Designing a Query Language for RDF: Marrying Open and Closed Worlds

Marcelo Arenas
Pontificia Universidad Católica de Chile
& Center for Semantic Web Research
marenas@ing.puc.cl

Martín Ugarte
Université Libre de Bruxelles
martin.ugarte@ulb.ac.be

Abstract

When querying an RDF graph, a prominent feature is the possibility of extending the answer to a query with optional information. However, the definition of this feature in SPARQL –the standard RDF query language– has raised some important issues. Most notably, the use of this feature increases the complexity of the evaluation problem, and its closed-world semantics is in conflict with the underlying open-world semantics of RDF. Many approaches for fixing such problems have been proposed, being the most prominent the introduction of the semantic notion of weakly-monotone SPARQL query. Weakly-monotone SPARQL queries have shaped the class of queries that conform to the open-world semantics of RDF. Unfortunately, finding an effective way of restricting SPARQL to the fragment of weakly-monotone queries has proven to be an elusive problem. In practice, the most widely adopted fragment for writing SPARQL queries is based on the syntactic notion of well designedness. This notion has proven to be a good approach for writing SPARQL queries, but its expressive power has yet to be fully understood.

The starting point of this paper is to understand the relation between well-designed queries and the semantic notion of weak monotonicity. It is known that every well-designed SPARQL query is weakly monotone; as our first contribution we prove that the converse does not hold, even if an extension of this notion based on the use of disjunction is considered. Given this negative result, we embark on the task of defining syntactic fragments that are weakly-monotone, and have higher expressive power than the fragment of well-designed queries. To this end, we move to a more general scenario where infinite RDF graphs are also allowed, so that interpolation techniques studied for first-order logic can be applied. With the use of these techniques, we are able to define a new operator for SPARQL that gives rise to a query language with the desired properties (over finite and infinite RDF graphs). It should be noticed that every query in this fragment is weakly monotone if we restrict to the case of finite RDF graphs. Moreover, we use this result to provide a simple characterization of the class of monotone CONSTRUCT queries, that is, the class of SPARQL queries that produce RDF graphs as output. Finally, we pinpoint the complexity of the evaluation problem for the query languages identified in the paper.

PODS '16, June 26–July 1, 2016, San Francisco, CA, USA.

ISBN 978-1-4503-4191-2/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2902251.2902298

1. INTRODUCTION

In the last fifteen years the Semantic Web initiative has received a lot of attention. From its initial steps, the goal of this initiative has been to build a World Wide Web with *machine-understandable* information [9]. To this end, a first step was to standardize a data model for the information in the Web. This gave rise to RDF, a graph-based data model for specifying relationships between resources in the Web [25]. By 2013 more than four million Web domains offered data stored as RDF graphs, creating what is known as Linked Open Data [33]. Jointly with the release of RDF as a recommendation of the World Wide Web Consortium (W3C), the natural problem of querying this data was raised. Several designs and proposals were presented to solve this issue [15], being the query language SPARQL the one that finally got more attention. SPARQL is an SQL-flavored query language for RDF that became a W3C recommendation in 2008 [32]. The current version of this language, SPARQL 1.1, was issued in 2013 [40].

The query language SPARQL was originally designed by looking at each desired feature in isolation, but it turned out to be a rather complicated language when all of these features were put together. In [27], the authors formalized the syntax and semantics of SPARQL, presenting the first step towards understanding the fundamental properties of this language. This work was followed by studies about the complexity of query evaluation [34, 23, 5, 29], query optimisation [22, 30, 12, 13], query federation [10], expressive power analysis [3, 31, 20], and provenance tracking [17, 18]. The theoretical study of SPARQL has impacted the Semantic Web in several ways, influencing the standard definition of SPARQL by the W3C and also the form in which users query RDF.

In spite of the advance in our understanding of SPARQL, there is still a fundamental issue in the definition of this language. The semantics of SPARQL is defined under a closed-world assumption, in fact there are SPARQL queries that cannot be answered without making the assumption that the unavailable data is false. However, the information in the Web is inherently incomplete and, therefore, making such assumption about unavailable data contradicts the underlying open-world semantics of RDF. To address this problem, the authors of [28] identify a condition that is satisfied by the SPARQL queries that are appropriate for the open-world semantics of RDF, namely *weak monotonicity*.

Weak monotonicity is an important notion for the study of the query language SPARQL. However, weak monotonicity is a semantic notion that does not provide much insight on how to write well-behaved queries. In fact, this notion is not well-suited for

practical applications, as the problem of verifying if a query is weakly-monotone is undecidable. Hence, finding a fragment of the class of weakly-monotone SPARQL queries with a simple syntactic definition is a fundamental task. This problem has been addressed by defining fragments of SPARQL based on some syntactic restrictions. Arguably, the most adopted of these restrictions is that of *well designedness* [28, 29, 22, 30, 1, 7, 20]. It is known that every well-designed query is weakly monotone [6], but whether the opposite direction holds is an open problem. As our first contribution, we provide a negative answer to this question proving that there are (disjunction-free) weakly-monotone queries in SPARQL that are not equivalent to any well-designed query. Moreover, we show that this is the case even if we extend well-designed queries with disjunction at the top-most level.

Given these negative results, we embark on the design of an RDF query language with a simple syntactic definition and the same expressive power as the fragment of SPARQL consisting of weakly-monotone queries. To this end, we move to a more general scenario where infinite RDF graphs are also allowed, so that interpolation techniques studied for first-order logic can be applied. Interpolation techniques have proved to be useful in establishing connections between semantic and syntactic notions for first-order logic, so they are a natural choice in this investigation. The application of these techniques to SPARQL is the most challenging contribution of this paper, and its consequences provide a significant improvement in our understanding of the notion of weak monotonicity. In particular, we make use of Lyndon's [24] and Otto's [26] interpolation theorems to obtain a result that establish a form of equivalence between the fragment of weakly-monotone SPARQL queries and the fragment of SPARQL that does not use the operator OPTIONAL (used in this query language to obtain optional information if available). This result leads to the definition of a new and simple operator for SPARQL, the not-subsumed operator (NS), that is used to remove redundant information from the answer to a query.

With the use of the operator NS, we proceed to introduce two novel fragments of weakly-monotone queries. We prove that these fragments are more expressive than the fragments based on the notion of well designedness, and we provide precise characterizations of their expressive power. In fact, we show that they capture classes of weakly-monotone queries that are widely used in practice. This, together with the fact that the syntactic definitions of these fragments are rather simple, shows that these new query languages fulfill our original goal. In fact, we think these fragments provide an interesting new approach for querying RDF graphs.

The input of a SPARQL query is an RDF graph, while its output is a set mappings. Thus, SPARQL queries cannot be composed in the sense that the result of a query cannot be used as the input of another query. To overcome this limitation, the standard definition of SPARQL by the W3C includes an operator CONSTRUCT [32, 40, 20] that can be used to produce an RDF graph as output (instead of a set of mappings). This operator is widely used in practice, so it is a relevant question whether its use in SPARQL is appropriate for the open-world semantics of RDF.

As opposed to the previous case, in the context of the CONSTRUCT operator monotonicity is the condition satisfied by the queries that are appropriate for the open-world semantics of RDF. Hence, we focus on this notion, and use the results obtained from interpolation to identify a fragment of the class of CONSTRUCT queries that captures monotonicity. This fragment has a simple syntactic definition, and, somewhat surprisingly, it uses neither the operator NS nor the operator OPTIONAL. All these properties make this fragment a promising query language that deserves further investigation.

Finally, we present a thorough study of the complexity of the evaluation problem for the query languages introduced in this paper.

Organization of the paper. We give in Section 2 the basic terminology used in the paper. Then we introduce in Section 3 the notions of well designedness and weak monotonicity, and prove that there are weakly-monotone queries in SPARQL that are not equivalent to any well-designed query. We use interpolation techniques in Section 4 to show a form of equivalence between the fragment of weakly-monotone SPARQL queries and the fragment of SPARQL that does not use the operator OPTIONAL. Inspired by this result, we define in Section 5 the operator NS. In this section, we also introduce and study two novel fragments of weakly-monotone queries. Then we consider the CONSTRUCT operator in Section 6, where we identify a query language with a simple syntactic definition that captures the class of monotone CONSTRUCT queries. Finally, we provide some concluding remarks and directions for future research in Section 8. Due to the lack of space, we only provide sketches for some of the proofs of the results in the paper. The complete proofs can be found in the extended version of the paper in https://www.dropbox.com/s/1opouuou3c4xp09/main.pdf.

2. PRELIMINARIES

In this section, we provide the basic terminology that will be used in the paper.

The Resource Description Framework (RDF) [25] is based upon *identifiers* representing Web resources. Assume that $\mathbf{I}$ is an infinite set of International Resource Identifiers (IRIs). Then a triple $(s, p, o) \in \mathbf{I} \times \mathbf{I} \times \mathbf{I}$ is called an RDF triple, where s, p and o are called the subject, predicate and object of the triple, respectively. Moreover, an RDF graph is defined to be a finite set of RDF triples. It should be noticed that constant values (like numbers and strings) and existential values (resources with unknown identifiers) are also allowed in RDF, but we disallow them here as the results of the paper are not affected by their presence. For the sake of readability, we also assume that every string can be used as an IRI, which violates the specification of these identifiers [14].

Example 2.1. Assume that we want to store in RDF information about the founders and supporters of different organizations. Then we need to state every relationship as an RDF triple; for instance, if a person s founded an organization o, then we store the triple $(s, \text{founder}, o)$. The following table stores such an RDF graph:

Subject	Predicate	Object
Gottfrid_Svartholm	founder	The_Pirate_Bay
Fredrik_Neij	founder	The_Pirate_Bay
Peter_Sunde	founder	The_Pirate_Bay
founder	sub_property	supporter
The_Pirate Bay	stands_for	sharing_rights
Carl_Lundström	supporter	The_Pirate_Bay

Notice that a resource mentioned in one triple as a property can also be mentioned in another triple as the subject or predicate. As RDF triples are relations between entities, it is also natural to represent RDF graphs as directed edge-labeled graphs, as shown in Figure 1. □

2.1 The RDF query language SPARQL

SPARQL is essentially a pattern-matching query language for RDF graphs. To define the syntax of SPARQL, assume there is an infinite set $\mathbf{V}$ of variables disjoint from $\mathbf{I}$. Elements of $\mathbf{V}$ are distinguished by using ? as a prefix (e.g. $?X$ and $?Y$ are variables).

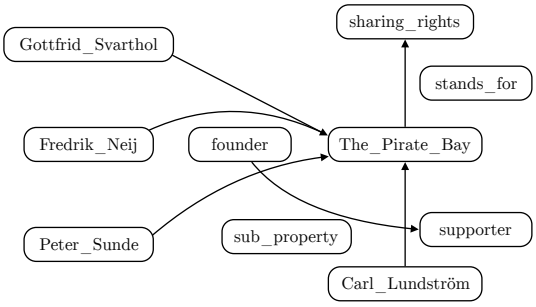

Figure 1: An RDF graph with information about founders and supporters of different organizations.

Then the set of SPARQL graph patterns is recursively defined as follows:

- A triple in $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V})$ is a graph pattern (called a triple pattern).
- If P_1, P_2 are graph patterns, then $(P_1$ UNION $P_2)$, $(P_1$ AND $P_2)$, $(P_1$ OPT $P_2)$ are graph patterns.
- If P is a graph pattern and V is a finite subset of $\mathbf{V}$, then (SELECT V WHERE P) is a graph pattern.
- If P is a graph pattern and R is a SPARQL built-in condition, then (P FILTER R) is a graph pattern.

In the previous definition, we have used the notion of SPARQL built-in condition, which is a propositional formula where atoms are equalities or inequalities over the set $(\mathbf{I} \cup \mathbf{V})$ together with some other features [32]. We restrict to the fragment of built-in conditions presented in [28], which is formally defined as follows:

- If $?X, ?Y \in \mathbf{V}$ and $c \in \mathbf{I}$, then bound($?X$), $?X = c$, $?X = ?Y$ are built in-conditions.
- 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.

For an operator O in the set {UNION, AND, OPT, FILTER, SELECT}, we say that a graph pattern P is O-free if O does not occur in P. To refer to a fragment of SPARQL in which only some operators are allowed, we use the first letter of the these operators. For example SPARQL[AFS] represents the fragment of SPARQL where only the operators AND, FILTER and SELECT are allowed.

To define the semantics of SPARQL we need to recall some further notation. If P is a graph pattern, then var(P) and $\mathbf{I}(P)$ denote the sets of all variables and IRIs mentioned in P, respectively. If R is a built-in condition, then var(R) denotes the set of all variables mentioned in R. A mapping μ is a partial function $\mu : \mathbf{V} \to \mathbf{I}$. The domain of μ is the subset of $\mathbf{V}$ where μ is defined, and is denoted by dom(μ). Given a mapping μ and a triple pattern t such that var(t) $\subseteq$ dom(μ), we have that $\mu(t)$ is the result of replacing every variable $?X \in$ var(t) by $\mu(?X)$. A mapping μ_1 is said to be compatible with a mapping μ_2, denoted by $\mu_1 \sim \mu_2$, if for every $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ it is the case that $\mu_1(?X) = \mu_2(?X)$. In this case, $\mu_1 \cup \mu_2$ denotes the extension of μ_1 to the variables in dom(μ_2) $\setminus$ dom(μ_1) defined according to μ_2. If two mappings μ_1 and μ_2 are not compatible, we write $\mu_1 \nsim \mu_2$.

Let Ω_1 and Ω_2 be two sets of mappings. Then the join of, union of, difference between, and left-outer join of Ω_1 and Ω_2 are defined, respectively, as follows [28]:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\} \\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\} \\
\Omega_1 \setminus \Omega_2 &= \{\mu \in \Omega_1 \mid \text{ for all } \mu' \in \Omega_2 : \mu \nsim \mu'\} \\
\Omega_1 \mathbin{⟕} \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2).
\end{aligned}$$

Finally, given a mapping μ and a set $V \subseteq \mathbf{V}$, the expression $\mu_{|V}$ represents the mapping that results from restricting μ to dom(μ)$\cap V$.

We have now the necessary terminology to define the semantics of SPARQL. Given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if one of the following conditions hold (omitting the usual rules for Boolean connectives):

- R is bound($?X$) and $?X \in \text{dom}(\mu)$;
- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$.

Moreover, given an RDF graph G and a SPARQL graph pattern P, the evaluation of P over G, denoted by $[\![P]\!]_G$, is recursively defined as follows:

- if P is a triple pattern, then $[\![P]\!]_G = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G\}$;
- if P is $(P_1$ AND $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$;
- if P is $(P_1$ OPT $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$;
- if P is $(P_1$ UNION $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$;
- if P is (SELECT V WHERE P'), then $[\![P]\!]_G = \{\mu_{|V} \mid \mu \in [\![P']\!]_G\}$;
- if P is (P' FILTER R), then $[\![P]\!]_G = \{\mu \mid \mu \in [\![P']\!]_G \text{ and } \mu \models R\}$

The following example illustrates the syntax and semantics of SPARQL.

Example 2.2. Let G be the RDF graph shown in Figure 1. Assume we want to retrieve the founders and supporters of organizations that stand for sharing rights. This is achieved by a graph pattern P = (SELECT $\{?p\}$ WHERE P_1), where P_1 is the the following graph pattern:

$$(?o, \text{stands\_for}, \text{sharing\_rights}) \text{ AND } ((?p, \text{founder}, ?o) \text{ UNION } (?p, \text{supporter}, ?o))$$

The evaluation of P over G is performed in a bottom-up fashion. We first evaluate the triple patterns, obtaining the following sets of mappings:

$[\![(?o, \text{stands\_for}, \text{sharing\_rights})]\!]_G =$

?o
The_Pirate_Bay

$[\![(?p, \text{founder}, ?o)]\!]_G =$

?p	?o
Gottfrid_Svartholm	The_Pirate_Bay
Fredrik_Neij	The_Pirate_Bay
Peter_Sunde	The_Pirate_Bay

$[\![(?p, \text{supporter}, ?o)]\!]_G =$

?p	?o
Carl_Lundström	The_Pirate_Bay

Then from the definition of the UNION operator, we obtain that $[\![(?p, \text{founder}, ?o) \text{ UNION } (?p, \text{supporter}, ?o)]\!]_G$ is equal to:

?p	?o
Gottfrid_Svartholm	The_Pirate_Bay
Fredrik_Neij	The_Pirate_Bay
Peter_Sunde	The_Pirate_Bay
Carl_Lundström	The_Pirate_Bay

The mappings in this table are combined with the only mapping in $[\![(?o, \text{stands\_for}, \text{sharing\_rights})]\!]_G$ through the operator AND, obtaining that $[\![P_1]\!]_G$ contains exactly the same set of mappings as in this table. Finally, the operator SELECT is used to keep only the values in the variable $?p$, thus generating the desired list of people:

?p
Gottfrid_Svartholm
Fredrik_Neij
Peter_Sunde
Carl_Lundström

□

In the previous example, we use a tabular notation for the result of a SPARQL query. In particular, a mapping μ with $\text{dom}(\mu) = \{?X_1, \ldots, ?X_n\}$ is represented as a row in a table with columns $?X_1, \ldots, ?X_n$. In what follows, we also refer to such mapping μ by using the notation $[?X_1 \to \mu(?X_1), \ldots, ?X_n \to \mu(?X_n)]$.

Finally, two graph patterns P_1 and P_2 are said to be equivalent, denoted by $P_1 \equiv P_2$, if for every RDF graph G, it holds that $[\![P_1]\!]_G = [\![P_2]\!]_G$. We use this notion of equivalence to compare fragments of SPARQL. In particular, we say that two such fragments $\mathcal{F}_1$ and $\mathcal{F}_2$ have the same expressive power if: (1) for every graph pattern $P_1 \in \mathcal{F}_1$, there exists a graph pattern $P_2 \in \mathcal{F}_2$ such that $P_1 \equiv P_2$; and (2) for every graph pattern $P_2 \in \mathcal{F}_2$, there exists a graph pattern $P_1 \in \mathcal{F}_1$ such that $P_2 \equiv P_1$. Moreover, we say $\mathcal{F}_1$ is strictly less expressive than $\mathcal{F}_2$ if: (1) for every graph pattern $P_1 \in \mathcal{F}_1$, there exists a graph pattern $P_2 \in \mathcal{F}_2$ such that $P_1 \equiv P_2$; and (2) there exists a graph pattern $P_2 \in \mathcal{F}_2$, for which there is no graph pattern $P_1 \in \mathcal{F}_1$ such that $P_2 \equiv P_1$.

3. WEAK MONOTONICITY VERSUS WELL DESIGNEDNESS

In this section, we formally introduce the notions of weak monotonicity and well designedness, and discuss their role in identifying fragments of SPARQL that are appropriate for the open-world semantics of RDF. Moreover, we show that there exist weakly monotone graph patterns that are not expressible as well-designed graph patterns, thus giving a negative answer to the question of whether these two notions are equivalent.

3.1 The notion of weak monotonicity

Intuitively, a query language is said to conform to the open-world assumption if its semantics is defined in a way such that, no assumption is made about the information that is not present when evaluating a query. This is particularly important when considering Web data, as in general we cannot make any assumption about the information that is not available. In fact, the semantics of RDF is based on an open-world assumption [25]. A notion that captures this idea for relational query languages is that of monotonicity. A query is said to be monotone if whenever it outputs an answer over a database, it outputs that same answer over all extensions of that database.

The notion of monotonicity is defined as follows for the case of SPARQL: a graph pattern P is said to be monotone if for every two RDF graphs G_1 and G_2 such that $G_1 \subseteq G_2$, it holds that $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. On the contrary to the relational scenario, monotonicity is not the right property when trying to capture the idea

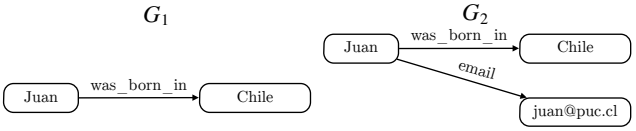

Figure 2: Two RDF graphs G_1 and G_2 such that $G_1 \subseteq G_2$.

of a query that is appropriate for the the open-world semantics of RDF. This occurs because SPARQL graph patterns allow for optional information, and hence one answer (mapping) to a query can contain *more information* than another answer to the same query. This does not occur in relational databases, where different answers to the same query have incomparable information.

Example 3.1. Consider the graph pattern

$$P = (?X, \text{was\_born\_in}, \text{Chile}) \text{ OPT } (?X, \text{email}, ?Y),$$

and let G_1 and G_2 be the RDF graphs shown in Figure 2. On the one hand, the answer to P over G_1 contains only the mapping $\mu_1 = [?X \to \text{juan}]$, as $(?X, \text{email}, ?Y)$ does not match any triple in G_1. On the other hand, the evaluation of P over G_2 contains only the mapping $\mu_2 = [?X \to \text{juan}, ?Y \to \text{juan@puc.cl}]$. Thus, we have that $[\![P]\!]_{G_1} \not\subseteq [\![P]\!]_{G_2}$ as μ_1 is not present in the answer to P over G_2, from which we conclude that P is not monotone as $G_1 \subseteq G_2$. However, in this case we can safely say that no information is lost when evaluating P over G_2, as every piece of information in μ_1 can be retrieved from the information in μ_2. □

In [6], the authors address the issue mentioned in the previous example by introducing a weaker notion of monotonicity that is appropriate for the semantics of SPARQL and, in particular, for the semantics of the OPT operator. To define this concept we need to introduce some terminology. Given two mappings μ_1 and μ_2, the mapping μ_1 is said to be subsumed by μ_2, denoted by $\mu_1 \preceq \mu_2$, if $\text{dom}(\mu_1) \subseteq \text{dom}(\mu_2)$ and $\mu_1(?X) = \mu_2(?X)$ for every $?X \in \text{dom}(\mu_1)$. Moreover, μ_1 is said to be properly subsumed by μ_2, denoted by $\mu_1 \prec \mu_2$, if $\mu_1 \preceq \mu_2$ and $\mu_1 \neq \mu_2$. Finally, given two sets of mappings Ω_1 and Ω_2, we have that Ω_1 is subsumed by Ω_2, denoted by $\Omega_1 \sqsubseteq \Omega_2$, if for every mapping $\mu_1 \in \Omega_1$, there is a mapping $\mu_2 \in \Omega_2$ such that $\mu_1 \preceq \mu_2$. With this notation we have the following:

Definition 3.2 ([6]). *A graph pattern P is said to be weakly monotone if for every two RDF graphs G_1 and G_2 such that $G_1 \subseteq G_2$, it is the case that $[\![P]\!]_{G_1} \sqsubseteq [\![P]\!]_{G_2}$.*

Weak-monotonicity overcomes the issues with monotonicity when capturing the idea of making no assumptions about unknown information when evaluating a SPARQL query. In fact, if a graph pattern P is weakly monotone and the evaluation of P over an RDF graph G produces a mapping μ, then the evaluation of P over any extension of G produces a mapping that contains at least as much information as μ. As an example of this, notice that the graph pattern in Example 3.1 is not monotone but weakly monotone.

3.2 The notion of well designedness

The operators in SPARQL are all positive in nature; in fact, neither a difference operator nor a general form of negation were included in the first version of this language [32]. However, as opposed to the intuition behind the OPT operator, there exist SPARQL graph patterns which are not weakly monotone.

Example 3.3. Let P be the graph pattern defined by:

$$P = (?X, \text{was\_born\_in}, \text{Chile}) \text{ AND } ((?Y, \text{was\_born\_in}, \text{Chile}) \text{ OPT } (?Y, \text{email}, ?X)),$$

and assume that G_1 and G_2 are the RDF graphs depicted in Figure 2. In the evaluation of P over G_1, we can see that $(?X, \text{was\_born\_in}, \text{Chile})$ and $(?Y, \text{was\_born\_in}, \text{Chile})$ match the RDF triple $(\text{Juan}, \text{was\_born\_in}, \text{Chile})$. Thus, given that the triple pattern $(?Y, \text{email}, ?X)$ does not match any triple in G_1, we obtain that $[\![P]\!]_{G_1} = \{[?X \rightarrow \text{Juan}, ?Y \rightarrow \text{Juan}]\}$. On the other hand, if we evaluate P over G_2, we obtain the same results for the triple patterns $(?X, \text{was\_born\_in}, \text{Chile})$ and $(?Y, \text{was\_born\_in}, \text{Chile})$. Nevertheless, in this case the triple $(?Y, \text{email}, ?X)$ matches $(\text{Juan}, \text{email}, \text{juan@puc.cl})$, from which we conclude that:

$$[\![(?Y, \text{was\_born\_in}, \text{Chile}) \text{ OPT } (?Y, \text{email}, ?X)]\!]_{G_2} = \{[?Y \rightarrow \text{Juan}, ?X \rightarrow \text{juan@puc.cl}]\}.$$

Hence, given that $[\![(?X, \text{was\_born\_in}, \text{Chile})]\!]_{G_2} = \{[?X \rightarrow \text{Juan}]\}$, the mappings coming from the two sides of the AND operator are not compatible. We conclude that $[\![P]\!]_{G_2} = \emptyset$ and, therefore, P is not weakly-monotone as $G_1 \subseteq G_2$ and $[\![P]\!]_{G_1} \not\subseteq [\![P]\!]_{G_2}$. □

Arguably, the graph pattern P in the previous example is unnatural. In fact, the triple pattern $(?Y, \text{email}, ?X)$ offers optional information to $(?Y, \text{was\_born\_in}, \text{Chile})$, but at the same time is intended to match the results of the triple pattern $(?X, \text{was\_born\_in}, \text{Chile})$. To avoid such patterns the notion of well designedness was introduced in [28], with the specific goal in mind of disallowing the odd use of variables shown in the previous example.

Definition 3.4 ([28]). *Let P be a graph pattern in* SPARQL[AOF]. *Then P is said to be well designed if it satisfies the following conditions:*

- *for every sub-pattern of P of the form $(P_1 \text{ FILTER } R)$, it is the case that $var(R) \subseteq var(P_1)$; and*
- *for every sub-pattern of P of the form $(P_1 \text{ OPT } P_2)$ and every variable $?X \in var(P_2)$, if $?X$ occurs in P outside $(P_1 \text{ OPT } P_2)$, then $?X \in var(P_1)$.*

For instance, the graph pattern P given in Example 3.3 is not well designed. To see why this is the case, consider the sub-pattern $(P_1 \text{ OPT } P_2)$ of P with $P_1 = (?Y, \text{was\_born\_in}, \text{Chile})$ and $P_2 = (?Y, \text{email}, ?X)$. We have that $?X \in \text{var}(P_2)$, and also that $?X$ occurs in P outside $(P_1 \text{ OPT } P_2)$ in the triple pattern $(?X, \text{was\_born\_in}, \text{Chile})$. However, $?X \notin \text{var}(P_1)$, thus violating the second condition in the definition of well designedness. On the other hand, the graph pattern shown in Example 3.1 is well designed.

3.3 On the relationship between weak monotonicity and well designedness

It is well known that well designed patterns are weakly monotone, and thus they are appropriate for the open-world semantics of RDF [28, 6]. However, the rather syntactic definition of well designedness does not shed light on how close is this notion to weak monotonicity. In fact, the question of whether every weakly-monotone graph pattern in SPARQL[AOF] is equivalent to a well-designed graph pattern is still open. The first contribution of this paper is to give a negative answer to this question.

Theorem 3.5. *There exists a weakly-monotone graph pattern in* SPARQL[AOF] *that is not equivalent to any well-designed graph pattern in* SPARQL[AOF].

The notion of well designedness was defined in [28] without considering the UNION operator. Thus, it is natural to ask whether the lack of disjunction is the reason behind Theorem 3.5. To show that this is not the case, consider the following extension of the notion of well designedness. A graph pattern in SPARQL[AUOF] is said to be well designed if it is of the form

$$P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n,$$

where every disjunct P_i $(1 \leq i \leq n)$ is a well-designed graph pattern in SPARQL[AOF]. It is important to notice that every graph pattern in this class is weakly monotone, and also that this class has been widely adopted as a good practice for writing SPARQL queries (see, e.g, [30]). However, the following result shows that this fragment of SPARQL is not expressive enough to capture weak monotonicity.

Theorem 3.6. *There exists a weakly-monotone graph pattern in* SPARQL[AUOF] *that is not equivalent to any well-designed graph pattern in* SPARQL[AUOF].

The previous theorems help to improve the understanding of the expressive power of well-designed graph patterns, and also motivate the search for more expressive weakly-monotone fragments of SPARQL with simple syntactic definitions.

4. CAPTURING WEAK MONOTONICITY UNDER SUBSUMPTION EQUIVALENCE

Interpolation techniques have proved to be useful in establishing connections between semantic and syntactic notions for first-order logic (FO); an example of this is the use of Lyndon's interpolation theorem [24] to show that the semantic notion of monotonicity for FO is characterized by the syntactic notion of being positive. Interpolation techniques have also proved to be useful in the database area, for instance to generate plans for answering queries over physical repositories [37] or with restricted access to some data sources [8]. Thus, they are an obvious choice to address the issue of defining an RDF query language that captures the fragment of weakly-monotone SPARQL queries, which is the motivation of this section.

It is important to notice that interpolations techniques are known to fail when restricted to finite models [2], so infinite database instances are considered in the investigations that use these techniques for relational databases [37, 8]. By following the same idea, we consider in this paper both finite and infinite RFD graphs, and we define an *unrestricted* RDF graph as a (possible infinite) subset of $\mathbf{I} \times \mathbf{I} \times \mathbf{I}$. It is also important to notice that in this new setting, the semantics of SPARQL is defined in the same way as for the finite case. Moreover, the notions of weak monotonicity and equivalence of graph patterns are also defined as for the finite case; but to avoid confusion we say that a graph pattern P is unrestricted weakly monotone if for every pair G_1, G_2 of unrestricted RDF graphs such that $G_1 \subseteq G_2$, it holds that $[\![P]\!]_{G_1} \sqsubseteq [\![P]\!]_{G_2}$, and we use notation $P_1 \equiv^{\text{inf}} P_2$ if for every unrestricted RDF graph G, it holds that $[\![P_1]\!]_G = [\![P_2]\!]_G$. Finally, it should also be observed that if a graph pattern is unrestricted weakly monotone then it is also weakly monotone. Therefore, any query language obtained by using the results of this section is appropriate for the open-world semantics of RDF, as every graph pattern in it would be weakly monotone (in the sense defined in the previous section).

Before stating the main result of this section, we need to dig deeper into the notion of equivalence for SPARQL graph patterns. So far we have considered the usual definition of equivalence for graph patterns P_1 and P_2, which imposes the condition that the

set of mappings $[\![P_1]\!]_G$ and $[\![P_2]\!]_G$ contain exactly the same elements for every (unrestricted) RDF graph G. But we have argued in Section 3 that if a mapping μ_1 is subsumed by a mapping μ_2, then μ_2 contains at least as much information as μ_1, so if $[\![P_1]\!]_G \sqsubseteq [\![P_2]\!]_G$ and $[\![P_2]\!]_G \sqsubseteq [\![P_1]\!]_G$ then we can claim that the set of mappings $[\![P_1]\!]_G$ and $[\![P_2]\!]_G$ are equally informative. Hence, it is also natural to consider a notion of equivalence of SPARQL graph patterns based on subsumption. More precisely, two graph patterns P_1 and P_2 are said to be subsumption-equivalent [1, 7], denoted by $P_1 \equiv_s P_2$ (resp., $P_1 \equiv_s^{\inf} P_2$), if for every RDF graph G (resp., unrestricted RDF graph G), it holds that $[\![P_1]\!]_G \sqsubseteq [\![P_2]\!]_G$ and $[\![P_2]\!]_G \sqsubseteq [\![P_1]\!]_G$.

We are finally ready to present the main result of this paper. Notice that this result is stated in terms of the notion of subsumption-equivalence, which is a concept that has received a lot of attention in the last few years [22, 30, 1, 7].

Theorem 4.1. *For every unrestricted weakly-monotone graph pattern P, there exists a graph pattern Q in* SPARQL[AUFS] *such that $P \equiv_s^{\inf} Q$.*

As a corollary of this theorem and the fact that every graph pattern in SPARQL[AUFS] is unrestricted weakly monotone (in fact, monotone), we obtain the following result that shows that the query language SPARQL[AUFS] captures the fragment of unrestricted weakly-monotone SPARQL queries under subsumption equivalence.

Corollary 4.2. *A* SPARQL *graph pattern P is unrestricted weakly-monotone if and only if there exists a graph pattern Q in* SPARQL[AUFS] *such that $P \equiv_s^{\inf} Q$.*

Proof Sketch of Theorem 4.1. This theorem is obtained by applying Lyndon's [24] and Otto's [26] interpolation theorems. Here we provide a sketch of the proof of this result.

Let P be a graph pattern. We first need to provide a translation from RDF and SPARQL to a setting in FO. This transformation is inspired by the translations given in [4, 31, 20], but with some modifications needed to apply interpolation techniques. Define $\mathcal{L}^P_{\text{RDF}}$ as the vocabulary that contains a ternary relation symbol T, a unary relation symbol Dom, a constant symbol c_i for each $i \in \mathbf{I}(P)$, and a constant symbol n. We say that an $\mathcal{L}^P_{\text{RDF}}$-structure $\mathfrak{A} = \langle D, T^{\mathfrak{A}}, Dom^{\mathfrak{A}}, \{c_i^{\mathfrak{A}}\}_{i\in I}, n^{\mathfrak{A}}\rangle$ corresponds to an unrestricted RDF graph G if

- D is a set of IRIs plus an additional element N;
- $G = T^{\mathfrak{A}} \cap (Dom^{\mathfrak{A}} \times Dom^{\mathfrak{A}} \times Dom^{\mathfrak{A}})$;
- for every $i \in \mathbf{I}(G)$, it is the case that $c_i^{\mathfrak{A}} = i$; and
- $n^{\mathfrak{A}} = N$ and N occurs neither in $Dom^{\mathfrak{A}}$ nor in $T^{\mathfrak{A}}$.

For every graph pattern P and unrestricted RDF graph G, there is an infinite set of $\mathcal{L}^P_{\text{RDF}}$-structures that correspond to G. We denote this set by $\mathcal{A}^P_G$. At this stage, the constant n and the unary relation Dom might seem unnecessary; their importance will become clear later.

Now we need to define a relation between mappings and tuples. To this end, assume an arbitrary order $\leq$ over the set of variables $\mathbf{V}$. Let $\{?X_1, \ldots, ?X_\ell\}$ be the set of variables in P ordered under $\leq$. Given a mapping $\mu : \text{var}(P) \to \mathbf{I}$, we define the extension of μ to var(P) as the function $\mu^P : \text{var}(P) \to \mathbf{I}$ such that $\mu^P(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu)$, and $\mu^P(?X) = N$ for $?X \in \text{var}(P) \setminus \text{dom}(\mu)$. Using this function we define the tuple corresponding to μ as $t^P_\mu = (\mu^P(?X_1), \ldots, \mu^P(?X_\ell))$. We extend the notion of subsumption to tuples: given two tuples $\bar{a} = (a_1, \ldots, a_\ell)$ and $\bar{b} = (b_1, \ldots, b_\ell)$, we define $\bar{a} \preceq \bar{b}$ as

$$\bar{a} \preceq \bar{b} \quad = \quad \bigwedge_{i\in[1..\ell]} a_i = b_i \vee a_i = N.$$

With the previous FO setting, we embark on the task of defining an FO formula $\varphi_P(\bar{x})$ that is equivalent to P in the following sense: for every mapping μ, unrestricted RDF graph G and $\mathcal{L}^P_{\text{RDF}}$-structure $\mathfrak{A} \in \mathcal{A}_G$, it is the case that $\mu \in [\![P]\!]_G$ if and only if $\mathfrak{A} \models \varphi_P(t^P_\mu)$. For the application of Otto's interpolation theorem, the formula φ_P should be quantified relative to Dom, meaning that every quantifier in φ_P is either of the form $\forall x(Dom(x) \to \psi)$ or $\exists x(Dom(x) \wedge \psi)$. This condition makes the construction of φ_P a particularly technical procedure which is performed by induction over the structure of P; we skip the details here. A corollary of this construction is that for every weakly monotone graph pattern there is an equivalent FO formula with Dom-relativized quantification.

Now we have a formula φ_P that is related to P under a precise notion of equivalence and is quantified relative to Dom. The next step is to define a formula asserting that φ_P corresponds to an unrestricted weakly-monotone graph pattern, for which we need to move to a new vocabulary. Let $\mathcal{L}^{P2}_{\text{RDF}}$ be defined as $\mathcal{L}^P_{\text{RDF}} \cup \{T', Dom'\}$, where T' is a ternary relation symbol and Dom' is a unary relation symbol. This vocabulary is intended to define structures *corresponding* to a pair of unrestricted RDF graphs. We say that an $\mathcal{L}^{P2}_{\text{RDF}}$-structure $\mathfrak{A}$ corresponds to an unrestricted RDF graph G if the restriction of $\mathfrak{A}$ to $\mathcal{L}^P_{\text{RDF}}$ corresponds to G.

Now consider the following $\mathcal{L}^{P2}_{\text{RDF}}$-formula:

$$[\varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \to \exists \bar{y}\, (\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y})). \quad (1)$$

Here $\varphi_P(T', Dom', \bar{y})$ represents the formula $\varphi_P(\bar{y})$ but replacing every occurrence of T by T', and every occurrence of Dom by Dom'. Besides, the sentence $T \subseteq T'$ indicates that T is contained in T', which is expressed in FO as $\forall u \forall v \forall w\,(T(u, v, w) \to T'(u, v, w))$, and likewise for the formula $Dom \subseteq Dom'$. It is not hard to prove that (1) is satisfied by every $\mathcal{L}^{P2}_{\text{RDF}}$-structure corresponding to an RDF graph if and only if P is unrestricted weakly-monotone. However, we will see that to apply interpolation we need a tautology, and we cannot assert that (1) is a tautology since not every $\mathcal{L}^{P2}_{\text{RDF}}$-structure corresponds to an unrestricted RDF graph. To overcome this problem, we define a formula Φ_{RDF} that is satisfied precisely by those $\mathcal{L}^{P2}_{\text{RDF}}$-structures that correspond to an unrestricted RDF graph. Now we add Φ_{RDF} to the left-hand side of the implication:

$$[\Phi_{\text{RDF}} \wedge \varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \to \exists \bar{y}(\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y})). \quad (2)$$

It is not hard to prove that if P is unrestricted weakly-monotone, then this formula is a tautology.

Now we proceed to apply interpolation. Lyndon's interpolation theorem [24] asserts that if an implication $\alpha \to \beta$ is a tautology, then there is a formula θ such that $\alpha \to \theta$ and $\theta \to \beta$ are both tautologies, and every relational symbol occurring on θ must occur in both α and β. Moreover, if a relation R only occurs positively in α or β, then R can only occur positively in θ. Otto extended this result by proving that if α and β are quantified relative to a set U, then θ is also quantified relative to U [26]. From these theorems and formula (2), we can deduce the existence of a new formula

$\theta(T', Dom', \bar{x})$ such that the following formulas are tautologies:

$$[\Phi_{\text{RDF}} \wedge \varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \rightarrow \theta(T', Dom', \bar{x}) \quad (3)$$

$$\theta(T', Dom', \bar{x}) \rightarrow \exists \bar{y}\,(\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y})) \quad (4)$$

Let G be an unrestricted RDF graph. By considering an $\mathcal{L}^{P2}_{\text{RDF}}$-structure $\mathfrak{A}$ that corresponds to G such that $T^{\mathfrak{A}} = T'^{\mathfrak{A}}$ and $Dom^{\mathfrak{A}} = Dom'^{\mathfrak{A}}$, and by carefully inspecting (3), we can deduce that for every $\mathcal{L}^{P}_{\text{RDF}}$-structure $\mathfrak{A} \in \mathcal{A}_G$, if $\mathfrak{A} \models \varphi_P(T, Dom, \bar{x})$, then $\mathfrak{A} \models \theta(T, Dom, \bar{x})$. Furthermore, from the same structure and (4) we can deduce that for every $\mathcal{L}^{P}_{\text{RDF}}$-structure $\mathfrak{A} \in \mathcal{A}_G$, if $\mathfrak{A} \models \varphi_P(T, Dom, \bar{x})$, then there is a tuple $\bar{y}$ such that $\bar{x} \preceq \bar{y}$ and $\mathfrak{A} \models \varphi_P(T, Dom, \bar{y})$. We conclude that for every tuple $\bar{a}$ and every structure $\mathfrak{A}$ corresponding to an unrestricted RDF graph, it holds that $\bar{a}$ is a maximal tuple such that $\mathfrak{A} \models \varphi_P(\bar{a})$ if and only if $\bar{a}$ is a maximal tuple such that $\mathfrak{A} \models \theta(\bar{a})$ (under the order $\preceq$) .

Now we proceed to inspect the syntax of formula $\theta(T', Dom', \bar{x})$. From the construction of (2), we can see that in the left-hand side of the implication the relations T' and Dom' occur only positively. Therefore, these two relations must occur positively in $\theta(T', Dom', \bar{x})$. Moreover, every quantifier in (2) is relativized w.r.t. either Dom or Dom', and hence $\theta(T', Dom', \bar{x})$ is quantified relative to Dom'. From these two facts we are able to prove that θ is equivalent to a UCQ with inequalities θ'. This proof requires several technical details that we have to skip due to the lack of space.

The final step is to define a translation from FO to SPARQL. The goal of this translation is to transform θ' (a UCQ with inequalities) into a SPARQL[AUFS] graph pattern Q. Although this translation might sound simple, it is not straightforward to define a graph pattern equivalent to θ' under the same notion of equivalence used for P and φ_P. Therefore, we use a weaker notion of equivalence: for every RDF graph G and every structure $\mathfrak{A} \in \mathcal{A}_G$ such that $T^{\mathfrak{A}} = G$, a mapping μ belongs to $[\![Q]\!]_G$ if and only if $\mathfrak{A} \models \theta'(t^Q_\mu)$. Due to the lack of space, we omit the details of the construction of the graph pattern Q satisfying the previous condition.

Summarizing the previous procedure, given a graph pattern P we constructed an FO-formula φ_P that is equivalent to P. By using interpolation techniques, we showed the existence of a new formula θ that is equivalent in terms of the obtained maximal tuples to φ_P, over structures corresponding to unrestricted RDF graphs. We defined a UCQ with inequalities θ' equivalent to θ, from which we obtained a SPARQL[AUFS] graph pattern Q that is equivalent to θ' under a weaker notion of equivalence. By inspecting every performed step, it can be shown that for every unrestricted RDF graph G, it is the case that $[\![P]\!]_G \subseteq [\![Q]\!]_G$ and $[\![Q]\!]_G \sqsubseteq [\![P]\!]_G$. This implies that $P \equiv^{\text{inf}}_s Q$, which was to be shown. □

Recall that we introduce in Section 3 the notion of well designed graph pattern, which is a stronger condition than weak monotonicity. It is interesting to notice that Theorem 4.1 can be proved for well-designed graph patterns using techniques similar to those presented in [20]. However, it is not clear how to prove it over unrestricted weakly-monotone graph patterns without using a translation to FO and interpolation techniques.

In the rest of the paper, we will see that Theorem 4.1 turns out to be a powerful tool for characterizing and capturing semantic properties over different fragments of SPARQL. It is important to mention that Theorem 4.1 provides, for every unrestricted weakly-monotone pattern P, a subsumption-equivalent graph pattern Q that is monotone (as Q is in SPARQL[AUFS]). Hence, we will need some further operators to obtain unrestricted weakly-monotone graph patterns, which is the motivation for the next section.

5. CAPTURING WEAKLY MONOTONE FRAGMENTS OF SPARQL

The goal of this section is to introduce RDF query languages that capture important (and widely used) weakly-monotone fragments of SPARQL. Inspired by the results in the previous section, we start by defining in Section 5.1 a new operator for SPARQL. Then we use this operator in Section 5.2 to define a query language with a simple syntax and capturing the fragment of unrestricted weakly-monotone SPARQL queries where subsumed answers are not allowed. Finally, we extend this result in Section 5.3 to consider the UNION operator.

5.1 A new operator for SPARQL

The result presented in Theorem 4.1 can be reformulated in terms of the notion of *maximal* answer for a graph pattern. More precisely, given a graph pattern P and an unrestricted RDF graph G, the set of maximal answers of P over G, denoted by $[\![P]\!]^{\max}_G$, is defined as the set of mappings $\mu \in [\![P]\!]_G$ for which there is no mapping $\mu' \in [\![P]\!]_G$ such that $\mu \prec \mu'$. Then Theorem 4.1 tell us that given an unrestricted weakly-monotone graph pattern P, there exists a graph pattern Q in SPARQL[AUFS] that preserves the maximal answer to P, that is, $[\![P]\!]^{\max}_G = [\![Q]\!]^{\max}_G$ for every unrestricted RDF graph G.

The idea of preserving only the maximal answers, or removing the properly subsumed answers, naturally gives rise to a "not subsumed" (NS) operator for SPARQL. More precisely, let NS–SPARQL be the extension of SPARQL with the following recursive rule for graph patterns:

- If P is a graph pattern, then NS(P) is a graph pattern. Moreover, given an unrestricted RDF graph G:

$$[\![\text{NS}(P)]\!]_G \quad = \quad [\![P]\!]^{\max}_G.$$

A graph pattern of the form (P_1 OPT P_2) is equivalent to NS(P_1 UNION (P_1 AND P_2)). Thus, the operator NS can be simply considered as an alternative way of obtaining optional information in the context of incomplete data. In fact, a similar operator for obtaining maximal answers called *minimal union relation* was already studied in the context of relational databases with incomplete information [16].

A first question about NS–SPARQL is whether it has the same expressive power as SPARQL, in the sense that for every graph pattern P in NS–SPARQL, there exists a graph pattern Q in SPARQL such that $P \equiv Q$ (the opposite direction trivially holds as NS–SPARQL is an extension of SPARQL). We have already shown that the operator OPT can be easily simulated by using the operator NS. But unlike that case, the simulation of the operator NS in SPARQL is not trivial. In fact, this is proven by providing an algorithm that takes as input a graph pattern P in NS–SPARQL, and outputs a graph pattern Q in SPARQL such that $P \equiv Q$ and the size of Q is double-exponential in the size of P.

THEOREM 5.1. *The languages* SPARQL *and* NS–SPARQL *have the same expressive power.*

Given that NS–SPARQL is an extension of SPARQL, we have that not every graph pattern in NS–SPARQL is weakly monotone. However, in the following sections we use the operator NS to identify query languages with simple syntactic definitions, with good expressive power and whose graph patterns are all weakly monotone.

5.2 Capturing weak monotonicity for subsumption-free graph patterns

As mentioned before, the fact that the answer to a SPARQL query can contain subsumed mappings has given rise to two different notions of equivalence for graph patterns. We start our search for a fragment of NS–SPARQL capturing weak monotonicity by putting us in an scenario where subsumption is no longer an issue. More precisely, a graph pattern P in SPARQL is said to be subsumption-free if for every unrestricted RDF graph G, it holds that $[\![P]\!]_G = [\![P]\!]_G^{\max}$. Notice that for every pair P_1, P_2 of subsumption-free graph patterns, it holds that $P_1 \equiv^{\inf} P_2$ if and only if $P_1 \equiv_s^{\inf} P_2$. Moreover, also notice that subsumption-free graph patterns are the rule, not the exception, in practice.

As a corollary of Theorem 4.1, we obtain the following:

Corollary 5.2. *Let P be a subsumption-free graph pattern. If P is unrestricted weakly monotone, then there exists a graph pattern Q in* SPARQL[AUFS] *such that $P \equiv^{\inf}$ NS(Q).*

This corollary motivates the introduction of the notion of simple graph pattern.

Definition 5.3. *A graph pattern is simple if it is of the form* NS(P)*, where P is in* SPARQL[AUFS].

Let SP–SPARQL be an RDF query language consisting of all simple graph patterns, that is, every query in SP–SPARQL is of the form NS(P) with P in SPARQL[AUFS]. From Corollary 5.2 and the fact that every simple graph pattern is subsumption-free, we obtain the following corollary showing that SP–SPARQL is a query language appropriate for the open-world semantics of RDF, with a simple syntactic definition and with good expressive power:

Theorem 5.4. *Over unrestricted RDF graphs,* SP–SPARQL *has the same expressive power as the fragment of unrestricted weakly-monotone and subsumption-free* SPARQL *graph patterns.*

Notice that in this theorem we compare the expressive power of two query languages over unrestricted RDF graphs; thus, we use $\equiv^{\inf}$ instead of $\equiv$ when indicating that the query languages have the same expressive power.

We conclude this section by considering the important fragments SPARQL[AOF] and SPARQL[AFS] of SPARQL. The latter includes the class of conjunctive queries with inequalities, and it is widely used in practice. The former was extensively studied in [28], where the notion of well designedness introduced in Section 3 was originally defined.

It is easy to see that every graph pattern in SPARQL[AFS] is subsumption free. Moreover, from the results proved in [28], it is possible to conclude that every graph pattern in SPARQL[AOF] is also subsumption free. Thus, we obtain from Theorem 5.4 the following result:

Corollary 5.5. *Let P be a graph pattern in* SPARQL[AOF] *or* SPARQL[AFS]*. If P is unrestricted weakly-monotone, then there exists a graph pattern in* SP–SPARQL *such that $P \equiv^{\inf} Q$.*

Moreover, for the case of (finite) RDF graphs, it is posible to establish the following connection between well designedness and simple graph patterns:

Proposition 5.6. *The fragment of well-designed graph patterns in* SPARQL[AOF] *is strictly less expressive than* SP–SPARQL.

The existence of a graph pattern in SP–SPARQL which is not equivalent to any well-designed graph pattern in SPARQL[AOF] is not surprising, as the operator UNION is allowed in SP–SPARQL. What is interesting from the previous result is that, a well-designed graph pattern containing arbitrarily nested OPT operators can always be translated into an equivalent graph pattern with a single operator NS on the top-most level.

5.3 Including the UNION operator

In Sections 4 and 5.2, we have argued that SPARQL[AUOF] and SP–SPARQL are good query languages for the open-world semantics of RDF, in particular because of their expressive power. But these languages are incomparable, as in the former every graph pattern is monotone but not necessarily subsumption-free, while in the latter every graph pattern is subsumption-free and weakly monotone but not necessarily monotone. Thus, it is natural to ask whether there exists a query language that contains both, and where every graph pattern is still weakly monotone. This is the motivating question for this section.

We start our search by allowing the use of disjunction in simple patterns.

Definition 5.7. *A graph pattern P is an* ns-pattern *if it is of the form (P_1* UNION $\cdots$ UNION *P_n), where each P_i ($1 \leq i \leq n$) is a simple pattern.*

Let USP–SPARQL be an RDF query language consisting of all ns-patterns. As our first result, we prove that USP–SPARQL is indeed more expressive than both SPARQL[AUFS] and SP–SPARQL.

Proposition 5.8. USP–SPARQL *is strictly more expressive that both* SPARQL[AUFS] *and* SP–SPARQL.

From the fact that every simple pattern is unrestricted weakly monotone, it is easy to conclude that every ns-pattern is unrestricted weakly monotone. Thus, we conclude from the previous result that USP–SPARQL is an answer to our motivating question. Moreover, we obtain the following corollary from Theorem 4.1:

Corollary 5.9. *Let P be a graph pattern in* SPARQL*. Then P is unrestricted weakly-monotone if and only if there exists a graph pattern Q in* USP–SPARQL *such that $P \equiv_s^{\inf} Q$.*

Finally, we obtain the following characterization of the expressive power of USP–SPARQL by using Theorem 5.4:

Corollary 5.10. *Over unrestricted RDF graphs,* USP–SPARQL *has the same expressive power as the fragment of unions of unrestricted weakly-monotone and subsumption-free* SPARQL *graph patterns.*

From the results of this section, we can conclude that USP–SPARQL is also an appropriate query language for the open-world semantics of RDF, in particular because of their expressive power and the fact that every graph pattern in USP–SPARQL is weakly monotone.

6. CAPTURING CONSTRUCT QUERIES

The input of a SPARQL graph pattern is an RDF graph, while its output is a set mappings. Thus, SPARQL graph patterns cannot be composed in the sense that the result of a query cannot be used as the input of another query. Besides, SPARQL queries cannot be used to define views that will later used by other queries, a common functionality in relational database systems. To overcome this limitation, the standard definition of SPARQL by the World Wide Web Consortium includes an operator CONSTRUCT [32] that can be used to produce as output an RDF graph instead of set of mapping.

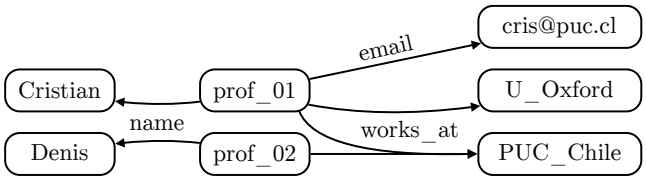

Figure 3: An RDF graph containing information about professors and universities.

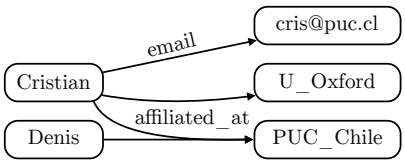

Figure 4: The RDF graph obtained by evaluating the CONSTRUCT query Q in Example 6.1 over the RDF graph in Figure 3.

This operator is widely used in practice, so it is a relevant question whether its use in SPARQL is appropriate for the open-world semantics of RDF.

The goal of this section is to answer this question. More precisely, we provide a formal definition of the CONSTRUCT operator in Section 6.1, and then we identify in Section 6.2 a query language with a simple syntactic definition and the same expressive power at the class of monotone queries using the CONSTRUCT operator. It should be noticed that Theorem 4.1 plays a crucial role in the proof of this latter result.

6.1 The CONSTRUCT operator

We follow the terminology introduced in [20] to define the CONSTRUCT operator. Let P be a SPARQL graph pattern and H a finite set of triple patterns. Then

$$Q \quad = \quad (\text{CONSTRUCT } H \text{ WHERE } P)$$

is a CONSTRUCT query, where P and H are called the graph pattern and the template of Q, respectively. Moreover, given an RDF graph G, the evaluation of Q over G is defined as follows:

$$\text{ans}(Q, G) = \{\mu(t) \mid \mu \in [\![P]\!]_G, t \in H \text{ and } \text{var}(t) \subseteq \text{dom}(\mu)\}.$$

Example 6.1. Let G be the RDF graph shown in Figure 3, which stores information about professors and universities. In this scenario, we want to construct an RDF graph that contains for each professor his/her name, the universities he/she is affiliated to, and his/her email if this information is available. We achieve this with a CONSTRUCT query Q of the form (CONSTRUCT H WHERE P), where $H = \{(?n, \text{affiliated\_to}, ?u), (?n, \text{email}, ?e)\}$ and P is the following graph pattern:

$$((?p, \text{name}, ?n) \text{ AND } (?p, \text{works\_at}, ?u)) \text{ OPT } (?p, \text{email}, ?e)$$

Notice that in this case the template H contains IRIs that are not mentioned in G. The evaluation of the graph pattern of P over G results in the following set of mappings:

	?p	?n	?u	?e
μ_1	prof_02	Denis	PUC_Chile	
μ_2	prof_01	Cristian	U_Oxford	cris@puc.cl
μ_3	prof_01	Cristian	PUC_Chile	cris@puc.cl

In the left-hand side of this table we have included names for the mappings. Then to evaluate Q, we consider the mapping in this table separately. For the mapping μ_1, we have that each variable in the triple pattern $(?n, \text{affiliated\_to}, ?u) \in H$ is contained in the domain of μ_1, so the triple $(\mu_1(?n), \text{affiliated\_to}, \mu_1(?u)) = (\text{Denis}, \text{affiliated\_to}, \text{PUC\_Chile})$ is included in the output $\text{ans}(Q, G)$. On the other hand, the variable $?e$ is not in the domain of μ_1, so no triple is produced by this mapping and the triple pattern $(?n, \text{email}, ?e) \in H$. The result of the evaluation process is the RDF graph depicted in Figure 4. Notice that the triple $(\text{Cristian}, \text{email}, \text{cris@puc.cl})$ is generated when considering the mapping μ_2 and μ_3 and the triple pattern $(?n, \text{email}, ?e) \in H$. However, as the semantics of the CONSTRUCT operator is defined as a set of triples, $(\text{Cristian}, \text{email}, \text{cris@puc.cl})$ can occur only once in the output $\text{ans}(Q, G)$. □

6.2 Capturing monotone CONSTRUCT queries

In Section 3, we concluded that the notion of monotonicity is too restrictive when trying to identify fragments of SPARQL that are appropriate for the open-world semantics of RDF. In that section, we also argue that the notion of weak monotonicity is appropriate for this goal, as to compare the answers of two graph patterns one has to consider the fact that a mapping can be more informative than another one. However, the situation is different for CONSTRUCT queries, as the answer for such a query is a set of RDF triples (an RDF graph), and an RDF triple is an atomic piece of information. In fact, one RDF triple cannot be more informative than another one. Thus, in the context of CONSTRUCT queries we consider the notion of monotonicity when trying to identify which fragment of these queries is appropriate for the open-world semantics of RDF. The notion of monotonicity is defined as follows in this context:

Definition 6.2. *A CONSTRUCT query Q is monotone if for every pair G_1, G_2 of RDF graphs such that $G_1 \subseteq G_2$, it holds that* $\text{ans}(Q, G_1) \subseteq \text{ans}(Q, G_2)$.

Exactly as in the case of weak monotonicity for graph patterns, this definition provides no insight about how to find a syntactic characterization of monotone CONSTRUCT queries, which is aggravated by the fact the problem of verifying whether a CONSTRUCT query is monotone is undecidable.[1] Nevertheless, CONSTRUCT queries enjoy some properties that allow us to address this issue. The key observation here is that when evaluating the graph pattern of a CONSTRUCT query, it suffices to look only at the non-subsumed graph patterns. This observation can be formalized by using the operator NS introduced in Section 5:

Lemma 6.3. *For every graph pattern P and template H, the query* (CONSTRUCT H WHERE P) *is equivalent to the query* (CONSTRUCT H WHERE NS(P)).

Next we show that this lemma plays a key role in the characterization of the monotone fragment of the class of CONSTRUCT

[1]This is a corollary of the fact that SPARQL and first-order logic have the same expressive power [4, 35] and the fact that the problem of verifying whether a first-order logic formula is monotone is undecidable. In turn, the latter result about monotonicity for first-order logic is a corollary of the undecidability of the finite satisfiability problem for this logic [38].

queries. As it is done in Sections 4 and 5, from now on we consider unrestricted RDF graphs, for which the semantics of the CONSTRUCT operator is defined in the same way as for the case of (finite) RDF graphs.

Lemma 6.3 has the following consequence. Assume that we have a CONSTRUCT query Q = (CONSTRUCT H WHERE P) in which P is unrestricted weakly monotone. From Theorem 4.1, we have that there exists a graph pattern $P^\star$ in SPARQL[AUFS] such that $P^\star \equiv_s^{\inf} P$. Thus, given that the condition $P^\star \equiv_s^{\inf} P$ holds if and only if the condition $\mathrm{NS}(P^\star) \equiv^{\inf} \mathrm{NS}(P)$ holds, we obtain the following by applying Lemma 6.3:

$$\begin{array}{lll} Q & = & (\text{CONSTRUCT } H \text{ WHERE } P) \\ & \equiv & (\text{CONSTRUCT } H \text{ WHERE } \mathrm{NS}(P)) \\ & \equiv^{\inf} & (\text{CONSTRUCT } H \text{ WHERE } \mathrm{NS}(P^\star)) \\ & \equiv & (\text{CONSTRUCT } H \text{ WHERE } P^\star) \end{array}$$

Hence, we obtain the following corollary.

Corollary 6.4. *Let Q = (CONSTRUCT H WHERE P) be a CONSTRUCT query in which P is unrestricted weakly monotone. Then there exists a graph pattern $P^\star$ in* SPARQL[AUFS] *such that $Q \equiv^{\inf}$ (CONSTRUCT H WHERE $P^\star$).*

It is easy to prove that if the graph pattern of a CONSTRUCT query is weakly monotone, then the CONSTRUCT query is monotone. However, the opposite direction is not true, as there exist CONSTRUCT queries that are monotone but whose graph patterns are not weakly monotone. This prevents the previous corollary from establishing a general characterization of monotone CONSTRUCT queries. However, we overcome this limitation with the following lemma.

Lemma 6.5. *For every monotone CONSTRUCT query Q, there is a template H and an unrestricted weakly monotone* SPARQL *graph pattern P such that $Q \equiv^{\inf}$ (CONSTRUCT H WHERE P).*

From this lemma and Corollary 6.4, we can finally obtain a syntactic characterization of monotonicity for CONSTRUCT queries. To simplify notation, given a set of SPARQL operators O, we denote by CONSTRUCT[O] the set of CONSTRUCT queries of the form (CONSTRUCT H WHERE P) such that P is a graph pattern in SPARQL[O].

Theorem 6.6. *Over unrestricted RDF graphs, the class of monotone CONSTRUCT queries has the same expressive power as* CONSTRUCT[AUFS].

Next we strengthen this result by proving that the SELECT operator can be removed from the fragment CONSTRUCT[AUFS].

Proposition 6.7. CONSTRUCT[AUF] *has the same expressive power as* CONSTRUCT[AUFS].

Thus, we obtain a simpler characterization of the class of monotone CONSTRUCT queries.

Corollary 6.8. *Over unrestricted RDF graphs, the class of monotone CONSTRUCT queries has the same expressive power as* CONSTRUCT[AUF].

This result culminates our study of CONSTRUCT queries. We have presented a clean and simple syntactic characterization of the class of monotone CONSTRUCT queries. It is interesting to notice that the only allowed operators in this characterization are FILTER, AND and UNION. We think this provides evidence to place CONSTRUCT[AUF] as an interesting query language for RDF that should be further investigated.

7. THE COMPLEXITY OF THE EVALUATION PROBLEM

So far, we have introduced a new operator and several syntactic fragments with good properties in terms of expressive power. At this point, it is natural to ask what is the complexity of the evaluation problem for these fragments, and whether this complexity is lower than for some well-known fragments of SPARQL. To answer this question, we study the combined complexity [39] of the evaluation problem. More precisely, we pinpoint the exact complexity of this problem for simple patterns and ns-patterns in Section 7.2, and for queries in CONSTRUCT[AUF] in Section 7.3.

7.1 A bit of background on computational complexity

We use complexity classes that might not be familiar to the reader, and hence we briefly recall their definition. In particular, we present the Boolean Hierarchy and the complexity class $\mathrm{P}_{\parallel}^{\mathrm{NP}}$.

The Boolean Hierarchy is an infinite family of complexity classes based on boolean combinations of languages in NP [41]. The most popular class in this hierarchy is DP, which consists of all languages that can be expressed as $L_1 \cap L_2$ with $L_1 \in$ NP and $L_2 \in$ coNP. The levels of the boolean hierarchy are denoted by $\{\mathrm{BH}_i\}_{i\in\mathbb{N}}$, and are recursively defined as follows:

- BH_1 is the complexity class NP.
- BH_{2k} consists of all languages that can be expressed as $L_1 \cap L_2$, where $L_1 \in \mathrm{BH}_{2k-1}$ and $L_2 \in$ coNP.
- BH_{2k+1} consists of all languages that can be expressed as $L_1 \cup L_2$, where $L_1 \in \mathrm{BH}_{2k}$ and $L_2 \in$ NP.

Notice that DP = BH_2. The complexity class $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ [19] contains all problems that can be solved in polynomial time by a Turing machine that can query a polynomial amount of times (in terms of the input's length) an NP oracle, with the restriction that all of these queries need to be issued in parallel. The parallel access to the NP oracle prevents the queries to depend on previous oracle answers. The class $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ is equivalent to $\Delta_2^p[\log n]$, the complexity class of all problems that can be solved in polynomial time by a Turing machine that can make $O(\log n)$ queries to an NP oracle, not necessarily in parallel [11].

7.2 The evaluation problem for simple patterns and ns-patterns

Consider a fragment $\mathcal{F}$ of NS–SPARQL. Then the evaluation problem for $\mathcal{F}$ is defined as follows:

Problem	:	Eval($\mathcal{F}$)
Input	:	An RDF graph G, a graph pattern $P \in \mathcal{F}$ and a mapping μ
Question	:	Is $\mu \in [\![P]\!]_G$?

As usual, we only consider inputs of finite length and, therefore, we do not consider in this section unrestricted RDF graphs. Our complexity results are built upon several studies of the complexity of evaluating SPARQL graph patterns [30, 28, 34, 6, 21]. In particular, the key ideas rely on the fact that the evaluation problem is NP-complete for SPARQL[AUFS] [34] and is coNP-complete for well-designed graph patterns in SPARQL[AOF] [27].

We start by considering the evaluation problem for simple graph patterns.

Theorem 7.1. Eval(SP–SPARQL) *is* DP*-complete.*

It is interesting to notice that the complexity of evaluating simple patterns is already higher than that of evaluating well-designed graph patterns in SPARQL[AOF], which is coNP-complete. This is to be expected as the former fragment is more expressive than the latter (see Proposition 5.6).

We continue our study by considering the evaluation problem for ns-patterns. As an ns-pattern is of the form $(P_1 \text{ UNION } \cdots \text{ UNION } P_k)$ with each P_i being a simple pattern, an important parameter for the evaluation problem in this context is the maximum number of disjunct P_i in these patterns. Let USP–SPARQL$_k$ be the fragment of USP–SPARQL consisting of all ns-patterns having at most k disjuncts each. Then we have that:

THEOREM 7.2. *For every $k > 0$, it holds that* EVAL(USP–SPARQL$_k$) *is* BH$_{2k}$*-complete.*

Finally, we obtain the following combined complexity when considering ns-patterns with an unbounded number of disjuncts.

THEOREM 7.3. EVAL(USP–SPARQL) *is* $\text{P}_{\parallel}^{\text{NP}}$*-complete.*

It is important to mention that, although the evaluation problem for well-designed graph patterns in SPARQL[AOF] is coNP-complete, these patterns do not allow for projection. If projection is allowed only on the top-most level, then the evaluation problem for well-designed graph patterns already increases to Σ_2^p-complete [22], which is higher than the complexity of the evaluation problem for USP–SPARQL (unless the polynomial-time hierarchy [36] collapses to its second level as $\text{P}_{\parallel}^{\text{NP}} \subseteq \Delta_2^P \subseteq \Sigma_2^P$).

7.3 The evaluation problem for CONSTRUCT queries

Consider a class $\mathcal{G}$ of CONSTRUCT queries. Then the evaluation problem for $\mathcal{G}$ is defined as follows:

Problem	:	EVAL($\mathcal{G}$)
Input	:	An RDF graph G, a CONSTRUCT query $Q \in \mathcal{G}$ and a tripe t
Question	:	Is $t \in \text{ans}(Q, G)$?

As mentioned before, the fragment CONSTRUCT[AUF] is one of the most important fragments defined in this paper, as it captures the class of CONSTRUCT queries that are monotone. It should be noticed that establishing the combined complexity of CONSTRUCT[AUF] is straightforward, as we rely on the fact that the evaluation problem for SPARQL[AUF] is NP-complete.

THEOREM 7.4. EVAL(CONSTRUCT[AUF]) *is* NP*-complete.*

This concludes our study of the complexity of the evaluation problem for the query languages introduced in this paper. As a final remark, it is important to mention that the results of this section provide more evidence in favor of CONSTRUCT[AUF] as an appropriate query language for RDF, as this language not only captures the notion of monotonicity for CONSTRUCT queries (over unrestricted RDF graphs), but also has an evaluation problem with a lower complexity than for well-designed graph patterns with projection on top (Σ_2^p-complete), and general CONSTRUCT queries (PSPACE-complete).

8. CONCLUDING REMARKS AND FUTURE WORK

We have presented a thorough study of the relationship between different fragments of SPARQL and the notion of weak monotonicity. We showed that one of the most adopted fragments of SPARQL, namely the class of unions of well-designed graph patterns, has lower expressive power than the fragment of weakly-monotone graph patterns. We further strengthen this result by proving that it also holds if disjunction is disallowed in both fragments. Given this negative result, we moved to a new setting in which RDF graphs can also be infinite. In this setting, we developed a framework for applying interpolation techniques for first-order logic to SPARQL, which resulted in a powerful theorem relating the fragment of weakly-monotone graph patterns with SPARQL[AUFS]. This theorem suggested the definition of the operator NS, which is a natural replacement for the operator OPTIONAL. Using the operator NS, we defined the weakly-monotone fragments of simple patterns and ns-patterns, and proved that they have higher expressive power than the fragments defined in terms of well designedness. Then we focused on the fragment of CONSTRUCT queries. We applied the results obtained from the use of interpolation techniques, from which we proved that the fragment of CONSTRUCT queries restricted to CONSTRUCT, AND, FILTER and UNION precisely characterizes the notion of monotonicity. Finally, we provided a thorough study of the combined complexity of the evaluation problem for the query languages introduced in the paper.

Our results open new research possibilities, starting by the search for useful extensions of the identified query languages. For example, allowing for projection on top of simple and ns-patterns preserves weak monotonicity, and hence this extension could lead to the definition of new weakly-monotone fragments with higher expressivity.

Finally, the focus of this paper has been mostly theoretical. Therefore, the development of more practical studies of the proposed query languages is a promising direction for future research. For instance, it is important to understand what are the practical consequences of replacing the operator OPTIONAL by the operator NS, and whether the aforementioned fragment of CONSTRUCT queries covers the needs of real-world applications. Moreover, these new lines of research are open to the development of implementations and optimizations, potentially leading to real-world applications of the techniques developed in this paper.

Acknowledgements

The authors would like to thank the anonymous referees for many helpful comments. M. Arenas was funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004, and M. Ugarte was partially funded by the scholarship CONICYT-PCHA - 21120368, by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004, and by the SPICES research project funded by Innoviris, the Brussels Institute for Research and Innovation under the Bridge strategic platform program.

9. REFERENCES

[1] S. Ahmetaj, W. Fischl, R. Pichler, M. Simkus, and S. Skritek. Towards reconciling SPARQL and certain answers. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015*, pages 23–33, 2015.

[2] M. Ajtai and Y. Gurevich. Monotone versus positive. *J. ACM*, 34(4):1004–1015, Oct. 1987.

[3] R. Angles and C. Gutierrez. The expressive power of SPARQL. In *ISWC*, pages 114–129, 2008.

[4] R. Angles and C. Gutierrez. *The expressive power of SPARQL*. Springer, 2008.

[5] 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 *Proceedings of the 21st international conference on World Wide Web*, pages 629–638. ACM, 2012.

[6] M. Arenas and J. Pérez. Querying semantic web data with sparql. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316. ACM, 2011.

[7] P. Barceló, R. Pichler, and S. Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015*, pages 131–144, 2015.

[8] M. Benedikt, B. ten Cate, and E. Tsamoura. Generating low-cost plans from proofs. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14*, pages 200–211, 2014.

[9] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[10] C. Buil-Aranda, M. Arenas, Ó. Corcho, and A. Polleres. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Sem.*, 18(1):1–17, 2013.

[11] S. R. Buss and L. Hay. On truth-table reducibility to sat. *Information and Computation*, 91(1):86 – 102, 1991.

[12] M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda. SPARQL query containment under $\mathcal{SHI}$ axioms. In *AAAI*, 2012.

[13] M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda. SPARQL query containment under RDFS entailment regime. In *IJCAR*, pages 134–148, 2012.

[14] M. Durst and M. Suignard. Rfc 3987, internationalized resource identifiers (iris), 2005.

[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. Springer, 2006.

[16] C. A. Galindo-Legaria. Outerjoins as disjunctions. *SIGMOD Rec.*, 23(2):348–358, May 1994.

[17] F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *ICDT*, 2013.

[18] H. Halpin and J. Cheney. Dynamic provenance for SPARQL updates. In *ISWC*, 2014.

[19] L. A. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299 – 322, 1989.

[20] E. V. Kostylev, J. L. Reutter, and M. Ugarte. CONSTRUCT queries in SPARQL. In *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, pages 212–229, 2015.

[21] A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS '12, pages 89–100. ACM, 2012.

[22] A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. *ACM TODS*, 38(4):25, 2013.

[23] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 101–112. ACM, 2012.

[24] R. C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific J. of Mathematics*, 9(1):129–142, 1959.

[25] F. Manola and E. Miller. RDF Primer. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[26] M. Otto. An interpolation theorem. *Bulletin of Symbolic Logic*, 6(4):447–462, 2000.

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

[28] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3), 2009.

[29] F. Picalausa and S. Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.

[30] R. Pichler and S. Skritek. Containment and equivalence of well-designed SPARQL. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–50. ACM, 2014.

[31] A. Polleres and J. P. Wallner. On the relation between SPARQL1.1 and answer set programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013.

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

[33] V. G. Ramanathan. Light at the end of the tunnel. 2013.

[34] M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33. ACM, 2010.

[35] J. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012*, pages 649–658, 2012.

[36] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[37] D. Toman and G. E. Weddell. *Fundamentals of Physical Design and Query Compilation*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[38] B. Trakhtenbrot. The impossibility of an algorithm for the decidability problem on finite classes. *Proceedings of the USSR Academy of Sciences (in Russian)*, 70(4):569–572, 1950.

[39] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[40] W3C SPARQL Working Group. SPARQL 1.1 Query language. W3C Recommendation, 21 March 2013. Available at http://www.w3.org/TR/sparql11-query/.

[41] G. Wechsung. On the boolean closure of NP. In *Fundamentals of Computation Theory*, pages 485–493. Springer, 1985.