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

MARCELO ARENAS, Pontificia Universidad Católica de Chile & Center for Semantic Web Research
MARTIN UGARTE, Université Libre de Bruxelles

When querying an Resource Description Framework (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, the most prominent being 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 article 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 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 the semantics to 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 article.

CCS Concepts: • **Information systems → Query languages for non-relational engines**; *Resource Description Framework (RDF)*; • **Theory of computation → Database query languages (principles)**; **Logic and databases**;

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, the Millennium Nucleus Center for Semantic Web Research under Grant NC120004, the SPICES research project funded by Innoviris, and the Brussels Institute for Research and Innovation under the Bridge strategic platform program.
Authors' addresses: Vicuna Mackenna 4860, Edificio San Agustin, 4to piso, Macul 7820436, Santiago, Chile; email: marenas@ing.uc.cl; M. Ugarte, Universite Libre de Bruxelles, Avenue F.D. Roosevelt 50, CP 165/15, Ixelles, 1050 Bruxelles, Belgium; email: mugartec@ulb.ac.be.

https://doi.org/10.1145/3129247

Additional Key Words and Phrases: Query languages, SPARQL, RDF, semantic Web, open-world assumption, monotonicity

ACM Reference format:
Marcelo Arenas and Martin Ugarte. 2017. Designing a Query Language for RDF: Marrying Open and Closed Worlds. *ACM Trans. Database Syst.* 42, 4, Article 21 (October 2017), 46 pages.
https://doi.org/10.1145/3129247

1 INTRODUCTION

In the last 15 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 (Berners-Lee et al. 2001). 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 (Manola and Miller 2004). By 2013, more than four million Web domains publicly offered data stored as Resource Description Framework (RDF) graphs, creating what is known as Linked Open Data (Guha 2013). Jointly with the release of RDF as a recommendation of the World Wide Web Consortium (W3C), the natural problem of querying data in RDF was raised. Several proposals were presented to solve this issue (Furche et al. 2006), the query language SPARQL being the one that finally got more attention. SPARQL is an SQL-flavored query language for RDF that became a W3C recommendation in 2008 (Prud'hommeaux and Seaborne 2008). The current version of this language, SPARQL 1.1, was issued in 2013 (Harris and Seaborne 2013).

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 Pérez et al. (2006), the authors formalized the syntax and semantics of SPARQL, presenting the first step towards understanding its fundamental properties. This work was followed by studies about the complexity of query evaluation (Schmidt et al. 2010; Losemann and Martens 2012; Arenas et al. 2012; Picalausa and Vansummeren 2011), query optimisation (Letelier et al. 2013; Pichler and Skritek 2014; Chekol et al. 2012a, 2012b), query federation (Buil-Aranda et al. 2013), expressive power analysis (Angles and Gutierrez 2008; Polleres and Wallner 2013; Kostylev et al. 2015), and provenance tracking (Geerts et al. 2013; Halpin and Cheney 2014). 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 graphs.

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 some unavailable data are 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, Pérez and collaborators identify a condition that is satisfied by those SPARQL queries that are appropriate for the open-world semantics of RDF, namely *weak monotonicity* (Pérez et al. 2009).

Although weak monotonicity is important for the study of SPARQL, it 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 whether 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* (Pérez et al. 2009; Picalausa and Vansummeren 2011; Letelier

et al. 2013; Pichler and Skritek 2014; Ahmetaj et al. 2015; Barceló et al. 2015; Kostylev et al. 2015). It is known that every well-designed SPARQL query is weakly monotone (Arenas and Pérez 2011), but whether the opposite direction holds is an open problem. As a first contribution of this article we provide a negative answer to this question. We show 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 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 article, and its consequences provide a significant improvement in our understanding of the notion of weak monotonicity. In particular, we make use of the interpolation theorems of Lyndon (1959) and Otto (2000) to obtain a result that establishes 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 when 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. We stress that 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 of 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 (Prud'hommeaux and Seaborne 2008; Harris and Seaborne 2013; Kostylev et al. 2015) 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. 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 article.

It is important to mention that the results in the article do not imply that every SPARQL graph pattern that is weakly monotone in the finite case is also weakly monotone in the unrestricted case, where infinite RDF graphs are also allowed. In fact, a simple example of a graph pattern separating the two cases can be obtained by considering a query that is not weakly monotone in

the unrestricted case, and which is based on a condition that is false for every finite RDF graph, thus leading to a trivially weakly monotone query in the finite scenario. For instance, consider a SPARQL query checking that an RDF graph represents a linear order without a maximum element; this query is not weakly monotone in the unrestricted case and is false for every finite RDF graph. During the course of this investigation, the only separating examples that we found are trivial examples as the previous one, or complex examples based on the separation between monotonicity and the syntactic property of being positive for first-order logic (Ajtai and Gurevich 1987). It remains as a fundamental open issue whether there exist queries of practical interest that are weakly monotone in the finite case but not in the unrestricted scenario.

Organization of the article. We give in Section 2 the basic terminology used in the article. 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. The complexity of the evaluation problem for the different fragments studied in the article is pinpointed in Section 7, while some practical implications of the results of the article are discussed in Section 8. Finally, we provide some concluding remarks and directions for future research in Section 9.

This article extends an earlier publication presented at the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems 2016 (Arenas and Ugarte 2016). Two of the fundamental results of this article are the proof that there exists a weakly monotone graph pattern that is not expressible as a well-designed graph pattern and the extension of this result to the case where the union operator is allowed. In Section 3, we provide these proofs with simpler versions of the separating examples, which give more information to the reader about the expressiveness of weakly monotone queries. Moreover, we include a new section in the article (Section 8) where we discuss three practical implications of the investigation carried out in this article. Finally, we provide proofs for all the results presented in Arenas and Ugarte (2016); in particular, we include an appendix where we carefully show how interpolation techniques can be used to characterize weak monotonicity (when infinite RDF graphs are allowed).

2 PRELIMINARIES

In this section, we provide the basic definitions used throughout this article concerning RDF and SPARQL.

The RDF (Manola and Miller 2004) is based on the idea that each resource on the Web should have an *identifier*. Assume that $\mathbf{I}$ is an infinite set of Internationalized 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 article 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 (Dürst and Suignard 2005).

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

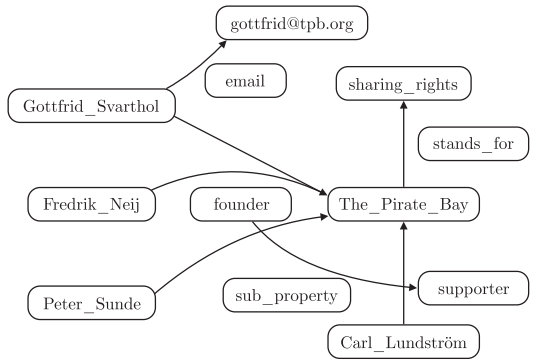

Fig. 1. An RDF graph with information about founders and supporters of organizations.

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., $?X$, $?Y$, and $?Z$ are variables in $\mathbf{V}$). 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 $(P_1 \text{ UNION } P_2)$, $(P_1 \text{ AND } P_2)$, and $(P_1 \text{ OPT } P_2)$ are graph patterns.[1]
- If P is a graph pattern and V is a finite subset of $\mathbf{V}$, then $(\text{SELECT } V \text{ WHERE } P)$ is a graph pattern.
- If P is a graph pattern and R is a SPARQL built-in condition, then $(P \text{ FILTER } R)$ 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 (Prud'hommeaux and Seaborne 2008). We restrict to the fragment of built-in conditions presented in Pérez et al. (2009), which is formally defined as follows:

[1]From now on, we use the term OPT as a shorthand of the operator OPTIONAL.

—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 occurring 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 $?X \in \text{var}(t)$ 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) \ dom(μ_1) defined according to μ_2. If two mappings μ_1 and μ_2 are not compatible, then 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 (Pérez et al. 2009):

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

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, and their emails if they are provided. This is achieved by a graph pattern P = (SELECT {$?p$} WHERE P_1) OPT ($?p$, email, $?e$),

where P_1 is defined as

$$P_1 = (?o, \text{stands\_for}, \text{sharing\_rights}) \text{ AND } ((?p, \text{founder}, ?o) \text{ UNION } (?p, \text{supporter}, ?o))$$

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

$[\![(?p, \text{email}, ?e)]\!]_G =$

?p	?e
Gottfrid_Svartholm	gottfrid@tpb.org

Now we evaluate the UNION pattern by combining both tables with founders and supporters:

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

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

The mappings presented in this latter table are combined with the only mapping in the first table through the operator AND. Notice that all of the mappings to be combined are compatible, which implies that $[\![P_1]\!]_G = [\![(?p, \text{founder}, ?o) \text{ UNION } (?p, \text{supporter}, ?o)]\!]_G$. Next, the SELECT operator is used to keep only the values in the variable $?p$, thus generating the corresponding list of people:

$[\![(\text{SELECT } \{?p\} \text{ WHERE } P_1)]\!]_G =$

?p
Gottfrid_Svartholm
Fredrik_Neij
Peter_Sunde
Carl_Lundström

Finally, these results will be optionally extended with the rows in $[\![(?p, \text{email}, ?e)]\!]_G$, producing the final result:

$[\![P]\!]_G =$

?p	?e
Gottfrid_Svartholm	gottfrid@tpb.org
Fredrik_Neij	
Peter_Sunde	
Carl_Lundström	

Note here that the OPT operator allows for people without an email to be part of the answer.

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 $[?X_1 \to \mu(?X_1), \ldots, ?X_n \to \mu(?X_n)]$.

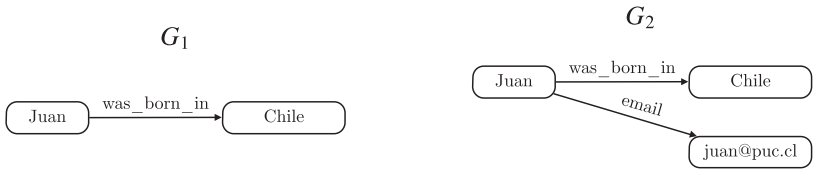

Fig. 2. Two RDF graphs G_1 and G_2 such that $G_1 \subseteq G_2$.

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 (Manola and Miller 2004). 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 $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. Contrary to what one could expect, monotonicity is not the right property when trying to capture the idea of a query that is appropriate for 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.

Example 3.1. Consider the graph pattern $P = (?X, \text{was\_born\_in}, \text{Chile})$ 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 $\mu_2 = [?X \to \text{juan}, ?Y \to \text{juan@puc.cl}]$. 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.

Arenas and Pérez 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 (Arenas and Pérez 2011). 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 (Arenas and Pérez 2011). 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 in the evaluation of 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 (Prud'hommeaux and Seaborne 2008). However, as opposed to the intuition behind the OPT operator, there exist SPARQL graph patterns that 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 $[\![P]\!]_{G_1} = \{[?X \to \text{Juan}, ?Y \to \text{Juan}]\}$. 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

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

Hence, given that $[\![(?X, \text{was\_born\_in}, \text{Chile})]\!]_{G_2} = \{[?X \to \text{Juan}]\}$, 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 $[\![P]\!]_{G_1} \not\sqsubseteq [\![P]\!]_{G_2}$.

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 Pérez et al. (2009), with the specific goal in mind of disallowing the odd use of variables shown in the previous example.

Definition 3.4 (Pérez et al. 2009). 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 $\text{var}(R) \subseteq \text{var}(P_1)$; and
— for every sub-pattern of P of the form $(P_1 \text{ OPT } P_2)$ and every variable $?X \in \text{var}(P_2)$, if $?X$ occurs in P outside $(P_1 \text{ OPT } P_2)$, then $?X \in \text{var}(P_1)$.

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 (Arenas and Pérez 2011; Pérez et al. 2009). However, the rather syntactic definition of well designedness does not shed light on how close this notion is 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 article is to give a negative answer to this question.

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

To prove this theorem, we use the following two results:

Proposition 3.6 (Pérez et al. 2009). *Every well-designed graph pattern in* SPARQL[AOF] *is equivalent to a well-designed graph pattern in* SPARQL[AOF] *that has the form*

$$(\cdots((P_1 \text{ OPT } P_2) \text{ OPT } P_3) \cdots \text{ OPT } P_n),$$

where P_1 is in SPARQL[AF] *and for each $i \in \{2, \ldots, n\}$ P_i is in* SPARQL[AOF].

Proposition 3.7 (Pérez et al. 2009). *Let P be a well-designed graph pattern in* SPARQL[AF]. *For every RDF graph G and every mapping $\mu \in [\![P]\!]_G$, it is the case that $dom(\mu) = var(P)$.*

Proof of Theorem 3.5. Define the graph pattern P as

$$P = [((a, b, c) \text{ OPT } (?X, d, e)) \text{ OPT } (?Y, f, g)] \text{ FILTER } (\text{bound}(?X) \vee \text{bound}(?Y)).$$

For the sake of simplicity, we refer to the left-hand side of the FILTER as P_{OPT}. Now we show that P is weakly monotone, but it is not equivalent to any well-designed graph pattern in SPARQL[AOF]. Let G_1 and G_2 be two RDF graphs such that $G_1 \subseteq G_2$, and let $\mu \in [\![P]\!]_{G_1}$. By the semantics of FILTER, we have that $\mu \in [\![P_{\text{OPT}}]\!]_{G_1}$. Since P_{OPT} is well designed, it is also weakly monotone and, therefore, there exists $\mu' \in [\![P_{\text{OPT}}]\!]_{G_2}$ such that $\mu \preceq \mu'$. As $\mu \models (\text{bound}(?X) \vee \text{bound}(?Y))$, we have that $\mu' \models (\text{bound}(?X) \vee \text{bound}(?Y))$ and, therefore, $\mu' \in [\![P]\!]_{G_2}$. Thus, we conclude that there exists $\mu' \in [\![P]\!]_{G_2}$ such that $\mu \preceq \mu'$ and, hence, P is weakly monotone as μ is an arbitrary mapping in $[\![P]\!]_{G_1}$. Now we show that P is not equivalent to any well-designed graph pattern in SPARQL[AOF]. For the sake of contradiction assume that Q is a well-designed graph pattern in SPARQL[AOF] such that $P \equiv Q$. From Proposition 3.6 we can assume without loss of generality that $Q = (\cdots((Q_1 \text{ OPT } Q_2) \text{ OPT } Q_3) \cdots \text{ OPT } Q_n)$, where Q_1 is in SPARQL[AF]. Let ℓ be an IRI not mentioned in Q_1, and define G_1 and G_2 as the RDF graphs $\{(a, b, c), (\ell, d, e)\}$ and $\{(a, b, c), (\ell, f, g)\}$, respectively. It is easy to see that $[\![P]\!]_{G_1} = [?X \to \ell]$ and $[\![P]\!]_{G_2} = [?Y \to \ell]$. By Proposition 3.7, this implies that the variables $?X$ and $?Y$ cannot be mentioned in Q_1, as otherwise they would both be

present in the evaluation of Q over every RDF graph, which given that $P \equiv Q$ would imply that $?X$ and $?Y$ occur in the evaluation of P over every RDF graph. Moreover, it is possible to conclude that Q_1 does not mention any variable by using the same argument. Now define the RDF graph $G = \{(a, b, c)\}$. Since Q_1 does not mention ℓ, it is clear that $[\![Q_1]\!]_G = [\![Q_1]\!]_{G_1}$ as no variable occurs in Q_1. Given that $[\![Q]\!]_{G_1} = [\![P]\!]_{G_1}$ and $[\![P]\!]_{G_1} \neq \emptyset$, we have that $[\![Q]\!]_{G_1} \neq \emptyset$ and, hence, $[\![Q_1]\!]_{G_1} \neq \emptyset$ as $Q = (\cdots((Q_1 \text{ OPT } Q_2) \text{ OPT } Q_3) \cdots \text{ OPT } Q_n)$. Thus, we conclude that $[\![Q_1]\!]_G \neq \emptyset$ as $[\![Q_1]\!]_G = [\![Q_1]\!]_{G_1}$. But this implies that $[\![Q]\!]_G \neq \emptyset$ as $Q = (\cdots((Q_1 \text{ OPT } Q_2) \text{ OPT } Q_3) \cdots \text{ OPT } Q_n)$, which leads to a contradiction, since $[\![P]\!]_G = \emptyset$ and we assume that $P \equiv Q$. □

It is interesting to see that the SPARQL graph pattern used in the proof of Theorem 3.5 can be expressed in the syntactic fragment of SPARQL studied in Kaminski and Kostylev (2016). Although not every query in this fragment is weakly monotone, it is an interesting open issue whether every weakly monotone query can be expressed in this fragment.

The notion of well designedness was defined in Pérez et al. (2009) 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., Pichler and Skritek (2014)). However, the following result shows that this fragment of SPARQL is not expressive enough to capture weak monotonicity.

Theorem 3.8. *There exists a weakly monotone graph pattern in* SPARQL[AUOF] *that is not equivalent to any well-designed graph pattern in* SPARQL[AUOF].

To prove this theorem, we use the following result:

Proposition 3.9 (Pérez et al. 2009). *Let P be a graph pattern in* SPARQL[AOF] *and G an RDF graph. Then, for every two distinct mappings $\mu_1, \mu_2 \in [\![P]\!]_G$, it is the case that $\mu_1 \nsim \mu_2$.*

In simple words, this proposition states that a graph pattern in SPARQL[AOF] cannot output two distinct compatible mappings.

Proof of Theorem 3.8. Let P be the graph pattern defined as

$$P = (?X, a, b) \text{ OPT } ((?X, c, ?Y) \text{ UNION } (?X, d, ?Z)).$$

Since in P both sides of the operator OPT are (strictly) monotone, we can directly conclude that P is weakly monotone. Now we prove that P is not equivalent to any well-designed graph pattern in SPARQL[AUOF]. Consider the following four RDF graphs:

$$G_1 = \{(1, a, b)\} \qquad G_2 = \{(1, a, b), (1, c, 2)\}$$
$$G_3 = \{(1, a, b), (1, d, 3)\} \qquad G_4 = \{(1, a, b), (1, c, 2), (1, d, 3)\}.$$

Evaluating P over these graphs, we obtain:

$$[\![P]\!]_{G_1} = \{[?X \to 1]\} \qquad [\![P]\!]_{G_2} = \{[?X \to 1, ?Y \to 2]\}$$
$$[\![P]\!]_{G_3} = \{[?X \to 1, ?Z \to 3]\} \qquad [\![P]\!]_{G_4} = \{[?X \to 1, ?Y \to 2], [?X \to 1, ?Z \to 3]\}.$$

Assume for the sake of contradiction that P is equivalent to the SPARQL[AUOF] well-designed graph pattern $P' = P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n$. By semantics of UNION, there must be an i for which $[\![P_i]\!]_{G_1} = \{[?X \to 1]\}$. Without loss of generality we can assume $i = 1$. Since $G_1 \subseteq G_2$

and P_1 is weakly monotone , we have that $[\![P_1]\!]_{G_1} \sqsubseteq [\![P_1]\!]_{G_2}$. Thus, given that $[\![P_1]\!]_{G_1} = \{[?X \to 1]\}$, there exists $\mu_2 \in [\![P_1]\!]_{G_2}$ such that $[?X \to 1] \preceq \mu_2$. But then given that $[\![P]\!]_{G_2} = \{[?X \to 1, ?Y \to 2]\}$, $[\![P]\!]_{G_2} = [\![P']\!]_{G_2}$, and $[\![P_1]\!]_{G_2} \subseteq [\![P']\!]_{G_2}$, we conclude that $\mu_2 = [?X \to 1, ?Y \to 2]$ and $[\![P_1]\!]_{G_2} = \{[?X \to 1, ?Y \to 2]\}$. Analogously, as $[\![P]\!]_{G_3} = \{[?X \to 1, ?Z \to 3]\}$ and $G_1 \subseteq G_3$, it is the case that $[\![P_1]\!]_{G_3} = \{[?X \to 1, ?Z \to 3]\}$.

Since $G_2 \subseteq G_4$ and P_1 is weakly monotone, we have that $[\![P_1]\!]_{G_2} \sqsubseteq [\![P_1]\!]_{G_4}$. Thus, given that $[\![P]\!]_{G_2} = \{[?X \to 1, ?Y \to 2]\}$, there exists $\mu_4 \in [\![P_1]\!]_{G_4}$ such that $[?X \to 1, ?Y \to 2] \preceq \mu_4$. But then given that $[\![P]\!]_{G_4} = \{[?X \to 1, ?Y \to 2], [?X \to 1, ?Z \to 3]\}$, $[\![P]\!]_{G_4} = [\![P']\!]_{G_4}$, and $[\![P_1]\!]_{G_4} \subseteq [\![P']\!]_{G_4}$, we conclude that $\mu_4 = [?X \to 1, ?Y \to 2]$. Hence, we have that $[?X \to 1, ?Y \to 2] \in [\![P_1]\!]_{G_4}$. The same analysis over G_3 and the mapping $[?X \to 1, ?Z \to 3]$ shows that $[?X \to 1, ?Z \to 3] \in [\![P_1]\!]_{G_4}$.

We conclude that $\{[?X \to 1, ?Y \to 2], [?X \to 1, ?Z \to 3]\} \subseteq [\![P_1]\!]_{G_4}$, showing that P_1 returns two distinct compatible mappings when evaluated over G_4. This contradicts Proposition 3.9 and therefore P and P' cannot be equivalent, which was to be shown. □

The previous theorems present an important improvement in understanding 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 (Lyndon 1959) 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 (Toman and Weddell 2011) or with restricted access to some data sources (Benedikt et al. 2014, 2016a). Thus, they are a natural 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 interpolation techniques are known to fail when restricted to finite models (Ajtai and Gurevich 1987; Libkin 2004), so infinite database instances are considered in the investigations that use these techniques for relational databases (Toman and Weddell 2011; Benedikt et al. 2014). By following the same idea, we consider in this article both finite and infinite RDF graphs, and we define an *unrestricted* RDF graph as a (possibly 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 $[\![P_1]\!]_G = [\![P_2]\!]_G$. It should 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 (Ahmetaj et al. 2015; Barceló et al. 2015), denoted by $P_1 \equiv_s P_2$ (respectively, $P_1 \equiv_s^{\inf} P_2$), if for every RDF graph G (resp., unrestricted RDF graph G), it holds that $[\![P_1]\!]_G \sqsubseteq [\![P_2]\!]_G$ and $[\![P_2]\!]_G \sqsubseteq [\![P_1]\!]_G$. This notion is central to our article and has received a lot of attention since the early 2010s (Letelier et al. 2013; Pichler and Skritek 2014; Ahmetaj et al. 2015; Barceló et al. 2015). We are now ready to present the main result of this article.

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

The proof of this theorem is rather involved. Since interpolation theorems are proved for FO, we first need to provide a translation from SPARQL to FO, then apply the theorems in FO, and, finally, translate the results back to SPARQL. As there are many definitions and intermediate propositions involved, we leave the full proof for Appendix A. However, we present in the next section a sketch of the proof that sheds lights on how interpolation theorems are applied to SPARQL.

As a corollary of Theorem 4.1 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 $P \equiv_s^{\inf} Q$.*

4.1 Sketch of the Proof of Theorem 4.1

The theorem is obtained by applying the interpolation theorems of Lyndon (1959) and Otto (2000). 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 Angles and Gutierrez (2008), Polleres and Wallner (2013), and Kostylev et al. (2015), but with some modifications needed to apply interpolation techniques.

Let P be a graph pattern. Define $\mathcal{L}_{\mathrm{RDF}}^P$ 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 $\mathcal{L}_{\mathrm{RDF}}^P$-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 $\mathcal{L}_{\mathrm{RDF}}^P$-structures that correspond to G. We denote this set by $\mathcal{A}_G^P$. 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 $\leq$ on the set of variables $\mathbf{V}$. Assume $?X_1, \ldots, ?X_\ell$ are the variables in P ordered under $\leq$. Given a mapping $\mu : \mathrm{var}(P) \to \mathbf{I}$, we define the extension of μ to $\mathrm{var}(P)$ as the function $\mu^P : \mathrm{var}(P) \to \mathbf{I}$ such that $\mu^P(?X) = \mu(?X)$ for every $?X \in \mathrm{dom}(\mu)$, and $\mu^P(?X) = N$ for $?X \in \mathrm{var}(P) \setminus \mathrm{dom}(\mu)$. Using this function, we define the tuple corresponding to μ as $t_\mu^P = (\mu^P(?X_1), \ldots, \mu^P(?X_\ell))$. 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

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

With the above 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 $\forall x(Dom(x) \rightarrow \psi)$ or $\exists x(Dom(x) \wedge \psi)$. This condition makes the construction of φ_P a particularly technical procedure; 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[2]:

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

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 $\forall u \forall v \forall w\, (T(u, v, w) \rightarrow T'(u, v, w))$ 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, to apply interpolation, we need a tautological implication, 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:

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

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 (Lyndon 1959) asserts that if an implication $\alpha \rightarrow \beta$ is a tautology, then there is a formula θ such that $\alpha \rightarrow \theta$ and $\theta \rightarrow \beta$ are both tautologies, and every relational symbol occurring in 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 (Otto 2000). From these theorems and formula (2), we can deduce the existence of a new formula $\theta(T', Dom', \bar{x})$ such that the following implications hold:

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

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

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

[2]For the sake of readability, in this section we assume that free variables are universally quantified.

every $\mathcal{L}^P_{\text{RDF}}$-structure $\mathfrak{A} \in \mathcal{A}_G$, if $\mathfrak{A} \models \varphi_P(T, Dom, \bar{a})$ for some tuple $\bar{a}$, then $\mathfrak{A} \models \theta(T, Dom, \bar{a})$. 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 $\mathfrak{A} \models \theta(T, Dom, \bar{a})$, then there is a tuple $\bar{b}$ such that $\bar{a} \preceq \bar{b}$ and $\mathfrak{A} \models \varphi_P(T, Dom, \bar{b})$. 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'. In summary, we have obtained a UCQ with inequalities that outputs the same set of maximal answers as the original pattern P (under our relation between mappings and tuples). Notice that this implies that P and θ' are actually subsumption-equivalent.

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 unrestricted 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 $\mathfrak{A} \models \theta'(t^Q_\mu)$. The details of the construction of the graph pattern Q can be found in Appendix A.

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 obtain 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 introduced in Section 3 the notion of a 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 Kostylev et al. (2015). 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 article, 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]). In the next section, we introduce some operators that allow us to go beyond monotone graph patterns and obtain certain equivalences with classes of unrestricted weakly monotone graph patterns.

5 CAPTURING WEAKLY MONOTONE FRAGMENTS OF SPARQL

The goal of this section is to introduce RDF query languages that capture important 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 to P over G, denoted by $[\![P]\!]_G^{\max}$, 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 tells that given an unrestricted weakly monotone graph pattern P, there exists a graph pattern Q in SPARQL[AUFS] that preserves the maximal answers to P, that is, $[\![P]\!]_G^{\max} = [\![Q]\!]_G^{\max}$ 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. Formally, let NS–SPARQL be the result of extending SPARQL with the following rule for graph patterns:

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

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

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 (Galindo-Legaria 1994).

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, we provide 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. Before providing this translation, we need to show that every graph pattern in SPARQL is equivalent to a graph pattern in a normal form. A graph pattern P is said to be in UNION-normal-form if $P = P_1$ UNION P_2 UNION $\cdots$ UNION P_n, where each P_i ($1 \leq i \leq n$) is UNION-free.

Proposition 5.1. *Every* SPARQL *graph pattern is equivalent to a graph pattern in UNION-normal-form.*

Proof. Let P be a SPARQL graph pattern. We proceed by induction over the structure of P. We only consider the case in which P = SELECT V WHERE Q, as the other cases have already been proved for SPARQL[AUOF] in Pérez et al. (2009).

Assume P = SELECT V WHERE Q. By induction hypothesis, we can assume that Q is equivalent to Q_1 UNION $\cdots$ UNION Q_n, where each disjunct is UNION-free. We show that

$$P \equiv (\text{SELECT } V \text{ WHERE } Q_1) \text{ UNION } \cdots \text{ UNION } (\text{SELECT } V \text{ WHERE } Q_n).$$

Let G be an RDF graph and μ be a mapping.

—[$\Rightarrow$] Assume $\mu \in [\![P]\!]_G$. Then, there is a mapping $\mu' \in [\![Q]\!]_G$ such that $\mu'_{|V} = \mu$. This implies there is an $i \in \{1, \ldots, n\}$ such that $\mu' \in [\![Q_i]\!]_G$. It follows that $\mu \in [\![\text{SELECT } V \text{ WHERE } Q_i]\!]_G$.

— [⇐] Assume $\mu \in [\![\text{SELECT } V \text{ WHERE } Q_i]\!]_G$ for some $i \in \{1, \ldots, n\}$. Then, there is a mapping $\mu' \in [\![Q_i]\!]_G$ such that $\mu'_{|V} = \mu$. It follows that $\mu' \in [\![Q]\!]_G$ and hence $\mu \in [\![P]\!]_G$. □

To prove that NS–SPARQL is contained in SPARQL, we actually make use of a stronger version of UNION-normal-form. We abuse notation by writing $D \in P$ whenever P is a graph pattern in UNION-normal-form and D one of the disjuncts in P.

Proposition 5.2. *Let P be a* SPARQL *graph pattern. Then, there is a graph pattern P' in UNION-normal-form such that $P' \equiv P$. Moreover, for every $D \in P'$ there is a set of variables $V_D \subseteq var(P)$ such that for every RDF graph G and every $\mu \in [\![D]\!]_G$, it is the case that $dom(\mu) = V_D$.*

Proof. Let P be a SPARQL graph pattern. For every $V \subseteq \text{var}(P)$, define the graph pattern

$$P_V = P \text{ FILTER} \left(\bigwedge_{?X \in V} \text{bound}(?X) \wedge \bigwedge_{?X \in \text{var}(P) \setminus V} \neg \text{bound}(?X) \right).$$

Notice that for every graph pattern G, $[\![P_V]\!]_G$ is the set of mappings μ in $[\![P]\!]_G$ such that $\text{dom}(\mu) = V$. Now, define the pattern P'_V as the transformation of P_V into UNION-normal-form. It is clear that the domain of every mapping that comes from the disjuncts of P'_V must be exactly V. Define P' as the disjunction (by means of UNION) of every P'_V ($V \subseteq \text{var}(P)$). We prove that P is equivalent to P'. Let G be an RDF graph and $\mu \in [\![P]\!]_G$. It is clear that $\mu \in [\![P_{\text{dom}(\mu)}]\!]_G$, and hence $\mu \in [\![P'_{\text{dom}(\mu)}]\!]_G$, which implies $\mu \in [\![P']\!]_G$. For the converse, let $\mu \in [\![P']\!]_G$. There is a set V of variables such that $\mu \in [\![P'_V]\!]_G$, and hence $\mu \in [\![P_V]\!]_G$, which implies $\mu \in [\![P]\!]_G$. Finally, as P' is a disjunction of graph patterns in UNION-normal-form, P' is also in UNION-normal-form. Moreover, the disjuncts in the UNION-normal-form version of P_V can only output mappings whose domain is precisely V, which concludes the proof. □

In simple words, the previous lemma allows us to distinguish the set of variables bound by the mappings coming from each disjunct. Now we proceed to prove that NS–SPARQL is contained in SPARQL. In the proof we use the MINUS[3] operator, which is defined as

$$(P_1 \text{ MINUS } P_2) = (P_1 \text{ OPT } (P_2 \text{ AND } (?x_1, ?x_2, ?x_3))) \text{ FILTER } \neg bound(?x_1),$$

where $?x_1$, $?x_2$, $?x_3$ are variables occurring neither in P_1 nor in P_2. Given an RDF graph G, the graph pattern $(P_1 \text{ MINUS } P_2)$ retrieves the mappings in $[\![P_1]\!]_G$ that are not compatible with any mapping in $[\![P_2]\!]_G$ (Angles and Gutierrez 2008).

Theorem 5.3. *The languages* SPARQL *and* NS–SPARQL *have the same expressive power.*

Proof. Given that NS–SPARQL is an extension of SPARQL, we only need to prove that every graph pattern in NS–SPARQL is equivalent to a graph pattern in SPARQL. Let P be a graph pattern in NS–SPARQL. We proceed by induction over the structure of P. The basic case is trivial as a triple pattern is already in SPARQL. For the inductive step assume that $P = \text{NS}(Q)$, which is the only nontrivial case. By hypothesis we can assume Q is in SPARQL (notice that OPT might occur in Q). From Proposition 5.2 we can suppose that Q is in UNION-normal-form and, moreover, that each disjunct of Q can only output mappings binding a fixed set of variables V_Q. Let Q' be a disjunct of Q, and assume $Q_1, \ldots, Q_n$ are all disjuncts of Q such that $V_{Q'} \subsetneq V_{Q_i}$. Define the graph pattern

$$Q'_{\text{NS}} = Q' \text{ MINUS } (Q_1 \text{ UNION } \cdots \text{ UNION } Q_n).$$

[3]Note that this is the MINUS operator introduced in Angles and Gutierrez [2008], which is called DIFF in the SPARQL 1.1 standard [Harris and Seaborne 2013].

Intuitively, this pattern outputs all mappings coming from Q' that are not compatible with a mapping containing strictly more variables. Naturally, this coincides with obtaining the patterns coming from Q' that are not subsumed. We show that this is true by proving that $\text{NS}(Q)$ is equivalent to the graph pattern $R = \text{UNION}_{Q' \in Q} Q'_{\text{NS}}$.

- [$\Rightarrow$] Let G be an RDF graph and let $\mu \in [\![\text{NS}(Q)]\!]_G$. We have that $\mu \in [\![Q']\!]_G$ for some disjunct Q' of Q, and that there is no mapping $\mu' \in [\![\text{NS}(Q)]\!]_G$ subsuming μ. It follows that there is no disjunct Q'' of Q and mapping $\mu'' \in [\![Q'']\!]_G$ such that $V_{Q'} \subsetneq V_{Q''}$ and $\mu \sim \mu''$, from which we can deduce that $\mu \in [\![Q'_{\text{NS}}]\!]_G$.
- [$\Leftarrow$] Let G be an RDF graph and let $\mu \in [\![R]\!]_G$. We have that $\mu \in [\![Q'_{\text{NS}}]\!]_G$ for some disjunct Q' of Q. Then, $\mu \in [\![Q']\!]_G$ and there is no disjunct Q'' of Q and mapping $\mu'' \in [\![Q'']\!]_G$ such that $V_{Q'} \subsetneq V_{Q''}$ and $\mu \sim \mu'$. This obviously implies that there is no mapping in $[\![Q]\!]_G$ subsuming μ, and hence $\mu \in [\![\text{NS}(Q)]\!]_G$.

Having that $\text{NS}(Q)$ is equivalent to a union of NS-free patterns, we conclude that the NS operator can be removed by introducing the operator MINUS, which is in turn defined in by means of operator OPT. We conclude that every graph pattern in NS–SPARQL is equivalent to a SPARQL graph pattern, which concludes the proof. □

Given that NS–SPARQL and SPARQL have the same expressive power, 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 considering a 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 $P_1 \equiv^{\text{inf}} P_2$ if and only if $P_1 \equiv_s^{\text{inf}} P_2$. It is important to mention that, in practice, subsumption-free graph patterns are the rule and not the exception.[4]

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

Corollary 5.4. *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 $P \equiv^{\text{inf}} \text{NS}(Q)$.*

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

Definition 5.5. A graph pattern in NS–SPARQL is called simple if it is of the form $\text{NS}(P)$, where P is in SPARQL[AUFS].

Let SP–SPARQL be the RDF query language consisting of all simple graph patterns, that is, every query in SP–SPARQL is of the form $\text{NS}(P)$ with P in SPARQL[AUFS]. From Corollary 5.2 and the fact that every simple graph pattern is subsumption-free, we obtain the following theorem

[4]It should be mentioned that the problem of verifying, given a graph pattern P, whether P is subsumption-free is undecidable. This is a corollary of the equivalence between the expressive power of SPARQL and first-order logic [Angles and Gutierrez 2008; Sequeda et al. 2012] and the undecidability of the finite satisfiability problem for this logic [Trakhtenbrot 1950].

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

PROOF. From Theorem 5.3, we know that every graph pattern in SP–SPARQL is equivalent to a SPARQL graph pattern. Thus, given that every pattern in SP–SPARQL is unrestricted weakly monotone and subsumption-free, we conclude that every graph pattern in SP–SPARQL is equivalent to an unrestricted weakly monotone and subsumption-free SPARQL graph pattern.

To prove the opposite direction, consider a subsumption-free and unrestricted weakly monotone graph pattern P in SPARQL. From Theorem 4.1, we know that there is a graph pattern $P' \in \text{SPARQL[AUFS]}$ such that $P \equiv_s^{\text{inf}} P'$. Now let G be an unrestricted RDF graph. Since P is subsumption-free, we have that $[\![P]\!]_G = [\![P]\!]_G^{\max}$. But as $P \equiv_s^{\text{inf}} P'$, it is the case that $[\![P]\!]_G^{\max} = [\![P']\!]_G^{\max} = [\![\text{NS}(P')]\!]_G$. Therefore P and $\text{NS}(P')$ are equivalent, which concludes the proof as $\text{NS}(P')$ is a simple pattern. □

Notice that in this theorem we compare the expressive power of two query languages over unrestricted RDF graphs; thus, we use $\equiv^{\text{inf}}$ 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 Pérez et al. (2009), 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 Proposition 3.9 we know that every graph pattern in SPARQL[AOF] is also subsumption free. Thus, we obtain from Theorem 5.6 the following result:

COROLLARY 5.7. *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 $P \equiv^{\text{inf}} Q$.*

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

PROPOSITION 5.8. *The fragment of well-designed graph patterns in* SPARQL[AOF] *is strictly less expressive than* SP–SPARQL.

PROOF. Provided Corollary 5.7, to prove that every well-designed graph pattern in SPARQL[AOF] is equivalent to a graph pattern in SP–SPARQL, it suffices to show that well-designed graph patterns in SPARQL[AOF] are unrestricted weakly monotone. This follows immediately from the proof that graph patterns in SPARQL[AOF] are weakly monotone ((Pérez et al. 2009), Lemma 4.3), since that proof does not impose any restriction on the cardinality of RDF graphs. To prove that the containment is strict, consider the graph pattern $P = \text{NS}((a, b, ?X) \text{ UNION } (d, e, ?Y))$ and the RDF graph $G = \{(a, b, c), (d, e, f)\}$. It is clear that $[\![P]\!]_G = \{[?X \to c], [?Y \to f]\}$. From Proposition 3.9 we know that if Q is a well-designed graph pattern in SPARQL[AOF], then $[\![Q]\!]_G$ cannot contain two compatible mappings. Hence, P cannot be equivalent to any well-designed graph pattern in SPARQL[AOF]. □.

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[AUFS] 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 5.9. A graph pattern P in NS–SPARQL is called an *ns-pattern* if it is of the form $(P_1 \text{ UNION } \cdots \text{ UNION } P_n)$, 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 5.10. USP–SPARQL *is strictly more expressive than both* SPARQL[AUFS] *and* SP–SPARQL.

To show that SPARQL[AUFS] is contained in USP–SPARQL, given a pattern in SPARQL[AUFS] one can transform it into UNION-normal form (Proposition 5.1). Moreover, it is not hard to see that each disjunct will be subsumption-free (Proposition 3.9). Therefore, this disjunction can be trivially transformed into an equivalent pattern in USP–SPARQL. The fact that the containments are proper holds trivially because SPARQL[AUFS] only contains monotone patterns and SP–SPARQL only contains subsumption-free patterns.

From the fact that every simple pattern is unrestricted weakly monotone, it is easy to deduce that every ns-pattern is also 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 5.11. *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 $P \equiv_s^{\text{inf}} Q$.*

Finally, we obtain the following characterization of the expressive power of USP–SPARQL by using Theorem 5.6:

Corollary 5.12. *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 it is strictly more expressive than all previously proposed weakly monotone fragments whose definition is syntactic.

6 CAPTURING CONSTRUCT QUERIES

The input of a SPARQL graph pattern is an RDF graph, while its output is a set of 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 (Prud'hommeaux and Seaborne 2008) that can be used to produce as

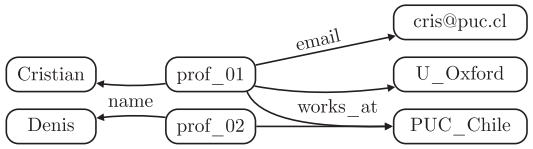

Fig. 3. An RDF graph containing information about professors and universities.

output an RDF graph. 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 unrestricted monotone queries using the CONSTRUCT operator. It should be noticed that, once more, 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 Kostylev et al. (2015) to define the CONSTRUCT operator. Let P be a SPARQL graph pattern and H a finite set of triple patterns. Then

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

is a CONSTRUCT query, where H and P are called the template and the graph pattern 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

Next to the table we have included names for the mappings. Now, to evaluate Q, we consider each 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 (Cristian, email, 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 operator CONSTRUCT is defined as a set of triples, (Cristian, email, cris@puc.cl) can occur only once in the output $\text{ans}(Q, G)$.

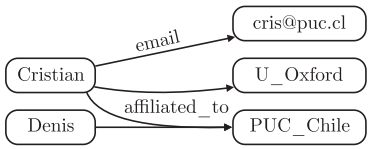

Fig. 4. The RDF graph obtained by evaluating the CONSTRUCT query Q in Example 6.1 over the RDF graph in Figure 3.

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. As such, 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 (respectively, unrestricted monotone) if for every pair G_1, G_2 of RDF graphs (respectively, unrestricted RDF graphs) such that $G_1 \subseteq G_2$, it holds that $\text{ans}(Q, G_1) \subseteq \text{ans}(Q, G_2)$.

Notice that in this definition we also considered 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.

Exactly as in the case of weak monotonicity for graph patterns, Definition 6.2 provides no insight about how to find a syntactic characterization of monotone CONSTRUCT queries, which is aggravated by the fact that the problem of verifying whether a CONSTRUCT query is monotone is undecidable.[5] Nevertheless, we can address this issue by focusing on unrestricted RDF graphs (as it is done in Sections 4 and 5) and considering some specific properties of CONSTRUCT queries. 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 pattern P and template H, it holds that:*

$$(\text{CONSTRUCT } H \text{ WHERE } P) \equiv^{\text{inf}} (\text{CONSTRUCT } H \text{ WHERE NS}(P)).$$

Proof. Let G be an unrestricted RDF graph and define the queries $Q = (\text{CONSTRUCT } H \text{ WHERE } P)$ and $Q' = (\text{CONSTRUCT } H \text{ WHERE NS}(P))$. Since $[\![\text{NS}(P)]\!]_G \subseteq [\![P]\!]_G$, it is trivial to show that $\text{ans}(Q', G) \subseteq \text{ans}(Q, G)$. It remains to show that $\text{ans}(Q, G) \subseteq \text{ans}(Q', G)$. Let $t \in \text{ans}(Q, G)$. By definition, there is a mapping $\mu \in [\![P]\!]_G$ and a triple pattern $T \in H$ such that $\text{var}(T) \subseteq \text{dom}(\mu)$ and $\mu(T) = t$. From the definition of NS there is a mapping $\mu' \in [\![\text{NS}(P)]\!]_G$ such that $\mu \preceq \mu'$, and therefore $\text{var}(T) \subseteq \text{dom}(\mu')$. Then, the triple $\mu'(T)$ belongs

[5]This is a corollary of the equivalence between the expressive power of SPARQL and first-order logic [Angles and Gutierrez 2008; Sequeda et al. 2012], 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 [Trakhtenbrot 1950].

to $\text{ans}(Q', G)$. But since $\mu \preceq \mu'$ and $\text{var}(T) \subseteq \text{dom}(\mu)$, we have that $\mu'(T) = \mu(T) = t$. We conclude that $t \in \text{ans}(Q', G)$. □

Lemma 6.3 has a strong consequence in the characterization of monotone queries. Consider a CONSTRUCT query $Q = (\text{CONSTRUCT } H \text{ WHERE } P)$ 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 $P^\star \equiv_s^{\text{inf}} P$. Thus, given that the condition $P^\star \equiv_s^{\text{inf}} P$ holds if and only if the condition $\text{NS}(P^\star) \equiv^{\text{inf}} \text{NS}(P)$ holds, we obtain the following by applying Lemma 6.3:

$$\begin{aligned} Q = & \quad (\text{CONSTRUCT } H \text{ WHEREP}) \\ \equiv^{\text{inf}} & \quad (\text{CONSTRUCT } H \text{ WHERE NS}(P)) \\ \equiv^{\text{inf}} & \quad (\text{CONSTRUCT } H \text{ WHERE NS}(P^\star)) \\ \equiv^{\text{inf}} & \quad (\text{CONSTRUCT } H \text{ WHERE} P^\star) \end{aligned}$$

Hence, we obtain the following corollary.

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

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. *Let Q be an unrestricted monotone CONSTRUCT query. Then, there exists a template H and an unrestricted weakly monotone* SPARQL *graph pattern P such that $Q \equiv^{\text{inf}}$ (CONSTRUCT H WHERE P).*

Proof. Let $Q = (\text{CONSTRUCT } H \text{ WHERE } P)$ be an unrestricted monotone CONSTRUCT query. We can assume without loss of generality that $\text{var}(H) \subseteq \text{var}(P)$, as every triple in H mentioning a variable not occuring in P can be safely removed. For every triple pattern $t \in H$ define a renaming function $\sigma_t : \mathbf{V} \to \mathbf{V}$ such that:

—For every $t, s \in H$ and every $v_1, v_2 \in \text{var}(P)$, it is the case that $\sigma_t(v_1) \neq \sigma_s(v_2)$.
—For every $t \in H$ and every $v \in \text{var}(P)$, $\sigma_t(v) \notin \text{var}(P)$.

Given a mapping μ and a triple $t \in H$, define $\sigma_t[\mu]$ as the mapping that results from replacing the domain of μ by its image under σ_t. For every $t \in H$ let $Adom(t)$ be the conjunction (by means of AND) of $Adom(?X)$ for each variable $?X$ in $\text{var}(t)$. If t has no variables, then $Adom(t)$ is considered to be a tautology.

For every $t \in H$ define the pattern P^t as the result of replacing in P every occurrence of a variable $?X$ by $\sigma_t(?X)$. For every two triples $t = (t_1, t_2, t_3)$ and $s = (s_1, s_2, s_3)$ in H define $R_{t,s}$ as the filter condition $(t_1 = \sigma_s(s_1) \wedge t_2 = \sigma_s(s_2) \wedge t_3 = \sigma_s(s_3))$, assuming, for the sake of simplicity, that $\sigma_s(a) = a$ for every $a \in \mathbf{I}$. Now we define the set of graph patterns that will serve as a basis for our construction. For each $t \in H$, define P_t as

$$P_t = \text{SELECT var}(t) \text{ WHERE } \Big(\Big[P \text{ UNION } \underset{s \in H \setminus \{t\}}{\text{UNION}} [(P^s \text{ AND } Adom(t)) \text{ FILTER } R_{t,s}] \Big] \text{ FILTER } (bound(\text{var}(t))) \Big), \tag{5}$$

where $Adom(t)$ states that the variables of t can be assigned any value in the active domain. Formally $Adom(?X)$ is defined as

$$Adom(?X) = \text{SELECT } \{?X\} \text{ WHERE } [(?X, ?Y, ?Z) \text{ UNION } (?Y, ?X, ?Z) \text{ UNION } (?Y, ?Z, ?X)]$$

and $Adom(t)$ is the conjunction of $Adom(?W)$ for each variable $?W \in \text{var}(t)$.

Now we prove that the following three properties hold for every $t \in H$:

(1) For every RDF graph G and every mapping $\mu \in [\![P]\!]_G$, if $\mu(t) \in \text{ans}(Q, G)$, then $\mu(t) \in \text{ans}(\text{CONSTRUCT } t \text{ WHERE } P_t, G)$.
(2) For every graph G, $\text{ans}(\text{CONSTRUCT } t \text{ WHERE } P_t, G) \subseteq \text{ans}(Q, G)$.
(3) P_t is unrestricted weakly monotone.

The first property immediately follows from the fact that P is one of the disjuncts of P_t, because if $\mu(t) \in \text{ans}(Q, G)$, then the variables in t are bounded by μ. Now we proceed with (2). Let G be an RDF graph and let μ be a mapping in $[\![P_t]\!]_G$ such that $\mu(t) \in \text{ans}(\text{CONSTRUCT } t \text{ WHERE } P_t, G)$. Hence, μ must come from one of the disjuncts in P_t. If that disjunct is P, then we have that μ is the projection over $\text{var}(t)$ of a mapping in $[\![P]\!]_G$, and hence $\mu(t) \in \text{ans}(Q, G)$. If not, then there is an $s \in H$ such that μ is subsumed by a mapping $\mu' \in [\![(P^s \text{ AND } Adom(t)) \text{ FILTER } R_{t,s}]\!]_G$. Then μ' is the join between two mappings. Let μ_s be the mapping in $[\![P^s]\!]_G$ of such join. Since P^s equals P by a renaming of all variables, the mapping $\sigma_s^{-1}[\mu_s]$ belongs to $[\![P]\!]_G$. Moreover, by the filter condition $R_{t,s}$, we know that $\sigma_s^{-1}[\mu_s]$ must bind all variables in $\text{var}(s)$, and hence $\sigma_s^{-1}[\mu_s](s) \in \text{ans}(Q, G)$. But from the filter condition we know that $\sigma_s^{-1}[\mu_s](s)$ equals $\mu(t)$, and hence $\mu(t)$ belongs to $\text{ans}(Q, G)$, which was to be shown.

Finally, we prove that P_t is unrestricted weakly monotone. Let G be an unrestricted RDF graph and $\mu \in [\![P_t]\!]_G$. We know that $\text{dom}(\mu) = \text{var}(t)$, and hence $\mu(t) \in \text{ans}(\text{CONSTRUCT } t \text{ WHERE } P_t, G)$. By property 2, this implies that $\mu(t) \in \text{ans}(Q, G)$. Let G' be an RDF graph such that $G \subseteq G'$. Since Q is unrestricted monotone, there must be a triple $s \in H$ and a mapping $\mu_s \in [\![P]\!]_{G'}$ such that $\mu_s(s) = \mu(t)$. Hence $\sigma_s[\mu_s] \in [\![P^s]\!]_{G'}$. Moreover, since $\mu_s(s) = \mu(t)$, we have that $\sigma_s[\mu_s] \bowtie \mu$ satisfies $R_{t,s}$. Hence, $\sigma_s[\mu_s] \bowtie \mu$ belongs to $[\![P^s \text{ AND } Adom(t) \text{ FILTER } R_{t,s}]\!]_{G'}$, and thus $\mu \in [\![P_t]\!]_{G'}$. This actually tells us that P_t is unrestricted monotone and therefore unrestricted weakly monotone.

Having defined the patterns P_t and proved the three properties above, we proceed with the main result, namely that Q can be rewritten as a CONSTRUCT query in which the graph pattern is unrestricted weakly monotone. First, define for each $t \in H$ the CONSTRUCT query Q_t as $(\text{CONSTRUCT } t' \text{ WHERE } P'_t)$, where t' and P'_t are the result of renaming the variables in t and P_t, respectively, in a consistent manner. Without loss of generality we can assume that for $t, s \in H$, the queries Q_t and Q_s have pairwise disjoint sets of variables. Notice, however, that for every $t \in H$ the query Q_t is equivalent to CONSTRUCT t WHERE P_t and hence satisfies the three properties mentioned above. Finally, define H' and P' as:

$$H' = \{t' \mid t \in H\} \qquad P' = \text{UNION}_{t \in H} P'_t.$$

Let $Q' = (\text{CONSTRUCT } H' \text{ WHERE } P')$. We prove that Q and Q' are equivalent. Let G be an RDF graph.

— [$\Rightarrow$] Let $\mu \in [\![P]\!]_G$ and $t \in H$ such that $\mu(t) \in \text{ans}(Q, G)$. By the first property proved above, we know that $\mu(t)$ is in the answer to CONSTRUCT t WHERE P_t over G, which implies that $\mu(t) \in \text{ans}(\text{CONSTRUCT } t' \text{ WHERE } P'_t, G)$. Since P'_t is one of the disjuncts of P' and $t' \in H'$, we have that $\mu(t) \in \text{ans}(Q', G)$.

—[⇐] Let $\mu \in [\![P']\!]_G$ and $t \in H'$ such that $\mu(t) \in \text{ans}(Q', G)$. We know that $\mu \in [\![P'_s]\!]_G$ for some $s' \in H'$. If $\text{var}(t) \neq \emptyset$, then such s' must be t' as P'_t and P'_s do not share variables. In this case, $\mu(t) \in \text{ans}(\text{CONSTRUCT } t' \text{ WHERE } P'_t, G)$, which implies that $\mu(t) \in \text{ans}(\text{CONSTRUCT } t \text{ WHERE } P_t, G)$. By the second property, this implies $\mu(t) \in \text{ans}(Q, G)$. On the other hand, if t has no variables, then we still know that $\mu \in [\![P'_s]\!]_G$ for some $s' \in H'$. This entails there is a mapping in $[\![P_s]\!]_G$. Hence, there is either a mapping μ' in $[\![P]\!]_G$ or in $[\![P^h]\!]_G$ for some $h \in H$. Since P^h is a renaming of P, in any case there must be a mapping $\mu' \in [\![P]\!]_G$. Finally, as t has no variables, $\mu'(t) = \mu(t) = t \in [\![P]\!]_G$.

We proved that Q is equivalent to $Q' = (\text{CONSTRUCT } H' \text{ WHERE } P')$. Since P' is a disjunction between unrestricted weakly monotone graph patterns, we know that P' is also unrestricted weakly monotone, concluding the proof. □

From Lemma 6.5 and Corollary 6.4, we can finally obtain a syntactic characterization of unrestricted 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].

Corollary 6.6. *Over unrestricted RDF graphs, the class of unrestricted 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]. To prove this proposition, we define an effective procedure for removing the SELECT operator from a CONSTRUCT query. This procedure is given by the following recursive definition:

Definition 6.7. Let P be an NS–SPARQL graph pattern. The SELECT-free version of P, denoted by P_{sf}, is recursively defined as follows:

—If P is a triple pattern, then $P_{\text{sf}} = P$.
—If $P = (\text{SELECT } V \text{ WHERE } P')$, then P_{sf} is the result of replacing in P'_{sf} every variable in $\text{var}(P') \setminus V$ by a fresh variable. Notice that the operator SELECT is removed.
—If P is $(P^1 \circ P^2)$, where $\circ$ is one of {AND, UNION, OPT}, then $P_{\text{sf}} = (P^1_{\text{sf}} \circ P^2_{\text{sf}})$, where the sets of variables $\text{var}(P^1_{\text{sf}}) \setminus \text{var}(P)$ and $\text{var}(P^2_{\text{sf}}) \setminus \text{var}(P)$ are disjoint.
—If $P = \text{NS}(P')$, then $P_{\text{sf}} = \text{NS}(P'_{\text{sf}})$.
—If $P = (P' \text{ FILTER } R)$, then $P = (P'_{\text{sf}} \text{ FILTER } R)$.

Now we need the following equivalence between a graph pattern and its SELECT-free version.

Lemma 6.8. *Let P be a graph pattern. For every RDF graph G, a mapping $\mu \in [\![P]\!]_G$ if and only if there is a mapping $\mu' \in [\![P_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$ and $dom(\mu) = dom(\mu') \cap var(P)$.*

Proof. We proceed by induction on the structure of P. Assume G is an RDF graph and let μ be a mapping.

—If P is a triple pattern, then the result immediately follows.
—If P is $(P^1 \text{ UNION } P^2)$, then $\mu \in [\![P]\!]_G$ if and only if $\mu \in [\![P^1]\!]_G \cup [\![P^2]\!]_G$. Assume without loss of generality that $\mu \in [\![P^1]\!]_G$. By hypothesis, this occurs if and only if there is a mapping $\mu' \in [\![P^1_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P^1)$. From the definition of P_{sf}, we know that $\text{dom}(\mu') \setminus \text{dom}(\mu)$ does not contain any variables occurring in P^2 or P^2_{sf}. We obtain that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$, concluding this case as μ' is clearly in $[\![P_{\text{sf}}]\!]_G$.

—Let $P = (P^1 \text{ AND } P^2)$. In this case, μ is in $[\![P]\!]_G$ if and only if there are two compatible mappings $\mu_1 \in [\![P^1]\!]_G$ and $\mu_2 \in [\![P^2]\!]_G$ such that $\mu = \mu_1 \cup \mu_2$. By hypothesis, this occurs if and only if there are two mappings $\mu_1' \in [\![P^1_{\text{sf}}]\!]_G$ and $\mu_2' \in [\![P^2_{\text{sf}}]\!]_G$ such that $\mu_1 \preceq \mu_1'$, $\mu_2 \preceq \mu_2'$, $\text{dom}(\mu_1) = \text{dom}(\mu_1') \cap \text{var}(P_1)$ and $\text{dom}(\mu_2) = \text{dom}(\mu_2') \cap \text{var}(P_2)$. But as the sets of variables $\text{var}(P^1_{\text{sf}}) \setminus \text{var}(P)$ and $\text{var}(P^2_{\text{sf}}) \setminus \text{var}(P)$ are disjoint, we have that $\text{dom}(\mu_1) \cap \text{dom}(\mu_2) = \text{dom}(\mu_1') \cap \text{dom}(\mu_2')$. Then, μ_1' and μ_2' are compatible and $\mu_1' \cup \mu_2' \in [\![P_{\text{sf}}]\!]_G$. Now we show that $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$. Notice that the variables in $\text{dom}(\mu_1') \setminus \text{var}(P_1)$ are fresh variables, so they cannot occur in $\text{var}(P_2)$. This implies that $\text{dom}(\mu_1') \cap \text{var}(P_1) = \text{dom}(\mu_1') \cap \text{var}(P)$, and the corresponding analysis over μ_2' and P_2 yields $\text{dom}(\mu_2') \cap \text{var}(P_2) = \text{dom}(\mu_2') \cap \text{var}(P)$. The equality is then obtained as follows:

$$\begin{aligned}\text{dom}(\mu) &= \text{dom}(\mu_1) \cup \text{dom}(\mu_2) = (\text{dom}(\mu_1') \cap \text{var}(P_1)) \cup (\text{dom}(\mu_2') \cap \text{var}(P_2)) \\ &= (\text{dom}(\mu_1') \cap \text{var}(P)) \cup (\text{dom}(\mu_2') \cap \text{var}(P)) = (\text{dom}(\mu_1') \cup \text{dom}(\mu_2')) \cap \text{var}(P) \\ &= \text{dom}(\mu') \cap \text{var}(P).\end{aligned}$$

—Let $P = (P^1 \text{ OPT } P^2)$. We know that $\mu \in [\![P]\!]_G$ if and only if $\mu \in [\![(P^1 \text{ MINUS } P^2)]\!]_G$ or $\mu \in [\![(P^1 \text{ AND } P^2)]\!]_G$. It suffices show that $\mu \in [\![(P^1 \text{ MINUS } P^2)]\!]_G$ if and only if there is a $\mu' \in [\![P^1_{\text{sf}} \text{ MINUS } P^2_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$, since the AND case was already provided. By definition, $\mu \in [\![(P^1 \text{ MINUS } P^2)]\!]_G$ if and only if $\mu \in [\![P^1]\!]_G$ and there is no $\mu' \in [\![P^2]\!]_G$ compatible with μ. By hypothesis, the former condition occurs if and only if there is a mapping $\mu_1 \in [\![P^1_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu_1$ and $\text{dom}(\mu) = \text{dom}(\mu_1) \cap \text{var}(P_1)$. We can easily see by contradiction, then, that there cannot be a $\mu_2 \in [\![P^2_{\text{sf}}]\!]_G$ compatible with μ_1, as if such μ_2 existed, then $\mu_1 \cup \mu_2$ would belong to $[\![P^1_{\text{sf}} \text{ AND } P^2_{\text{sf}}]\!]_G$. Thus, by the AND case this would imply the existence of a mapping in $[\![P^1 \text{ AND } P^2]\!]_G$ extending μ, contradicting the fact that $\mu \in [\![(P^1 \text{ MINUS } P^2)]\!]_G$. We finally need to show that $\text{dom}(\mu) = \text{dom}(\mu_1) \cap \text{var}(P)$, but in this case this is trivial, since we have $\text{dom}(\mu) = \text{dom}(\mu_1) \cap \text{var}(P_1)$, and by construction the variables in $\text{dom}(\mu_1) \setminus \text{var}(P_1)$ cannot occur in P_2.

—Let $P = (P' \text{ FILTER } R)$. Then $\mu \in [\![P]\!]_G$ if and only if $\mu \in [\![P']\!]_G$ and $\mu \models R$. By hypothesis, we have that $\mu \in [\![P']\!]_G$ if and only if there is a mapping $\mu' \in [\![P'_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$. As the variables in $\text{dom}(\mu') \setminus \text{dom}(\mu)$ are not mentioned in $\text{var}(P)$, they cannot occur in R. Hence $\mu \models R$ if and only if $\mu' \models R$, concluding the proof.

—If $P = \text{NS}(P')$, then a mapping μ is in $[\![P]\!]_G$ if and only if μ is a maximal mapping in $[\![P']\!]_G$. By hypothesis, this occurs if and only if there is a mapping μ' in $[\![P'_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$. Such a μ' exists if and only if there is a μ'' that is maximal in $[\![P'_{\text{sf}}]\!]_G$ satisfying $\mu' \preceq \mu''$. Notice that this implies $\text{dom}(\mu) = \text{dom}(\mu'') \cap \text{var}(P)$ as μ, μ', and μ'' must agree in all variables mentioned in P. This concludes the NS case, since $\mu \preceq \mu' \preceq \mu''$.

—Let $P = \text{SELECT } V \text{ WHERE } P'$. By definition of SELECT, we have that $\mu \in [\![P]\!]_G$ if and only if there is a mapping $\mu' \in [\![P']\!]_G$ such that $\mu = \mu'_{|V}$. From our hypothesis, this occurs if and only if there is a mapping $\mu'' \in [\![P'_{\text{sf}}]\!]_G$ such that $\mu' \preceq \mu''$ and $\text{dom}(\mu') = \text{dom}(\mu'') \cap \text{var}(P')$. As P_{sf} is simply the result of renaming in P' the variables not occurring in V, there must be a mapping $\mu'' \in [\![P_{\text{sf}}]\!]_G$ that is equivalent to the same renaming over μ'. Since $\mu = \mu'_{|V}$, we know that $\text{dom}(\mu) = \text{dom}(\mu') \cap V$. As μ'' is a renaming of μ' in which all variables not in V are replaced by fresh variables (not in P), we also have that $\text{dom}(\mu) = \text{dom}(\mu'') \cap \text{var}(P)$. Moreover, as μ' and μ'' agree on all variables in V and $\mu \preceq \mu'$, we have $\mu \preceq \mu''$, concluding the proof. □

Finally, we are ready to prove our main proposition.

Proposition 6.9. *Over unrestricted RDF graphs,* CONSTRUCT[AUFS] *and* CONSTRUCT[AUF] *have the same expressive power.*

Proof. We only need to show that every query in CONSTRUCT[AUFS] is equivalent to a query in CONSTRUCT[AUF]. Let $Q = (\text{CONSTRUCT } H \text{ WHERE } P)$ be a query in CONSTRUCT[AUFS]. We can assume w.l.o.g that $\text{var}(H) \subseteq \text{var}(P)$. We prove that $Q \equiv (\text{CONSTRUCT } H \text{ WHERE } P_{\text{sf}})$. Let G be an RDF graph.

- [⇒] Assume $\mu(t) \in \text{ans}(Q, G)$, where $\mu \in [\![P]\!]_G$ and $t \in H$. Then, there is a mapping μ' in $[\![P_{\text{sf}}]\!]_G$ such that $\mu \preceq \mu'$. This implies that $\mu'(t)$ is a triple in $\text{ans}((\text{CONSTRUCT } H \text{ WHERE } P_{\text{sf}}), G)$. As $\mu \preceq \mu'$, we obtain that $\mu'(t) = \mu(t)$, concluding this direction.
- [⇐] Assume $\mu'(t) \in \text{ans}((\text{CONSTRUCT } H \text{ WHERE } P_{\text{sf}}), G)$, where $\mu' \in [\![P_{\text{sf}}]\!]_G$ and $t \in H$. Then, there is a mapping μ in $[\![P]\!]_G$ such that $\mu \preceq \mu'$ and $\text{dom}(\mu) = \text{dom}(\mu') \cap \text{var}(P)$. Since $\text{var}(t) \subseteq \text{var}(P)$ and $\text{var}(t) \subseteq \text{dom}(\mu')$, we obtain that $\text{var}(t) \subseteq \text{dom}(\mu)$. Therefore, we have that $\mu(t) \in \text{ans}(Q, G)$. □

This result finally gives a simpler characterization of the class of monotone CONSTRUCT queries.

Corollary 6.10. *Over unrestricted RDF graphs, the class of unrestricted 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 unrestricted 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

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 study the complexity of the evaluation problem for these fragments and see whether this complexity is lower than for some well-known fragments of SPARQL. In this section, we study the combined complexity (Vardi 1982) 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 definitions. In particular, we present the Boolean Hierarchy and the complexity class $\text{P}_{||}^{\text{NP}}$.

The Boolean Hierarchy is an infinite family of complexity classes based on Boolean combinations of languages in NP (Wechsung 1985). 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 $L_1 \in \text{NP}$ and $L_2 \in \text{coNP}$. The levels of the Boolean hierarchy are denoted by $\{\text{BH}_i\}_{i \in \mathbb{N}}$ and are recursively defined as follows:

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

Notice that $\text{DP} = \text{BH}_2$. The complexity class $\text{P}_{||}^{\text{NP}}$ (Hemachandra 1989) contains all problems that can be solved in polynomial time by a Turing machine that can query a polynomial number 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 $\mathrm{P}_{||}^{\mathrm{NP}}$ 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 (Buss and Hay 1991).

7.2 The Evaluation Problem for Simple Patterns and NS-Patterns

Consider a fragment $\mathcal{F}$ of NS–SPARQL. 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 on several studies of the complexity of evaluating SPARQL graph patterns (Arenas and Pérez 2011; Letelier et al. 2012; Pérez et al. 2009; Pichler and Skritek 2014; Schmidt et al. 2010). In particular, the key ideas rely on the fact that the evaluation problem is NP-complete for SPARQL[AUFS] (Schmidt et al. 2010) and is coNP-complete for well-designed graph patterns in SPARQL[AOF] (Pérez et al. 2006).

We start by considering the evaluation problem for simple graph patterns. To prove this theorem, we make use of the following two lemmas.

Lemma 7.1 ((Pérez et al. 2009), Theorem 3.2). *There is a polynomial-time algorithm that, given a propositional formula φ, generates a mapping μ_φ, a graph pattern P_φ in* SPARQL[AUF] *and an RDF graph G_φ, such that:*

(1) $dom(\mu_\varphi) = var(P_\varphi)$ and $\mathbf{I}(P_\varphi) = \mathbf{I}(G_\varphi)$;
(2) every triple pattern in P_φ mentions both variables and IRIs;
(3) if φ is satisfiable, then $[\![P]\!]_{G_\varphi} = \{\mu_\varphi\}$; and
(4) if φ is unsatisfiable, then $[\![P]\!]_{G_\varphi} = \emptyset$.

Lemma 7.2. *Let G_1 and G_2 be two RDF graphs such that $\mathbf{I}(G_1) \cap \mathbf{I}(G_2) = \emptyset$, and let P be a graph pattern in* NS–SPARQL. *If P is free from variable-only triple patterns and $\mathbf{I}(P) \subseteq \mathbf{I}(G_1)$, then $[\![P]\!]_{G_1 \cup G_2} = [\![P]\!]_{G_1}$.*

Proof. Proceed by induction over the structure of P. If P is a triple pattern, then it must mention some IRI in that is in $\mathbf{I}(G_1) \setminus \mathbf{I}(G_2)$. Hence, P can only match triples in G_1 and $[\![P]\!]_{G_1 \cup G_2} = [\![P]\!]_{G_1}$. The remaining cases are proven directly from the inductive definition of graph patterns, and therefore we omit them. □

We are finally ready to pinpoint the exact complexity of Eval(SP–SPARQL).

Theorem 7.3. Eval(SP–SPARQL) *is DP-complete.*

Proof. According to the definition, the evaluation problem for simple graph patterns corresponds to the language of all triples (G, P, μ) such that $\mu \in [\![P]\!]_G$, where $P = \mathrm{NS}(P')$ is a simple pattern. This language is in DP, since it can be expressed as the intersection of the following two languages.

$$\{(G, P, \mu) \mid P = \mathrm{NS}(P') \text{ is a simple pattern and } \mu \in [\![P']\!]_G\}, \tag{6}$$

$$\{(G, P, \mu) \mid P = \mathrm{NS}(P') \text{ is a simple pattern and there is no } \mu' \in [\![P']\!]_G \text{ s.t. } \mu \prec \mu'\}. \tag{7}$$

Language (6) is trivially reducible to Eval(SPARQL[AUFS]), which is in NP (Schmidt et al. 2010). The second language is in coNP as its complement consists of the triples $(G, \text{NS}(P), \mu)$ where $P = NS(P')$ (which is polynomially verifiable) and there is a mapping $\mu' \in [\![P']\!]_G$ such that $\mu \prec \mu'$. This is also in NP as μ' can be guessed nondeterministically before evaluating P' (which is in SPARQL[AUFS]).

Now we show that the evaluation problem for simple patterns is DP-hard. We provide a reduction from the well-known DP-complete problem SAT-UNSAT (Papadimitriou and Yannakakis 1982). This is the problem of deciding, given a pair of propositional formulas (φ, ψ), whether φ is satisfiable and ψ is unsatisfiable. Let (φ, ψ) be a pair of propositional formulas. Let μ_φ, P_φ, G_φ and μ_ψ, P_ψ, G_ψ be the elements provided by Lemma 7.1 corresponding to φ and ψ, respectively. By renaming variables and IRIs, we can assume w.l.o.g. that the IRIs and variables mentioned in μ_φ, P_φ, G_φ are disjoint from those mentioned in μ_ψ, P_ψ, G_ψ. Consider now the graph pattern $P = \text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))$. We show that $\mu_\varphi \in [\![P]\!]_{G_\varphi \cup G_\psi}$ if and only if $(\varphi, \psi) \in$ SAT-UNSAT. Notice that by Lemma 7.2 we have that $\mu_\varphi \in [\![P_\varphi]\!]_G$ if and only if $\mu_\varphi \in [\![P_\varphi]\!]_{G_\varphi}$ and that $\mu_\psi \in [\![P_\psi]\!]_G$ if and only if $\mu_\varphi \in [\![P_\psi]\!]_{G_\psi}$.

- [$\Rightarrow$] Suppose for the sake of contradiction that $\mu_\varphi \in [\![\text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))]\!]_G$ and that φ is unsatisfiable or ψ is satisfiable. We analyze these cases separately.
  - If φ is not satisfiable, then we know by Lemma 7.1 that $[\![P_\varphi]\!]_G = \emptyset$, which implies that $[\![\text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))]\!]_G = \emptyset$.
  - If ψ is satisfiable, then we have by Lemma 7.1 that $\mu_\psi \in [\![P_\psi]\!]_G$. Since $\text{var}(P_\varphi) \cap \text{var}(P_\psi) = \emptyset$ and $[\![P_\psi]\!]_G \neq \emptyset$, every mapping in $[\![P_\varphi]\!]_G$ is subsumed by some mapping in $[\![P_\varphi \text{ AND } P_\psi]\!]_G$. Hence, we obtain that

$$[\![\text{NS}(P_\varphi \text{ AND } P_\psi)]\!]_G \equiv [\![\text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))]\!]_G.$$

    We know by Lemma 7.1 that the empty mapping does not belong to $[\![P_\psi]\!]_G$, and therefore every mapping in $[\![P_\varphi \text{ AND } P_\psi]\!]_G$ mentions some variable in $\text{var}(P_\psi)$. As $\text{var}(\mu_\varphi) \cap \text{var}(P_\psi) = \emptyset$, we conclude that $\mu_\varphi \notin [\![P_\varphi \text{ AND } P_\psi]\!]_G$. Thus, μ_φ is not in $[\![\text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))]\!]_G$, which contradicts our initial assumption.
- [$\Leftarrow$] Assume φ is satisfiable and ψ is unsatisfiable. By Lemma 7.1, this implies that $[\![P_\psi]\!]_G = \emptyset$. Hence, in this case we have that $[\![\text{NS}(P_\varphi \text{ UNION } (P_\varphi \text{ AND } P_\psi))]\!]_G$ is the same as $[\![\text{NS}(P_\varphi)]\!]_G$. From Lemma 7.1, we have that $[\![P_\varphi]\!]_G = \{\mu_\varphi\}$ and, therefore, $\mu_\varphi \in [\![\text{NS}(P_\varphi)]\!]_G$, concluding the proof. □

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 5.8).

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 in these patterns. Let USP–SPARQL$_k$ be the fragment of USP–SPARQL consisting of all ns-patterns having at most k disjuncts each. To study the complexity of USP–SPARQL$_k$, we need to prove the following lemma.

Lemma 7.4. *Let $n \in \mathbb{N}$, and for every $i \in \{1, \ldots, n\}$ let μ_i, G_i and P_i be a mapping, an RDF graph and a graph pattern, respectively. If the following conditions hold, then*

- *for every $i, j \in \{1, \ldots, n\}$ with $i \neq j$, the variables and IRIs mentioned in (μ_i, P_i, G_i) are disjoint from the variables and IRIs mentioned in (μ_j, P_j, G_j);*

—*for every $i \in \{1, \ldots, n\}$, it is the case that P_i is a simple pattern which does not mention variable-only triple patterns,*
then there is a mapping μ, an ns-pattern P and an RDF graph G such that $\mu \in [\![P]\!]_G$ if and only if $\mu_i \in [\![P_i]\!]_{G_i}$ for some $i \in \{1, \ldots, n\}$. Moreover, μ, P, and G can be computed in polynomial time.

Proof. Let $n \in \mathbb{N}$, and for every $i \in \{1, \ldots, n\}$ let μ_i, G_i and P_i be a mapping, an RDF graph, and a graph pattern, respectively. First, define the mapping μ as $\mu_1 \cup \mu_2 \cup \cdots \cup \mu_n$. This mapping is correctly defined, since $\text{var}(\mu_i) \cap \text{var}(\mu_j) = \emptyset$ for every $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

Now define the RDF graph G as

$$G = \left(\bigcup_{i \in \{1, \ldots, n\}} G_i \right) \cup \left(\bigcup_{?X \in \text{dom}(\mu)} (\mu(?X), c^{?X}, d^{?X}) \right),$$

where $c^{?X}$ and $d^{?X}$ are distinct fresh IRIs for every $?X \in \text{dom}(\mu)$. Adding the new IRIs and their corresponding triples allows us to trivially match the graph to include the assignment $?X \to \mu(?X)$ in any mapping not mentioning $?X$. Based on this intuition, we proceed to create the ns-pattern P. Let $i \in \{1, \ldots, n\}$, and assume that $\text{dom}(\mu) \setminus \text{dom}(\mu_i) = \{?X_1, \ldots, ?X_\ell\}$. Assuming that $P_i = NS(Q_i)$, define the pattern P'_i as

$$P'_i = \text{NS}\left(Q_i \text{ AND } (?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})\right).$$

Finally, define the graph pattern P by

$$P = P'_1 \text{ UNION } P'_2 \text{ UNION } \cdots \text{ UNION } P'_n.$$

It is clear that the above elements μ, P, and G can be computed in polynomial time. Notice that if $\mu_i \in [\![P_i]\!]_{G_i}$, then μ_i will appear in the answer to Q_i over G, as $G_i \subseteq G$ and Q_i is monotone. Moreover, for every $?X \in \text{dom}(\mu) \setminus \text{dom}(\mu_i)$ the triple pattern $(?X, c^{?X}, d^{?X})$ will trivially match the RDF triple $(\mu(?X), c^{?X}, d^{?X})$.

Now that we have defined μ, P, and G, we formally prove that μ is in $[\![P]\!]_G$ if and only if $\mu_i \in [\![P_i]\!]_{G_i}$ for some $i \in \{1, \ldots, n\}$. Since $P = P'_1$ UNION P'_2 UNION $\cdots$ UNION P'_n, we know that $\mu \in [\![P]\!]_G$ if and only if $\mu \in [\![P'_i]\!]_G$ for some $i \in \{1, \ldots, n\}$. Thus, it is sufficient to show that for each $i \in \{1, \ldots, n\}$ it is the case that

$$\mu_i \in [\![P_i]\!]_{G_i} \text{ if and only if } \mu \in [\![P'_i]\!]_G. \tag{8}$$

Let $i \in \{1, \ldots, n\}$. Define the mapping μ_{-i} as μ restricted to $\text{dom}(\mu) \setminus \text{dom}(\mu_i)$. Assume $\text{dom}(\mu_{-i}) = \{?X_1, \ldots, ?X_\ell\}$. We have

$$P'_i = \text{NS}\left(Q_i \text{ AND } (?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})\right).$$

Since G contains every triple of the form $(\mu(?X), c^{?X}, d^{?X})$, and the IRIs $c^{?X}$ and $d^{?X}$ are not mentioned anywhere else in G, we know that

$$[\![(?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})]\!]_G = \{\mu_{-i}\}. \tag{9}$$

We make use of this fact to prove both directions of (8).

—[$\Rightarrow$] Assume $\mu_i \in [\![P_i]\!]_{G_i}$. By semantics of NS, it is the case that $\mu_i \in [\![Q_i]\!]_{G_i}$. Since Q_i is monotone and $G_i \subseteq G$, we have $\mu_i \in [\![Q_i]\!]_G$. As μ_i and μ_{-i} are compatible, by Equation (9) we obtain that

$$\mu_i \cup \mu_{-i} \in [\![Q_i \text{ AND } (?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})]\!]_G.$$

Finally, as $\mu_i \cup \mu_{-i} = \mu$ and $\text{dom}(\mu) = \text{var}(P'_i)$, we have $\mu \in [\![P'_i]\!]_G$.

—[⇐] Assume $\mu \in [\![P'_i]\!]_G$. By the semantics of the NS operator, we know that

$$\mu \in [\![Q_i \text{ AND } (?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})]\!]_G.$$

Provided that $[\![(?X_1, c^{?X_1}, d^{?X_1}) \text{ AND } \cdots \text{ AND } (?X_\ell, c^{?X_\ell}, d^{?X_\ell})]\!]_G = \{\mu_{-i}\}$, we have that $[\![Q_i]\!]_G$ must contain a mapping subsuming μ_i. This mapping must be exactly μ_i, as $\text{dom}(\mu_i) = \text{var}(Q_i)$. We conclude that $\mu_i \in [\![\text{NS}(Q_i)]\!]_G = [\![P_i]\!]_G$. From Lemma 7.2, we know that $[\![P_i]\!]_G = [\![P_i]\!]_{G_i}$, concluding the proof. □

We are now ready to pinpoint the exact complexity of Eval(USP–SPARQL$_k$).

Theorem 7.5. *For every $k > 0$, it holds that* Eval(USP–SPARQL$_k$) *is* BH$_{2k}$*-complete.*

Proof. We start by showing by induction that Eval(USP–SPARQL$_k$) $\in$ BH$_{2k}$. For $k = 1$, we have the evaluation problem for simple patterns, which we know is complete for DP $= BH_2$ (Theorem 7.3). For the inductive case let, $k > 0$ and assume Eval(USP–SPARQL$_k$) $\in$ BH$_{2k}$. We want to show that Eval(USP–SPARQL$_{k+1}$) $\in$ BH$_{2(k+1)}$. Consider the following two languages:

$$L_1 = \{(G, P_1 \text{ UNION } \cdots \text{ UNION } P_j, \mu) \mid j \leq k+1, \text{for each } i \in [1, j], P_i \text{ is a simple pattern, and } \mu \in [\![P_i]\!]_G \text{ for some } i \in [1, k]\}$$

$$L_2 = \{(G, P_1 \text{ UNION } \cdots \text{ UNION } P_{k+1}, \mu) \mid \text{ foreach } i \in [1, k+1], P_i \text{ is a simple pattern, and } \mu \in [\![P_{k+1}]\!]_G\}$$

Since Eval(USP–SPARQL$_k$) is in BH_{2k}, it is trivial to prove that $L_1 \in BH_{2k}$. Moreover, since Eval(SP–SPARQL) is in DP, it is trivial to show that L_2 is also in DP. By simply inspecting L_1 and L_2, we can see that $L_1 \cup L_2 =$ USP–SPARQL$_{k+1}$. We obtain that Eval$_{k+1}$ is the union between a problem in BH_{2k} and a problem in DP. It is known that such a union belongs to BH_{2k+2} (Wagner 1987) which concludes the containment part.

Let $k > 0$. To prove that Eval(USP–SPARQL$_k$) is BH_{2k}-hard, we make a reduction from the problem of knowing if a graph has chromatic number in the set $M_k = \{6k+1, 6k+3 \ldots, 8k-1\}$. This problem is known as Exact-M_k-Colorability and is BH_{2k}-complete (Riege and Rothe 2006). We will create a function that takes a graph H as input and generates an RDF graph G, a mapping μ and an ns-pattern $P = P_1$ UNION $\cdots$ UNION P_k, such that the chromatic number of G is in M_k if and only if $\mu \in [\![P]\!]_G$. Let H be a graph. Denote by $\{m_1, \ldots, m_k\}$ the elements in M_k. We know that the problem of verifying if a graph has chromatic number m is in DP for every m in M_k (Riege and Rothe 2006). Since the evaluation problem for simple patterns is DP-complete, for every $i \in \{1, \ldots, k\}$ we can generate in polynomial time an RDF graph G_i, a mapping μ_i, and a simple pattern P_i, such that $\mu_i \in [\![P_i]\!]_{G_i}$ if and only if H has chromatic number m_i. Moreover, we can assume w.l.o.g. that for $i \neq j$, the variables and IRIs mentioned in μ_i, G_i, and P_i are disjoint from those mentioned in μ_j, G_j, and P_j. Hence, by Lemma 7.4, we can construct in polynomial time a mapping μ, an ns-pattern P with k disjuncts, and an RDF graph G such that $\mu \in [\![P]\!]_G$ if and only if $\mu_i \in [\![P_i]\!]_{G_i}$ for some $i \in \{1, \ldots, k\}$. But as mentioned before, this occurs if and only if H has chromatic number in M_k. This implies that $\mu \in [\![P]\!]_G$ if and only if the chromatic number of H is in M_k, concluding the proof. □

Finally, we study the combined complexity of ns-patterns when the number of disjuncts is unbounded.

Theorem 7.6. Eval(USP–SPARQL) *is* $\mathrm{P}_{||}^{\mathrm{NP}}$*-complete.*

Proof. Let $P = P_1$ UNION P_2 UNION $\cdots$ UNION P_n be a graph pattern where every P_i ($1 \leq i \leq n$) is a simple pattern. Let G be an RDF graph and μ be a mapping. Since for every i the problem of deciding if $\mu \in [\![P_i]\!]_G$ belongs to DP, it can be solved by two parallel calls to an NP oracle. Thus, by making $2n$ calls in parallel to the NP oracle, one can decide whether μ belongs to $[\![P_i]\!]_G$ for some $i \in \{1, \ldots, n\}$. Therefore, deciding whether μ belongs to $[\![P]\!]_G$ can be achieved by a polynomial-time Turing machine that asks $2n$ queries to an NP oracle in parallel. We conclude that Eval(USP–SPARQL) $\in \mathrm{P}_{||}^{\mathrm{NP}}$.

Now we prove the problem is $\mathrm{P}_{||}^{\mathrm{NP}}$-hard by providing a reduction from the problem MAX-ODD-SAT. This is the problem of deciding, given a propositional formula φ, whether the truth-assignment that assigns true to the largest number of variables while satisfying φ, assigns true to an odd number of variables. This problem is shown to be $\mathrm{P}_{||}^{\mathrm{NP}}$-complete in Spakowski (2005).

Let φ be a propositional formula with m variables. We can assume without loss of generality that m is even (if not, consider the formula $\varphi \wedge \neg r$ for a fresh variable r). We want to create an ns-pattern P, an RDF graph G, and a mapping μ such that μ belongs to $[\![P]\!]_G$ if and only if φ belongs to MAX-ODD-SAT. It is easy to see that given a number k between 1 and m, the problem of deciding whether there is a truth assignment that satisfies φ and assigns true to at least k variables is in NP. Thus, by Cook's theorem, we can create a propositional formula φ_k such that φ_k is satisfiable if and only if there is a truth assignment that satisfies φ and assigns true to at least k variables. Hence, φ belongs to MAX-ODD-SAT if and only if $(\varphi_k, \varphi_{k+1})$ belongs to SAT-UNSAT for some odd k between 1 and $m-1$. By Theorem 7.3, for every such k, we can create a simple pattern P_k, a mapping μ_k, and an RDF graph G_k such that μ_k belongs to $[\![P_k]\!]_{G_k}$ if and only if $(\varphi_k, \varphi_{k+1}) \in$ SAT-UNSAT. We can assume without loss of generality that for every $j, k \in \{1, 3, \ldots, m-1\}$ with $j \neq k$, it is the case that $(\mathrm{dom}(\mu_j) \cup \mathrm{var}(P_j)) \cap (\mathrm{dom}(\mu_k) \cup \mathrm{var}(P_k)) = \emptyset$ and $(\mathrm{range}(\mu_j) \cup \mathbf{I}(P_j) \cup \mathbf{I}(G_j)) \cap (\mathrm{range}(\mu_k) \cup \mathbf{I}(P_k) \cup \mathbf{I}(G_k)) = \emptyset$. Hence, by Lemma 7.4, we can construct in polynomial time a mapping μ, an ns-pattern P, and an RDF graph G such that $\mu \in [\![P]\!]_G$ if and only if $\mu_i \in [\![P_i]\!]_{G_i}$ for some $i \in \{1, 3, \ldots, m-1\}$. As mentioned before, this occurs if and only if φ belongs to MAX-ODD-SAT, concluding the proof. □

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 (Letelier et al. 2013), which is higher than the complexity of the evaluation problem for USP–SPARQL (unless the polynomial-time hierarchy (Stockmeyer 1976) collapses to its second level as $\mathrm{P}_{||}^{\mathrm{NP}} \subseteq \Delta_2^P \subseteq \Sigma_2^P$).

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 $t \in \mathrm{ans}(Q, G)$?

As mentioned before, the fragment CONSTRUCT[AUF] is one of the most important fragments defined in this article, 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 7.7. EVAL(CONSTRUCT[AUF]) *is* NP*-complete.*

PROOF. Let $Q = (\text{CONSTRUCT } H \text{ WHERE } P)$ and let G and t be an RDF graph and an RDF triple, respectively. To decide if t is in $\text{ans}(Q, G)$ in polynomial time with a nondeterministic Turing machine, it suffices to guess a triple $h \in H$ and a mapping $\mu \in [\![P]\!]_G$ such that $\mu(h) = t$. It is clear that this can be done in NP, since EVAL(SPARQL[AUF]) is in NP. Now, to show NP-hardness, we provide a simple reduction from SAT. Let φ be a propositional formula. We want to define a construct query Q, an RDF graph G, and an RDF triple t such that $t \in \text{ans}(Q, G)$ if and only if φ is satisfiable. From Lemma 7.1, we know that there is a graph pattern P and an RDF graph G such that $[\![P]\!]_G$ is empty if and only if φ is unsatisfiable. Now define Q as CONSTRUCT (a, a, a) WHERE P. It is then trivial to show that the triple (a, a, a) belongs to $\text{ans}(Q, G)$ if and only if φ is satisfiable. Moreover, from Lemma 7.1 we know that the previous steps can be performed in polynomial time, which concludes our proof. □

This concludes our study of the complexity of the evaluation problem for the query languages introduced in this article. 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 PRACTICAL IMPLICATIONS

In this section, we discuss three practical implications of the results of this article.

8.1 Recursion in SPARQL

A first natural application of the characterization of monotone CONSTRUCT queries is an extension of SPARQL to support recursion. This topic has been previously studied in Reutter et al. (2015), where a recursive SPARQL query is defined as

$$\text{WITH RECURSIVE } g \text{ AS } Q_1\, Q_2, \tag{10}$$

where g is the *identifier* of an RDF graph, Q_1 a CONSTRUCT query, and Q_2 either a graph pattern, a CONSTRUCT query, or a recursive SPARQL query. In the following example, we illustrate how the semantics of such query is defined.

Example 8.1. Assume that we want to compute the transitive closure over the flight relation of an airline. Suppose this relation is encoded in triples of the form $(a, \textit{flight}, b)$, indicating that there is a direct flight from location a to location b. Assume also that the graph that is computed recursively has as identifier g_{tmp}. Then the transitive closure of flight is computed by the query:

$$Q = \text{WITH RECURSIVE } g_{\text{tmp}} \text{ AS } Q_1\, Q_2,$$

where Q_1 and Q_2 are defined as follows:

$$Q_1 = \text{CONSTRUCT } (?x, \text{route}, ?z) \text{ WHERE } [(?x, \text{flight}, ?z) \text{ UNION } (\text{GRAPH } g_{\text{tmp}} ((?x, \text{route}, ?y) \text{ AND } (?y, \text{route}, ?z)))]$$

$$Q_2 = \text{SELECT } \{?x, ?y\} \text{ WHERE } (\text{GRAPH } g_{\text{tmp}} (?x, \text{route}, ?y)).$$

Notice that in Q_1 the second disjunct of the WHERE clause uses the GRAPH operator of SPARQL. The syntax of this operator is (GRAPH g P), where g is the identifier of an RDF graph and P is a graph pattern. The result of evaluating the expression (GRAPH g P) is $[\![P]\!]_H$ assuming that H is the RDF graph identified by g.

Assume that we want to evaluate Q over an RDF graph G. Then we first need to evaluate Q_1 recursively until we obtain a fixed point, which is an RDF graph G_{tmp} with identifier g_{tmp}. In the computation of such fixed point, the graph pattern $(?x, flight, ?z)$ is evaluated over G, while $((?x, route, ?y)$ AND $(?y, route, ?z))$ is evaluated over the current version of G_{tmp}, since the GRAPH operator is used. Initially G_{tmp} is assumed to be empty. Then at the end of the first iteration for computing the fixed point, G_{tmp} contains one triple $(a, route, b)$ for each triple $(a, flight, b)$ in G. Moreover, at the end of the second iteration G_{tmp} contains one triple $(a, route, b)$ for each pair a, b of locations that are at distance at most two in G. Finally, the fixed point is reached when no new triples are added to G_{tmp}, in which case G_{tmp} contains the transitive closure of the relation flight in G. Once this fixed point has been reached, the query Q_2 is evaluated over G and G_{tmp} if the GRAPH operator indicates so. In fact, in this example the output of Q_2 is computed just by considering G_{tmp}, and it consists of all pairs (a, b) of locations for which there is a route from a to b.

The evaluation of a recursive query ends if the fixed point exists, which corresponds to the least fixed point given the evaluation procedure just described. As noticed in Reutter et al. (2015), it follows from the Knaster-Tarski Theorem (Libkin 2004) that such a least fixed point exists if the query Q_1 is monotone. However, no practical (or decidable) characterization of monotone queries existed when this recursive language was introduced, and therefore the authors of Reutter et al. (2015) simply decided to remove all forms of negation by disallowing the operator OPT. Our work provides a solid argument for this decision and, moreover, show that the fragment proposed contains an exhaustive syntax for unrestricted monotone queries. Indeed, from Corollary 6.4 it readily follows that by disallowing the operator OPT one obtains precisely the fragment of unrestricted monotone CONSTRUCT queries.

It should be noticed that SPARQL 1.1 allows the use of regular expressions (called property paths) to express navigation patterns on RDF graphs (Harris and Seaborne 2013). The use of such expressions provides a form of recursion; in fact, the query in Example 8.1 can be expressed in SPARQL 1.1. However, there exist natural queries that cannot be expressed in SPARQL 1.1 (Libkin et al. 2013; Rudolph and Krötzsch 2013) and that require the more general recursion mechanism proposed in Reutter et al. (2015).

8.2 Query Reformulation

The idea of reformulating queries is motivated by scenarios in which an original query refers to datasets that are partially or entirely not accessible, due to physical restrictions or logical constraints, and therefore it makes sense to re-write the query to only use a particular set of datasets or views (if possible). It turns out that this problem is closely related to interpolation (Segoufin and Vianu 2005; Nash et al. 2010; Benedikt et al. 2014); in fact, recent research in reformulation of relational queries has been driven by techniques similar to the ones used in the proof of Theorem 4.1 (Benedikt et al. 2014, 2016a, 2016b).

In the context of SPARQL, the notion of *restricted access* does not carry over directly from relational databases, as most translations from SPARQL to FO result in a single table containing all triples. However, by the very nature of RDF one expects that datasets are distributed between different locations, and therefore there are several situations in which one would prefer to avoid querying certain datasets (e.g., to reduce network or server loads). Formally, this can be formulated

in terms of the SPARQL operator GRAPH, with which queries can mention different RDF graphs where sub-queries have to be evaluated.[6]

Carrying reformulation techniques from the relational setting to SPARQL requires several steps. First, it is necessary to define an FO setting that allows for multiple RDF graphs. Then, given a SPARQL query mentioning the GRAPH operator and multiple RDF graphs, an equivalent FO formula over this new setting must be constructed. The resulting formula can be reformulated using interpolation techniques to only mention the desired RDF graphs. Finally, the reformulated formula needs to be translated back to SPARQL. For the sake of simplicity, the constructions performed in the proof of Theorem 4.1 do not consider the GRAPH operator. However, they can be extended to a more general setting in which RDF datasets consist of a set of RDF graphs. This setting can be translated to FO by considering a relation of arity 4, in which the first element of a tuple is the *name* of a graph, and the other three elements form an RDF triple in that graph (see Kostylev et al. (2015) for details). In this way, the interpolation techniques developed in this article can be used in a setting that allows for multiple RDF graphs and, in particular, could be of help for solving the query reformulation problem for SPARQL.

8.3 Evaluation of Weakly Monotone Graph Patterns

As already mentioned, the complexity of evaluating SPARQL graph patterns has been widely studied in the literature. However, the lack of a characterization of weakly monotone graph patterns has prevented the development of efficient algorithms for their evaluation. More precisely, the combined complexity of the evaluation problem for weakly monotone graph patterns is only known to be in PSPACE, which is not surprising as the evaluation of full SPARQL is already in PSPACE.

Contrary to weakly monotone graph patterns, other fragments of SPARQL have been proven to have lower complexity. For example, the complexity of evaluating disjunctions of well-designed graph patterns is complete for coNP (Pérez et al. 2009). Moreover, disjunctions of well-designed graph patterns can be efficiently recognized. These two facts already provide an optimization to the evaluation of SPARQL graph patterns. However, we know from Theorem 3.8 that this will not be an optimization for all weakly monotone queries.

On the other hand, the characterization presented in Corollary 5.12 tells that every union of unrestricted weakly monotone and subsumption-free SPARQL graph patterns can be written as an ns-pattern, which in turn can be evaluated more efficiently than an arbitrary SPARQL query. The evaluation problem for ns-patterns is $\mathrm{P}_{||}^{\mathrm{NP}}$-complete, while for arbitrary SPARQL queries it is PSPACE-complete. In particular, ns-patterns can be evaluated by solving a linear amount of NP problems in parallel, and then performing a polynomial number of steps.

9 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. Further, we proved that this also holds if disjunction is disallowed in both fragments. Given these negative results, 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 from first-order logic to SPARQL, which resulted in a theorem relating the fragment of weakly monotone graph patterns with SPARQL[AUFS]. This theorem

[6]This could also be formulated in terms of the SPARQL operator SERVICE, which allows to evaluate some sub-queries of a given query in remote datasets. However, from a theoretical point of view the effect of using GRAPH would be the same as the effect of using SERVICE.

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. We provided a thorough study of the combined complexity of the evaluation problem for the query languages introduced in the article and, finally, discussed potential applications of the main contributions of this article.

An issue that we did not address in this article is the role of the different operators in our results. For example, the FILTER and UNION operators are heavily used in our proofs. A natural question then is whether the notion of well-designedness captures weak-monotonicity for the fragment SPARQL[AO], which is still open.

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 expressive power.

Finally, the focus of this article 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 the practical consequences of replacing the operator OPTIONAL by the operator NS and whether the fragment of monotone 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 article.

APPENDIX

A PROOF OF THEOREM 4.1

In this appendix, we provide the complete proof of Theorem 4.1, that is, we show that 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 discussed in the article, the poof of this theorem is based on the interpolation theorems of Lyndon (1959) and Otto (2000). We start by presenting a translation from graph patterns to first-order formulas that will further allow us to apply these results over SPARQL in the unrestricted RDF setting.

Given a graph pattern P, 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 $\mathfrak{A} = \langle D, T^{\mathfrak{A}}, Dom^{\mathfrak{A}}, \{c_i^{\mathfrak{A}}\}_{i \in I(P)}, n^{\mathfrak{A}} \rangle$ 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 $i \in \mathbf{I}(P)$, 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 corresponds to G. We denote this set by $\mathcal{A}^P_G$, and when P is clear from the context, we simply write $\mathcal{A}_G$. 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 $X = \{?X_1, \ldots, ?X_\ell\}$ be a set of variables ordered under $\leq$. Given a mapping μ, we define the extension of μ to X as the function $\mu^X : \text{dom}(\mu) \cup X \to \mathbf{I}$ such that $\mu^X(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu)$, and $\mu^X(?X) = N$ for $?X \in X \setminus \text{dom}(\mu)$. Using this function we define the tuple corresponding to μ under X as $t_\mu^X = (\mu^X(?X_1), \ldots, \mu^X(?X_\ell))$. Given a graph pattern P, and a mapping μ, we define t_μ^P as $t_\mu^{\text{var}(P)}$. Moreover, we simplify notation by writing t_μ instead of $t_\mu^{\text{dom}(\mu)}$. Finally, if $\varphi(x_1, \ldots, x_n)$ is a first-order formula and $t = (i_1, \ldots, i_n)$ is a tuple of IRIs, then we abuse notation and write $\varphi(t)$ to represent $\varphi(c^{i_1}, \ldots, c^{i_n})$.

Having defined 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 $\mathfrak{A} \models \varphi_P(t_\mu^P)$. To this end, we need to take an intermediate step: We create one formula for each subset of $\text{var}(P)$. Intuitively, the formula corresponding to a subset X of $\text{var}(P)$ will *generate* the tuples corresponding to mappings that bind exactly X. We abuse notation by treating FO and SPARQL variables indistinctly. We also abuse notation by extending every mapping μ to $\text{dom}(\mu) \cup \mathbf{I}$, where $\mu(i) = i$ for every $i \in \mathbf{I}$.

Lemma A.1. *For every* SPARQL *graph pattern P, there is a set $\{\varphi_X^P(\bar{X})\}_{X \subseteq var(P)}$ of formulas in $\mathcal{L}^P_{RDF}$ satisfying the following condition: Given a mapping μ, an unrestricted RDF graph G and a $\mathcal{L}^P_{RDF}$ structure $\mathfrak{A} \in \mathcal{A}_G$, it is the case that $\mu \in [\![P]\!]_G$ if and only if $\mathfrak{A} \models \varphi^P_{dom(\mu)}(t_\mu)$.*

Proof. Let P be a SPARQL graph pattern. We proceed by induction on the structure of P.

—Let $P = (t_1, t_2, t_3)$ be a triple pattern and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. Since for every RDF graph G and mapping $\mu \in [\![P]\!]_G$ we have $\text{var}(P) = \text{dom}(\mu)$, define $\varphi_X^P(\bar{X})$ as a contradiction for every $X \subsetneq \text{var}(P)$. For $X = \text{var}(P)$, define

$$\varphi_X^P(\bar{X}) = T(t_1, t_2, t_3) \wedge Dom(t_1) \wedge Dom(t_2) \wedge Dom(t_3).$$

A mapping μ belongs to $[\![P]\!]_G$ if and only if $(\mu(t_1), \mu(t_2), \mu(t_3))$ belongs to G, which occurs if and only if $\mu(t_1), \mu(t_2), \mu(t_3) \in Dom^{\mathfrak{A}}$ and $(\mu(t_1), \mu(t_2), \mu(t_3)) \in T^{\mathfrak{A}}$, concluding this case.

—Let $P = P_1$ UNION P_2 and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. For every $X \subseteq \text{var}(P)$, define $\varphi_X^P(\bar{X})$ as

$$\varphi_X^P(\bar{X}) = \varphi_X^{P_1}(\bar{X}) \vee \varphi_X^{P_2}(\bar{X}).$$

Let μ be a mapping and $X = \text{dom}(\mu)$. By semantics of UNION, $\mu \in [\![P]\!]_G$ if and only if $\mu \in [\![P_1]\!]_G \cup [\![P_2]\!]_G$. Hence, by hypothesis we have $\mu \in [\![P]\!]_G$ if and only if $\mathfrak{A} \models \varphi_X^{P_1}(t_\mu)$ or $\mathfrak{A} \models \varphi_X^{P_2}(t_\mu)$, which is the semantic definition of $\mathfrak{A} \models \varphi_X^{P_1}(t_\mu) \vee \varphi_X^{P_2}(t_\mu)$.

—Let $P = P_1$ AND P_2 and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. For every $X \subseteq \text{var}(P)$ define the formula $\varphi_X^P(\bar{X})$ as

$$\varphi_X^P(\bar{X}) = \bigvee_{X_1 \cup X_2 = X} \left[\varphi_{X_1}^{P_1}(\bar{X_1}) \wedge \varphi_{X_2}^{P_2}(\bar{X_2})\right].$$

Let μ be a mapping and let $X = \text{dom}(\mu)$. If μ belongs to $[\![P]\!]_G$, then there are two compatible mappings $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ such that $\mu = \mu_1 \cup \mu_2$. Let $X_1 = \text{dom}(\mu_1)$ and $X_2 = \text{dom}(\mu_2)$. By hypothesis, we know that $\mathfrak{A} \models \varphi_{X_1}^{P_1}(t_{\mu_1})$ and $\mathfrak{A} \models \varphi_{X_2}^{P_2}(t_{\mu_2})$, which is equivalent to $\mathfrak{A} \models \varphi_{X_1}^{P_1}(t_{\mu_1}) \wedge \varphi_{X_2}^{P_2}(t_{\mu_2})$. As $X_1 \cup X_2 = X$, we have $\mathfrak{A} \models \varphi_X^P(t_\mu)$.

For the converse, if $\mathfrak{A} \models \varphi^P_{\text{dom}(\mu)}(t_\mu)$, then there are two sets X_1 and X_2 such that $X_1 \cup X_2 = X$ and both $\mathfrak{A} \models \varphi_{X_1}^{P_1}(t_\mu^{X_1})$ and $\mathfrak{A} \models \varphi_{X_2}^{P_1}(t_\mu^{X_2})$ hold. Define μ_i as μ restricted to X_i ($i \in \{1, 2\}$). It follows from the hypothesis that $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$. Since μ_1 and μ_2 are compatible

(they are both restrictions of μ) and $\mu = \mu_1 \cup \mu_2$, this implies $\mu \in [\![P]\!]_G$, which was to be shown.

—Let $P = P_1$ OPT P_2, and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. For every $X \subseteq \text{var}(P)$ define the formula $\varphi^P_X(\bar{X})$ as

$$\varphi^P_X(\bar{X}) = \varphi^{P_1 \text{ AND } P_2}_X(\bar{X}) \vee \varphi^P_{\text{MINUS}\,X}(\bar{X}),$$

where $\varphi^P_{\text{MINUS}\,X}(\bar{X})$ is defined as

$$\varphi^{P_1}_X(\bar{X}) \wedge \neg \bigvee_{X' \subseteq \text{var}(P_2)} \exists \overline{X' \setminus X} \left(\bigwedge_{x' \in X'} Dom(x') \wedge \varphi^{P_2}_{X'}(\bar{X}') \right).$$

Here the notation $\exists \overline{S}$ for a set $S = \{s_1, \ldots, s_n\}$ stands for $\exists s_1 \cdots \exists s_n$. Let now μ be a mapping, and let $X = \text{dom}(\mu)$. Notice that μ belongs to $[\![P_1 \text{ AND } P_2]\!]_G$ or to $[\![P_1]\!]_G \setminus [\![P_2]\!]_G$. In the first case, we know that $\mathfrak{A} \models \varphi^{P_1 \text{ AND } P_2}_X(t_\mu)$. It remains show that if $\mu \in [\![P_1]\!]_G \setminus [\![P_2]\!]_G$, then $\mathfrak{A} \models \varphi^P_{\text{MINUS},X}(t_\mu)$. As $\mu \in [\![P_1]\!]_G$, we know $\mathfrak{A} \models \varphi^{P_1}_X(t_\mu)$, so we only need to prove that there is no set $X' \subseteq \text{var}(P_2)$ such that $\mathfrak{A} \models \varphi^{P_2}_{X'}(t_{\mu'})$ for some mapping μ' compatible with μ. But if this was the case, then, by hypothesis, we would have $\mu' \in [\![P_2]\!]_G$. This contradicts the fact that $\mu \in [\![P_1]\!]_G \setminus [\![P_2]\!]_G$ as μ' and μ are obviously compatible.

For the converse, assume $\mathfrak{A} \models \varphi^P_X(t_\mu)$, where $X = \text{dom}(\mu)$. If $\mathfrak{A} \models \varphi^{P_1 \text{ AND } P_2}_X(t_\mu)$, then we know by the AND case that $\mu \in [\![P_1 \text{ AND } P_2]\!]_G$ and hence $\mu \in [\![P]\!]_G$. The remaining case is when $\mathfrak{A} \models \varphi^P_{\text{MINUS}\,X}(t_\mu)$. If this is the case, then by hypothesis it readily follows that $\mu \in [\![P_1]\!]_G$. Now we have to prove that μ is not compatible with any mapping in $[\![P_2]\!]_G$. Proceed by contradiction. Assume there is a mapping $\mu' \in [\![P_2]\!]_G$ compatible with μ. We know $\mathfrak{A} \models \varphi^{P_2}_{X'}(t_{\mu'})$ where $X' = \text{dom}(\mu')$. Since μ and μ' are compatible, μ' can be obtained by extending the assignments in μ, and thus $\mathfrak{A}$ would not satisfy $\varphi^P_{\text{MINUS}\,X}(t_\mu)$, which leads to a contradiction.

—Let $P = \text{SELECT}\,V$ WHERE Q, and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. For every $X \subseteq \text{var}(P)$ such that $X \not\subseteq V$, the formula $\varphi^P_X(\bar{X})$ is defined as a contradiction. It is immediate to show that this satisfies the equivalence, as P cannot output variables not mentioned in V. Now, for every $X \subseteq \text{var}(P) \cap V$ define the formula $\varphi^P_X(\bar{X})$ as

$$\varphi^P_X(\bar{X}) = \bigvee_{X \subseteq Y \subseteq \text{var}(P)} \exists \overline{Y \setminus X} \left(\bigwedge_{y \in Y} Dom(y) \wedge \varphi^Q_Y(\bar{Y}) \right).$$

Let μ be a mapping and let $X = \text{dom}(\mu)$. If μ belongs to $[\![P]\!]_G$, then there is a mapping $\mu' \in [\![Q]\!]_G$ such that $\mu'_{|X} = \mu$. Let $Y = \text{dom}(\mu')$. We have by hypothesis that $\mathfrak{A} \models \varphi^Q_Y(t_{\mu'})$, where $Y = \text{dom}(\mu')$. Since $t_{\mu'}$ is a tuple extending t_μ, by replacing the free variables in X according to t_μ, we obtain that $\mathfrak{A} \models \exists(Y \setminus X)\varphi^Q_Y(t_{\mu'})$. It readily follows that $\mathfrak{A} \models \varphi^P_X(t_\mu)$.

For the converse, if $\mathfrak{A} \models \varphi^P_{\text{dom}(\mu)}(t_\mu)$, then there must be a tuple $\bar{a}$ that extends t_μ and a set of variables Y with $X \subseteq Y \subseteq \text{var}(P)$, such that $\mathfrak{A} \models \varphi^Q_Y(\bar{a})$. Let μ' be the mapping such that $t_{\mu'} = \bar{a}$. By hypothesis, we have that $\mu' \in [\![Q]\!]_G$. But since μ' corresponds to $\bar{a}$ and $\bar{a}$ extends t_μ, we have that $\mu'_{|X} = \mu$. We conclude that $\mu \in [\![\text{SELECT}\,V \text{ WHERE } Q]\!]_G$.

—Let $P = P_1$ FILTER R and let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G. For every $X \subseteq \text{var}(P)$, define $\varphi^P_X(\bar{X})$ as

$$\varphi^P_X(\bar{X}) = \varphi^{P_1}_X(\bar{X}) \wedge \varphi_R(\bar{X}),$$

where $\varphi_R(\bar{X})$ is inductively defined as follows:

—If R is an equality and $\text{var}(R) \not\subseteq X$, then $\varphi_R = \text{False}$.
—If R is an equality and $\text{var}(R) \subseteq X$, then $\varphi_R = R$.
—If $R = bound(x)$ and $x \notin X$, then $\varphi_R = \text{False}$.
—If $R = bound(x)$ and $x \in X$, then $\varphi_R = \text{True}$.
—If R is of the form $\neg R_1$, $R_1 \wedge R_2$, or $R_1 \vee R_2$ for filter conditions R_1 and R_2, then φ_R is the corresponding Boolean combination of φ_{R_1} and φ_{R_2}.

Let μ be a mapping and let $X = \text{dom}(\mu)$. It is easy to see from the definition of φ_R that $\mathfrak{A} \models \varphi_R(t_\mu)$ if and only if $\mu \models R$. By hypothesis, we have $\mu \in [\![P_1]\!]_G$ if and only if $\mathfrak{A} \models \varphi^{P_1}_{\text{dom}(\mu)}(t_\mu)$, and hence it readily follows that $\mathfrak{A} \models \varphi^{P_1}_{\text{dom}(\mu)}(t_\mu) \wedge \varphi_R(\bar{X})$ if and only if $\mu \in [\![P_1]\!]_G$ and $\mu \models R$, which was to be shown. □

Given a SPARQL graph pattern, the previous lemma allows us to construct a set of formulas that together, in a sense, are equivalent to P. We now need to transform such set into one single formula. As it can be foreseen, the main issue in this transformation is that mappings do not bind every variable in $\text{var}(P)$, while in an FO formula the set of free variables is fixed and output tuples must bind all free variables.

Lemma A.2. *For every* SPARQL *graph pattern P there is a first-order formula φ_P in $\mathcal{L}^P_{RDF}$ that is equivalent to P.*

Proof. Let P be a SPARQL graph pattern, and let $\{\varphi^P_X(\bar{X})\}_{X \subseteq \text{var}(P)}$ be the set of formulas obtained from applying Lemma A.1 to P. Let $Y = \text{var}(P)$ and define the first-order formula $\varphi_P(Y)$ as follows:

$$\varphi_P(Y) = \bigvee_{X \subseteq Y} \left[\varphi^P_X(X) \wedge \bigwedge_{z \in Y \setminus X} z = n \right].$$

We show that P and φ_P are equivalent. Let $\mathfrak{A} \in \mathcal{A}_G$ for some unrestricted RDF graph G.

—[⇒] Let $\mu \in [\![P]\!]_G$ and let $X = \text{dom}(\mu)$. By definition, t^P_μ assigns n to all variables in $\text{var}(P) \setminus X$. It follows that $\mathfrak{A}$ satisfies $y = n$ for every $y \in \text{var}(P) \setminus X$ when variables are assigned according to t^P_μ. Also, we know from Lemma A.1 that $\mathfrak{A} \models \varphi^P_X(t_\mu)$. Hence we have that $\mathfrak{A} \models \varphi_P(t^P_\mu)$, concluding this direction.
—[⇐] Let μ be a mapping such that $\mathfrak{A} \models \varphi_p(\mu^P_t)$. Since the variables equal to n in t^P_μ are precisely those not in $\text{dom}(\mu)$, the only disjunct that can be satisfied is that in which $X = \text{dom}(\mu)$. Hence, we know that $\mathfrak{A} \models \varphi^P_{\text{dom}(\mu)}(t_\mu)$. From Lemma A.1, this directly implies $\mu \in [\![P]\!]_G$, concluding the proof. □

It is important to mention that the previous transformation creates an FO formula with the same set of free variables as in the original graph pattern. Having a transformation from SPARQL to FO, we can now apply the previously mentioned interpolation techniques. Notice that the formula resulting from a graph pattern can have equalities, and hence we use the version of Otto's interpolation theorem in which equalities are allowed.

Next, we need to write a formula expressing weak monotonicity in our FO setting. Given two tuples $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$ of the same length, define the formula $\bar{x} \preceq \bar{y}$ by

$$\bar{x} \preceq \bar{y} = \bigwedge_{i=1}^{n} (x_i = y_i \vee x_i = n).$$

It can be seen that given two mappings μ_1 and μ_2, it is the case that μ_1 is subsumed by μ_2 if and only if $t^V_{\mu_1} \preceq t^V_{\mu_2}$ for every set V of variables. Define now the extended vocabulary $\mathcal{L}^{P2}_{\text{RDF}}$ as $\mathcal{L}^P_{\text{RDF}} \cup$

$\{T', Dom'\}$, where T' is a ternary predicate symbol and Dom' is a unary predicate symbol. We say that an $\mathcal{L}^{P2}_{\text{RDF}}$-structure $\mathfrak{A}$ corresponds to a graph G if the restriction of $\mathfrak{A}$ to $\mathcal{L}^{P}_{\text{RDF}}$ corresponds to G. Let P be a SPARQL graph pattern and let be the $\mathcal{L}^{P}_{\text{RDF}}$-formula obtained from applying Lemma A.2 to a SPARQL graph pattern φ_P. Consider the following $\mathcal{L}^{P2}_{\text{RDF}}$-formula:

$$[\varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \rightarrow \exists \bar{y}(\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y})). \tag{11}$$

The idea behind this formula is to state that P is a weakly monotone graph pattern. Indeed, it is not hard to see that if P is a weakly monotone graph pattern, then Equation (11) is satisfied by structures that represent RDF graphs. However, there are structures that do not correspond to any RDF graph, and therefore we cannot assert that the previous formula is a tautology. As we will see later, interpolation techniques can only be applied over tautologies. To overcome this problem, we define a sentence for stating that the structure corresponds to an RDF graph. This sentence is defined as

$$\Sigma_{\text{RDF}} = \bigwedge_{i \in I} c_i \neq n \wedge \bigwedge_{i \neq j} c_i \neq c_j \wedge \neg Dom(n).$$

It is easy to see that an $\mathcal{L}^{P2}_{\text{RDF}}$-structure satisfies Σ_{RDF} if and only if it corresponds to some RDF graph. By including Σ_{RDF} in the left-hand side of the implication we obtain a proper tautology:

$$[\Sigma_{\text{RDF}} \wedge \varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \rightarrow \exists \bar{y}(\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y})). \tag{12}$$

Assume now that P is weakly monotone. Since φ_P is equivalent to P (which is weakly monotone), we know that the above implication is a tautology. Therefore, we can obtain an interpolant $\theta(T', Dom', \bar{x})$, satisfying the conditions from Ottos's interpolation theorem. Unfortunately, and as opposed to the applications of interpolation in FO, this interpolant is not necessarily equivalent to φ_P. In particular, we will see that the interpolant is not only weakly monotone but also monotone. However, there is a strong connection between φ_P and θ, which is again stated in terms of subsumption equivalence.

Definition A.3. Let $\varphi(\bar{x})$ and $\psi(\bar{x})$ be two FO formulas. We say that φ and ψ are subsumption-equivalent under RDF graphs if for every structure $\mathfrak{A}$ corresponding to an RDF graph and every tuple $\bar{a}$, if $\mathfrak{A} \models \varphi(\bar{a})$, then there is a tuple $\bar{b}$ such that $\mathfrak{A} \models \psi(\bar{b})$ and $a \preceq b$ and vice versa.

Now we are ready to prove the main result obtained from translating SPARQL to FO and applying interpolation.

Theorem A.4. *Let P be a weakly monotone graph pattern. Then there are two $\mathcal{L}^{P}_{RDF}$-formulas φ and ψ such that φ is equivalent to P, ψ is subsumption-equivalent to φ under RDF graphs, and ψ is positive existential.*

Proof. Let P be a weakly monotone graph pattern and let φ_P be the $\mathcal{L}^{P}_{\text{RDF}}$-formula obtained from Theorem A.2. Since φ_P is equivalent to P, which is weakly monotone, we know that formula (12) is a tautology. Then, there is an interpolant θ that satisfies the following conditions:

(1) $[\Sigma_{\text{RDF}} \wedge \varphi_P(T, Dom, \bar{x}) \wedge T \subseteq T' \wedge Dom \subseteq Dom'] \rightarrow \theta(T', Dom', \bar{x})$ is a tautology,
(2) $\theta(T', Dom', \bar{x}) \rightarrow \exists \bar{y}(\bar{x} \preceq \bar{y} \wedge \varphi_P(T', Dom', \bar{y}))$ is a tautology,
(3) θ only mentions the predicates T' and Dom',
(4) θ is positive in both T' and Dom'.

As φ_P and Σ_{RDF} are $\{Dom, Dom'\}$-relativized and Dom is not mentioned in θ, we can deduce that θ is $\{Dom'\}$-relativized. Moreover, Dom' is mentioned positively in θ from which we conclude that θ is an existential formula. Now it only remains to show that φ_P and θ are subsumption-equivalent under RDF graphs. Let $\mathfrak{A}$ be an $\mathcal{L}^{P}_{\text{RDF}}$-structure corresponding to an RDF graph G. Define $\mathfrak{A}'$ as the structure that results from extending $\mathfrak{A}$ with $T' = T$ and $Dom' = Dom$.

— [⇒] For this direction we prove a stronger result: every answer to $\varphi_P(T, Dom)$ is also an answer to $\theta(T, Dom)$. Let $\bar{a}$ be a tuple such that $\mathfrak{A} \models \varphi_P(T, Dom, \bar{a})$. Since $\varphi_P(T, Dom, \bar{x})$ does not mention T' nor Dom', we know that $\mathfrak{A}' \models \varphi_P(T, Dom, \bar{a})$. We also know that both $T \subseteq T'$ and $Dom \subseteq Dom'$ hold in $\mathfrak{A}$, so by condition (1) we have that $\mathfrak{A}' \models \theta(T', Dom', \bar{a})$. It immediately follows that $\mathfrak{A}' \models \theta(T, Dom, \bar{a})$.
— [⇐] Let $\bar{a}$ be a tuple such that $\mathfrak{A} \models \theta(T, Dom, \bar{a})$. Since $\theta(T, Dom, \bar{x})$ mentions neither T' nor Dom', and in $\mathfrak{A}'$ we have $T = T'$ and $Dom = Dom'$, we know that $\mathfrak{A}' \models \theta(T', Dom', \bar{a})$. By condition (2), there must be a tuple $\bar{b}$ such that $\bar{a} \preceq \bar{b}$ and $\mathfrak{A}' \models \varphi_P(T', Dom', \bar{b})$. Since $T = T'$ and $Dom = Dom'$, we conclude that $\varphi_P(T, Dom, \bar{b})$.

We have that θ is a positive existential formula that is subsumption-equivalent under RDF graphs to φ_P, which is equivalent to P, concluding the proof. □

The previous theorem establishes what we obtain from applying interpolation to formulas related to weakly monotone SPARQL graph patterns. We now know that given a weakly monotone SPARQL graph pattern P, there is a positive existential formula φ_P that is subsumption-equivalent under RDF graphs to a formula that is equivalent to P. To simplify notation, whenever this is the case we say that φ_P is subsumption-equivalent to P, and write $\varphi_P \equiv_s P$. Next we discuss how to transform such formula φ_P back into SPARQL, and what is the syntactic form of the obtained graph pattern. In this transformation, we do not need the equivalence for all structures that correspond to RDF graphs. Instead, we use a weaker notion of equivalence in which the correspondence between the formula and the graph pattern only holds in very specific structures.

Definition A.5. Let G be an unrestricted RDF graph and let P be a graph pattern. The first-order structure that represents G for P is denoted by G^P_{FO} and is defined as the only $\mathcal{L}^P_{\text{RDF}}$-structure with domain $\mathbf{I}(G) \cup \{N\}$ that corresponds to G.

Notice that in a structure representing an unrestricted RDF graph G, Dom is interpreted as $\mathbf{I}(G)$ and T corresponds precisely to the triples in G. Now we define the notion of equivalence that will hold when transforming a first-order positive existential formula into a SPARQL graph pattern.

Definition A.6. Given an $\mathcal{L}^P_{\text{RDF}}$-formula φ_P and a graph pattern P, we say that P and φ_P are equivalent in RDF structures, denoted by $P \equiv_{\text{RDF}} \varphi$, if for every mapping μ and *RDF* graph G, it is the case that $\mu \in [\![P]\!]_G$ if and only if $G^P_{\text{FO}} \models \varphi(t^P_\mu)$.

We also use the relation $\equiv_{\text{RDF}}$ between two FO formulas to assert that they coincide in every structure representing an RDF graph for some pattern. Now we proceed to define the aforementioned transformation: Given a positive existential formula φ_P, we construct a graph pattern P such that $\varphi \equiv_{\text{RDF}} P$. To this end, we first transform our formula into a union of conjunctive queries (UCQ) with inequalities. A UCQ with inequalities is a formula of the form $\varphi_P(\bar{x}) = \exists \bar{y}_1 \varphi_1(\bar{y}_1, \bar{x}) \vee \cdots \vee \exists \bar{y}_n \varphi_n(\bar{y}_n, \bar{x})$, where for every $i \in \{1, \ldots, n\}$:

— φ_i is a conjunction of equalities, inequalities, and positive occurrences of predicates, and
— the set of free variables in φ_i is precisely the set of free variables in φ_P.

Denote by $\text{UCQ}^{\neq}$ the set of all UCQs with inequalities. Before proceeding with our translation, we need to prove that positive existential formulas can be translated to UCQs with inequalities under certain conditions.

Lemma A.7. *Let G be an unrestricted RDF graph and let P be a graph pattern. Let φ_P be a positive existential $\mathcal{L}^P_{RDF}$-formula that is subsumption-equivalent to P. There is an $\mathcal{L}^P_{RDF}$-formula γ in $\text{UCQ}^{\neq}$ such that*

—*The predicate Dom does not occur in* γ,
—*Every equality and inequality in* γ *contains at least one variable,*
—$\varphi_P \equiv_{\text{RDF}} \gamma$.

Proof. Let φ_P be a positive existential formula in $\mathcal{L}^P_{\text{RDF}}$. Define

$$Adom(x) = \exists y \exists z (T(x, y, z) \vee T(y, x, z) \vee T(y, z, x)),$$

and let φ_T be the result of replacing in φ_P every occurrence of *Dom* by *Adom*. It is clear that φ_T is also a positive existential formula and that it does not mention the predicate *Dom*. It is also clear that $\varphi_T \equiv_{\text{RDF}} \varphi_P$, since in every structure representing an RDF graph *Dom* and *Adom* are equivalent. Since φ_T is positive existential, we can assume w.l.o.g. that $\varphi_T(\bar{x}) = \exists \bar{y}_1 \varphi_1(\bar{y}_1, \bar{x}_1) \vee \cdots \vee \exists \bar{y}_n \varphi_n(\bar{y}_n, \bar{x}_n)$, where each φ_i is a conjunction of equalities, inequalities, and positive occurrences of predicates (Abiteboul et al. 1995) . Notice, however, that the free variables in the disjuncts are not necessarily $\bar{x}$. Let ψ be the result of applying the next procedure over φ_T:

(1) remove every equality between two equal constants,
(2) remove every disjunct with an occurrence of T mentioning the constant n,
(3) remove every disjunct with an equality between two distinct constants.

Since equalities between equal constants are tautologies, the first operation does not alter the formula. In a structure corresponding to an RDF graph, the element associated with the constant n does not appear in any occurrence of the predicate T, so the second operation preserves the equivalence $\equiv_{\text{RDF}}$. Moreover, in structures corresponding to RDF graphs every two constants have different interpretation, and hence the third operation also preserves the equivalence in these structures. It follows that $\varphi_T \equiv_{\text{RDF}} \psi$. Finally, we transform ψ into a formula γ such that every disjunct of γ has the same free variables as ψ. Assume that $\psi(\bar{x}) = \exists \bar{y}_1 \psi_1(\bar{y}_1, \bar{x}_1) \vee \cdots \vee \exists \bar{y}_n \psi_n(\bar{y}_n, \bar{x}_n)$, where $\bar{x}$ is the set of free variables in ψ and $\bar{x}_i$ is the set of free variables in ψ_i. For $i \in \{1, \ldots, n\}$, define

$$\gamma_i(\bar{x}) = \bigvee_{X \subseteq \bar{x} \setminus \bar{x}_i} \exists \bar{y}_i \left(\psi_i(\bar{x}_i, \bar{y}_i) \wedge \bigwedge_{x \in X} Adom(x) \wedge \bigwedge_{x \in \bar{x} \setminus (\bar{x}_i \cup X)} x = n \right).$$

Finally, let $\gamma(\bar{x}) = \gamma_1(\bar{x}) \vee \cdots \vee \gamma_n(\bar{x})$. We show that $\gamma(\bar{x}) \equiv_{\text{RDF}} \psi(\bar{x})$.

—Let G be an RDF graph and let $\bar{a}$ be a tuple such that $G^P_{\text{FO}} \models \gamma(\bar{a})$. Hence, $G^P_{\text{FO}} \models \gamma_i(\bar{a})$ for some $i \in \{1, \ldots, n\}$. This implies that $G^P_{\text{FO}} \models \exists \bar{y}_i(\psi(\bar{a}_i, \bar{y}_i))$, where $\bar{a}_i$ is the tuple that results from restricting $\bar{a}$ to $\bar{x}_i$. By the definition of ψ, this implies that $G^P_{\text{FO}} \models \psi(\bar{a})$.
—Let G be an RDF graph and let $\bar{a}$ be a tuple such that $G^P_{\text{FO}} \models \psi(\bar{a})$. Hence, $G^P_{\text{FO}} \models \exists \bar{y}_i(\psi(\bar{a}_i, \bar{y}_i))$ for some $i \in \{1, \ldots, n\}$, where a_i is the tuple that results from restricting $\bar{a}$ to $\bar{x}_i$. Let X be the set of variables in $\bar{a} \setminus \bar{a}_i$ that are assigned to elements mentioned in $T^{G^P_{\text{FO}}}$. Since the only element not mentioned in $T^{G^P_{\text{FO}}}$ is N, and N is assigned to the constant n, it is clear that

$$G^P_{\text{FO}} \models \exists \bar{y}_i \left(\psi(\bar{x}_i, \bar{y}_i) \wedge \bigwedge_{x \in X} Adom(x) \wedge \bigwedge_{x \in \bar{x} \setminus (\bar{x}_i \cup X)} x = n \right)$$

and hence $G^P_{\text{FO}} \models \gamma(\bar{a})$.

We have that γ satisfies the desired conditions. In particular, the set of free variables in every disjunct of γ is $\bar{x}$. This concludes the proof, since $\varphi_P \equiv_{\text{RDF}} \varphi_T \equiv_{\text{RDF}} \psi \equiv_{\text{RDF}} \gamma$. □

Having this equivalence, we can now present the main transformation from positive existential formulas to SPARQL. As previously mentioned, we are particularly interested in the syntactic form of the resulting graph pattern.

THEOREM A.8. *Let φ_P be a positive existential formula. There is a graph pattern P in* SPARQL[AUFS] *such that $\varphi \equiv_{\text{RDF}} P$.*

PROOF. By Lemma A.7 we can assume w.l.o.g. that (1) φ_P is in UCQ$^{\neq}$, (2) in φ_P the predicate *Dom* is not mentioned, (3) the constant n is not present in any occurrence of T, and (4) every equality and inequality mentions at least one variable. We proceed by transforming the disjuncts of φ_P into SPARQL graph patterns. Suppose that φ_P is the following formula:

$$\varphi(\bar{x}) = \exists \bar{y}_1 \varphi_1(\bar{y}_1, \bar{x}) \vee \cdots \vee \exists \bar{y}_j \varphi_j(\bar{y}_j, \bar{x}).$$

Fix $k \in \{1, \ldots, j\}$ and assume

$$\varphi_k = T(u_1, v_1, w_1) \wedge \cdots \wedge T(u_n, v_n, w_n) \wedge a_1 = b_1 \wedge \cdots \wedge a_m = b_m \wedge c_1 \neq d_1 \wedge \cdots \wedge c_\ell \neq d_\ell,$$

Where u_i, v_i, and w_i are either variables or IRIs, and a_i, b_i, c_i, and d_i are variables, IRIs, or the constant n (for i in the suitable intervals). For every equality $a_i = b_i$, define the filter condition R_i piecewise as $\neg\text{bound}(?X)$ if $\{a_i, b_i\} = \{n, ?X\}$ and $a_i = b_i$ otherwise. For every inequality $c_i \neq d_i$, define the filter condition S_i piecewise as $\text{bound}(?X)$ if $\{c_i, d_i\} = \{n, ?X\}$ and $c_i \neq d_i$ otherwise. Define the graph pattern Q_k as

$$Q_k = ((u_1, v_1, w_1) \text{ AND } \cdots \text{ AND } (u_n, v_n, w_n)) \text{ FILTER } (R_1 \wedge \cdots \wedge R_m \wedge S_1 \wedge \cdots \wedge S_\ell)\,.$$

By the conditions of Lemma A.7, this graph pattern is well defined, and the free variables in φ_k and Q_k are exactly $\bar{x}$. We now need to prove that $\varphi_k \equiv_{\text{RDF}} Q_k$.

- [$\Rightarrow$] Let G be an RDF graph and $\mu \in [\![Q_k]\!]_G$. This implies that $\mu((u_i, v_i, w_i)) \in G$. By the definition of G^P_{FO} and $t^{Q_k}_\mu$, we have $G^P_{\text{FO}} \models T(u_i, v_i, w_i)$ for each $i \in \{1, \ldots, n\}$ when replacing variables according to $t^{Q_k}_\mu$. Now let $i \in \{1, \ldots, n\}$. Recall from the definition of $t^{Q_k}_\mu$ that variables that are not bound in μ are assigned to N in $t^{Q_k}_\mu$. Hence, as $\mu \models R_i$ and $\mu \models S_i$, it is easy to see that $G^P_{\text{FO}} \models (a_i = b_i)$ and $G^P_{\text{FO}} \models (c_i \neq d_i)$ when replacing variables according to $t^{Q_k}_\mu$. We conclude that G^P_{FO} satisfies each conjunct of φ_k when variables are replaced according to μ, and therefore $G^P_{\text{FO}} \models \varphi_k(t^{Q_k}_\mu)$.
- [$\Leftarrow$] Let G be an RDF graph and let μ be a mapping such that $G^P_{\text{FO}} \models \varphi_k(t^{Q_k}_\mu)$. We have that $G^P_{\text{FO}} \models T(u_i, v_i, w_i)$ when replacing variables according to $t^{Q_k}_\mu$, and hence $\mu((u_i, v_i, w_i)) \in G$ for each $i \in \{1, \ldots, n\}$. Again, the variables assigned to N by $t^{Q_k}_\mu$ are precisely those variables not bound by μ. Hence, as $G^P_{\text{FO}} \models (a_i = b_i)$ and $G^P_{\text{FO}} \models (c_i \neq d_i)$ when replacing variables according to $t^{Q_k}_\mu$, it is easy to see that $\mu \models R_i$ and $\mu \models S_i$. By the semantics of AND and FILTER, we conclude that $\mu \in [\![Q_k]\!]_G$.

We have transformed the conjunctive part of each disjunct of φ_P into a first-order formula. Now we need to include in our transformation the existential quantification. This is achieved by means of SELECT.

Define the pattern P_k as SELECT $\bar{x}$ WHERE Q_k. We show that $P_k \equiv_{\text{RDF}} \exists \bar{y}_k \varphi_k(\bar{y}_k, \bar{x})$. Let G be an RDF graph.

- [$\Rightarrow$] Let $\mu \in [\![P_k]\!]_G$. By the semantics of SELECT, there must be a mapping $\mu' \in [\![Q_k]\!]_G$ such that $\mu'_{|\bar{x}} = \mu$. Since $Q_k \equiv_{\text{RDF}} \varphi_k$, this implies that $G^P_{\text{FO}} \models \varphi_k(t^{\bar{x} \cup \bar{y}_k}_{\mu'})$. Since the projection of $t^{\bar{x} \cup \bar{y}_k}_{\mu'}$ to $\bar{x}$ is precisely $t^{\bar{x}}_\mu$, we have that $G^P_{\text{FO}} \models \exists \bar{y}_k \varphi_k(\bar{y}_k, t^{\bar{x}}_\mu)$, concluding the first direction.

— [⇐] Let μ be a mapping such that $G^P_{\text{FO}} \models \exists \bar{y}_k \varphi_k(\bar{y}_k, t^{\bar{x}}_\mu)$. Then, there is a tuple a that extends $t^{\bar{x}}_\mu$ by assigning an IRI or the value N to each variable in $\bar{y}_k$. Hence, a corresponds to $t^{\bar{x}\cup\bar{y}_k}_{\mu'}$ for some mapping μ'. Since $G^P_{\text{FO}} \models \varphi_k(t^{\bar{x}\cup\bar{y}_k}_{\mu'})$ and $\varphi_k \equiv_{\text{RDF}} Q_k$, this means that $\mu' \in [\![P_k]\!]_G$. As the restriction of μ' to $\bar{x}$ is μ, we have that $\mu \in [\![\text{SELECT } \bar{x} \text{ WHERE } P_k]\!]_G$.

Having for each disjunct φ_P an equivalent graph pattern, we finally proceed to create a graph pattern that is equivalent to φ_P. This graph pattern is defined, as expected, as the disjunction between the previously constructed patterns. Let $P = P_1$ UNION $\cdots$ UNION P_j. It is immediate to prove that $P \equiv_{\text{RDF}} \varphi$: Let G be an RDF graph. A mapping μ belongs to $[\![P]\!]_G$ if an only if there is a $k \in \{1, \ldots, j\}$ such that $\mu \in [\![P_k]\!]_G$. We already proved this is the case if and only if there is a $k \in \{1, \ldots, j\}$ such that $G^P_{\text{FO}} \models \exists \bar{y}_k \varphi_k(\bar{y}_k, t^{\bar{x}}_\mu)$, concluding the proof. □

At this point, we have one transformation from SPARQL to FO and one transformation from FO to graph patterns in SPARQL[AUFS]. The composition of these two transformations does not preserve all the answers but only the maximal ones. Notice that this is expected, as it is well known that graph patterns in SPARQL[AUFS] are not only weakly monotone but also monotone. We are finally ready to prove Theorem 4.1, that is, to show that for every unrestricted weakly monotone graph pattern P, there exists a graph pattern Q in SPARQL[AUFS] such that $P \equiv^{\text{inf}}_s Q$.

Proof of Theorem 4.1. Let P be an unrestricted weakly monotone SPARQL graph pattern. Let φ_P be the existential first-order formula such that $P \equiv_s \varphi_P$ obtained from applying Lemma A.1 and Theorem A.4. Denote by ψ the UCQ with inequalities equivalent to φ_P constructed by applying Lemma A.7. Now let Q be the graph pattern such that $Q \equiv_{\text{RDF}} \psi$ obtained from applying Theorem A.8. We prove that Q is equivalent in maximal answers to P.

— [⇒] Let G be an unrestricted RDF graph and let μ be a mapping in $[\![P]\!]_G$. Then, we know that $G^P_{\text{FO}} \models \varphi(t^P_\mu)$, and hence there is a tuple $\bar{a}$ such that $t^P_\mu \preceq \bar{a}$ and $G^P_{\text{FO}} \models \psi(\bar{a})$. Let $\mu_{\bar{a}}$ be the mapping corresponding to $\bar{a}$. It is easy to see that $\mu \preceq \mu_{\bar{a}}$. Moreover, from the equivalence between ψ and Q, we obtain that $\mu_{\bar{a}} \in [\![Q]\!]_G$.

— [⇐] Let G be an RDF graph and let μ be a mapping in $[\![Q]\!]_G$. This implies that $G^P_{\text{FO}} \models \psi(t^P_\mu)$. Since ψ and φ are subsumption-equivalent, there is a tuple $\bar{a}$ such that $t^P_\mu \preceq \bar{a}$ and $G^P_{\text{FO}} \models \varphi(\bar{a})$. Let $\mu_{\bar{a}}$ be the mapping corresponding to $\bar{a}$. It is easy to see that $\mu \preceq \mu_{\bar{a}}$. Moreover, since φ and P are equivalent, we conclude that $\mu_{\bar{a}} \in [\![P]\!]_G$. □

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their careful reading of the paper and for providing many helpful comments.

REFERENCES

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Vol. 8. Addison-Wesley, Reading.

Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, and Sebastian Skritek. 2015. Towards reconciling SPARQL and certain answers. In *Proceedings of the 24th International Conference on World Wide Web*. 23–33.

Miklos Ajtai and Yuri Gurevich. 1987. Monotone versus positive. *J. ACM* 34, 4 (1987), 1004–1015.

Renzo Angles and Claudio Gutierrez. 2008. The expressive power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference*. 114–129.

Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. 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*. 629–638.

Marcelo Arenas and Jorge Pérez. 2011. Querying semantic web data with SPARQL. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 305–316.

Marcelo Arenas and Martín Ugarte. 2016. Designing a query language for RDF: Marrying open and closed worlds. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'16)*. 225–236.

Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. 2015. Efficient evaluation and approximation of well-designed pattern trees. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS'15)*. 131–144.

Michael Benedikt, Julien Leblay, Balder ten Cate, and Efthymia Tsamoura. 2016a. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Morgan & Claypool Publishers.

Michael Benedikt, Balder ten Cate, and Efthymia Tsamoura. 2014. Generating low-cost plans from proofs. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*. 200–211.

Michael Benedikt, Balder Ten Cate, and Efthymia Tsamoura. 2016b. Generating plans from proofs. *ACM Trans. Database Syst.* 40, 4, Article 22 (2016), 45 pages.

Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Sci. Am.* 284, 5 (2001), 28–37.

Carlos Buil-Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. 2013. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Sem.* 18, 1 (2013), 1–17.

Samuel R. Buss and Louise Hay. 1991. On truth-table reducibility to SAT. *Inf. Comput.* 91, 1 (1991), 86–102.

Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. 2012a. SPARQL Query containment under $\mathcal{SHI}$ axioms. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*.

Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. 2012b. SPARQL Query containment under RDFS entailment regime. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*. 134–148.

M. Dürst and M. Suignard. 2005. Rfc 3987, Internationalized Resource Identifiers (IRIs). Retrieved from http://www.ietf.org/rfc/rfc3987.txt.

Tim Furche, Benedikt Linse, François Bry, Dimitris Plexousakis, and Georg Gottlob. 2006. RDF querying: Language constructs and evaluation methods compared. In *Reasoning Web*. Springer, 1–52.

César A. Galindo-Legaria. 1994. Outerjoins as disjunctions. *SIGMOD Rec.* 23, 2 (May 1994), 348–358.

F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. 2013. Algebraic structures for capturing the provenance of SPARQL queries. In *Proceedings of the 16th International Conference on Database Theory*. 153–164.

Ramanathan V. Guha. 2013. Light at the end of the tunnel. In *Proceedings of the12th International Semantic Web Conference*.

Harry Halpin and James Cheney. 2014. Dynamic provenance for SPARQL updates. In *Proceedings of the 13th International Semantic Web Conference*. 425–440.

Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 query language. Retrieved from https://www.w3.org/TR/sparql11-query/.

Lane A. Hemachandra. 1989. The strong exponential hierarchy collapses. *J. Comput. System Sci.* 39, 3 (1989), 299–322.

Mark Kaminski and Egor V. Kostylev. 2016. Beyond well-designed SPARQL. In *Proceedings of the 19th International Conference on Database Theory*. 5:1–5:18.

Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. 2015. CONSTRUCT queries in SPARQL. In *Proceedings of the 18th International Conference on Database Theory*. 212–229.

Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2012. Static analysis and optimization of semantic web queries. In *Proceedings of the 31st Symposium on Principles of Database Systems*. 89–100.

Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2013. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.* 38, 4 (2013), 25.

Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

Leonid Libkin, Juan Reutter, and Domagoj Vrgoč. 2013. TriAL for RDF: Adapting graph query languages for RDF data. In *Proceedings of the 32nd symposium on Principles of database systems*. ACM, 201–212.

Katja Losemann and Wim Martens. 2012. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st Symposium on Principles of Database Systems*. ACM, 101–112.

Roger C. Lyndon. 1959. An interpolation theorem in the predicate calculus. *Pacific J. Math.* 9, 1 (1959), 129–142.

Frank Manola and Eric Miller. 2004. RDF Primer. *W3C Recommendation*. Retrieved from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

Alan Nash, Luc Segoufin, and Victor Vianu. 2010. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.* 35, 3, Article 21 (July 2010), 41 pages.

Martin Otto. 2000. An interpolation theorem. *Bull. Symbol. Logic* 6, 4 (2000), 447–462.

C. H. Papadimitriou and M. Yannakakis. 1982. The complexity of facets (and some facets of complexity). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*. 255–260.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2006. Semantics and complexity of SPARQL. In *Proceedings of the 5th International Semantic Web Conference*. 30–43.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009).

François Picalausa and Stijn Vansummeren. 2011. What are real SPARQL queries like? In *Proceedings of the International Workshop on Semantic Web Information Management*.

Reinhard Pichler and Sebastian Skritek. 2014. Containment and equivalence of well-designed SPARQL. In *Proceedings of the 33rd ACM Symposium on Principles of Database Systems*. 39–50.

Axel Polleres and Johannes Peter Wallner. 2013. On the relation between SPARQL1.1 and answer set programming. *J. Appl. Non-Class. Logics* 23, 1–2 (2013), 159–212.

Eric Prud'hommeaux and Andy Seaborne. 2008. SPARQL Query Language for RDF. *W3C Recommendation*. Retrieved from http://www.w3.org/TR/rdf-sparql-query/.

Juan L. Reutter, Adrián Soto, and Domagoj Vrgoč. 2015. Recursion in SPARQL. In *Proceedings of the 14th International Conference on the Semantic Web*. 19–35.

Tobias Riege and Jörg Rothe. 2006. Completeness in the boolean hierarchy: Exact-four-colorability, minimal graph uncolorability, and exact domatic number problems—A survey. *The Journal of Universal Computer Science* 12, 5 (May 2006), 551–578.

Sebastian Rudolph and Markus Krötzsch. 2013. Flag & check: Data access with monadically defined queries. In *Proceedings of the 32nd ACM Symposium on Principles of Database Systems*. 151–162.

Michael Schmidt, Michael Meier, and Georg Lausen. 2010. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*. 4–33.

Luc Segoufin and Victor Vianu. 2005. Views and queries: Determinacy and rewriting. In *Proceedings of the Twenty-fourth ACM Symposium on Principles of Database Systems*. 49–60.

Juan Sequeda, Marcelo Arenas, and Daniel P. Miranker. 2012. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st World Wide Web Conference 2012*. 649–658.

Holger Spakowski. 2005. *Completeness for Parallel Access to NP and Counting Class Separations*. Ph.D. Thesis.

Larry J. Stockmeyer. 1976. The polynomial-time hierarchy. *Theor. Comput. Sci.* 3, 1 (1976), 1–22.

David Toman and Grant E. Weddell. 2011. *Fundamentals of Physical Design and Query Compilation*. Morgan & Claypool Publishers.

Boris Trakhtenbrot. 1950. The impossibility of an algorithm for the decidability problem on finite classes. *Proc. USSR Acad. Sci.* 70, 4 (1950), 569–572. [in Russian]

Moshe Y. Vardi. 1982. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*. ACM, 137–146.

K. W. Wagner. 1987. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.* 51, 1–2 (March 1987), 53–80.

Gerd Wechsung. 1985. On the boolean closure of NP. In *Fundamentals of Computation Theory*. Springer, 485–493.

Received December 2016; revised May 2017; accepted July 2017