Querying Semantic Data on the Web*

Marcelo Arenas
PUC Chile &
U. of Oxford

Claudio Gutierrez
Comp. Science
U. de Chile

Daniel P. Miranker
U. of Texas
at Austin

Jorge Pérez
Comp. Science
U. de Chile

Juan F. Sequeda
U. of Texas
at Austin

1 Introduction

The Semantic Web is the initiative of the W3C to make information on the Web readable not only by humans but also by machines. RDF is the data model for Semantic Web data, and SPARQL is the standard query language for this data model. In recent years, we have witnessed a constant growth in the amount of RDF data available on the Web, which has motivated the theoretical study of fundamental aspects of RDF and SPARQL.

The goal of this paper is two-fold: to introduce SPARQL, which is a fundamental technology for the development of the Semantic Web, and to present some interesting and non-trivial problems on RDF data management at a Web scale, that we think the database community should address.

2 Semantic Web Data

The RDF specification [26] considers two types of values: *resource identifiers* (in the form of URIs [10]) to denote Web resources, and *literals* to denote values such as natural numbers, Booleans, and strings. In this paper, we use $\mathbf{U}$ to denote the set of all URIs and $\mathbf{L}$ to denote the set of all literals, and we assume that these two sets are disjoint. RDF also considers a special type of objects to describe *anonymous resources*, called blank nodes in the RDF data model. Essentially, blank nodes are existentially quantified variables that can be used to make statements about unknown (but existent) resources [34]. In this paper, we do not consider blank nodes, that is, we focus on what are called *ground* RDF graphs. Formally, an RDF triple is a tuple:

$$(s, p, o) \quad \in \quad \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L}),$$

where s is the *subject*, p the *predicate* and o the *object*. An RDF graph is a finite set of RDF triples.

*Database Principles Column. Column editor: Pablo Barceló, Department of Computer Science, Universidad de Chile, Santiago, Chile. E-mail: pbarcelo@dcc.uchile.cl.

Figure 2 shows an example of an RDF graph with data from the RNA Comparative Analysis Database[1], RNA Ontology[2], Gene Ontology[3], TaxonConcept[4] and DBpedia[5]. Since URIs can be long, they can be abbreviated by assigning a prefix string to a URI. For example, the prefix `tc` is assigned the string `http://lod.taxonconcept.org/ses/` in this example. Then adding another string after the prefix, separated by a colon (:), creates a new URI. For example, `tc:T9nAS` is equivalent to concatenating `T9nAS` to the string assigned to `tc`.

The RDF graph shown in Figure 2 states that the Sequence identified by `seq:237860` has a length of 118 and is part of the taxon identified by `tax:36178`, which corresponds to the following RDF triples:

(`seq:2378690`, `seq:length`, `"118"`)
(`seq:2378690`, `seq:taxonomy`, `tax:36178`)

Notice that literals, such as 118, are denoted between quotation marks (i.e. `"118"`). Additionally, `seq:237860` is located in a cell location identified by `obo:GO_0005634`, which is a sub class of `obo:GO_0043231`. Furthermore, sequence `seq:237860` is of type `seqtype:3`, which is the same as `rnao:16S_rRNA` that comes from the RNA Ontology. Consequently, `tax:36178` is the same as taxon `tc:T9nAS` that comes from the TaxonConcept ontology. Finally, the taxon `tc:T9nAS` is the same as `dbpedia:Pallid_sturgeon` from DBpedia, which is the subject of `dbpedia:Endemic_fauna_of_the_United_States`.

2.1 SPARQL 1.0: Syntax, semantics and complexity

Jointly with the release of RDF in 1999 as Recommendation of the W3C, the natural problem of querying RDF

[1] `http://www.rna.icmb.utexas.edu/DAT/`
[2] `http://bioportal.bioontology.org/ontologies/1500`
[3] `http://www.geneontology.org/`
[4] `http://www.taxonconcept.org/`
[5] `http://dbpedia.org/`

```
prefix : <http://ribs.csres.utexas.edu/rcad/>  prefix seq: <http://ribs.csres.utexas.edu/rcad/SequenceMain/>
prefix obo: <http://purl.obolibrary.org/obo/>  prefix seqtype: <http://ribs.csres.utexas.edu/rcad/SequenceType/>
prefix tc: <http://lod.taxonconcept.org/ses/>  prefix tax: <http://ribs.csres.utexas.edu/rcad/Taxonomy/>
prefix dbpedia: <http://dbpedia.org/resource/> prefix rnao: <http://purl.obolibrary.org/obo/rnao.owl#>
```

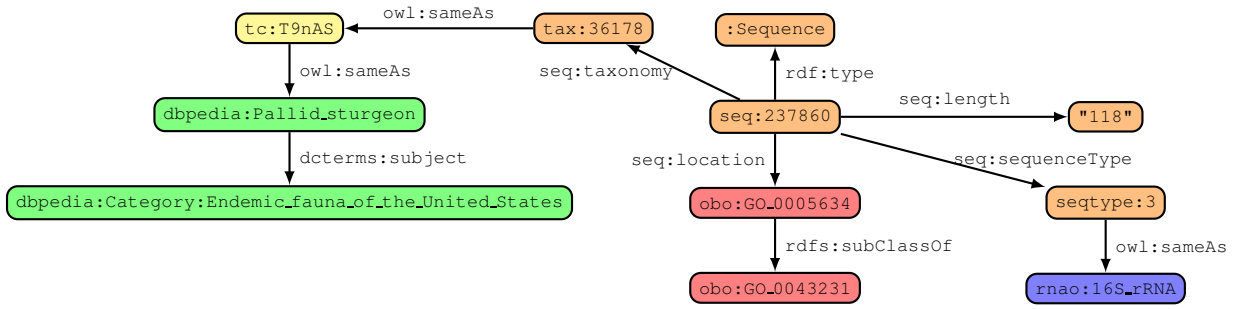

Figure 1: RDF triples containing biological information from five different sources: The RNA Comparative Analysis Database (orange nodes), The RNA Ontology (blue node), The Gene Ontology (red nodes), TaxonConcept (yellow node), and DBpedia (green nodes).

data was raised. Since then, several designs and implementations of RDF query languages have been proposed [15]. In 2004, the RDF Data Access Working Group released a first public working draft of a query language for RDF, called SPARQL [44]. Currently, SPARQL is a W3C recommendation, and has become the standard language for querying RDF data. In this section, we give an algebraic formalization of the core fragment of SPARQL, and we provide some results about the complexity of the evaluation problem for this query language. It is important to notice that there is an extended version of this query language called SPARQL 1.1 that is currently under development [18], and which is studied in Section 2.2. Thus, in this section we use the term SPARQL 1.0 to refer to the first standard version of SPARQL defined in [44].

2.1.1 Syntax and semantics of SPARQL 1.0

To present the syntax of SPARQL 1.0, we use the algebraic formalism for this query language proposed in [39, 40, 41]. More specifically, assume that $\mathbf{V}$ is an infinite set of variables disjoint from $\mathbf{U}$ and $\mathbf{L}$, and assume that the elements from $\mathbf{V}$ are prefixed by the symbol ?. Then a SPARQL 1.0 graph pattern is recursively defined as follows:

- A tuple from $(\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{L} \cup \mathbf{V})$ is a graph pattern (a *triple pattern*).
- If P_1 and P_2 are graph patterns, then the expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns.
- If P is a graph pattern and R is a built-in condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern.

Moreover, a SPARQL 1.0 query is defined by either adding the possibility of selecting some values from a graph pattern or asking whether a graph pattern has a solution (which corresponds to the notion of Boolean query):

- If P is a graph pattern and W is a finite set of variables, then $(\text{SELECT } W\ P)$ is a SPARQL 1.0 query.
- If P is a graph pattern, then $(\text{ASK } P)$ is a SPARQL 1.0 query.

Notice that the notion of built-in condition is used in the definitions of graph patterns and SPARQL 1.0 queries. A built-in condition is a Boolean combination of terms constructed by using equality ($=$) among elements of $(\mathbf{U} \cup \mathbf{L} \cup \mathbf{V})$, and the unary predicate bound over variables.[6] Formally,

- if $?X,\ ?Y \in \mathbf{V}$ and $c \in (\mathbf{U} \cup \mathbf{L})$, then $\text{bound}(?X)$, $?X = c$ and $?X = ?Y$ are built-in conditions; and
- if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions.

[6]For simplicity, we omit here other built-in predicates such as isIRI, isLiteral and isBlank, and other features such as comparisons ($<$, $>$,$\leq$, $\geq$), data type conversion and string functions. We refer the reader to [44, Section 11.3] for details.

Example 2.1 *In the running example shown in Figure 2, the following is a SPARQL 1.0 query that intuitively selects sequences that have length 118:*

(SELECT $\{S\}$
  $((?S, \texttt{seq:length}, ?L)$ FILTER $(?L = \texttt{"118"}))$)

To define the semantics of SPARQL 1.0 queries, we need to borrow some terminology from [39, 40, 41]. A *mapping* μ is a partial function $\mu : \mathbf{V} \to (\mathbf{U} \cup \mathbf{L})$. The domain of μ, denoted by $\mathrm{dom}(\mu)$, is the subset of $\mathbf{V}$ where μ is defined. Two mappings μ_1 and μ_2 are *compatible*, denoted by $\mu_1 \sim \mu_2$, when for every $?X \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$. Notice that if $\mu_1 \sim \mu_2$ holds, then $\mu_1 \cup \mu_2$ is also a mapping. Moreover, notice that two mappings with disjoint domains are always compatible, and that the empty mapping $\mu_\emptyset$ (i.e. the mapping with empty domain) is compatible with any other mapping. Finally, given a mapping μ and a set W of variables, the restriction of μ to W, denoted by $\mu_{|_W}$, is a mapping such that $\mathrm{dom}(\mu_{|_W}) = (\mathrm{dom}(\mu) \cap W)$ and $\mu_{|_W}(?X) = \mu(?X)$ for every $?X \in (\mathrm{dom}(\mu) \cap W)$.

The semantics of SPARQL 1.0 is defined by considering four basic operators on sets of mappings. More precisely, given sets Ω_1 and Ω_2 of mappings, the join of, the union of, the difference between, and the left-outer join between Ω_1 and Ω_2 are defined as follows [39, 40, 41]:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\},\\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\},\\
\Omega_1 \setminus \Omega_2 &= \{\mu \in \Omega_1 \mid \forall \mu' \in \Omega_2\colon \mu \not\sim \mu'\}\\
\Omega_1 \leftmodels \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2).
\end{aligned}$$

Notice that in the definition of $\Omega_1 \setminus \Omega_2$, notation $\mu \not\sim \mu'$ is used to indicate that mappings μ, μ' are not compatible. Intuitively, $\Omega_1 \bowtie \Omega_2$ is the set of mappings that result from extending mappings in Ω_1 with their compatible mappings in Ω_2, and $\Omega_1 \setminus \Omega_2$ is the set of mappings in Ω_1 that cannot be extended with any mapping in Ω_2. Finally, a mapping μ is in $\Omega_1 \leftmodels \Omega_2$ if it is the extension of a mapping of Ω_1 with a compatible mapping of Ω_2, or if it belongs to Ω_1 and cannot be extended with any mapping of Ω_2.

We are now ready to define the semantics of SPARQL 1.0. First, we define the semantics of built-in conditions. Given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if [39, 40, 41]:

- R is $?X = c$, where $c \in \mathbf{U}$, $?X \in \mathrm{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X \in \mathrm{dom}(\mu)$, $?Y \in \mathrm{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$;
- R is $(\neg R_1)$, and it is not the case that $\mu \models R_1$;
- R is $(R_1 \vee R_2)$, and $\mu \models R_1$ or $\mu \models R_2$;
- R is $(R_1 \wedge R_2)$, $\mu \models R_1$ and $\mu \models R_2$.

Second, we define the semantics of graph patterns. Given a triple pattern t, denote by $\mathrm{var}(t)$ the set of variables mentioned in t, and given a mapping μ such that $\mathrm{var}(t) \subseteq \mathrm{dom}(\mu)$, denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. Then given an RDF graph G and a graph pattern P, the evaluation of P over G, denoted by $[\![P]\!]_G$, is defined recursively as follows [39, 40, 41]:

- if P is a triple pattern t, then $[\![P]\!]_G = \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\}$.
- if P is $(P_1 \text{ AND } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$.
- if P is $(P_1 \text{ OPT } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \leftmodels [\![P_2]\!]_G$.
- if P is $(P_1 \text{ UNION } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$.
- if P is $(P_1 \text{ FILTER } R)$, then $[\![P]\!]_G = \{\mu \in [\![P_1]\!]_G \mid \mu \models R\}$.

Moreover, given a SPARQL 1.0 query $Q = (\text{SELECT } W\ P)$, define the evaluation of Q over an RDF graph G as $[\![Q]\!]_G = \{\mu_{|_W} \mid \mu \in [\![P]\!]_G\}$ [40]. Finally, given a SPARQL 1.0 query $Q = (\text{ASK } P)$, define the evaluation of Q over an RDF graph G as:

$$[\![Q]\!]_G = \begin{cases} \textit{yes} & [\![P]\!]_G \neq \emptyset \\ \textit{no} & \text{otherwise} \end{cases}$$

It should be noticed that the idea behind the OPT operator is to allow for optional matching of graph patterns. Consider graph pattern expression $(P_1 \text{ OPT } P_2)$ and let μ_1 be a mapping in $[\![P_1]\!]_G$. If there exists a mapping $\mu_2 \in [\![P_2]\!]_G$ such that μ_1 and μ_2 are compatible, then $\mu_1 \cup \mu_2$ belongs to $[\![(P_1 \text{ OPT } P_2)]\!]_G$. But if no such a mapping μ_2 exists, then μ_1 belongs to $[\![(P_1 \text{ OPT } P_2)]\!]_G$. Thus, operator OPT allows information to be added to a mapping μ if the information is available, instead of just rejecting μ whenever some part of the pattern does not match. This feature of optional matching is crucial in Semantic Web applications, and more specifically in RDF

data management, where it is assumed that every application have only partial knowledge about the resources being managed.

Assume that μ is a mapping such that $\mathrm{dom}(\mu) = \{?X_1, \ldots, ?X_k\}$ and $\mu(?X_i) = a_i$ for every $i \in \{1, \ldots, k\}$. From now on, we also use notation $\{?X_1 \to a_1, \ldots, ?X_k \to a_k\}$ to represent such a mapping.

Example 2.2 *Consider again the RDF graph G shown in Figure 2. The following SPARQL 1.0 graph pattern is used to return the list of sequences in this graph, together with the taxa they are part of and their lengths:*

$$P_1 = ((?S, \texttt{seq:taxonomy}, ?T) \text{ AND } (?S, \texttt{seq:length}, ?L)).$$

In this case, we have that $[\![P_1]\!]_G = \{\mu_1\}$, where μ_1 is the mapping $\{?S \to \texttt{seq:237860}, ?T \to \texttt{tax:36178}, ?L \to \texttt{"118"}\}$. Moreover, the following SPARQL 1.0 graph pattern is used to retrieve the list of sequences in G, together with their locations and names, if the latter information is available:

$$P_2 = ((?S, \texttt{seq:location}, ?L) \text{ OPT } (?S, \texttt{seq:name}, ?N)).$$

In this case, we have that $[\![P_2]\!]_G = \{\mu_2\}$, where μ_2 is the mapping $\{?S \to \texttt{seq:237860}, ?L \to \texttt{obo:GO\_0005634}\}$. Notice that in the mapping μ_2 we do not have any value associated with the variable $?N$, as we have no information about the name of the sequence with id `seq:237860` *in the graph G. Also notice that if P_2 is replaced by the graph pattern:*

$$P_3 = ((?S, \texttt{seq:location}, ?L) \text{ AND } (?S, \texttt{seq:name}, ?N)),$$

then we obtain the empty set of mappings when evaluating P_3 over G, as in this case we do not use the optional feature of SPARQL 1.0 when retrieving the names of the sequences in G.

2.1.2 Complexity of the evaluation problem

In this section, we present a survey of the results on the complexity of the evaluation of SPARQL 1.0 graph patterns, that is, without considering the SELECT operator. In this study, we consider several fragments built incrementally, and present complexity results for each such fragment. Among other results, we show that the complexity of the evaluation problem for general SPARQL 1.0 graph patterns is PSPACE-complete, and that this high complexity is obtained as a consequence of unlimited use of nested optional parts.

As is customary when studying the complexity of the evaluation problem for a query language [49], we consider its associated decision problem. We denote this problem by EVALUATION and we define it as follows:

PROBLEM	:	EVALUATION
INPUT	:	An RDF graph G, a graph pattern P and a mapping μ
QUESTION	:	Is $\mu \in [\![P]\!]_G$?

Notice that the pattern and the graph are both input for EVALUATION. Thus, we study the *combined complexity* of the query language [49].

We start this study by considering the fragment consisting of graph pattern expressions constructed by using only the operators AND and FILTER. In what follows, we call AND-FILTER to this fragment.[7] Given an RDF graph G, a graph pattern P in this fragment and a mapping μ, it is possible to efficiently check whether $\mu \in [\![P]\!]_G$ by using the following simple algorithm [39]. First, for each triple t in P, verify whether $\mu(t) \in G$. If this is not the case, then return *false*. Otherwise, by using a bottom-up approach, verify whether the expression generated by instantiating the variables in P according to μ satisfies the FILTER conditions in P. If this is the case, then return *true*, else return *false*. Thus, assuming that $|G|$ denotes the size of an RDF graph G and $|P|$ denotes the size of a graph pattern P, we have that:

Theorem 2.3 ([39, 41]) EVALUATION *can be solved in time $O(|P| \cdot |G|)$ for the* AND-FILTER *fragment of* SPARQL 1.0.

We continue this study by adding the UNION operator to the AND-FILTER fragment. It is important to notice that the inclusion of UNION in SPARQL 1.0 was one of the most controversial issues in the definition of the language. The following theorem shows that the inclusion of this operator makes the evaluation problem for SPARQL 1.0 graph patterns considerably harder.

Theorem 2.4 ([39, 41]) EVALUATION *is* NP*-complete for the* AND-FILTER-UNION *fragment of* SPARQL 1.0.

In [45], the authors strengthen the above result by showing that the complexity of evaluating graph pattern expressions constructed by using only AND and UNION operators is already NP-hard. Thus, we have the following result.

[7] We use a similar notation for other combinations of SPARQL 1.0 operators. For example, the AND-FILTER-UNION fragment of SPARQL 1.0 is the fragment consisting of all the graph patterns constructed by using only the operators AND, FILTER and UNION.

Theorem 2.5 ([45]) EVALUATION *is* NP*-complete for the* AND-UNION *fragment of* SPARQL 1.0*.*

We now consider the OPT operator. The following theorem proved in [39] shows that when considering all the operators in SPARQL 1.0 graph patterns, the evaluation problem becomes considerably harder.

Theorem 2.6 ([39, 41]) EVALUATION *is* PSPACE*-complete.*

To prove the PSPACE-hardness of EVALUATION, the authors show in [41] how to reduce in polynomial time the quantified boolean formula problem (QBF) to EVALUATION. An instance of QBF is a quantified propositional formula φ of the form $\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_m \exists y_m \, \psi$, where ψ is a quantifier-free formula of the form $C_1 \wedge \cdots \wedge C_n$, with each C_i ($i \in \{1, \ldots, n\}$) being a disjunction of literals, that is, a disjunction of propositional variables x_i and y_j, and negations of propositional variables. Then the problem is to verify whether φ is valid. It is known that QBF is PSPACE-complete [16]. In the encoding presented in [41], the authors use a fixed RDF graph G and a fixed mapping μ. Then they encode formula φ with a pattern P_φ that uses nested OPT operators to encode the *quantifier alternation* of φ, and a graph pattern without OPT to encode the satisfiability of formula ψ. By using a similar idea, it is shown in [45] how to encode formulas φ and ψ by using only the OPT operator, thus strengthening Theorem 2.6.

Theorem 2.7 ([45]) EVALUATION *is* PSPACE*-complete even for the* OPT *fragment of* SPARQL 1.0*.*

When verifying whether $\mu \in [\![P]\!]_G$, it is natural to assume that the size of P is considerably smaller than the size of G. This assumption is formalized by means of the notion of data complexity [49], which is defined as the complexity of the evaluation problem for a fixed query. More precisely, for the case of SPARQL 1.0, given a graph pattern expression P, the evaluation problem for P, denoted by EVALUATION(P), has as input an RDF graph G and a mapping μ, and the problem is to verify whether $\mu \in [\![P]\!]_G$.

Theorem 2.8 ([41]) EVALUATION(P) *is in* LOGSPACE *for every* SPARQL 1.0 *graph pattern expression* P*.*

2.1.3 Well-designed patterns: On the use of the OPT operator in SPARQL 1.0

One of the most delicate issues in the definition of a semantics for graph pattern expressions is the semantics of the OPT operator. As we have mentioned before, the idea behind this operator is to allow for optional matching of patterns, that is, to allow information to be added if it is available, instead of just rejecting whenever some part of a pattern does not match. However, this intuition fails in some simple examples.

Example 2.9 *Consider again the RDF graph shown in Figure 2, and let P be the following graph pattern:*

$$((?X, \texttt{seq:length}, \texttt{"118"}) \text{ AND } (?Y, \texttt{owl:sameAs}, \texttt{tc:T9nAS})),$$

which retrieves in $?X$ the identifiers of the sequences that have length 118 and retrieves in $?Y$ the identifiers of the taxa that are the same as the taxon with identifier tc:T9nAS. Moreover, let P' be the graph pattern obtained from P by replacing the triple pattern $(?Y, \texttt{owl:sameAs}, \texttt{tc:T9nAS})$ by the following graph pattern using the OPT *operator:*

$$((?Y, \texttt{owl:sameAs}, \texttt{tc:T9nAS}) \text{ OPT } (?X, \texttt{seq:label}, ?Z)). \quad (1)$$

Finally, let G be an RDF graph obtained by adding the triple

$$(\texttt{seq:504416}, \texttt{seq:label}, \texttt{"ID 504416"})$$

to the RDF graph shown in Figure 2. Given that P' is obtain by adding an OPT *operator to P, one would expect that the information extracted from an RDF graph by using P is contained in the information extracted by using P'. However, one can use RDF graph G to show that this is not the case in general. In fact, it is straightforward to see that $[\![P]\!]_G = \{\mu\}$, where μ is the mapping $\{?X \to \texttt{seq:237860}, ?Y \to \texttt{tax:36178}\}$, while $[\![P']\!]_G = \emptyset$. To see why the latter holds, notice that the evaluation of triple pattern $(?X, \texttt{seq:length}, \texttt{"118"})$ over G gives as result a set consisting of mapping $\mu_1 = \{?X \to \texttt{seq:237860}\}$, while the evaluation of graph pattern* (1) *over G gives as result a set consisting of mapping $\mu_2 = \{?X \to \texttt{seq:504416}, ?Y \to \texttt{tax:36178}, ?Z \to \texttt{"ID 504416"}\}$, and mappings μ_1, μ_2 are not compatible as $\mu_1(?X) \neq \mu_2(?X)$.*

The pattern P' in the previous example is unnatural as the triple pattern $(?X, \texttt{seq:label}, ?Z)$ seems to be giving optional information for $(?X, \texttt{seq:length}, \texttt{"118"})$ (they share variable $?X$), but in P' it is giving optional information for $(?Y, \texttt{owl:sameAs}, \texttt{tc:T9nAS})$ (see pattern (1) above). In fact, it is possible to find a common characteristic in the examples that contradict the intuition behind the definition of the OPT operator: A graph pattern

P mentions an expression $Q = (P_1 \text{ OPT } P_2)$ and a variable $?X$ occurring both inside P_2 and outside Q, but not occurring in P_1. In [39], the authors introduce a syntactic restriction that forbids the form of interaction between variables discussed above. To present this restriction, we need to introduce some terminology. A graph pattern P is said to be safe if for every sub-pattern $(P_1 \text{ FILTER } R)$ of P, every variable mentioned in R is also mentioned in P_1. Then a graph pattern P in the AND-FILTER-OPT fragment of SPARQL 1.0 is said to be *well designed* [39] if: P is safe, and for every sub-pattern $Q = (P_1 \text{ OPT } P_2)$ of P and variable $?X$, if $?X$ occurs both inside P_2 and outside Q, then it also occurs in P_1. For instance, pattern P' in Example 2.9 is not well designed.

In [39], the notion of being well designed was introduced in an attempt to regulate the scope of variables in the OPT operator. Interestingly, well-designed graph patterns also have good properties regarding the complexity of the evaluation problem. As shown in Theorem 2.7, the evaluation problem for SPARQL 1.0 is PSPACE-complete even if only the OPT operator is considered. However,

Theorem 2.10 ([41]) EVALUATION *is coNP-complete for the fragment of* SPARQL 1.0 *consisting of well-designed patterns.*

It is important to notice that it was also shown in [39, 41, 11, 32] that well-designed patterns are suitable for reordering and optimization, demonstrating the significance of this class of queries from a practical point of view.

2.2 SPARQL 1.1

The SPARQL Recommendation [44] is not the last step towards the definition of the right language for querying RDF, and the W3C groups involved in the design of the language are currently working on the new version of the standard, the upcoming SPARQL 1.1 [18]. This new version will include several interesting and useful features for querying RDF. Among the multiple design issues to be considered, there are three important problems that have been in the focus of attention: federation of queries, the use of navigation capabilities and the possibility of nesting queries. These features have a clear motivation in the context of querying distributed graph-shaped linked data. In this section, we study these features paying special attention to the theoretical and practical challenges that arise from them. It is important to mention that due to the lack of space, we do not cover in this section other important features of SPARQL 1.1 like the use of aggregates and negation, and the inclusion in the language of some entailment regimes [17, 30] to deal with the RDFS [26] and OWL [38, 28] vocabularies.

2.2.1 Federation

Since the release of SPARQL 1.0 in 2008, the Web has witnessed a constant growth in the amount of RDF data publicly available on-line. Nowadays, several RDF repositories provide SPARQL interfaces to directly querying their data, which has led the W3C to standardize some constructs for accessing these repositories by means of so called SPARQL endpoints. All these constructs are part of the federation extensions of SPARQL 1.1 [18, 43], which extends the syntax of SPARQL 1.0 graph patterns presented in Section 2.1 by including the following rule:

- If P is a graph pattern and $c \in \mathbf{U} \cup \mathbf{V}$ then $(\text{SERVICE } c\, P)$ is a graph pattern.

In the above expression, P is a graph pattern expression that has to be evaluated over the SPARQL endpoint represented by c. Notice that c can be a variable, thus the definition of the semantics of the SERVICE operator is not immediately evident. To formalize this semantics, assume the existence of a partial function $\text{ep}(\cdot)$ from the set of URIs to the set of all RDF graphs such that for every $c \in \mathbf{U}$, if $\text{ep}(c)$ is defined, then $\text{ep}(c)$ is the RDF graph associated with the endpoint accessible via URI c. Then given an RDF graph G and a graph pattern $P = (\text{SERVICE } c\, P_1)$, the evaluation of P over G, denoted by $[\![P]\!]_G$, is defined by considering the following cases:

- if $c \in \text{dom}(\text{ep})$, then $[\![P]\!]_G = [\![P_1]\!]_{\text{ep}(c)}$;
- if $c \in \mathbf{U} \setminus \text{dom}(\text{ep})$, then $[\![P]\!]_G = \{\mu_\emptyset\}$ (recall that $\mu_\emptyset$ is the mapping with empty domain); and
- if $c \in \mathbf{V}$, then

$$[\![P]\!]_G \quad = \quad \bigcup_{a \in \text{dom}(\text{ep})} \left([\![P_1]\!]_{\text{ep}(a)} \bowtie \{\mu_{c \to a}\} \right),$$

  where $\mu_{c \to a}$ is a mapping such that $\text{dom}(\mu_{c \to a}) = \{c\}$ and $\mu_{c \to a}(c) = a$.

The previous definition was proposed in [11, 12] to formalize the semantics for the SERVICE operator introduced in [43]. The goal of this definition is to state in an unambiguous way what the result of evaluating an expression containing the operator SERVICE should be, and as such it should not be considered as a straightforward basis for the implementation of the language. In fact, a direct implementation of the semantics for $(\text{SERVICE } ?X\ P)$ would involve evaluating P in every possible SPARQL endpoint, which is obviously infeasible in practice.

Given the definition of the semantics of the SERVICE operator, it is natural to ask in which cases a query containing a graph pattern (SERVICE $?X$ P_1) can be evaluated in practice. This issue was considered in [11, 12], where the authors study some restrictions that ensure that SERVICE patterns can be evaluated by only considering a finite set of SPARQL endpoints. More specifically, the first restriction considered in [11, 12] is based on a notion of boundedness, which is formalized as follows. A variable $?X$ is said to be bound [11, 12] in a graph pattern P if for every RDF graph G and every $\mu \in [\![P]\!]_G$, it holds that $?X \in \mathrm{dom}(\mu)$ and $\mu(?X)$ is mentioned in G. Then one can ensure that a SPARQL pattern P can be evaluated in practice by imposing the restriction that for every sub-pattern (SERVICE $?X$ P_1) of P, it holds that $?X$ is bound in P. Unfortunately, this simple condition turned out to be not completely appropriate, as shown in the following example.

Example 2.11 *Assume first that P_1 is the following graph pattern:*

$$P_1 = [(?X, \texttt{service\_description}, ?Z) \text{ UNION } ((?X, \texttt{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \texttt{email}, ?E)))].$$

That is, either $?X$ and $?Z$ store the name of a SPARQL *endpoint and a description of its functionalities, or $?X$ and $?Y$ store the name of a* SPARQL *endpoint and the IRI where it is located (together with a list of names and email addresses retrieved from that location). Variable $?Y$ is not bound in P_1. However, there is a simple strategy that ensures that P_1 can be evaluated over an RDF graph G: first compute $[\![(?X, \texttt{service\_description}, ?Z)]\!]_G$, then compute $[\![(?X, \texttt{service\_address}, ?Y)]\!]_G$, and finally for every μ in the set of mappings $[\![(?X, \texttt{service\_address}, ?Y)]\!]_G$, compute $[\![(\text{SERVICE } a\ (?N, \texttt{email}, ?E))]\!]_G$ with $a = \mu(?Y)$. In fact, the reason why P_1 can be evaluated in this case is that $?Y$ is bound in the following sub-pattern of P_1:*

$$((?X, \texttt{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \texttt{email}, ?E))).$$

As a second example, assume that G is an RDF graph that uses triples of the form $(a_1, \texttt{related\_with}, a_2)$ to indicate that the SPARQL endpoints located at the IRIs a_1 and a_2 store related data. Moreover, assume that P_2 is the following graph pattern:

$$[(?U_1, \texttt{related\_with}, ?U_2) \text{ AND } (\text{SERVICE } ?U_1\ ((?N, \texttt{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2\ (?N, \texttt{phone}, ?F))))].$$

When this query is evaluated over the RDF graph G, it returns for every tuple $(a_1, \texttt{related\_with}, a_2)$ in G, the list of names and email addresses that that can be retrieved from the SPARQL *endpoint located at a_1, together with the phone number for each person in this list for which this data can be retrieved from the* SPARQL *endpoint located at a_2 (recall that pattern (SERVICE $?U_2$ $(?N, \texttt{phone}, ?F)$) is nested inside the first SERVICE operator in P_2). To evaluate this query over an RDF graph, first it is necessary to determine the possible values for variable $?U_1$, and then to submit the query*

$$((?N, \texttt{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2\ (?N, \texttt{phone}, ?F))) \quad (2)$$

to each one of the endpoints located at the IRIs stored in $?U_1$. In this case, variable $?U_2$ is bound in P_2. However, this variable is not bound in the graph pattern (2)*, which has to be evaluated in some of the* SPARQL *endpoints stored in the RDF graph where P_2 is being evaluated, something that is infeasible in practice. It is important to notice that the difficulties in evaluating P_2 are caused by the nesting of* SERVICE *operators (more precisely, by the fact that P_2 has a sub-pattern of the form (SERVICE $?X_1$ Q_1), where Q_1 has in turn a sub-pattern of the form (SERVICE $?X_2$ Q_2) such that $?X_2$ is bound in P_2 but not in Q_1).*

To overcome the limitations of the notion of boundedness mentioned in the previous example, the authors introduce in [11, 12] the notion of service-boundedness. To present this notion, we need to introduce some terminology. Given a graph pattern P, assume that $\mathcal{T}(P)$ is the *parse* tree of P, in which every node corresponds to a sub-pattern of P. For example, Figure 2 shows the parse tree of a graph pattern P. In this figure, $u_1, u_2, u_3, u_4, u_5, u_6$ are the identifiers of the nodes of the tree, which are labeled with the sub-patterns of P. It is important to notice that this tree does not make any distinction between the different operators in SPARQL, it just uses the child relation to store the structure of the sub-patterns of a SPARQL query. Then a graph pattern P is said to be service-bound [11, 12] if for every node u of $\mathcal{T}(P)$ with label (SERVICE $?X$ P_1), it holds that:

- there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $?X$ is bound in P_2;
- P_1 is service-bound.

For example, query Q in Figure 2 is service-bound. In fact, the first condition above is satisfied as u_5 is the only node in $\mathcal{T}(Q)$ having as label a SERVICE

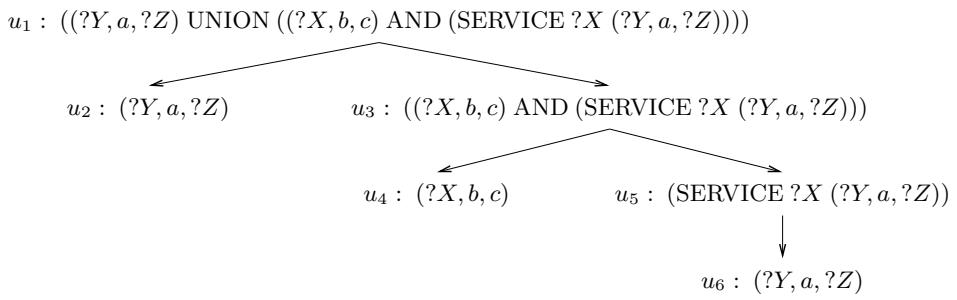

Figure 2: Parse tree $\mathcal{T}(P)$ of a graph pattern $P = [(?Y, a, ?Z)$ UNION $((?X, b, c)$ AND (SERVICE $?X$ $(?Y, a, ?Z))))]$.

graph pattern, in this case (SERVICE $?X$ $(?Y, a, ?Z)$), and for the node u_3, it holds that: u_3 is an ancestor of u_5 in $\mathcal{T}(P)$, the label of u_3 is $P = ((?X, b, c)$ AND (SERVICE $?X$ $(?Y, a, ?Z)))$ and $?X$ is bound in P. Moreover, the second condition above is satisfied as the sub-pattern $(?Y, a, ?Z)$ of the label of u_5 is also service-bound.

The notion of service-boundedness captures our intuition about the condition that a SPARQL query containing the SERVICE operator should satisfy. Unfortunately, the following theorem shows that such a condition is undecidable and, thus, a SPARQL query engine would not be able to check it in order to ensure that a query can be evaluated.

Theorem 2.12 ([11, 12]) *The problem of verifying, given a* SPARQL 1.1 *query Q, whether Q is service-bound is undecidable.*

Given this undecidability result, the authors proposed in [11, 12] a decidable sufficient condition for service-boundedness, which is formalized as follows. Let P be a graph pattern. Then the set of strongly bound variables in P, denoted by $\mathrm{SB}(P)$, is recursively defined as follows:

- if $P = t$, where t is a triple pattern, then $\mathrm{SB}(P) = \mathrm{var}(t)$;
- if $P = (P_1$ AND $P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \mathrm{SB}(P_2)$;
- if $P = (P_1$ UNION $P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cap \mathrm{SB}(P_2)$;
- if $P = (P_1$ OPT $P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$;
- if $P = (P_1$ FILTER $R)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$;
- if $P = (\mathrm{SERVICE}\ c\ P_1)$, with $c \in \mathbf{U} \cup \mathbf{V}$, then $\mathrm{SB}(P) = \emptyset$;

Moreover, graph pattern P is said to be service-safe [11, 12] if for every node u of $\mathcal{T}(P)$ with label (SERVICE $?X$ P_1), it holds that:

- there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $?X \in \mathrm{SB}(P_2)$;
- P_1 is service-safe.

That is, the notion of service-safeness is obtained from the notion of service-boundedness by replacing the restriction that variables are bound by the syntactic restriction that variables are strongly bound. In fact, it is possible to prove that service-safeness is a sufficient condition for service-boundedness.

Proposition 2.13 ([11, 12]) *If a graph pattern P is service-safe, then P is service-bound.*

It is easy to see that one can efficiently verify whether a graph pattern is service-safe. In fact, the notion of service-safeness is used in the system presented in [11, 12] to verify that a graph pattern can be evaluated in practice.

2.2.2 Property paths

Navigational features have been largely recognized as fundamental for graph database query languages. This fact has motivated several authors to propose RDF query languages with navigational capabilities [37, 2, 29, 6, 3, 42], and, in fact, it was the motivation to include the property-path feature in SPARQL 1.1 [18]. Property paths are essentially regular expressions, that are used to retrieve pairs of nodes from an RDF graph if they are connected by paths conforming to those expressions. In this section, we formalize the syntax and semantics of property paths, and study the complexity of evaluating them. It is important to mention that this formalization considers a set semantics for SPARQL queries, so it does not suffer from the complexity issues identified in [8, 33].

According to [18], a property path is recursively defined as follows: (1) if $u \in \mathbf{U}$, then u is a property path, and (2) if p_1 and p_2 are property paths, then $(p_1|p_2)$, (p_1/p_2) and (p_1^*) are property paths. Thus, from a syntactical point of view, property paths are regular expressions over the vocabulary $\mathbf{U}$, being $|$ disjunction, $/$ concatenation and $(\ )^*$ the Kleene star. It should be noticed that the definition of property paths in [18] includes some additional features that are common in regular expressions, such as $p?$ (zero or one occurrences of p) and p^+ (one or more occurrences of p). In this section, we focus on the core operators $|$, $/$ and $(\ )^*$, as the other operators can be easily defined in terms of them.

A property-path triple is a tuple t of the form (v, p, w), where $v, w \in (\mathbf{U} \cup \mathbf{V})$ and p is a property path. SPARQL 1.1 includes as atomic formulas triple patterns and property-path triples. Thus, to complete the definition of the semantics of SPARQL 1.1, we need to specify how property-path triples are evaluated over RDF graphs, that is, we need to extend the definition of the function $[\![\cdot]\!]_G$ to include property-path triples. In order to do this, we first overload the meaning of $[\![\cdot]\!]_G$ to also consider property paths. More precisely, given an RDF graph G and a property path p, the evaluation of p over G, denoted by $[\![p]\!]_G$, is recursively defined as follows:

- if $p = u$, where $u \in \mathbf{U}$, then $[\![p]\!]_G = \{(a, b) \mid (a, u, b) \in G\}$;
- if $p = (p_1|p_2)$, then $[\![p]\!]_G = [\![p_1]\!]_G \cup [\![p_2]\!]_G$;
- if $p = (p_1/p_2)$, then $[\![p]\!]_G = \{(a, b) \mid \exists u \in \mathbf{U}$: $(a, u) \in [\![p_1]\!]_G$ and $(u, b) \in [\![p_2]\!]_G\}$;
- if $p = (p_1^*)$, then

$$[\![p]\!]_G = \{(a, a) \mid a \in \mathbf{U} \text{ and } a \text{ is mentioned in } G\} \cup \left(\bigcup_{n \geq 1} [\![p_1^n]\!]_G \right),$$

  where p_1^n ($n \geq 1$) is the property path obtained by concatenating n copies of p_1.

Then given an RDF graph G and a property-path triple t of the form $(?X, p, ?Y)$, the evaluation of t over G, denoted by $[\![t]\!]_G$, is defined as:

$$\{\mu \mid \mathrm{dom}(\mu) = \{?X, ?Y\} \text{ and } (\mu(?X), \mu(?Y)) \in [\![p]\!]_G\}.$$

Moreover, the semantics of a property-path triple of the form either $(a, p, ?Y)$ or $(?X, p, b)$ or (a, p, b), where $a, b \in \mathbf{U}$, is defined in an analogous way. Notice that for every property-path triple t of the form (v, u, w), where $u \in \mathbf{U}$ and $v, w \in (\mathbf{U} \cup \mathbf{V})$, the semantics of t according to the previous definition coincides with the semantics for t if we consider it as a triple pattern.

To study the complexity of evaluating property paths, we define the following decision problem.

PROBLEM	:	EVALUATIONPROPERTYPATH
INPUT	:	An RDF graph G, a property-path triple t and a mapping μ
OUTPUT	:	Is $\mu \in [\![t]\!]_G$?

Notice that with EVALUATIONPROPERTYPATH, we are measuring the combined complexity of evaluating a property-path triple. The following result shows that EVALUATIONPROPERTYPATH is tractable. This is a corollary of some well-known results on graph databases (e.g. see Section 3.1 in [42]). In the result, we use $|G|$ to denote the size of an RDF graph G and $|t|$ to denote the size of a property-path triple t.

Proposition 2.14 EVALUATIONPROPERTYPATH *can be solved in time* $O(|G| \cdot |t|)$.

Thus, the use of property-path triples under the semantics presented in this section does not significantly increase the complexity of the evaluation problem for SPARQL.

2.2.3 Sub-queries

The advantages of having subqueries and composition in a query language are well known; among the most important for SPARQL we can mention incorporation of views, reuse of queries, query rewriting and optimization, and facilitating distributed queries.

SPARQL 1.0 only allows SELECT as the outermost operator in a query (see Section 2.1.1). On the other hand, motivated by the advantages of having subqueries in a query languages, SPARQL 1.1 allows the possibility of nesting SELECT operators. More precisely, if W is a finite set of variables and P is a graph pattern, then (SELECT W P) is a graph pattern in SPARQL 1.1 [18]. Moreover, the evaluation of such an expression over an RDF graph G is defined exactly as for the case of SPARQL 1.0: $[\![(\text{SELECT } W \ P)]\!]_G = \{\mu_{|W} \mid \mu \in [\![P]\!]_G\}$.

Assume that $?X$ is a variable occurring in a graph pattern P, W is a set of variables not including $?X$ and Q is a SPARQL 1.1 query mentioning graph pattern (SELECT W P). Due to the semantics of SPARQL 1.1, the value of $?X$ cannot be used in the remaining part of Q after evaluating (SELECT W P). As an example of this, recall that a graph pattern expression $P_1 = (?X, a, ?Y)$ AND

(SELECT $\{?X\}$ $(?X, b, ?Y)$) is equivalent to $P_2 = (?X, a, ?Y)$ AND (SELECT $\{?X\}$ $(?X, b, ?Z)$) according to the semantics of SPARQL 1.1 (that is, for every RDF graph G, it holds that $[\![P_1]\!]_G = [\![P_2]\!]_G$). Hence, the two occurrences of the variable $?Y$ in P_1 are not correlated.

It is not clear whether there is a natural way to correlate variables when using sub-queries in SPARQL 1.1, a functionality that has proved to be very useful in other query languages such as SQL. This drawback, and other limitations of the sub-query functionality of SPARQL 1.1, are studied in [4, 5], where the authors propose some extensions to SPARQL 1.1 to solve these problems. In what follows, we present one of these additions, and show how it can be used to correlate variables in a natural way. More precisely, the following rule is included in [4, 5] when defining graph patterns: If P_1, P_2 are graph patterns, then $(P_1$ FILTER $(\text{ASK } P_2))$, $(P_1$ FILTER $\neg(\text{ASK } P_2))$ are graph patterns.

To define the semantics of the expressions just presented, we need to introduce some terminology. Given a graph pattern P and a mapping μ, define $\mu(P)$ as the graph pattern obtained from P by replacing every variables $?X \in \text{dom}(\mu)$ occurring in P by $\mu(P)$. Then given an RDF graph G:

$$[\![P_1 \text{ FILTER } (\text{ASK } P_2))]\!]_G = \{\mu \in [\![P_1]\!]_G \mid [\![(\text{ASK } \mu(P_2))]\!]_G = \textit{yes}\}$$

$$[\![P_1 \text{ FILTER } \neg(\text{ASK } P_2))]\!]_G = \{\mu \in [\![P_1]\!]_G \mid [\![(\text{ASK } \mu(P_2))]\!]_G = \textit{no}\}$$

In the following example, we show a query where the possibility of correlating variables is needed, and we show how it can be expressed by using the extension to SPARQL 1.1 just introduced.

Example 2.15 *Assume that we have an RDF graph storing bibliographic data. In this graph, a triple of the form* $(a, \texttt{name}, b)$ *is used to indicate that* b *is the name of an author with identifier* a*, and a triple of the form* $(a, \texttt{series}, b)$ *is used to indicate that* a *is an identifier of a particular edition of a conference with identifier* b *(for example,* $(\texttt{SIGMOD\_11}, \texttt{series}, \texttt{SIGMOD})$ *indicates that* `SIGMOD_11` *is a particular edition of* `SIGMOD`*, in this case the 2011 edition). Moreover, a triple of the form* $(a, \texttt{isPartOf}, b)$ *is used in* G *to indicate that the article with identifier* a *was published in the conference with identifier* b*, and a triple of the form* $(a, \texttt{isAuthorOf}, b)$ *is used to indicate that* a *is the identifier of one of the authors of the article with identifier* b*.*

Assume that we want to retrieve from G *the list of authors who have published a paper in every edition of* `SIGMOD`*. Given a particular author identifier* `id`*, we can retrieve the* `SIGMOD` *editions where she/he did not publish a paper by using the following graph pattern:*

$$(?C, \texttt{series}, \texttt{SIGMOD}) \text{ FILTER } \neg(\text{ASK } ((?P, \texttt{isPartOf}, ?C) \text{ AND } (\texttt{id}, \texttt{isAuthorOf}, ?P)))$$

Thus, the following graph pattern can be used to answer our initial query, where identifier `id` *is replaced by a variable* $?A$*:*

$$(\text{SELECT } \{?N\} \; (?A, \texttt{name}, ?N) \text{ FILTER } \neg(\text{ASK } (?C, \texttt{series}, \texttt{SIGMOD}) \text{ FILTER } \neg(\text{ASK } ((?P, \texttt{isPartOf}, ?C) \text{ AND } (?A, \texttt{isAuthorOf}, ?P)))))$$

3 The Challenges of Data Management at Web Scale

Since its creation, in the early nineties, the Web has been the object of study of the database community in areas such as querying the Web, information extraction and integration, website restructuring, semi-structured data models and query languages, etc. Although aware that database techniques were not "the magic bullet that will solve all Web management information problems", most of this research focused in extending classical database techniques to this new scenario [50].

Since the early 2000 we are witnessing the emergence of the tip of an iceberg showing that drastic changes are happening to the area of Web data management. If we had to summarize them in one sentence, it would be: *real distribution of big data*.

A nice laboratory for these trends is Linked Data. Linked Data defines a set of best practices in order to treat data as a distributed interconnected graph, just as the Web, through hyperlinks [27]. Linked Data is based on the RDF data model which uses URIs. By definition, each URI will be associated with an Internet server. The Linked Data principles stipulate that when a URI is dereferenced, the server should return a set of RDF triples [9]. Those triples, in turn, may contain URIs for different servers. Thus, there is a potential for a triple on one server to logically connect to a triple on another server, such that additional graph structured data may be gathered from distributed servers. This is shown in Figure 2, where an RDF graph is composed of data coming from five different servers. Therefore, heterogeneous distributed datasets, with their own schemas, coming form diverse sources, are being linked together enabling a Web of Data.

Linked Data has highlighted aspects of the cycle of data management that in the classical setting did not occur, did not have relevance, or were addressed by other communities. In what follows, we list some challenges of data management at Web scale, with the goal of showing the reader that there are lots of interesting and non-trivial problems to solve in this area.

Publication: Publishing means to prepare data for public exposure. Berners-Lee introduced the Linked Data principles consisting of four rules [9]: 1) Use URIs as names for things, 2) Use HTTP URIs, 3) When a URI is dereferenced, provide useful information in RDF and 4) Include links to other URIs so more things can be discovered. If we assume distributed publication, the issues of handling identifiers and mapping data to RDF have to be addressed.

URIs are global unique identifiers of resources. How these URIs should be created? And given that a concept can have several URIs identifying it, how can different URIs that identify the same concept be managed and controlled? Additionally, given that Linked Data is based on the RDF data model, data in different formats must be mapped to RDF. How can different formats (relational database, logs, XML, spreadsheets, csv, etc) be mapped into the RDF model? Consequently, a schema must be chosen to describe the data. Which schemas should be chosen? How are schemas mapped at a Web-scale? Mapping relational data to RDF has fostered standardizations [7, 13] and the study of fundamental properties and optimizations [46, 47].

Discovery: Distributed publication implies the notion of discovery. One approach to discover data on the Web is to follow the same approach that is done currently on the Web: crawl webpages by following the links. This means that data must be stored in centralized datasources giving the advantage that data can be accessed quickly and statistics can be created to enable discovery [19]. However, the opportunity to access fresh data is missing and discovery of new data is bounded to the centralized repository.

A decentralized approach does not assume prior information about sources to be available, and executes queries directly on the web discovering new sources on the fly. This approach, also known as Link Traversal Based Query Execution, can be seen as a combination of querying and crawling [24, 25, 36]. Given a SPARQL query, if a triple satisfies just one clause, then the connected components of that triple, linked by URIs, may satisfy other query clauses. Thus, in the course of evaluating a SPARQL query, for each such URI, it may be necessary to go to a server and collect an additional set of triples.

A hybrid approach combines the two previous approaches by assuming that information about some sources is already available and more information can be obtained during query execution [31, 48].

Querying: Given a set of data sources on the Web, how can a query be executed in a reasonable amount of time over the distributed and linked data sources? What should be the syntax and semantics of a query language for the Web? Is SPARQL the right query language for this? What type of Web queries would a user like to express? What is the complexity of evaluating a query over the distributed data on the Web? What should the result of a query be? Should it be a SPARQL solution mapping or an RDF graph? Do we want a sound and complete answer? Or a few good answers quickly is enough? Models of the Web that could be used to solve these problems have been developed [35, 1], and some initial results in the context of Linked Data have been obtained [23, 20].

Navigation: The natural counterpart of querying in Linked Data is navigation. Data sources are discovered by following links, and navigating over the links among datasets. How can the scope of this navigation be defined? Does there need to be specific language to describe navigation? What if there are several alternatives during the navigation process? Which alternatives should be chosen? What if there are no alternatives? Fionda et al. introduced a declarative language that is designed to specify navigation patterns over the Web of Data [14].

Trust, Quality and Provenance: Data, and thus query results may not be considered trustworthy by certain users. On the other hand, users may want to track the provenance of data [21]. Should query results be associated with its provenance? How can a source and a query results be trusted? Should query results include their trustworthiness scores? Trust-aware extensions to SPARQL have been introduced [22], but should trust be a factor/operator of the query language?

Acknowledgments. M. Arenas and C. Gutierrez were supported by Fondecyt grant #1110287, J. Pérez was supported by Fondecyt grant #11110404, and J. F. Sequeda was supported by the NSF Graduate Research Fellowship.

References

[1] S. Abiteboul and V. Vianu. Queries and computation on the Web. *Theor. Comput. Sci.*, 239(2):231–255, 2000.

[2] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Constrained regular expressions in SPARQL. In *SWWS*, pages 91–99, 2008.

[3] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem.*, 7(2):57–73, 2009.

[4] R. Angles and C. Gutierrez. SQL nested queries in SPARQL. In *AMW*, 2010.

[5] R. Angles and C. Gutierrez. Subqueries in SPARQL. In *AMW*, 2011.

[6] K. Anyanwu, A. Maduko, and A. P. Sheth. Sparq2l: towards support for subgraph extraction queries in rdf databases. In *WWW*, pages 797–806, 2007.

[7] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. F. Sequeda. A direct mapping of relational data to RDF. W3C Recommendation 27 September 2012, http://www.w3.org/TR/rdb-direct-mapping/.

[8] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *WWW*, pages 629–638, 2012.

[9] T. Berners-Lee. Principles of design. `http://www.w3.org/DesignIssues/Principles.html`.

[10] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. `http://www.ietf.org/rfc/rfc3986.txt`, 2005.

[11] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC*, 2011.

[12] C. Buil-Aranda, M. Arenas, O. Corcho, and A. Polleres. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. Submitted for journal publication.

[13] S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to RDF mapping language. W3C Recommendation 27 September 2012, http://www.w3.org/TR/r2rml/.

[14] V. Fionda, C. Gutierrez, and G. Pirró. Semantic navigation on the web of data: specification of routes, web fragments and actions. In *WWW*, pages 281–290, 2012.

[15] T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. RDF querying: Language constructs and evaluation methods compared. In *Reasoning Web*, pages 1–52, 2006.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[17] B. Glimm and C. Ogbuji. SPARQL 1.1 entailment regimes. W3C Working Draft 05 January 2012, http://www.w3.org/TR/sparql11-entailment/.

[18] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C working draft. http://www.w3.org/TR/sparql11-query/, July 2012.

[19] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, pages 411–420, 2010.

[20] A. Harth and S. Speiser. On completeness classes for query evaluation on linked data. In *AAAI*, pages 613–619, 2012.

[21] O. Hartig. Provenance information in the web of data. In *LDOW*, 2009.

[22] O. Hartig. Querying trust in RDF data with tSPARQL. In *ESWC*, pages 5–20, 2009.

[23] O. Hartig. SPARQL for a web of linked data: Semantics and computability. In *ESWC*, pages 8–23, 2012.

[24] O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL queries over the web of linked data. In *ISWC*, pages 293–309, 2009.

[25] O. Hartig and J.-C. Freytag. Foundations of traversal based query execution over linked data. In *HT*, pages 43–52, 2012.

[26] P. Hayes. RDF semantics, W3C recommendation, February 2004.

[27] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 2011.

[28] P. Hitzler, M. Krtzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web ontology language primer. W3C Recommendation 27 October 2009, http://www.w3.org/TR/owl2-primer/.

[29] K. Kochut and M. Janik. SPARQLeR: Extended sparql for semantic association discovery. In *ESWC*, pages 145–159, 2007.

[30] I. Kollia, B. Glimm, and I. Horrocks. SPARQL query answering over OWL ontologies. In *ESWC*, 2011.

[31] G. Ladwig and T. Tran. Linked data query processing strategies. In *ISWC*, 2010.

[32] A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. In *PODS*, pages 89–100, 2012.

[33] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *PODS*, pages 101–112, 2012.

[34] A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On blank nodes. In *ISWC*, pages 421–437, 2011.

[35] A. O. Mendelzon and T. Milo. Formal models of Web queries. *Inf. Syst.*, 23(8):615–637, 1998.

[36] D. P. Miranker, R. K. Depena, H. Jung, J. F. Sequeda, and C. Reyna. Diamond: A SPARQL query engine, for linked data based on the rete match. In *Workshop on Artificial Intelligence meets the Web of Data*, 2012.

[37] M. Olson and U. Ogbuji. The Versa specification. http://uche.ogbuji.net/tech/rdf/versa/etc/versa-1.0.xml.

[38] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web ontology language semantics and abstract syntax. W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-semantics/.

[39] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, pages 30–43, 2006.

[40] J. Pérez, M. Arenas, and C. Gutierrez. Semantics of SPARQL. Technical report, Universidad de Chile, 2006. Dept. Computer Science, Universidad de Chile, TR/DCC-2006-17.

[41] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[42] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[43] E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 federated query. W3C Working Draft 17 November 2011, http://www.w3.org/TR/sparql11-federated-query/.

[44] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, http://www.w3.org/TR/rdf-sparql-query/.

[45] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL query optimization. In *ICDT*, pages 4–33, 2010.

[46] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and owl. In *WWW*, 2012.

[47] J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. Technical Report TR-12-10, University of Texas at Austin, Department of Computer Sciences, 2012.

[48] J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Hybrid SPARQL queries: fresh vs. fast results. In *ISWC*, 2012.

[49] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.

[50] V. Vianu. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27:59–74, 1998.