes) of the graph. It is not difficult to see that this testing can be done in time $\mathcal{O}(v^n)$, where $v$ is the set of nodes of the largest graph and $n$ is the number of blank nodes.*

For simple graphs, the concept of lean graph corresponds exactly to minimal representations as the next theorem shows:

THEOREM 5. *If $G$ is simple, then* core$(G)$ *is the unique (up to isomorphism) minimal (w.r.t number of triples) graph equivalent to $G$.*

## 3.2 Semantics of RDF graphs with RDFS Vocabulary

In what follows, we present an operational semantics for the notion of entailment for graphs with rdfs-vocabulary. It is based on a sound and complete set of rules for $\models$ given in [23].

### 3.2.1 Rules

The set of rules is arranged in four groups. Group A describes the semantics of blank nodes, which is essentially the semantics of simple RDF graphs. Group B describes the semantics of `sp`, stating that it is a transitive relation. Group C describes similar semantics for `sc`. Group D states the semantics of `dom` and `range`, the domain and range of a relation. We omit another group of rules that has to with internal relationships of the RDF model itself and that we do not consider in this paper; see [23] for details.

**GROUP A (Existential)** For a map $\mu : G' \to G$:

$$\frac{G}{G'} \tag{1}$$

**GROUP B (Subproperty)**

$$\frac{(a, \texttt{type}, \texttt{prop})}{(a, \texttt{sp}, a)} \tag{2}$$

$$\frac{(a, \texttt{sp}, b) \;\; (b, \texttt{sp}, c)}{(a, \texttt{sp}, c)} \tag{3}$$

$$\frac{(a, \texttt{sp}, b) \;\; (x, a, y)}{(x, b, y)} \tag{4}$$

**GROUP C (Subclass)**

$$\frac{(a, \texttt{type}, \texttt{class})}{(a, \texttt{sc}, a)} \tag{5}$$

$$\frac{(a, \texttt{sc}, b) \;\; (b, \texttt{sc}, c)}{(a, \texttt{sc}, c)} \tag{6}$$

$$\frac{(a, \texttt{sc}, b) \;\; (x, \texttt{type}, a)}{(x, \texttt{type}, b)} \tag{7}$$

**GROUP D (Typing)**

$$\frac{(a, \texttt{dom}, c) \;\; (x, a, y)}{(x, \texttt{type}, c)} \tag{8}$$

$$\frac{(a, \texttt{range}, d) \;\; (x, a, y)}{(y, \texttt{type}, d)} \tag{9}$$

The following deductive system based on the rules presented, is sound and complete for entailment of RDF graphs with rdfs vocabulary.

DEFINITION 3. *Let $G$ be a graph. For each rule $r : \frac{A}{B}$ above, define $G \vdash_r G \cup \mu(B)$ iff there is a map $\mu : A \to G$. Also define $G \vdash_s G'$ iff $G'$ is a subgraph of $G$.*

*Define $G \vdash G'$ if there is a finite sequence of graphs $G_1, \ldots, G_n$ such that (1) $G = G_1$; (2) $G' = G_n$; and (3) for each $i$, either, $G_i \vdash_r G_{i+1}$ for some $r$, or $G_i \vdash_s G_{i+1}$.*

The following theorem relieves us from having to describe the model theoretic semantics and will suffice for our purposes. The proof given in [23] with slight modifications works here.

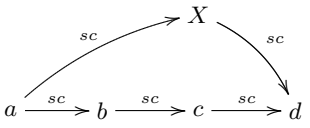
THEOREM 6 (CF. [23]). *The deductive system of Definition 3 is sound and complete for $\models$. That is, $G_1 \vdash G_2$ if and only if $G_1 \models G_2$.*

### 3.2.2 Characterization of entailment

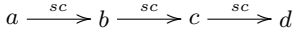
When working with non-simple graphs, Theorem 3 does not hold, because with rdfs vocabulary issues like transitivity come into play, as the next example shows. We will show that there is a notion of normal forms for RDF graphs for which such a characterization can be given.

EXAMPLE 4. *Theorem 3 is not true in general for graphs with rdfs-vocabulary. Although one can prove that the following three graphs are equivalent, there are no maps, for example, from $G_1 \to G_2$ nor $G_3 \to G_1$ (recall that* `sc` *denotes subClassOf).*

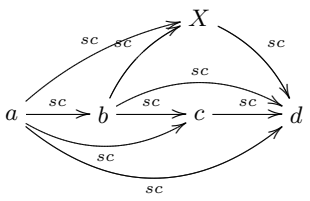
$G_1$ :




$G_2$ :



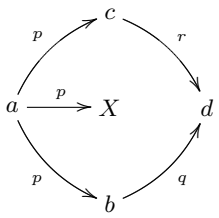
$G_3$ :




To avoid the problems of Example 4, the simplest idea is to "close" the graph with all possible triples entailed from the existing ones.

Definition 4. *A* closure *of a graph $G$ is a maximal set of triples $G'$ over* universe($G$) *plus the rdfs vocabulary such that $G'$ contains $G$ and is equivalent to it.*

Example 5. *There could be more than one closure of a graph. For example the graph*




*where $p, q, r$ are different properties, has two different non-isomorphic closures, namely, either adding the triple $(X, r, d)$ or the triple $(X, q, d)$ (but not both).*

A slightly different notion of closure is defined in [23]. In that document, the *"RDFS-closure"* of a graph $G$ is defined as the closure of $G$ under all the rules of Section 3.2.1 except (1). With this notion, the following result is proved.

Lemma 1 (cf. [23]). *$G_1 \models G_2$ if and only if there is a map from $G_2$ to the* RDFS-closure *of $G_1$.*

Using this lemma it can be proved:

Theorem 7. *Deciding entailment of RDF graphs with rdfs vocabulary is NP-complete.*

It turns out that from a data representation point of view, the notions of "RDFS-closure" introduced in [23] and of "closure" from our Definition 4 are not the best choices to work with because they may have redundancies. Similarly, the operator core in the case of non-simple graphs does not eliminate all redundant triples: in Example 4, $G_1$, $G_2$ and $G_3$ are lean graphs, but $G_1$ and $G_3$ have redundancies.

We next introduce a notion that avoids these problems.

Definition 5. *For a graph $G$, define its* normal form, *denoted* nf($G$), *as* core($G'$) *for a closure (as in Definition 4) $G'$ of $G$.*

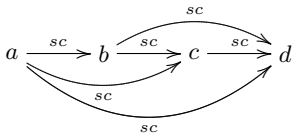
Theorem 8 (Normal forms for RDF graphs). *Let $G$ be an RDF graph. Then:*

*1. The normal form* nf($G$) *is unique (up to isomorphism).*

*2. $G_1 \models G_2$ if and only if* nf($G_2$) $\rightarrow$ nf($G_1$)

*3. $G_1 \equiv G_2$ if and only if* nf($G_1$) $\cong$ nf($G_2$).

Example 6. *The normal form for any of the graphs of Example 4 is:*




### 3.2.3 Redundancy elimination

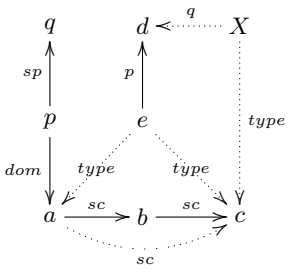
Normal forms are not the most compact representation in general, as Example 6 shows. In this example, in order to eliminate redundant triples of the resulting graph one needs a transitive reduction of `sc`.

Definition 6. *A* reduction *of a graph $G$ is a minimal graph $G_r$ equivalent to $G$ and contained in $G$.*

Proposition 1. *The following algorithm gives the reduction of a graph. (reverse rule means deleting the triple deduced by the rule).*

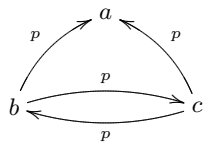
*1. $G \leftarrow$ nf($G$).*
*2. Apply reverse rule (7) until no longer applicable.*
*3. Apply reverse rules (8) and (9) until no longer applicable.*
*4. Apply reverse rule (4) until no longer applicable.*
*5. Apply transitive reduction of* `sp` *and* `sc`.
*6. Apply any reverse rule in any order until no longer applicable.*

Example 7. *Reduction of a graph. Consider:*



*Its reduction is the subgraph given by the solid lines. One must be careful in the order in which the "reverse" rules are applied. For example, to eliminate* $(e, \texttt{type}, c)$ *using rule (7) from Section 3.2.1 one needs* $(a, \texttt{sc}, c)$ *and* $(e, \texttt{type}, a)$.

EXAMPLE 8. *There could be more than one reduction of a given graph. This follows from the transitive property of* `sc` *and* `sp` *and classical results on transitive reduction on graphs [1]. The standard example is:*




*where* $p$ *is a transitive property. The reduction is obtained by deleting either* $(b, p, a)$ *or* $(c, p, a)$*, but not both.*

THEOREM 9 (ACYCLIC REDUCTION). *Let* $G$ *be subproperty- and subclass- acyclic. Then*

*1. The reduction of* $G$ *is unique (up to isomorphism), denoted* $\mathrm{red}(G)$.

*2.* $G_1 \models G$ *if and only if there is a map* $\mathrm{red}(G) \rightarrow \mathrm{nf}(G_1)$.

Graphs whose subgraphs defined by `sp` and `sc` are acyclic form an important class. In modeling, this is considered good practice [9].

### 3.2.4 Complexity of the closure and the reduction

The notion of normal form for RDF graphs permits to have a clean operational semantics for entailment, and will be essential in the next sections when discussing the query language. On the other hand, the reduction of a graph can save considerable space compared to the original graph. Unfortunately, computing normal forms and reductions is a hard problem.

THEOREM 10. *Let* $G, G'$ *be graphs.*

*1. The problem of deciding if* $G'$ *is the closure of* $G$ *is DP-complete.*

*2. The problem of deciding if* $G'$ *is the normal form of* $G$ *is DP-complete.*

*3. The problem of deciding if* $G'$ *is the reduction of* $G$ *is DP-complete.*

# 4. QUERYING RDF DATABASES

An RDF graph can be considered a standard relational database: a relation of triples with the attributes Subject, Predicate, and Object. In what follows, an RDF *database* will be simply an RDF graph.

## 4.1 Query Language

Let $V$ be a set of variables (disjoint from UBL). Individual variables will be denoted $?X, ?Y, ?Person$, etc.

As query language, we will use the notion of *tableau* borrowed from the database literature (see for example [2]) 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 with some UBLs replaced by variables in $V$, $B$ has no blank nodes, 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

$$(?A, \text{creates}, ?Y) \leftarrow (?A, \text{type}, Flemish), \\ (?A, \text{paints}, ?Y), (?Y, \text{exhibited}, \text{Uffizi})$$

where identifiers preceded by ? are variables, intuitively defines the artifacts created by Flemish artists being exhibited at Uffizi Gallery.

DEFINITION 7. *A* query *is a tableau* $(H, B)$ *plus a set of premises* $P$ *and a set of constraints* $C$*, where* $P$ *is a graph over UBL (i.e. with no variables) and* $C$ *is a subset of the variables occurring in* $H$*. In other words, a query is a tuple* $(H, B, P, C)$.

When $P$ is omitted we assume the premise is empty, i.e. write $(H, B, C)$ instead of $(H, B, \emptyset, C)$. Similarly for the set of constraints $C$ or both.

The set of constraints $C$ gives the user the possibility to discriminate between blank and ground nodes in answers and plays the same role as IS NOT NULL in SQL. For example, the tableau above is a query with no constraints. We can add to it the constraint $\{?A\}$; intuitively, as we will formalize in the next subsection, this means that the $?A$ variable must be bound to a non-blank element in each answer to the query.

The premise $P$ represents information the user supplies to the database to be queried in order to answer the query. For example, the query:

$(?X, \text{relative}, Peter) \leftarrow (?X, \text{relative}, Peter)$
with premise $P = \{(son, \texttt{sp}, relative)\}$ ask for all relatives of Peter knowing that "son" is a subproperty of "relative".

NOTE 2. *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 in Section 4.4.) Their semantics is explained in the next section.*

## 4.2 Answers to a query

Let $q = (H, B, P, C)$ be a query, $D$ a database, and $V$ a set of variables. This section defines the semantics of the query $q$ over the database $D$.

A *valuation* is a function $v : V \to \text{UBL}$. For a set $C \subseteq V$ of variables, the valuation $v$ satisfies the constraint $C$ (denoted $v \models C$) if for all $x \in C$, $v(x)$ is not blank.[3] We define $v(B)$ as the graph obtained after replacing every occurrence of a variable $x$ in $B$ by $v(x)$.

A *matching* of the graph $B$ in database $D$ is a valuation $v$ such that $v(B) \subseteq \text{nf}(D)$. The matchings that interest us are those that satisfy the constraints $C$.

The semantics includes, for each blank node $N$ occurring in $H$, a Skolem function $f_N : \text{UBL}^k \to C$, where $k$ is the number of distinct variables occurring in $B$ and $C$ a set of blank nodes disjoint with any appearing in the query or the database. For each valuation $v$, $v(H)$ is the graph obtained by replacing each variable $x$ occurring in $H$ by $v(x)$ and each blank node $N$ occurring in $H$ by $f_N(v(x_1), \ldots, v(x_k))$ where $x_1, \ldots, x_k$ are the variables occurring in $B$.

DEFINITION 8. *Let $q = (H, B, P, C)$ be a query and $D$ a database. A* pre-answer *to $q$ over $D$ is the set*

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

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

NOTE 3. *Some clarifications about the notion of matching:*

*We need* nf$(D)$ *instead of just $D$ in order to deal with rdfs vocabulary because entailment in this case is characterized in terms of* nf *(cf. Theorem 8). On the other hand, using a closure $D'$ of $D$ instead of* nf$(D)$ *would not give unique answers. Consider the database $D$ defined by the graph in Example 5, and the query $(H, B)$ with $H = B = \{(a, p, ?Y), (?Y, q, d)\}$. It is not difficult to see that $\{(a, p, b), (b, q, d)\}$ should be the unique answer. But if we match against a closure of $D$, each one of the two closures of $D$ would give one more answer, and different for each closure.*

*A more general definition of matching obtained by replacing "$v(B) \subseteq \text{nf}(D)$" by "$D \models v(B)$" does not work properly because it could give infinite answers. For example, given $D = \{(a, b, c)\}$ and the query defined as $(?X, ?Y, ?Z) \leftarrow (?X, ?Y, ?Z)$, the answers would be $D$ union all triples of the form $(N, b, M)$ with $N, M$ blank nodes.*

A desirable property a query language for RDF should have is compositionality, i.e., the property that complex queries can be composed from simpler ones [4]. For this, we need to output results in the same format as input data. In our case, we can combine single answers in several different ways to obtain as final answer an RDF graph. We concentrate on two approaches to do so.

1. $\text{ans}_\cup(q, D)$ is the *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, $\text{ans}_+(q, D)$, 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.

The merge-semantics could be useful when asking a query to several sources (e.g. several different files of metadata corresponding to different web pages). In this case we do not want clashes of blank nodes of different specifications. One important drawback of the merge-semantics is that there could be no data-independent query that retrieves the whole database. An approach similar to merge-semantics can be found in query languages for semistructured data [**?**].

The union-semantics is more intuitive. First, there exists an identity query (see Note 4 below). As another illustration, consider a database $D$ which has a blank node $N$ with several properties, i.e., there are in $D$ several triples $(N, p_1, z_1), (N, p_2, z_3), \ldots$. If we follow the merge-semantics, we cannot retrieve the properties of $N$ with a data independent query. On the other hand, if we follow the union-semantics, the query $(?X, \mathit{feature}, ?Y) \leftarrow (?X, ?Y, ?Z)$ will do it.

PROPOSITION 2. *Let $q$ be a query.*

*(1) For both semantics, if $D' \models D$ then* $\text{ans}(q, D') \models \text{ans}(q, D)$.

*(2) For all $D$,* $\text{ans}_\cup(q, D) \models \text{ans}_+(q, D)$.

NOTE 4 (THE IDENTITY QUERY). *The identity query is defined as $(H, B)$ with $H = B = \{(?X, ?Y, ?Z)\}$.*

*Observe that this query works as identity only with the union-semantics. Consider the database $D = \{(X, b, c), (X, b, d)\}$. Then* $\text{ans}_\cup(q, D) = D$, *but* $\text{ans}_+(q, D) = \{(X, b, c), (Y, b, d)\}$, *which is not equivalent to $D$ because there is no map from $D$ to* $\text{ans}_+(q, D)$. *This shows also that the converse of Proposition 2, item 2, does not hold.*

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

[3] This constraint is called a *must-bind variable* in DQL [21].

## 4.3 Premises

Having premises in queries extends classical querying in several aspects: The possibility of simulating if-then queries while still remaining within the expressiveness of the language; hypothetical analysis of information; and the ability to query incomplete information by supplying information not in the database.

Our definition of premises differs from Bonner's [5] in that we have one fixed premise for the whole query. We also allow blank nodes, but not variables, in the premise.

It is important to remark that premises cannot be simulated with Datalog programs. For example consider the following query:

$(?X, \text{relative}, Mary) \leftarrow (?X, \text{relative}, Mary)$
with premise $P = \{(son, \texttt{sp}, descendant)\}$.

It is not possible to write a data-independent Datalog-like query equivalent to it. The reason is that we do not know in advance the existence, in a given database, of triples like $(descendant, \texttt{sp}, relative)$ that indirectly links "son" with "relative" via the transitive relation `sp`.

## 4.4 Reification

One of the main motivations to have blank nodes in heads of queries is the reification vocabulary as the following example shows.

EXAMPLE 9. *(Following an example in Yang and Kifer [20]). All statements made by Encyclopedia Britannica are true. If we assume that Encyclopedia Britannica is a database containing all its statements (triples), the following query would do the work:*

$$(N, \texttt{value}, \texttt{true}), (N, \texttt{type}, \texttt{stat}), (N, \texttt{subj}, ?X), (N, \texttt{pred}, ?Y), (N, \texttt{obj}, ?Z) \leftarrow (?X, ?Y, ?Z).$$

In the semantics of RDF given in [23] a statement does not have an identifier associated to it. To refer to a statement, one has to give [associate?] a name (a blank node) to it, a process known as *reification*. There can be several different such names. Observe –using Theorem 3– that a triple does not entail its reification and its reification does not entail the triple. An alternative approach is to assume that the triple itself is an object of the universe and has a unique identifier (a URI). In [20] the advantages of such an approach are argued.

This issue has some implications from a database point of view. With the semantics of [23], 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 sets 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.

# 5. QUERY CONTAINMENT

In this section we explore different notions of query containment and their characterizations. Any reasonable notion of query containment $q \sqsubseteq q'$ should embody the idea that $\text{ans}(q', D)$ comprises all the information of $\text{ans}(q, D)$. In relational databases, set-theoretical inclusion of tuples captures this requirement. When databases are viewed as knowledge bases having a notion of entailment $\models$, the information comprised by a database is all that can be entailed from it. Hence the right notion of $q \sqsubseteq q'$ is $\text{ans}(q', D) \models \text{ans}(q, D)$ for all $D$. In the relational case both notions coincide. This is not the case in our context.

In what follows we will discuss these two versions of containment which corresponds to the following:

DEFINITION 9. *Let $q, q'$ be queries.*

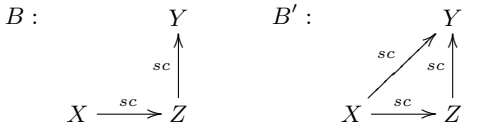
*1. $q \sqsubseteq_p q'$ iff for all databases $D$, $\text{preans}(q, D) \subseteq \text{preans}(q', D)$ modulo isomorphism, that is, for each $G \in \text{preans}(q, D)$, there is $G' \in \text{preans}(q', D)$ with $G' \cong G$.*

*2. $q \sqsubseteq_m q'$ iff for all databases $D$, $\text{ans}(q', D) \models \text{ans}(q, D)$.*

PROPOSITION 3. *$\sqsubseteq_p$ implies $\sqsubseteq_m$.*

The converse of this proposition is not true as the next example shows.

EXAMPLE 10. *When working with rdfs vocabulary, containment characterizations are more complex. In the following queries, the head is the same as the body.*




*Clearly, $q' \sqsubseteq_m q$ and $q \sqsubseteq_m q'$. But $q \not\sqsubseteq_p q'$ nor $q' \not\sqsubseteq_p q$.*

The next theorem characterizes the $\sqsubseteq_p$ notion for simple RDF graphs.

THEOREM 11. *Consider the queries $q = (H, B, P, C)$ and $q' = (H', B', P', C)$, and assume $H, H', B, B', P, P'$ are simple graphs.*

*Then $q \sqsubseteq_p q'$ if and only if for each map $\mu$ on the variables of $B$, there is a substitution (of variables and blank nodes) $\theta_\mu$ such that:*

*1. $\theta_\mu(B') \subseteq P' + (B - \mu(B, P))$, where $\mu(B, P)$ is the set of triples $t$ of $B$ such that $\mu(t) \in P$,*

*2. $\theta_\mu(H') = H$,*

*3. $\theta_\mu(C') \subseteq C$.*

The particular case of the previous theorem for queries without premises resembles containment of conjunctive relational queries: $q \sqsubseteq_p q'$ if and only if there is a substitution $\theta$ such that 1. $\theta(B') \subseteq B$; 2. $\theta(H') = H$; and 3. $\theta(C') \subseteq C$.

In the remainder of this section we study the notion $\sqsubseteq_m$. We introduce an auxiliary notion that avoids the problems that arise from blank nodes in databases (not in the query). Define $q \sqsubseteq_g q'$ iff for all ground databases $D$ (i.e. with no blank nodes) $\text{ans}(q', D) \models \text{ans}(q, D)$. Also, define $G_1 \models G_2$ for graphs $G_1, G_2$ containing variables, as $v(G_1) \models v(G_2)$, where $v$ is a valuation sending the variables to fresh constants.

PROPOSITION 4. *Let $q = (H, B)$ and $q' = (H', B')$ be queries. Then $q \sqsubseteq_g q'$ if and only if there are substitutions $\theta_1, \ldots, \theta_n$ (of variables) such that*

*1. $\theta_j(B') \subseteq \text{nf}(B)$,*

*2. $\bigcup_j \theta_j(H') \models H$.*

Notice that the set of constraints $C$ does not play any role when working with ground databases.

The next proposition shows that blank nodes in RDF databases play a similar role to ground elements.

PROPOSITION 5. *Let $q, q'$ be queries without premises and constraints. Then: $q' \sqsubseteq_g q$ if and only if $q' \sqsubseteq_m q$.*

The following theorem follows from the two previous propositions.

THEOREM 12. *Let $q = (H, B, C)$ and $q' = (H', B', C')$ be queries. Then $q \sqsubseteq_m q'$ if and only if there are substitutions $\theta_1, \ldots, \theta_n$ (of variables) such that*

*1. $\theta_j(B') \subseteq \text{nf}(B)$,*

*2. $\bigcup_j \theta_j(H') \models H$,*

*3. $\theta_j(C') \subseteq C$.*

Next we give complexity bounds for containment.

THEOREM 13. *Assume all graphs are simple. The following problems are NP-complete: 1. For queries $q_i = (H, B, C, P)$: Is $q_1 \sqsubseteq_p q_2$?; 2. For queries without premises: Is $q \sqsubseteq_m q'$?*

# 6. COMPLEXITY OF QUERY ANSWERING

## 6.1 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.

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 14. *The evaluation problem is NP-complete for the query complexity version, and polynomial for the data complexity version.*

From the proof of Theorem 14 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 normal form 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.

## 6.2 Redundancy elimination

We give some observations on redundancies in queries, databases and set of answers.

It is desirable and possible to have queries with lean heads. Otherwise, the answer generated will have redundancies which could have been avoided.

On the contrary, it is not always possible to have lean graphs in body of queries. For example, consider the query $q = (H, B, \emptyset)$, where $H = (?Course, \text{related}, \text{“DB”})$ and $B = (?Dept, \text{offers}, \text{“DB”}), (?Dept, \text{offers}, ?Course)$. $B$ is not lean and is equivalent to the lean graph $B' = (?Dept, \text{offers}, \text{“DB”})$. It turns out that there is no query equivalent to $q$ with body $B'$ (using any notion of equivalence).

Even having lean databases and queries with lean heads and bodies does not avoid redundancies in the answer set. Consider the lean graph $G_2$ in Example 1, and the query $(?Z, p, ?U) \leftarrow (?Z, p, ?U)$. The answer set is $G_1$ which is not lean.

Answers to queries in RDF usually have redundancies. Ideally, the answer set $\text{ans}(q, D)$ should reduce these redundancies to the minimum, i.e. to an equivalent lean graph. The naive approach to eliminate redundancy in answers is to compute: (1) $\text{ans}(q, D)$, and (2) a lean equivalent to $\text{ans}(q, D)$. The next theorem shows that in the worst case there is no better approach.

THEOREM 15. *Given a lean database $D$ and a query $q$, to decide whether $\text{ans}_\cup(q, D)$ is lean is coNP-complete (in the size of $D$).*

The theorem follows from the fact that there is a query that computes the identity and from Theorem 2.

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

THEOREM 16. *Given a lean database $D$ and a query $q$, deciding whether $\text{ans}_+(q, D)$ is lean can be done in polynomial time in the size of $D$.*

## 7. CONCLUSIONS AND FUTURE WORK

The RDF data model poses new challenges to query languages. We have studied some fundamental problems introduced by this model. This formalization needs to be extended to richer properties and mechanisms of current working query languages for RDF, to 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 must be studied.

**Acknowledgments.** C. Gutierrez was supported by FONDECYT No 1030810. C. Hurtado was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Mideplan. A.O. Mendelzon's visit to the University of Chile was supported by Mecesup project No. UCH0109, Millenium Nucleus, Center for Web Research (P01-029-F), and by the Natural Sciences and Engineering Research Council of Canada.

## 8. REFERENCES

[1] A. V. Aho, M. R. Garey, J. D. Ullman, *The transitive reduction of a directed graph*, SIAM J. Comput. 1 (1972) 131–137.

[2] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley Publishing Co., 1995.

[3] F. Baader, T. Nipkow, *Term Rewriting and All That*, Cambridge Univ. Press, 1998.

[4] J. Broeskstra, A. Kampman, *SeRQL: A Second Generation RDF Query Language*, SWAD-Europe Workshop on Semantic Web Storage and Retrieval, 13-14 November 2003, Vrije Universiteit, Amsterdam, Netherlands.

[5] A. Bonner, Hypothetical Datalog: Complexity and Expressibility. Proceedings ICDT 1988, 144-160.

[6] Y. Kanza, W. Nutt, Y. Sagiv, *Queries with incomplete answers over semistructured data*, Proceedings PODS 1999, 227-236.

[7] R. Fagin, Ph. G. Kolaitis, *Data Exchange: Getting to the Core*, Proceedings PODS 2003, 90-101.

[8] D. M. Gabbay, U. Reyle, *N-Prolog: an Extension of Prolog with Hypothetical Implications. I*, Journal of Logic Programming (JLP), 1(4):319-355, 1984.

[9] A. Gomez-Perez, M. C. Suarez-Figueroa, *Results of Taxonomic Evaluation of RDF(S) and DAML+OIL Ontologies using RDF(S) and DAML+OIL Validation Tools and Ontology Platforms Import Services*, II ISWC Workshop on Evaluation of Ontology-based Tools, EON2003, CEUR Workshop Proc. Vol 87.

[10] R. V. Guha, *rdfDB Query Language*, in `http://www.guha.com/rdfdb/query.html`

[11] C. Gutierrez, *Normal forms for connectedness in categories*, Annals of Pure and Applied Logic 108 (2001) 237-247.

[12] P. Hell, J. Nesetril, *The core of a graph*, Discrete Math. 109 (1992), 117-126.

[13] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, *RQL: A Declarative Query Language for RDF*, Proceedings WWW 2002, 592-603.

[14] 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.

[15] L. Miller, A. Seaborne, A. Reggiori, *Three Implementations of SquishQL, a Simple RDF Query Language*, Proc. 1st. International Semantic Web Conference, 2002, 399-403.

[16] *RDF Query Exchange Language (QEL)*, Edit. M. Nilsson, W. Siberski, `http://edutella.jxta.org/spec/qel.html`

[17] *OWL Web Ontology Language Reference*, W3C Candidate Recommendation 18 August 2003, Editors: M. Dean, G. Schreiber.

[18] E. Prud'hommeaux, B. Grosof, *RDF Query and Rules: A Framework and Survey*, `http://www.w3.org/2001/11/13-RDF-Query-Rules/`

[19] M. Sintek, S. Decker, *TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web*, Proc. 1st International Semantic Web Conference, 2002, 364-378.

[20] G. Yang, M. Kifer, *On the Semantics of Anonymous Identity and Reification* Proc. First International Conference on Ontologies, Databases and Applications of Semantics (ODBASE), 2002, 1047-1066.

[21] *DAML Query Language (DQL)*, April 2003, Abstract Specification. DAML Joint Committee, R. Fikes, P. Hayes, I. Horrocks, Ed.

[22] *Resource description framework (RDF) model and syntax specification*, Edit. O. Lassila, R. Swick, Working draft, W3C, 1998.

[23] *RDF Semantics, W3C Working Draft, 1 October 2003* Edit. Patrick Hayes

[24] *RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003*, Edit. Dan Brickley, R.V. Guha.

[25] *RDF Concepts and Abstract Syntax*, Edit. G. Klyne, J. J. Carroll. W3C Working Draft 10 October 2003.

[26] *RDF Primer*, Edit. F. Manola, E. Miller, W3C Working Draft 10 October 2003.