Semantics and Complexity of SPARQL

JORGE PÉREZ and MARCELO ARENAS
Pontificia Universidad Católica de Chile
and
CLAUDIO GUTIERREZ
Universidad de Chile

SPARQL is the standard language for querying RDF data. In this article, we address systematically the formal study of the database aspects of SPARQL, concentrating in its graph pattern matching facility. We provide a compositional semantics for the core part of SPARQL, and study the complexity of the evaluation of several fragments of the language. Among other complexity results, we show that the evaluation of general SPARQL patterns is PSPACE-complete. We identify a large class of SPARQL patterns, defined by imposing a simple and natural syntactic restriction, where the query evaluation problem can be solved more efficiently. This restriction gives rise to the class of well-designed patterns. We show that the evaluation problem is coNP-complete for well-designed patterns. Moreover, we provide several rewriting rules for well-designed patterns whose application may have a considerable impact in the cost of evaluating SPARQL queries.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms: Algorithms, Languages, Performance, Theory

Additional Key Words and Phrases: Complexity, query language, RDF, semantic Web, SPARQL

ACM Reference Format:

Pérez, J., Arenas, M., and Gutierrez, C. 2009. Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34, 3, Article 16 (August 2009), 45 pages.
DOI = 10.1145/1567274.1567278 http://doi.acm.org/10.1145/1567274.1567278.

1. INTRODUCTION

The Resource Description Framework (RDF) [Manola and Miller 2004] is a data model for representing information about World Wide Web resources. Jointly

This article is an extended and revised version of Pérez et al. [2006a].
M. Arenas was supported by Fondecyt grants 1070732 and 1090565; C. Gutierrez was supported by Fondecyt grant 1070348; J. Pérez was supported by Conicyt Ph.D. Scholarship.

Authors' addresses: J. Pérez, M. Arenas, Department of Computer Science, Pontificia Universidad Católica de Chile, Chile; email: {jperez,marenas}@ing.puc.cl; C. Gutierrez, Department of Computer Science, Universidad de Chile, Chile; email: cgutierr@dcc.uchile.cl.

DOI 10.1145/1567274.1567278 http://doi.acm.org/10.1145/1567274.1567278

with its release in 1998 as Recommendation of the World Wide Web Consortium (W3C), the natural problem of querying RDF data was raised. Since then, several designs and implementations of RDF query languages have been proposed (see Haase et al. [2004] and Furche et al. [2006] for detailed comparisons of RDF query languages). In 2004, the RDF Data Access Working Group, part of the W3C Semantic Web Activity, released a first public working draft of a query language for RDF, called SPARQL [Prud'hommeaux and Seaborne 2008].[1] Since then, SPARQL has been rapidly adopted as the standard for querying semantic Web data. In January 2008, SPARQL became a W3C Recommendation.

RDF is a directed labeled graph data format and, thus, SPARQL is essentially a graph-matching query language. SPARQL queries are composed by three parts. The *pattern matching part*, includes several interesting features of pattern matching of graphs, like optional parts, union of patterns, nesting, filtering values of possible matchings, and the possibility of choosing the data source to be matched by a pattern. The *solution modifiers*, once the output of the pattern has been computed (in the form of a table of values of variables), allow to modify these values applying classical operators like projection, distinct, order, and limit. Finally, the *output* of a SPARQL query can be of different types: yes/no queries, selections of values of the variables which match the patterns, construction of new RDF data from these values, and descriptions of resources.

The definition of a formal semantics for SPARQL has played a key role in the standardization process of this query language. Although taken one by one the features of SPARQL are intuitive and simple to describe and understand, it turns out that the combination of them makes SPARQL into a complex language. Reaching a consensus in the W3C standardization process about a formal semantics for SPARQL was not an easy task. The initial efforts to define SPARQL were driven by use cases, mostly by specifying the expected output for particular example queries. In fact, the interpretations of examples and the exact outcomes of cases not covered in the initial drafts of the SPARQL specification were a matter of long discussions in the W3C mailing lists. In the conference version of this article (see Pérez et al. [2006a]), we presented one of the first formalizations of a semantics for a fragment of the language. Currently, the official specification of SPARQL [Prud'hommeaux and Seaborne 2008], endorsed by the W3C, formalizes a semantics based on our work [Pérez et al. 2006a, 2006b; Arenas et al. 2007].

A formalization of a semantics for SPARQL is beneficial for several reasons, including to serve as a tool to identify and derive relations among the constructors that stay hidden in the use cases, identify redundant and contradicting notions, to drive and help the implementation of query engines, and to study the complexity, expressiveness, and further natural database questions like rewriting and optimization. The broad goal of our work is the formalization and study of the database aspects of SPARQL.

In this article, we present a thorough study of the pattern-matching facility of SPARQL, which constitutes the core of the language. In this direction,

[1] The name SPARQL is a recursive acronym that stands for *SPARQL Protocol and RDF Query Language*.

we address three main areas. We first provide a streamlined version of the core fragment of SPARQL with precise algebraic syntax and a formal compositional semantics. One of the delicate issues in the definition of a semantics for SPARQL is the treatment of *optional matching* and incomplete answers. The idea behind optional matching is to allow information to be added if the information is available in the data source, instead of just failing to give an answer whenever some part of the pattern does not match. This feature of optional matching is crucial in semantic Web applications, and more specifically in RDF data management, where it is assumed that every application has only partial knowledge about the resources being managed. We formalize the semantics of SPARQL by using *partial mappings* between variables in the patterns and actual values in the data source. This formalization allows us to deal with partial answers in a clean way. Our formalization is based on the extension of some classical relational algebra operators to work over sets of partial mappings.

A fundamental issue in every query language is the complexity of query evaluation and, in particular, what is the influence of each component of the language in this complexity. As a second contribution, we present a thorough study of the complexity of the evaluation of SPARQL graph patterns. In this study, we consider several fragments of SPARQL built incrementally, and present complexity results for each such fragment. Among other results, we show that the complexity of the evaluation problem for general SPARQL graph patterns is PSPACE-complete, and that this high complexity is obtained as a consequence of unlimited use of nested optional parts.

Given the high complexity of the evaluation problem for general SPARQL graph patterns, an important question is whether one can find interesting classes of patterns where the query evaluation problem can be solved more efficiently. Our third contribution is the identification of a large class of patterns with the previous characteristic, and defined by a simple and natural syntactic restriction. This class of patterns is obtained by forbidding a special form of interaction between variables appearing in optional parts. We call *well-designed patterns* the patterns satisfying this condition. We show that some counterintuitive outcomes may be obtained when evaluating nonwell-designed patterns. For instance, we have mentioned that the intuition behind the optional matching is to allow information to be added if the information is available, but not to reject if the information is not present. We formalize this intuition and show that it does hold for well-designed patterns but fails in the nonwell-designed case.

Well-designed patterns form a natural fragment of SPARQL that is very common in practice. Furthermore, well-designed patterns have several interesting features. We show that the complexity of the query evaluation problem for well-designed patterns is considerably lower, namely coNP-complete. We also prove that the property of being well designed has important consequences for the optimization of SPARQL queries. We present several rewriting rules for well-designed patterns whose application may have a considerable impact in the cost of evaluating SPARQL queries, and prove the existence of a normal form for well-designed patterns based on the application of these rewriting rules.

Organization of the Article. Section 2 presents a formalized algebraic syntax and a compositional semantics for SPARQL. Section 3 presents the complexity study of the language. Section 4 introduces the fragment of well-designed patterns and presents its properties. Finally, Section 5 discusses related work and Section 6 gives some concluding remarks. For the sake of readability, some proofs and technical results are included in the Appendix.

2. SYNTAX AND SEMANTICS OF SPARQL

In this section, we give an algebraic formalization of the core fragment of SPARQL over simple RDF, that is, RDF without RDFS vocabulary and literal rules. This allows us to take a close look at the core components of the language and identify some of its fundamental properties.

We introduce first the necessary notions about RDF (for details on RDF formalization see Gutierrez et al. [2004], or Marin [2004] for a complete reference including RDFS vocabulary). Assume there are pairwise disjoint infinite sets I, B, and L (IRIs [Durst and Suignard 2005], Blank nodes, and Literals, respectively). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. In this tuple, s is the *subject*, p the *predicate*, and o the *object*. Assume additionally the existence of an infinite set V of variables disjoint from the previous sets.

Definition 2.1. An *RDF graph* [Klyne et al. 2004] is a set of RDF triples. In our context, we refer to an RDF graph as an *RDF dataset*, or simply a *dataset*.

SPARQL is essentially a graph-matching query language. A SPARQL query is of the form $H \leftarrow B$, where B, the *body* of the query, is a complex RDF graph pattern expression that may include RDF triples with variables, conjunctions, disjunctions, optional parts, and constraints over the values of the variables, and H, the *head* of the query, is an expression that indicates how to construct the answer to the query. The evaluation of a query Q against a dataset D is done in two steps: The body of Q is matched against D to obtain a set of bindings for the variables in the body, and then using the information on the head of Q, these bindings are processed applying classical relational operators (projection, distinct, etc.) to produce the answer to the query, which can have different forms, such as a yes/no answer, a table of values, or a new RDF dataset. In this article, we concentrate on the body of SPARQL queries, that is, in the graph pattern-matching facility.

2.1 Syntax of SPARQL Graph Pattern Expressions

The official syntax of SPARQL [Prud'hommeaux and Seaborne 2008] considers operators `OPTIONAL`, `UNION`, and `FILTER`, and *concatenation* via a point symbol (`.`), to construct graph pattern expressions. The syntax also considers `{ }` to group patterns, and some implicit rules of precedence and association. For example, the point symbol (`.`) has precedence over `OPTIONAL`, and `OPTIONAL` is left associative. In order to avoid ambiguities in the parsing, we present the syntax of SPARQL graph patterns in a more traditional algebraic formalism, using binary operators AND (`.`), UNION (`UNION`), OPT (`OPTIONAL`), and FILTER (`FILTER`). We fully parenthesize expressions making explicit the precedence and association of operators. A SPARQL graph pattern expression is defined recursively as follows.

(1) A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (a *triple pattern*).
(2) If P_1 and P_2 are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns (*conjunction graph pattern*, *optional graph pattern*, and *union graph pattern*, respectively).
(3) If P is a graph pattern and R is a SPARQL *built-in* condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern (a *filter graph pattern*).

A SPARQL *built-in* condition is constructed using elements of the set $I \cup L \cup V$ and constants, logical connectives ($\neg$, $\wedge$, $\vee$), inequality symbols ($<$, $\leq$, $\geq$, $>$), the equality symbol ($=$), unary predicates like bound, isBlank, and isIRI, plus other features (see Prud'hommeaux and Seaborne [2008] for a complete list). In this article, we restrict to the fragment where the built-in condition is a Boolean combination of terms constructed by using $=$ and bound, that is:

(1) If $?X, ?Y \in V$ and $c \in I \cup L$, then bound($?X$), $?X = c$ and $?X = ?Y$ are built-in conditions.
(2) 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.

Let P be a SPARQL graph pattern. In the rest of the article, we use var(P) to denote the set of variables occurring in P. In particular, if t is a triple pattern, then var(t) denotes the set of variables occurring in the components of t. Similarly, for a built-in condition R, we use var(R) to denote the set of variables occurring in R.

2.2 Semantics of SPARQL Graph Pattern Expressions

To define the semantics of SPARQL graph pattern expressions, we need to introduce some terminology. A *mapping* μ from V to U is a partial function $\mu : V \rightarrow U$. Abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. The domain of μ, denoted by dom(μ), is the subset of V where μ is defined. Two mappings μ_1 and μ_2 are *compatible* when for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$, that is, when $\mu_1 \cup \mu_2$ is also a mapping. Intuitively, μ_1 and μ_2 are compatibles if μ_1 *can be extended* with μ_2 to obtain a new mapping, and vice versa. Note that two mappings with disjoint domains are always compatible, and that the empty mapping $\mu_\emptyset$ (i.e., the mapping with empty domain) is compatible with any other mapping.

Let Ω_1 and Ω_2 be sets of mappings. We define the join of, the union of, and the difference between Ω_1 and Ω_2 as

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\},\\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\},\\
\Omega_1 \smallsetminus \Omega_2 &= \{\mu \in \Omega_1 \mid \text{ for all } \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible}\}.
\end{aligned}$$

Based on the previous operators, we define the left outer-join as

$$\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2).$$

Intuitively, $\Omega_1 \bowtie \Omega_2$ is the set of mappings that result from extending mappings in Ω_1 with their compatible mappings in Ω_2, and $\Omega_1 \smallsetminus \Omega_2$ is the set of mappings in Ω_1 that cannot be extended with any mapping in Ω_2. The operation $\Omega_1 \cup \Omega_2$ is the usual set-theoretical union. A mapping μ is in $\Omega_1 \mathbin{⟕} \Omega_2$ if it is the extension of a mapping of Ω_1 with a compatible mapping of Ω_2, or if it belongs to Ω_1 and cannot be extended with any mapping of Ω_2. These operations resemble relational algebra operations over sets of mappings (partial functions).

We are ready to define the semantics of graph pattern expressions as a function $[\![\cdot]\!]_D$ which takes a pattern expression and returns a set of mappings. We follow the approach in Gutierrez et al. [2004] defining the semantics as the set of mappings that matches the dataset D. For the sake of readability, the semantics of filter expressions is presented in a separate definition.

Definition 2.2. The *evaluation* of a graph pattern P over an RDF dataset D, denoted by $[\![P]\!]_D$, is defined recursively as follows.

(1) If P is a triple pattern t, then $[\![P]\!]_D = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in D\}$.
(2) If P is $(P_1 \text{ AND } P_2)$, then $[\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$.
(3) If P is $(P_1 \text{ OPT } P_2)$, then $[\![P]\!]_D = [\![P_1]\!]_D \mathbin{⟕} [\![P_2]\!]_D$.
(4) If P is $(P_1 \text{ UNION } P_2)$, then $[\![P]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$.

The idea behind the OPT operator is to allow for *optional matching* of patterns. Consider pattern expression $(P_1 \text{ OPT } P_2)$ and let μ_1 be a mapping in $[\![P_1]\!]_D$. If there exists a mapping $\mu_2 \in [\![P_2]\!]_D$ such that μ_1 and μ_2 are compatible, then $\mu_1 \cup \mu_2$ belongs to $[\![(P_1 \text{ OPT } P_2)]\!]_D$. But if no such a mapping μ_2 exists, then μ_1 belongs to $[\![(P_1 \text{ OPT } P_2)]\!]_D$. Thus, operator OPT allows information to be added to a mapping μ if the information is available, instead of just rejecting μ whenever some part of the pattern does not match. This feature of *optional matching* is crucial in semantic Web applications, and more specifically in RDF data management, where it is assumed that every application has only partial knowledge about the resources being managed.

The semantics of filter expressions goes as follows. Given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if:

(1) R is bound$(?X)$ and $?X \in \text{dom}(\mu)$;
(2) R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
(3) R is $?X = ?Y$, $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
(4) R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\mu \models R_1$;
(5) R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions, and $\mu \models R_1$ or $\mu \models R_2$;
(6) R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\mu \models R_1$ and $\mu \models R_2$.

Definition 2.3. Given an RDF dataset D and a filter expression $(P \text{ FILTER } R)$,

$$[\![(P \text{ FILTER } R)]\!]_D = \{\mu \in [\![P]\!]_D \mid \mu \models R\}.$$

In the rest of the article, we usually represent sets of mappings as tables where each row represents a mapping in the set. We label every row with the name of a mapping, and every column with the name of a variable. If a mapping is not defined for some variable, then we simply leave empty the corresponding position. For instance, the table

	$?X$	$?Y$	$?Z$	$?V$	$?W$
μ_1 :	a	b			
μ_2 :		c			d
μ_3 :			e		

represents the set $\Omega = \{\mu_1, \mu_2, \mu_3\}$ where

—dom(μ_1) = $\{?X, ?Y\}$, $\mu_1(?X) = a$, and $\mu_1(?Y) = b$,
—dom(μ_2) = $\{?Y, ?W\}$, $\mu_2(?Y) = c$, and $\mu_2(?W) = d$,
—dom(μ_3) = $\{?Z\}$, and $\mu_3(?Z) = e$.

Sometimes we write $\{\{?X \to a, ?Y \to b\}, \{?Y \to c, ?W \to d\}, \{?Z \to e\}\}$ for the preceding set of mappings.

Example 2.4. Consider an RDF dataset D storing information about professors in a university.

$$
D = \{ \quad \begin{array}{llllll}
(B_1, & \text{name}, & \text{paul}), & (B_1, & \text{phone}, & \text{777-3426}), \\
(B_2, & \text{name}, & \text{john}), & (B_2, & \text{email}, & \text{john@acd.edu}), \\
(B_3, & \text{name}, & \text{george}), & (B_3, & \text{webPage}, & \text{www.george.edu}), \\
(B_4, & \text{name}, & \text{ringo}), & (B_4, & \text{email}, & \text{ringo@acd.edu}), \\
(B_4, & \text{webPage}, & \text{www.starr.edu}), & (B_4, & \text{phone}, & \text{888-4537})
\end{array} \quad \}
$$

The following are graph pattern expressions and their evaluations over D according to the previous semantics:

—P_1 = ((?A, email, ?E) OPT (?A, webPage, ?W)). Then

$[\![P_1]\!]_D =$

	$?A$	$?E$	$?W$
μ_1 :	B_2	john@acd.edu	
μ_2 :	B_4	ringo@acd.edu	www.starr.edu

—P_2 = (((?A, name, ?N) OPT (?A, email, ?E)) OPT (?A, webPage, ?W)). Then

$[\![P_2]\!]_D =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		www.george.edu
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

—P_3 = ((?A, name, ?N) OPT ((?A, email, ?E) OPT (?A, webPage, ?W))). Then

$[\![P_3]\!]_D =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

Notice the difference between $[\![P_2]\!]_D$ and $[\![P_3]\!]_D$. These two examples show that $[\![((A \text{ OPT } B) \text{ OPT } C)]\!]_D \neq [\![(A \text{ OPT } (B \text{ OPT } C))]\!]_D$ in general.

- $P_4 = ((?A, \text{name}, ?N) \text{ AND } ((?A, \text{email}, ?E) \text{ UNION } (?A, \text{webPage}, ?W)))$. Then

$[\![P_4]\!]_D =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_2	john	john@acd.edu	
μ_2 :	B_3	george		www.george.edu
μ_3 :	B_4	ringo	ringo@acd.edu	
μ_4 :	B_4	ringo		www.starr.edu

- $P_5 = (((?A, \text{name}, ?N) \text{ OPT } (?A, \text{phone}, ?P)) \text{ FILTER } ?N = \text{paul})$. Then

$[\![P_5]\!]_D =$

	$?A$	$?N$	$?P$
μ_1 :	B_1	paul	777-3426

2.3 Simple Algebraic Properties

We say that two graph patterns P_1 and P_2 are *equivalent*, denoted by $P_1 \equiv P_2$, if $[\![P_1]\!]_D = [\![P_2]\!]_D$ for every RDF dataset D. The following lemma states some simple algebraic properties of AND and UNION operators. These properties are direct consequence of the semantics of AND and UNION, both based on set-theoretical union.

LEMMA 2.5. *The operators* AND *and* UNION *are associative and commutative and the operator* AND *distributes over* UNION. *That is, if* P_1, P_2, *and* P_3 *are graph patterns, then it holds that:*

- $(P_1 \text{ AND } P_2) \equiv (P_2 \text{ AND } P_1)$*;*
- $(P_1 \text{ UNION } P_2) \equiv (P_2 \text{ UNION } P_1)$*;*
- $(P_1 \text{ AND } (P_2 \text{ AND } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ AND } P_3)$*;*
- $(P_1 \text{ UNION } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ UNION } P_2) \text{ UNION } P_3)$*;*
- $(P_1 \text{ AND } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3))$*.*

The previous lemma permits us to avoid parentheses when writing sequences of either AND operators or UNION operators. This is consistent with the definitions of *Group Graph Pattern* and *Union Graph Pattern* in Prud'hommeaux and Seaborne [2008]. We use Lemma 2.5 to simplify the notation in the following sections.

2.4 On the Semantics of SPARQL by the W3C

The definition of a formal semantics for SPARQL has played a key role in the standardization process of the language. In the conference version of this article (see Pérez et al. [2006a]), we presented one of the first formalizations of a semantics for a fragment of the language (we have included in Section 5 a discussion about other approaches to formalize the semantics of SPARQL). Currently, the official specification of SPARQL [Prud'hommeaux and Seaborne 2008], that is a W3C Recommendation since January 2008, formalizes a semantics based on our work [Pérez et al. 2006a, 2006b; Arenas et al. 2007]. Nevertheless, the semantics that we present in this article has some differences with the standard semantics. We review them in this section.

Set semantics versus bag semantics. The official specification of SPARQL uses a *bag (multiset) semantics* for the evaluation of graph patterns. Specifically, the evaluation of a graph pattern is a set of mappings together with a *cardinality function* that states how many duplicates of a mapping occurs in the evaluation. In the technical report [Pérez et al. 2006b], we formalize a semantics for SPARQL by considering bags of mappings. That work was the starting point of the bag semantics for SPARQL currently adopted by the W3C. In this article, we have decided to follow the same path as in the study of relational query languages, and we have focused on the set-based formalization of SPARQL. Although in practice SQL has a bag semantics, the study of relational algebra with set semantics has shown to be of fundamental importance in the development and implementation of SQL. We expect that the same will happen in the context of SPARQL.

It should be pointed out that the complexity results that we present in the following sections rely on the assumption that the evaluation of a SPARQL pattern is a set of mappings. We left for future work the extension of these results to the bag-semantics case.

FILTER *evaluation.* One of the main differences between the formalization presented in this article and the official specification is the evaluation of built-in conditions. The standard SPARQL language uses three values to evaluate a condition: *true*, *false*, and *error*. In this article, we have adopted a standard two-valued semantics for a query language. An example in which this issue makes a difference is the evaluation of a condition with unbounded variables. For instance, if μ is a mapping and $?X$ is a variable such that $?X \notin \text{dom}(\mu)$, then according to our definition $\mu \not\models (?X = a)$ and, thus, $\mu \models \neg(?X = a)$. Therefore, we assign value *true* to the expression $\neg(?X = a)$. On the other hand, the SPARQL standard specification assigns value *error* to both $(?X = a)$ and $\neg(?X = a)$. Thus, if D is an RDF graph and P a pattern such that $\mu \in [\![P]\!]_D$, then according to our definition $\mu \in [\![(P \text{ FILTER } \neg(?X = a))]\!]_D$, while according to the official specification, the mapping μ is not in the evaluation of pattern $(P \text{ FILTER } \neg(?X = a))$ over D.

Although this difference between the two semantics could be considered important, it does not play a substantial role in any of the results presented in this article. In fact, we have not used this difference to obtain any of the complexity and optimization results presented in the following sections.

Evaluation of FILTER *in* OPT *expressions.* The standard semantics of SPARQL has mostly adopted a compositional form of evaluation in the spirit of our work. However, there is still *a single case* in which a SPARQL expression must be evaluated in a noncompositional way according to the standard semantics. This happens when the righthand side of an OPT operator is a FILTER expression, that is, for expressions of the form $(P_1 \text{ OPT } (P_2 \text{ FILTER } R))$. As an example, consider the following pattern.

$$(?X, a, ?Y) \text{ OPT } ((?X, b, ?Z) \text{ FILTER } ?Y = ?Z) \qquad (1)$$

Notice that the variable $?Y$ mentioned in the condition $?Y = ?Z$ does not occur in the lefthand side of the FILTER expression. Under our compositional semantics, the preceding expression is evaluated unambiguously by following the parentheses, that is, expression $((?X, b, ?Z) \text{ FILTER } ?Y = ?Z)$ is evaluated first.

To evaluate expression (1) according to the official SPARQL semantics, one has to follow a different process, since in this case the FILTER condition must be used as a *left-outer join condition* when evaluating OPT [Prud'hommeaux and Seaborne 2008]. In other words, the evaluation of (1) over a graph D is given by

$$[\![(?X, a, ?Y)]\!]_D \;⟕_{(?Y=?Z)}\; [\![(?X, b, ?Z)]\!]_D.$$

Although this mismatch could be considered a significant difference, it was recently shown in Angles and Gutierrez [2008] that in terms of expressiveness, the compositional semantics of SPARQL that we use in this article is equivalent to the noncompositional semantics proposed by the W3C. It should be pointed out that we do not use this difference to obtain any of the complexity results shown in the following section. In particular, we focus in Section 4 on the class of well-designed patterns that only consider *safe* FILTER expressions, that is, FILTER expressions of the form $(P \text{ FILTER } R)$ where all the variables that occur in R also occur in P, excluding expressions like (1).

3. COMPLEXITY OF EVALUATING GRAPH PATTERN EXPRESSIONS

A fundamental issue in every query language is the complexity of query evaluation and, in particular, what is the influence of each component of the language in this complexity. In this section, we address these issues for graph pattern expressions.

As is customary when studying the complexity of the evaluation problem for a query language [Vardi 1982], we consider its associated decision problem. We denote this problem by Evaluation and we define it as follows.

INPUT	:	An RDF dataset D, a graph pattern P and a mapping μ.
QUESTION	:	Is $\mu \in [\![P]\!]_D$?

It is important to notice that the evaluation problem that we study considers the mapping as part of the input. In other words, we study the complexity by measuring how difficult it is to verify whether a given mapping is a solution for a pattern evaluated over an RDF dataset. This is the standard *decision* problem considered when studying the complexity of a query language [Vardi 1982], as opposed to the *computation* problem of actually listing the set of solutions (finding all the mappings). To focus on the decision problem allows us to obtain a fine-grained analysis of the complexity of the evaluation problem, classifying the complexity for different fragments of SPARQL in terms of standard complexity classes. Also notice that the pattern and the dataset are both inputs for Evaluation. Thus, we study the *combined complexity* of the query language [Vardi 1982].

In this section, we present a thorough analysis of the complexity of the evaluation problem for SPARQL graph patterns. In particular, we show that the unlimited use of the optional operator leads to high complexity, namely PSPACE-completeness. Let us give some intuition about the reasons for this high complexity. Given an RDF dataset D, a mapping μ is in $[\![(P_1 \text{ OPT } P_2)]\!]_D$ if either μ is $[\![(P_1 \text{ AND } P_2)]\!]_D$, or μ is in $[\![P_1]\!]_D$ and for every mapping μ' in $[\![P_2]\!]_D$, μ is not compatible with μ'. Thus, the evaluation of $(P_1 \text{ OPT } P_2)$ implicitly refers to a *universal quantification* (*for every mapping* μ') and a *negation* (μ' *is not compatible with* μ). In particular, this shows that evaluating a nested optional pattern $(P_1 \text{ OPT } (P_2 \text{ OPT } P_3))$ implies a quantifier *alternation*. It is well known that the alternation of quantifiers in some problems related to propositional logic and first-order logic leads to PSPACE-completeness, and this fact is used in the proof of Theorem 3.3 to show the PSPACE-completeness of the evaluation problem for SPARQL patterns including the OPT operator. In this section, we also consider some fragments of SPARQL that do not use the OPT operator, and show that the evaluation problem can be solved more efficiently for these fragments.

We start this study by considering the fragment consisting of graph pattern expressions constructed by using only AND and FILTER operators. This simple fragment is interesting as it does not use the two most complicated operators in SPARQL, namely UNION and OPT. Given an RDF dataset D, a graph pattern P in this fragment, and a mapping μ, it is possible to efficiently check whether $\mu \in [\![P]\!]_D$ by using the following algorithm. First, for each triple t in P, verify whether $\mu(t) \in D$. If this is not the case, then return *false*. Otherwise, by using a bottom-up approach, verify whether the expression generated by instantiating the variables in P according to μ satisfies the FILTER conditions in P. If this is the case, then return *true*, else return *false*. Thus, we conclude the next theorem.

Theorem 3.1. Evaluation *can be solved in time* $O(|P| \cdot |D|)$ *for graph pattern expressions constructed by using only* AND *and* FILTER *operators.*

We continue this study by adding the UNION operator to the AND-FILTER fragment. It is important to notice that the inclusion of UNION in SPARQL is one of the most controversial issues in the definition of this language. In the following theorem, we show that the inclusion of UNION operator makes the evaluation problem for SPARQL considerably harder.

Theorem 3.2. Evaluation *is NP-complete for graph pattern expressions constructed by using only* AND*,* FILTER*, and* UNION *operators.*

Proof. It is straightforward to prove that Evaluation is in NP for the case of graph pattern expressions constructed by using only AND, UNION, and FILTER operators. To prove the NP-hardness of Evaluation for this case, we show how to reduce in polynomial time the satisfiability problem for propositional formulas in CNF (SAT-CNF) to our problem. An instance of SAT-CNF is a propositional formula φ of the form $C_1 \wedge \cdots \wedge C_n$, where each C_i ($i \in [1, n]$) is a clause, that is, a disjunction of propositional variables and negations of propositional variables. Then the problem is to verify whether there exists a truth assignment satisfying φ. It is well known that SAT-CNF is NP-complete [Garey and Johnson 1979].

In the reduction from SAT-CNF, we use a fixed RDF dataset $D = \{(\mathtt{a}, \mathtt{b}, \mathtt{c})\}$. Assume that $x_1, \ldots, x_m$ is the list of propositional variables mentioned in φ. For each x_i ($i \in [1, m]$), we use SPARQL variables $?X_i$, $?Y_i$ to represent x_i and $\neg x_i$, respectively. Then for each clause C in φ of the form

$$x_{i_1} \vee \cdots \vee x_{i_k} \vee \neg x_{j_1} \vee \cdots \vee \neg x_{j_\ell},$$

we define a graph pattern P_C as

$$\begin{aligned}&(\,(\mathtt{a}, \mathtt{b}, ?X_{i_1})\ \text{UNION}\ \cdots\ \text{UNION}\ (\mathtt{a}, \mathtt{b}, ?X_{i_k})\ \text{UNION}\\ &\qquad\qquad\qquad (\mathtt{a}, \mathtt{b}, ?Y_{j_1})\ \text{UNION}\ \cdots\ \text{UNION}\ (\mathtt{a}, \mathtt{b}, ?Y_{j_\ell})\,),\end{aligned}$$

and we define a graph pattern P_φ for φ as

$$(P\ \text{AND}\ ((P_{C_1}\ \text{AND}\ \cdots\ \text{AND}\ P_{C_n})\ \text{FILTER}\ R)),$$

where

$$\begin{aligned}P &= ((\mathtt{a}, \mathtt{b}, ?X_1)\ \text{AND}\ (\mathtt{a}, \mathtt{b}, ?Y_1)\ \text{AND}\ \cdots\ \text{AND}\ (\mathtt{a}, \mathtt{b}, ?X_m)\ \text{AND}\ (\mathtt{a}, \mathtt{b}, ?Y_m)),\\ R &= ((\neg\,\text{bound}(?X_1) \vee \neg\,\text{bound}(?Y_1)) \wedge \cdots \wedge (\neg\,\text{bound}(?X_m) \vee \neg\,\text{bound}(?Y_m))).\end{aligned}$$

Let $\mu = \{?X_1 \to \mathtt{c}, \ldots, ?X_m \to \mathtt{c}, ?Y_1 \to \mathtt{c}, \ldots, ?Y_m \to \mathtt{c}\}$. We now prove that φ is satisfiable if and only if $\mu \in [\![P_\varphi]\!]_D$.

Assume first that $\mu \in [\![P_\varphi]\!]_D$, then we need to show that φ is satisfiable. Let Q_φ be the pattern $((P_{C_1}\ \text{AND}\ \cdots\ \text{AND}\ P_{C_n})\ \text{FILTER}\ R)$. Since $P_\varphi = (P\ \text{AND}\ Q_\varphi)$ and $\mu \in [\![P]\!]_D$, there exists a mapping ν that is compatible with μ and such that $\nu \in [\![Q_\varphi]\!]_D$. Consider the truth assignment σ defined as

$$\sigma(x_i) = \begin{cases} 1 & \text{if } ?X_i \in \text{dom}(\nu)\\ 0 & \text{if } ?X_i \notin \text{dom}(\nu)\end{cases}$$

for every variable x_i in φ ($i \in [1, m]$). We claim that σ is a truth assignment that satisfies φ. By the construction of Q_φ and since $\nu \in [\![Q_\varphi]\!]_D$, we know that for each clause C in φ there exists a variable $?V$ in P_C such that $?V \in \text{dom}(\nu)$. Now, if $?V$ is equal to $?X_i$ for some $i \in [1, m]$, then we have that x_i is a literal in C and that $\sigma(x_i) = 1$ and, thus, σ satisfies clause C. On the other hand, if $?V$ is equal to $?Y_i$ for some $i \in [1, m]$, then we know that $\neg x_i$ is a literal in C. In this case, since $(\neg\,\text{bound}(?X_i) \vee \neg\,\text{bound}(?Y_i))$ is a conjunction in the FILTER condition of Q_φ, we obtain that $?X_i \notin \text{dom}(\nu)$ and thus $\sigma(x_i) = 0$. Hence, since $\neg x_i$ is a literal in the clause C and $\sigma(x_i) = 0$, we have that σ satisfies C. We have shown that for every clause C in φ, it holds that σ satisfies C and, thus, σ satisfies φ. This shows that φ is satisfiable.

To prove the reverse implication, assume that φ is satisfiable. We need to show that $\mu \in [\![P_\varphi]\!]_D$. Let σ be a truth assignment that satisfies φ. Consider a mapping ν constructed from σ as follows.

$$\nu = \{?X_i \rightarrow \mathtt{c} \mid \sigma(x_i) = 1\} \cup \{?Y_i \rightarrow \mathtt{c} \mid \sigma(x_i) = 0\}$$

In other words, if $\sigma(x_i) = 1$, then dom(ν) contains the variable $?X_i$ and $\nu(?X_i) = \mathtt{c}$, and if $\sigma(x_i) = 0$, then dom(ν) contains the variable $?Y_i$ and $\nu(?Y_i) = \mathtt{c}$. Notice that since σ is a truth assignment, the variables $?X_i$ and $?Y_i$ do not simultaneously belong to dom(ν), for every $i \in [1, m]$. Thus, we have that $\nu \models R$. Moreover, since σ satisfies each clause in φ, it is easy to see that $\nu \in [\![(P_{C_1} \text{ AND } \cdots \text{ AND } P_{C_n})]\!]_D$. Therefore, we have that $\nu \in [\![((P_{C_1} \text{ AND } \cdots \text{ AND } P_{C_n}) \text{ FILTER } R)]\!]_D$. Finally, since $\mu \in [\![P]\!]_D$, μ and ν are compatible, and $\mu \cup \nu = \mu$, we obtain that $\mu \in [\![(P \text{ AND } ((P_{C_1} \text{ AND } \cdots \text{ AND } P_{C_n}) \text{ FILTER } R))]\!]_D = [\![P_\varphi]\!]_D$. This concludes the proof of the theorem. □

We study now the problem when the OPT operator is added to the AND-FILTER fragment. The OPT operator is probably the most complicated in graph pattern expressions and, definitively, the most difficult to define. The following theorem shows that the inclusion of OPT to the AND-FILTER fragment makes the evaluation problem even harder compared with the AND-FILTER-UNION fragment.

Theorem 3.3. Evaluation *is PSPACE-complete for graph pattern expressions constructed by using only* AND, FILTER, *and* OPT *operators.*

Proof. The membership in PSPACE is given by Algorithm 1. Given a mapping μ, a (general) pattern P, and an RDF dataset D, the algorithm verifies whether $\mu \in [\![P]\!]_D$. In the procedure, we use pos(P, D) to denote the set of mappings ν such that dom(ν) $\subseteq$ var(P) and for every variable $?X \in$ dom(ν), it holds that $\nu(?X)$ is a value in D.

It is easy to see that the procedure is correct (it is essentially applying the definition of the semantics of every SPARQL operator). Given that the size needed to store the name of a variable in var(P) is $O(\log |P|)$ and the size needed to store an element of D is $O(\log |D|)$, we obtain that the size of a mapping in pos(P, D) is $O(|P| \cdot (\log |P| + \log |D|))$. Thus, given that the depth of the tree of recursive calls to **Eval** is $O(|P|)$, we have that procedure **Eval** can be implemented by using a polynomial amount of space. It should be pointed out that this shows that the evaluation problem for general graph patterns (constructed by using AND, FILTER, UNION, and OPT) is in PSPACE. Thus, in particular, we obtain that the evaluation problem is in PSPACE for pattern constructed by using only AND, FILTER and OPT operators.

To prove the PSPACE-hardness of Evaluation for the case of graph pattern expressions constructed by using only AND, FILTER, and OPT operators, we show how to reduce in polynomial time the Quantified Boolean Formula problem (QBF) to our problem. An instance of QBF is a quantified propositional formula φ of the form

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \forall x_3 \exists y_3 \cdots \forall x_m \exists y_m \, \psi,$$

where ψ is a quantifier-free formula of the form $C_1 \wedge \cdots \wedge C_n$, with each C_i ($i \in [1, n]$) being a disjunction of literals, that is, a disjunction of propositional

Algorithm 1. Eval (μ: mapping, P: graph pattern, G: RDF graph)

```
case:
$P$ is a triple pattern $t$:
  if dom($\mu$) = var($t$) and $\mu(t) \in D$ then return true
  return false
$P$ is a pattern of the form ($P_1$ FILTER $R$):
  if Eval($\mu$, $P_1$, $D$) = true and $\mu \models R$ then return true
  return false
$P$ is a pattern of the form ($P_1$ UNION $P_2$):
  if Eval($\mu$, $P_1$, $D$) = true or Eval($\mu$, $P_2$, $D$) = true then return true
  return false
$P$ is a pattern of the form ($P_1$ AND $P_2$):
  for each pair of mappings $\mu_1 \in$ pos($P_1$, $D$) and $\mu_2 \in$ pos($P_2$, $D$)
    if Eval($\mu_1$, $P_1$, $D$) = true and Eval($\mu_2$, $P_2$, $D$) = true and $\mu = \mu_1 \cup \mu_2$ then
      return true
  return false
$P$ is a pattern of the form ($P_1$ OPT $P_2$):
  if Eval($\mu$, ($P_1$ AND $P_2$), $D$) = true then return true
  if Eval($\mu$, $P_1$, $D$) = true then
    for each mapping $\mu' \in$ pos($P_2$, $D$)
      if Eval($\mu'$, $P_2$, $D$) = true and $\mu$ is compatible with $\mu'$ then return false
    return true
  return false
```

variables x_i and y_i, and negations of propositional variables. Then the problem is to verify whether φ is valid. It is known that QBF is PSPACE-complete [Garey and Johnson 1979].

In the reduction from QBF, we use a fixed RDF dataset.

$$D = \{(\texttt{a}, \texttt{tv}, 0),\ (\texttt{a}, \texttt{tv}, 1),\ (\texttt{a}, \texttt{false}, 0),\ (\texttt{a}, \texttt{true}, 1)\}$$

Moreover, the SPARQL variables $?X_1, \ldots, ?X_m$ and $?Y_1, \ldots, ?Y_m$ represent propositional variables $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$, respectively.

To represent the propositional formula ψ, consider first a filter expression R_ψ constructed from ψ by maintaining the conjunctions and disjunctions of ψ, and replacing every positive occurrence of x_i and y_i by $?X_i = 1$ and $?Y_i = 1$, respectively, and every occurrence of $\neg x_i$ and $\neg y_i$ by $?X_i = 0$ and $?Y_i = 0$, respectively. For instance, if ψ is the formula $(x_1 \vee \neg y_2) \wedge (\neg x_2 \vee y_1)$, then R_ψ is the filter expression $(?X_1 = 1 \vee ?Y_2 = 0) \wedge (?X_2 = 0 \vee ?Y_1 = 1)$. Then we define a graph pattern P_ψ as

$$(((\texttt{a}, \texttt{tv}, ?X_1) \text{ AND } \cdots \text{ AND } (\texttt{a}, \texttt{tv}, ?X_m) \text{ AND }$$
$$(\texttt{a}, \texttt{tv}, ?Y_1) \text{ AND } \cdots \text{ AND } (\texttt{a}, \texttt{tv}, ?Y_m)) \text{ FILTER } R_\psi).$$

It is easy to see that ψ is satisfiable if and only if there exists a mapping $\mu \in [\![P_\psi]\!]_D$. In particular, for each mapping μ, there exists a truth assignment

σ_μ defined as $\sigma_\mu(v) = \mu(?V)$ for every variable v in ψ, such that $\mu \in [\![P_\psi]\!]_D$ if and only if σ_μ satisfies ψ.

Now we explain how we represent a quantified propositional formula φ as a graph pattern expression P_φ. In P_φ we use SPARQL variables $?A_0, ?A_1, \ldots, ?A_m, ?B_0, ?B_1, \ldots, ?B_m$ and nested OPT and AND operators to represent the quantifier sequence $\forall x_1 \exists y_1 \cdots \forall x_m \exists y_m$. More precisely, for every $i \in [1, m]$, we define graph pattern expressions P_i and Q_i as follows.

$$\begin{aligned}
P_i \ &:= \ \big((\mathtt{a}, \mathtt{tv}, ?X_1) \text{ AND } \cdots \text{ AND } (\mathtt{a}, \mathtt{tv}, ?X_i) \text{ AND} \\
&\qquad (\mathtt{a}, \mathtt{tv}, ?Y_1) \text{ AND } \cdots \text{ AND } (\mathtt{a}, \mathtt{tv}, ?Y_{i-1}) \text{ AND} \\
&\qquad (\mathtt{a}, \mathtt{false}, ?A_{i-1}) \text{ AND } (\mathtt{a}, \mathtt{true}, ?A_i)\big), \\
Q_i \ &:= \ \big((\mathtt{a}, \mathtt{tv}, ?X_1) \text{ AND } \cdots \text{ AND } (\mathtt{a}, \mathtt{tv}, ?X_i) \text{ AND} \\
&\qquad (\mathtt{a}, \mathtt{tv}, ?Y_1) \text{ AND } \cdots \text{ AND } (\mathtt{a}, \mathtt{tv}, ?Y_i) \text{ AND} \\
&\qquad (\mathtt{a}, \mathtt{false}, ?B_{i-1}) \text{ AND } (\mathtt{a}, \mathtt{true}, ?B_i)\big)
\end{aligned}$$

Then we define P_φ as

$$\begin{aligned}
&((\mathtt{a}, \mathtt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ OPT } (P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots \\
&\qquad\qquad\qquad\qquad (P_m \text{ OPT } (Q_m \text{ AND } P_\psi)) \cdots))))))
\end{aligned}$$

Next we show that we can use graph expression P_φ to check whether φ is valid. More precisely, we show that φ is valid if and only if $\mu \in [\![P_\varphi]\!]_D$, where μ is a mapping such that $\mathrm{dom}(\mu) = \{?B_0\}$ and $\mu(?B_0) = 1$.

($\Leftarrow$) Assume that $\mu \in [\![P_\varphi]\!]_D$. Given that $P_1 = (\mathtt{a}, \mathtt{tv}, ?X_1)$ AND $(\mathtt{a}, \mathtt{false}, ?A_0)$ AND $(\mathtt{a}, \mathtt{true}, ?A_1)$, we have that $[\![P_1]\!]_D = \{\mu_0, \mu_1\}$, where $\mu_0 = \{?X_1 \to 0, ?A_0 \to 0, ?A_1 \to 1\}$ and $\mu_1 = \{?X_1 \to 1, ?A_0 \to 0, ?A_1 \to 1\}$. Given that P_1 mentions triple $(\mathtt{a}, \mathtt{true}, ?A_1)$ and P_2 mentions triple $(\mathtt{a}, \mathtt{false}, ?A_1)$, there is no mapping in $[\![P_1]\!]_D$ compatible with some mapping in $[\![P_2]\!]_D$. Thus, since μ_0 and μ_1 are compatible with μ and $\mu \in [\![P_\varphi]\!]_D$, there exist mappings ν_0 and ν_1 in $[\![Q_1]\!]_D$ such that μ_0, ν_0 are compatible, μ_1, ν_1 are compatible, and

$$\begin{aligned}
\mu_0 \cup \nu_0 \ &\in \ [\![(P_1 \text{ OPT } (Q_1 \text{ OPT } (P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots \\
&\qquad\qquad (P_m \text{ OPT } (Q_m \text{ AND } P_\psi)) \cdots)))))]\!]_D, \qquad (2) \\
\mu_1 \cup \nu_1 \ &\in \ [\![(P_1 \text{ OPT } (Q_1 \text{ OPT } (P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots \\
&\qquad\qquad (P_m \text{ OPT } (Q_m \text{ AND } P_\psi)) \cdots)))))]\!]_D. \qquad (3)
\end{aligned}$$

We note that $\nu_0(?X_1) = \mu_0(?X_1) = 0$, $\nu_1(?X_1) = \mu_1(?X_1) = 1$ and $\nu_0(?Y_1)$, $\nu_1(?Y_1)$ are not necessarily distinct.

Given Q_1 mentions $(\mathtt{a}, \mathtt{true}, ?B_1)$ and Q_2 mentions triple $(\mathtt{a}, \mathtt{false}, ?B_1)$, there is no mapping in $[\![Q_1]\!]_D$ compatible with some mapping in $[\![Q_2]\!]_D$. Thus, given that (2) holds, for every mapping $\zeta \in [\![P_2]\!]_D$, we have that if ν_0 and ζ are compatible, then there exists $\xi \in [\![Q_2]\!]_D$ such that ζ and ξ are compatible and

$$\zeta \cup \xi \ \in \ [\![(P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots (P_m \text{ OPT } (Q_m \text{ AND } P_\psi)) \cdots)))]\!]_D.$$

There are two mappings in $[\![P_2]\!]_D$ which are compatible with ν_0.

$$\begin{aligned}\mu_{00} &= \{?X_1 \to 0,\ ?X_2 \to 0,\ ?Y_1 \to \nu_0(?Y_1),\ ?A_1 \to 0,\ ?A_2 \to 1\}\\ \mu_{01} &= \{?X_1 \to 0,\ ?X_2 \to 1,\ ?Y_1 \to \nu_0(?Y_1),\ ?A_1 \to 0,\ ?A_2 \to 1\}\end{aligned}$$

Thus, from the previous discussion we conclude that there exist mappings ν_{00} and ν_{01} such that μ_{00}, ν_{00} are compatible, μ_{01}, ν_{01} are compatible, and

$$\begin{aligned}\mu_{00} \cup \nu_{00} &\in [\![(P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots(P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D,\\ \mu_{01} \cup \nu_{01} &\in [\![(P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots(P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D.\end{aligned}$$

Similarly, there are two mappings in $[\![P_2]\!]_D$ which are compatible with ν_1.

$$\begin{aligned}\mu_{10} &= \{?X_1 \to 1,\ ?X_2 \to 0,\ ?Y_1 \to \nu_1(?Y_1),\ ?A_1 \to 0,\ ?A_2 \to 1\}\\ \mu_{11} &= \{?X_1 \to 1,\ ?X_2 \to 1,\ ?Y_1 \to \nu_1(?Y_1),\ ?A_1 \to 0,\ ?A_2 \to 1\}\end{aligned}$$

Thus, given that (3) holds, we conclude that there exist mappings ν_{10} and ν_{11} such that μ_{10}, ν_{10} are compatible, μ_{11}, ν_{11} are compatible, and

$$\begin{aligned}\mu_{10} \cup \nu_{10} &\in [\![(P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots(P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D,\\ \mu_{11} \cup \nu_{11} &\in [\![(P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots(P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D.\end{aligned}$$

If we continue in this fashion, we conclude that for every $i \in [2, m-1]$ and $n_1 \cdots n_i \in \{0,1\}^i$, and for the following mappings in $[\![P_{i+1}]\!]_D$

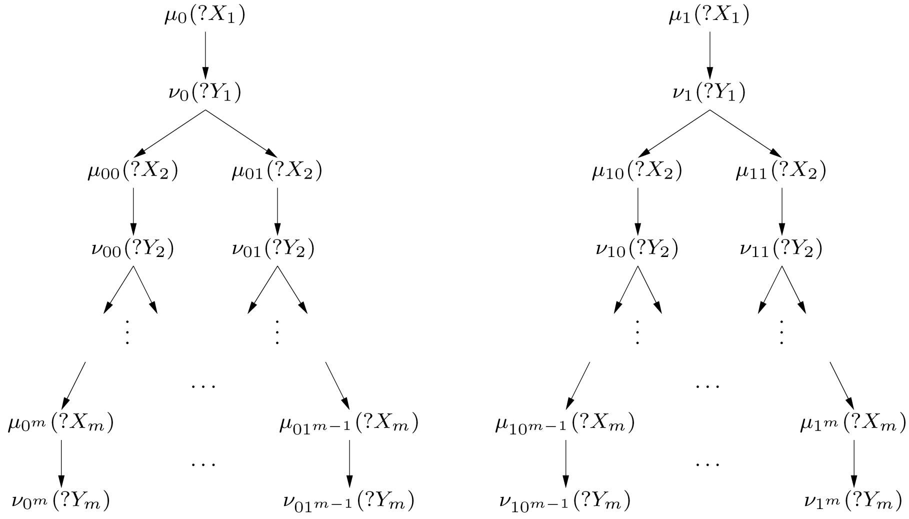
$$\begin{aligned}\mu_{n_1\cdots n_i 0} &= \{?X_1 \to n_1,\ \ldots,\ ?X_i \to n_i,\ ?X_{i+1} \to 0,\\ &\qquad ?Y_1 \to \nu_{n_1}(?Y_1),\ \ldots,\ ?Y_i \to \nu_{n_1\cdots n_i}(?Y_i),\ ?A_{i-1} \to 0,\ ?A_i \to 1\}\\ \mu_{n_1\cdots n_i 1} &= \{?X_1 \to n_1,\ \ldots,\ ?X_i \to n_i,\ ?X_{i+1} \to 1,\\ &\qquad ?Y_1 \to \nu_{n_1}(?Y_1),\ \ldots,\ ?Y_i \to \nu_{n_1\cdots n_i}(?Y_i),\ ?A_{i-1} \to 0,\ ?A_i \to 1\}\end{aligned}$$

there exist mappings $\nu_{n_1\cdots n_i 0}$ and $\nu_{n_1\cdots n_i 1}$ in $[\![Q_{i+1}]\!]_D$ such that $\mu_{n_1\cdots n_i 0}$, $\nu_{n_1\cdots n_i 0}$ are compatible, $\mu_{n_1\cdots n_i 1}$, $\nu_{n_1\cdots n_i 1}$ are compatible, and

$$\begin{aligned}\mu_{n_1\cdots n_i 0} \cup \nu_{n_1\cdots n_i 0} &\in [\![(P_{i+1} \text{ OPT } (Q_{i+1} \text{ OPT } (\cdots\\ &\qquad\qquad (P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D,\\ \mu_{n_1\cdots n_i 1} \cup \nu_{n_1\cdots n_i 1} &\in [\![(P_{i+1} \text{ OPT } (Q_{i+1} \text{ OPT } (\cdots\\ &\qquad\qquad (P_m \text{ OPT } (Q_m \text{ AND } P_\psi))\cdots)))]\!]_D.\end{aligned}$$

In particular, for every $n_1 \cdots n_m \in \{0,1\}^m$, given that $\nu_{n_1\cdots n_m} \in [\![(Q_m \text{ AND } P_\psi)]\!]_D$, Q_m is a conjunction of triple patterns and $\text{var}(P_\psi) \subseteq \text{var}(Q_m)$, we conclude that $\nu_{n_1\cdots n_m} \in [\![P_\psi]\!]_D$. Hence, if $\sigma_{n_1\cdots n_m}$ is a truth assignment defined as $\sigma_{n_1\cdots n_m}(x) = \nu_{n_1\cdots n_m}(?X)$ for every variable x in ψ, then $\sigma_{n_1\cdots n_m}$ satisfies ψ. Next we show this implies that φ is valid.

To prove that φ is valid, consider the following tree.



Given that for every $n_1 \cdots n_m \in \{0, 1\}^m$, it holds that $\sigma_{n_1 \cdots n_m}$ satisfies ψ and

$$\begin{array}{ccccccr}
\mu_{n_1 \cdots n_i}(?X_j) & = & \nu_{n_1 \cdots n_i}(?X_j) & = & \mu_{n_1 \cdots n_m}(?X_j) & & i \in [1, m],\ j \in [1, i] \\
\mu_{n_1 \cdots n_i}(?Y_k) & = & \nu_{n_1 \cdots n_i}(?Y_k) & = & \mu_{n_1 \cdots n_m}(?Y_k) & & i \in [1, m],\ k \in [1, i-1] \\
 & & \nu_{n_1 \cdots n_i}(?Y_i) & = & \mu_{n_1 \cdots n_m}(?Y_i) & & i \in [1, m]
\end{array}$$

we have that every path from the root to a leaf in the preceding tree represents a satisfying assignment for ψ. Thus, we conclude that φ is valid from the fact that for every $i \in [1, m]$ and $n_1 \cdots n_{i-1} \in \{0, 1\}^{i-1}$, it holds that $\mu_{n_1 \cdots n_{i-1} 0}(?X_i) = 0$ and $\mu_{n_1 \cdots n_{i-1} 1}(?X_i) = 1$.

($\Rightarrow$) The proof that φ is valid implies $\mu \in [\![P_\varphi]\!]_D$ is similar to the previous proof. □

The following result shows that the combination of OPT and UNION is also an important source of complexity when evaluating SPARQL patterns.

THEOREM 3.4. EVALUATION *is PSPACE-complete for graph pattern expressions constructed by using only* AND*,* UNION*, and* OPT *operators.*

PROOF. Membership in PSPACE follows from the proof of Theorem 3.3 (procedure **Eval** in that proof considers operators AND, FILTER, UNION, and OPT). To prove the PSPACE-hardness of EVALUATION, we use a reduction from QBF similar to the one used in the proof of Theorem 3.3. In that proof, we encode a quantified propositional formula φ of the form $\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_m \exists y_m\, \psi$, with a pattern that uses AND and FILTER operators to encode the satisfiability of ψ, and uses OPT and AND operators to encode the quantifier alternation. We show here how to encode the satisfiability of ψ by using the UNION operator instead of the FILTER operator.

Formula ψ is a quantifier-free formula of the form $C_1 \wedge \cdots \wedge C_n$, with each C_i ($i \in [1, n]$) being a disjunction of literals, that is, a disjunction of propositional variables x_i and y_i, and negations of propositional variables. In the reduction we use the same dataset $D = \{(\mathtt{a}, \mathtt{tv}, 0),\ (\mathtt{a}, \mathtt{tv}, 1),\ (\mathtt{a}, \mathtt{false}, 0),\ (\mathtt{a}, \mathtt{true}, 1)\}$ as in the proof of Theorem 3.3. Consider now the graph pattern expression Q_ψ

constructed from ψ as follows: (a) Replace every positive occurrence of x_i and y_i in ψ by the triple patterns $(\texttt{a}, \texttt{true}, ?X_i)$ and $(\texttt{a}, \texttt{true}, ?Y_i)$, respectively, (b) replace every occurrence of $\neg x_i$ and $\neg y_i$ by the triple patterns $(\texttt{a}, \texttt{false}, ?X_i)$ and $(\texttt{a}, \texttt{false}, ?Y_i)$, respectively, and (c) replace every disjunction by a UNION operator, and every conjunction by an AND operator. For instance, if ψ is the formula $(x_1 \vee \neg y_2) \wedge (\neg x_2 \vee y_1)$, then Q_ψ is the pattern expression

$$((\texttt{a}, \texttt{true}, ?X_1) \text{ UNION } (\texttt{a}, \texttt{false}, ?Y_2)) \text{ AND } ((\texttt{a}, \texttt{false}, ?X_2) \text{ UNION } (\texttt{a}, \texttt{true}, ?Y_1)).$$

We define then the pattern S_ψ as

$$(Q_\psi \text{ AND } (\texttt{a}, \texttt{tv}, ?X_1) \text{ AND } (\texttt{a}, \texttt{tv}, ?Y_1) \text{ AND } \cdots \text{ AND } (\texttt{a}, \texttt{tv}, ?X_m) \text{ AND } (\texttt{a}, \texttt{tv}, ?Y_m)).$$

It is straightforward to show that ψ is satisfiable if and only if there exists a mapping $\mu \in [\![S_\psi]\!]_D$. Moreover, for each such mapping μ, the truth assignment σ_μ defined as $\sigma_\mu(v) = \mu(?V)$ for every propositional variable v in ψ, is such that $\mu \in [\![S_\psi]\!]_D$ if and only if σ_μ satisfies ψ. The rest of the proof follows the proof of Theorem 3.3 but replacing pattern P_ψ by S_ψ. □

Notice that procedure **Eval** in the proof of Theorem 3.3 shows that the evaluation problem for general graph patterns is in PSPACE. Then as a corollary of the two previous theorems we obtain that, in general, the evaluation problem for SPARQL is PSPACE-complete.

Corollary 3.5. Evaluation *is PSPACE-complete.*

When verifying whether $\mu \in [\![P]\!]_D$, it is natural to assume that the size of P is considerably smaller than the size of D. This assumption is very common when studying the complexity of a query language. In fact, it is named *data complexity* in the database literature [Vardi 1982], and it is defined as the complexity of the evaluation problem for a fixed query. More precisely, for the case of SPARQL, given a graph pattern expression P, the evaluation problem for P, denoted by Evaluation(P), has as input an RDF dataset D and a mapping μ, and the problem is to verify whether $\mu \in [\![P]\!]_D$.

Theorem 3.6. Evaluation(P) *is in LOGSPACE for every graph pattern expression* P.

Proof. Consider the procedure **Eval** in the proof of Theorem 3.3. Given that the size needed to store a mapping in pos(P, D) is $O(|P| \cdot (\log |P| + \log |D|))$, this bound becomes $O(\log |D|)$ when P is considered to be fixed. Thus, given that the depth of the tree of recursive calls to **Eval** is a fixed constant if P is considered to be fixed, we obtain that **Eval** can be implemented by using logarithmic space in this case. □

An important question is whether one can find interesting classes of graph patterns, constructed by imposing simple and natural syntactic restrictions, such that one can obtain lower complexity bounds for the evaluation problem on these classes. In the following section, we introduce a first such restriction.

3.1 A Simple Normal Form for Graph Patterns

Recall that two graph patterns P_1 and P_2 are *equivalent*, denoted by $P_1 \equiv P_2$, if $[\![P_1]\!]_D = [\![P_2]\!]_D$ for every RDF dataset D. In Lemma 2.5, we show that AND distributes over UNION. The following lemma states that OPT and FILTER also distribute over UNION. The proof of this lemma can be found in the Appendix.

LEMMA 3.7. *Let P_1, P_2, and P_3 be graph pattern expressions and R a built-in condition. Then:*

(1) $((P_1 \text{ UNION } P_2) \text{ OPT } P_3) \equiv ((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))$.
(2) $((P_1 \text{ UNION } P_2) \text{ FILTER } R) \equiv ((P_1 \text{ FILTER } R) \text{ UNION } (P_2 \text{ FILTER } R))$.

The application of Lemmas 2.5 and 3.7 allows to translate every graph pattern into an equivalent one where UNION is the outermost operator. More precisely, we say that a pattern P is UNION-free if P is constructed by using only operators AND, OPT, and FILTER. Then we have the following proposition.

PROPOSITION 3.8. *Every graph pattern P is equivalent to a pattern of the form*

$$(P_1 \text{ UNION } P_2 \text{ UNION } P_3 \text{ UNION } \cdots \text{ UNION } P_n), \tag{4}$$

where each P_i $(1 \leq i \leq n)$ is UNION*-free.*

Notice that we omit the parentheses in the expression (4) given the associativity of UNION.

PROOF. By induction on the structure of P. If P is a triple pattern, then the property trivially holds. Assume that the property holds for patterns P_1 and P_2, that is, there exist expressions $(P_1^1 \text{ UNION } \cdots \text{ UNION } P_m^1)$ and $(P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$ such that: (1) $P_1 \equiv (P_1^1 \text{ UNION } \cdots \text{ UNION } P_m^1)$, (2) $P_2 \equiv (P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$, (3) each pattern P_i^1 $(i \in \{1, \ldots, m\})$ is UNION-free, and (4) each pattern P_i^2 $(i \in \{1, \ldots, n\})$ is UNION-free. Next we show that the property holds for $(P_1 \text{ UNION } P_2)$, $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ FILTER } R)$, where R is a built-in condition.

It is easy to see that the property holds for $P = (P_1 \text{ UNION } P_2)$. Thus, we consider first the case of $P = (P_1 \text{ AND } P_2)$. By induction hypothesis and Lemma 2.5, we have that

$$\begin{aligned} P \;\equiv\; & \big(P_1^1 \text{ AND } P_1^2\big) \text{ UNION } \cdots \text{ UNION } \big(P_1^1 \text{ AND } P_n^2\big) \text{ UNION } \cdots \\ & \text{UNION } \big(P_m^1 \text{ AND } P_1^2\big) \text{ UNION } \cdots \text{ UNION } \big(P_m^1 \text{ AND } P_n^2\big). \end{aligned}$$

Second, if $P = (P_1 \text{ FILTER } R)$, then we conclude from Lemma 2.5, Lemma 3.7, and induction hypothesis that

$$\big(P_1 \text{ FILTER } R\big) \equiv \big(P_1^1 \text{ FILTER } R\big) \text{ UNION } \cdots \text{ UNION } \big(P_m^1 \text{ FILTER } R\big).$$

Finally, assume that $P = (P_1 \text{ OPT } P_2)$. By induction hypothesis and Lemma 3.7, we conclude that P is equivalent to the graph pattern expression

$$\big(P_1^1 \text{ OPT } P_2\big) \text{ UNION } \cdots \text{ UNION } \big(P_m^1 \text{ OPT } P_2\big).$$

Thus, to conclude the proof, we need to show that the condition of the proposition holds for each one of the expressions $(P_i^1 \text{ OPT } P_2)$. To show this, we use the following claim.

CLAIM 3.9. *Let Q be a* UNION*-free graph pattern. For every RDF dataset D and pair of mappings $\mu_1, \mu_2 \in [\![Q]\!]_D$, if μ_1 and μ_2 are compatible, then $\mu_1 = \mu_2$.*

PROOF OF CLAIM 3.9. By induction on the structure of the UNION-free pattern Q. If Q is a triple pattern, then the property trivially holds. Assume first that $Q = (Q_1 \text{ AND } Q_2)$, where Q_1 and Q_2 satisfy the condition, that is, if $\xi, \zeta \in [\![Q_i]\!]_D$ $(i = 1, 2)$ and ξ, ζ are compatible, then $\xi = \zeta$. Let μ_1 and μ_2 be compatible mappings in $[\![Q]\!]_D$. Then there exist $\nu_1, \omega_1 \in [\![Q_1]\!]_D$ and $\nu_2, \omega_2 \in [\![Q_2]\!]_D$ such that $\mu_1 = \nu_1 \cup \omega_1$ and $\mu_2 = \nu_2 \cup \omega_2$. Given that μ_1 and μ_2 are compatible, we have that ν_1, ν_2 are compatible and ω_1, ω_2 are compatible. Thus, by induction hypothesis we have that $\nu_1 = \nu_2$ and $\omega_1 = \omega_2$ and, hence, $\mu_1 = \mu_2$. Second, assume that $Q = (Q_1 \text{ OPT } Q_2)$, and let μ_1 and μ_2 be compatible mappings in $[\![Q]\!]_D$. We consider four cases.

(1) If there exist $\nu_1, \omega_1 \in [\![Q_1]\!]_D$ and $\nu_2, \omega_2 \in [\![Q_2]\!]_D$ such that $\mu_1 = \nu_1 \cup \omega_1$ and $\mu_2 = \nu_2 \cup \omega_2$, then we conclude that $\mu_1 = \mu_2$ as in the case $Q = (Q_1 \text{ AND } Q_2)$.
(2) If $\mu_1, \mu_2 \in [\![Q_1]\!]_D$ and both are not compatible with any mapping in $[\![Q_2]\!]_D$, then by induction hypothesis we conclude that $\mu_1 = \mu_2$.
(3) If $\mu_1 \in [\![Q_1]\!]_D$, μ_1 is not compatible with any mapping in $[\![Q_2]\!]_D$, $\mu_2 = \nu_2 \cup \omega_2$, $\nu_2 \in [\![Q_1]\!]_D$ and $\omega_2 \in [\![Q_2]\!]_D$, then given that μ_1 and μ_2 are compatible, we have that μ_1 and ν_2 are compatible. Thus, by induction hypothesis we conclude that $\mu_1 = \nu_2$ and, therefore, μ_1 is compatible with $\omega_2 \in [\![Q_2]\!]_D$, which contradicts our initial assumption.
(4) If $\mu_1 = \nu_1 \cup \omega_1$, $\nu_1 \in [\![Q_1]\!]_D$, $\omega_1 \in [\![Q_2]\!]_D$, $\mu_2 \in [\![Q_1]\!]_D$ and μ_2 is not compatible with any mapping in $[\![Q_2]\!]_D$, then we obtain a contradiction as in the previous case.

Finally, assume that $Q = (Q_1 \text{ FILTER } R)$, where Q_1 satisfies the condition of the claim. Let μ_1 and μ_2 be compatible mappings in $[\![Q]\!]_D$. Then $\mu_1 \in [\![Q_1]\!]_D$, $\mu_1 \models R$, $\mu_2 \in [\![Q_1]\!]_D$, $\mu_2 \models R$ and, thus, $\mu_1 = \mu_2$ by induction hypothesis. This concludes the proof of the claim. □

Let $i \in \{1, \ldots, m\}$. Next we show that the condition of the proposition holds for $(P_i^1 \text{ OPT } P_2)$. Assume that $?X_1, \ldots, ?X_n, ?Y_1, \ldots, ?Y_n, ?Z_1, \ldots, ?Z_n$ are variables that are mentioned neither in P_i^1 nor in P_2. Moreover, for every $j \in \{1, \ldots, n\}$, let Q_j be the graph pattern $(P_i^1 \text{ OPT } (P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j)))$, and assume that Q is the graph pattern $(Q_1 \text{ AND } \cdots \text{ AND } Q_n)$. Next we show that $(P_i^1 \text{ OPT } P_2)$ is equivalent to

$$\big(P_i^1 \text{ AND } P_1^2\big) \text{ UNION } \cdots \text{ UNION } \big(P_i^1 \text{ AND } P_n^2\big) \text{ UNION } \\ \big(Q \text{ FILTER } \big(\neg\, \text{bound}(?X_1) \wedge \cdots \wedge \neg\, \text{bound}(?X_n)\big)\big). \quad (5)$$

We note that the previous formula is of the form (4), from which we conclude that the proposition holds for $(P_i^1 \text{ OPT } P_2)$ and, more generally, holds for $(P_1 \text{ OPT } P_2)$.

($\Rightarrow$) Assume that D is an RDF dataset and $\mu \in [\![(P_i^1 \text{ OPT } P_2)]\!]_D$. Then we have that either $\mu \in ([\![P_i^1]\!]_D \bowtie [\![P_2]\!]_D)$ or $\mu \in ([\![P_i^1]\!]_D \smallsetminus [\![P_2]\!]_D)$. If $\mu \in ([\![P_i^1]\!]_D \bowtie [\![P_2]\!]_D)$, we have that there exist compatible mappings $\mu_1 \in [\![P_i^1]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$ such that $\mu = \mu_1 \cup \mu_2$. Thus, given that $P_2 \equiv (P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$, we have that $\mu_2 \in [\![P_j^2]\!]_D$ for some $j \in \{1, \ldots, n\}$. We conclude that $\mu \in [\![(P_i^1 \text{ AND } P_j^2)]\!]_D$ and, thus, μ belongs to the evaluation of (5) over D. If $\mu \in ([\![P_i^1]\!]_D \smallsetminus [\![P_2]\!]_D)$, we have that $\mu \in [\![P_i^1]\!]_D$ and no mapping in $[\![P_2]\!]_D$ is compatible with μ. Let $j \in \{1, \ldots, n\}$. Given that no mapping in $[\![P_2]\!]_D$ is compatible with μ and $P_2 \equiv (P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$, we have that no mapping in $[\![P_j^2]\!]_D$ is compatible with μ and, therefore, no mapping in $[\![(P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j))]\!]_D$ is compatible with μ. Thus, we have that $\mu \in [\![(P_i^1 \text{ OPT } (P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j)))]\!]_D$. Moreover, given that $?X_j$, $?Y_j$ and $?Z_j$ are mentioned neither in P_i^1 nor in P_2, we have that $?X_j \notin \text{dom}(\mu)$, that is, $\neg\,\text{bound}(?X_j)$ holds for μ. Given that the previous conditions hold for every $j \in \{1, \ldots, n\}$, we conclude that μ belongs to the evaluation of (5) over D.

($\Leftarrow$) Assume that D is an RDF dataset and μ belongs to the evaluation of (5) over D. To prove that $\mu \in [\![(P_i^1 \text{ OPT } P_2)]\!]_D$, we consider two cases. Assume first that there exists $j \in \{1, \ldots, n\}$ such that $\mu \in [\![(P_i^1 \text{ AND } P_j^2)]\!]_D$. Then we have that there exist compatible mappings $\mu_1 \in [\![P_i^1]\!]_D$ and $\mu_2 \in [\![P_j^2]\!]_D$ such that $\mu = \mu_1 \cup \mu_2$. Given that $P_2 \equiv (P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$, we have that $\mu_2 \in [\![P_2]\!]_D$ and, therefore, $\mu \in [\![(P_i^1 \text{ OPT } P_2)]\!]_D$. Second, assume that

$$\mu \in [\![(Q \text{ FILTER } (\neg\,\text{bound}(?X_1) \wedge \cdots \wedge \neg\,\text{bound}(?X_n)))]\!]_D.$$

Then we have that $\mu \in [\![Q]\!]_D$ and $\mu \models (\neg\,\text{bound}(?X_1) \wedge \cdots \wedge \neg\,\text{bound}(?X_n))$, that is, $?X_j \notin \text{dom}(\mu)$ for every $j \in \{1, \ldots, n\}$. By definition of Q, we have that there exist compatible mappings $\mu_1 \in [\![Q_1]\!]_D$, $\ldots$, $\mu_n \in [\![Q_n]\!]_D$ such that $\mu = \mu_1 \cup \cdots \cup \mu_n$. Let $j \in \{1, \ldots, n\}$. Given that $?X_j \notin \text{dom}(\mu_j)$ and $\mu_j \in [\![Q_j]\!]_D$, we have that $\mu_j \in [\![P_i^1]\!]_D$ and μ_j is not compatible with any mapping in $[\![(P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j))]\!]_D$. Next we use this fact to prove that μ_j is not compatible with any mapping in $[\![P_j^2]\!]_D$. For the sake of contradiction, assume that there exists a mapping $\xi \in [\![P_j^2]\!]_D$ such that μ_j and ξ are compatible, and let ξ' be an arbitrary mapping in $[\![(?X_j, ?Y_j, ?Z_j)]\!]_D$. Given that $?X_j$, $?Y_j$, and $?Z_j$ are not mentioned in P_j^2, we have that ξ and ξ' are compatible and, hence, $\xi \cup \xi' \in [\![(P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j))]\!]_D$. But $?X_j$, $?Y_j$, and $?Z_j$ are not mentioned in P_i^1 and, thus, $\xi \cup \xi'$ is compatible with μ_j (since ξ is compatible with μ_j), which contradicts the fact that no mapping in $[\![(P_j^2 \text{ AND } (?X_j, ?Y_j, ?Z_j))]\!]_D$ is compatible with μ_j. Therefore, we have that $\mu_j \in [\![P_i^1]\!]_D$ and no mapping in $[\![P_j^2]\!]_D$ is compatible with μ_j, for every $j \in \{1, \ldots, n\}$. Given that P_i^1 is a UNION-free graph pattern, we have from Claim 3.9 that $\mu_1 = \mu_2 = \cdots = \mu_n$. Thus, given that $\mu = \mu_1 \cup \cdots \cup \mu_n$, we have that $\mu \in [\![P_i^1]\!]_D$ and no mapping in $[\![P_j^2]\!]_D$ is compatible with μ, for every $j \in \{1, \ldots, n\}$. But we know that $P_2 \equiv (P_1^2 \text{ UNION } \cdots \text{ UNION } P_n^2)$, from which we conclude that $\mu \in [\![P_i^1]\!]_D$ and no mapping in $[\![P_2]\!]_D$ is compatible with μ. Hence, we have that $\mu \in [\![(P_i^1 \text{ OPT } P_2)]\!]_D$. This concludes the proof of the proposition.

We say that a graph pattern is in UNION *normal form* if the pattern is in the form (4).[2] The following result shows that for graph patterns in UNION normal form that do not use the OPT operator, the evaluation problem can be solved efficiently. It is a direct consequence of Theorem 3.1.

COROLLARY 3.10. EVALUATION *can be solved in time* $O(|P| \cdot |D|)$ *for graph patterns in* UNION *normal form constructed by using only* AND*,* FILTER*, and* UNION *operators.*

We have managed to lower the complexity of the AND-FILTER-UNION fragment by imposing a simple normal form. However, Theorem 3.3 implies that when the OPT operator is allowed in graph patterns, the complexity of the evaluation problem is PSPACE-hard even if we restrict to patterns in UNION normal form. In the following section, we introduce a simple and natural syntactic condition that patterns usually satisfy in practice. Under this condition, the complexity of the evaluation of graph patterns in UNION normal form is lower even if the OPT operator is allowed.

4. WELL-DESIGNED GRAPH PATTERNS

The exact semantics of graph pattern expressions has been extensively discussed on the mailing list of the W3C. One of the most delicate issues in the definition of a semantics for graph pattern expressions is the semantics of the OPT operator. The idea behind the OPT operator is to allow for *optional matching* of patterns, that is, to allow information to be added if it is available, instead of rejecting whenever some part of a pattern does not match. This intuition fails in some simple, but unnatural, examples. For instance, consider the graph pattern

$$P \;=\; ((?X,\ \text{name},\ \text{john})\ \text{OPT}\ ((?Y,\ \text{name},\ \text{mick})\ \text{OPT}\ (?X,\ \text{email},\ ?Z))). \quad (6)$$

What is unnatural about graph pattern P is the fact that $(?X, \text{email}, ?Z)$ is giving optional information for $(?X, \text{name}, \text{john})$, but in P appears as giving optional information for $(?Y, \text{name}, \text{mick})$. For example, $(B_2, \text{name}, \text{john})$ and $(B_2, \text{email}, \text{john@ac.edu})$ are triples in the dataset D of Example 2.4, but the evaluation of P results in the set $\{\{?X \to B_2\}\}$ (since $[\![(?Y, \text{name}, \text{mick})]\!]_D = \emptyset$) without giving information about the email of john.

A careful examination of the examples that produce conflicts reveals a common pattern: A graph pattern P mentions an expression $P' = (P_1 \text{ OPT } P_2)$ and a variable $?X$ occurring both inside P_2 and outside P' but not occurring in P_1. In general, graph pattern expressions satisfying this condition are not natural.

To present the main definition of this section, we need to introduce some terminology. We say that a graph pattern Q is *safe* if for every subpattern

[2] In the conference version of this article [Pérez et al. 2006a], the proof of the existence of a UNION normal form uses the equivalence $(P_1 \text{ OPT } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ OPT } P_2) \text{ UNION } (P_1 \text{ OPT } P_3))$ (see Proposition 1 in Pérez et al. [2006a]). It was pointed out to us by M. Schmidt that the proof of this rule in Pérez et al. [2006a] is incorrect, and that this rule does not hold in general (see Schmidt et al. [2008]). In the proof of Proposition 3.8, we use a corrected version of this rule (see expression (5)).

$(P \text{ FILTER } R)$ of Q, it holds that $\text{var}(R) \subseteq \text{var}(P)$. This safety condition is present in many database query languages.

Definition 4.1. A UNION-free graph pattern P is *well designed* if P is safe and, for every subpattern $P' = (P_1 \text{ OPT } P_2)$ of P and for every variable $?X$ occurring in P, the following condition holds.

if $?X$ occurs both inside P_2 and outside P', then it also occurs in P_1.

For instance, pattern (6) given earlier is not well designed, while all the UNION-free patterns in Example 2.4 are well designed. We can extend Definition 4.1 to patterns in UNION normal form: We say that a pattern of the form $(P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n)$ is well designed if every P_i $(1 \leq i \leq n)$ is a UNION-free well-designed graph pattern.

The pattern used to obtain a PSPACE lower bound in Theorem 3.3 is not well designed. In that proof, pattern P_φ is defined as

$$((\texttt{a}, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ OPT } (P_2 \text{ OPT } (Q_2 \text{ OPT } (\cdots \\ (P_m \text{ OPT } (Q_m \text{ AND } P_\psi)) \cdots)))))),$$

and for every $i \in [1, m-1]$, there exists a variable that occurs in P_i and in P_{i+1} that does not occur in Q_i. Thus, an immediate question is whether the complexity of evaluating well-designed graph pattern expressions is lower than in the general case. We show in this section that this is indeed the case, and we prove a coNP upper bound for the case of well-designed graph patterns. But not only that; in this section we also consider the problem of optimizing well-designed graph patterns. Since the beginning of the relational data model, several techniques for optimizing the evaluation of relational algebra expressions have been developed. In fact, one of the reasons why relational algebra is so extensively used to implement SQL is the existence of simple reordering and optimization rules for this language. Unfortunately, the development of this type of rules for SPARQL is limited by the presence of the OPT operator. However, we show in this section that well-designed patterns are suitable for reordering and optimization, demonstrating the significance of this class of queries from the practical point of view.

We note first that the well-designed property can be checked efficiently by a straightforward procedure. Let P be a pattern. Then for every subpattern P' of P of the form $(P_1 \text{ OPT } P_2)$, we construct three sets: sets V_{P_1} and V_{P_2}, containing the variables occurring in P_1 and P_2, respectively, and set $O_{P'}$ containing the variables that occur *outside* P'. To construct V_{P_1}, we collect variables by making a bottom-up traversal of the subpatterns of P_1. We repeat this procedure in P_2 to construct V_{P_2}. To construct $O_{P'}$, we make a bottom-up traversal of the entire pattern P, but not taking into consideration P'. Having these three sets, we check whether $V_{P_2} \cap O_{P'} \subseteq V_{P_1}$, that is, we check whether every variable that occurs inside P_2 and outside P' also occurs inside P_1, which is exactly the well-designed condition. We must repeat this test for every OPT subpattern of P. Notice that the test for every OPT subpattern takes linear time in the size of P, and then, the entire process takes time proportional to the size of P times the number of OPT subpatterns of P. We can then state the following proposition.

Proposition 4.2. *Testing if a pattern P is well designed can be done in time $O(|P|^2)$.*

The rest of this section is organized as follows. In Section 4.1, we present a characterization of the evaluation of well-designed graph patterns based on the notions of *reduction* of patterns and *subsumption* of mappings. We then use this characterization to derive a coNP upper bound for the complexity of the evaluation problem for well-designed patterns. In Section 4.2, we further show that well-designed patterns are suitable for reordering and optimization. We present several optimization rules and define a normal form for well-designed patterns.

4.1 Complexity of Evaluating Well-Designed Patterns

Intuitively, if we *delete* some optional parts of a pattern P to obtain a new pattern P', the mappings in the evaluation of P' over a dataset D could not be more informative than the mappings in the evaluation of P over D. In other words, the optional matchings of a pattern must only serve to *extend* solutions with new information, but not to reject solutions if some information is not provided. We formalize this intuition and show, in Lemma 4.3, that the intuition is indeed correct for the case of well-designed graph patterns. We then use this lemma to develop a characterization of the evaluation of well-designed graph patterns.

We say that a mapping μ is *subsumed* by a mapping μ', denoted by $\mu \sqsubseteq \mu'$, if μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$, that is, μ is subsumed by μ' if μ agrees with μ' in every variable for which μ is defined. For sets of mappings Ω and Ω', we write $\Omega \sqsubseteq \Omega'$ if for every mapping $\mu \in \Omega$, there exists a mapping $\mu' \in \Omega'$ such that $\mu \sqsubseteq \mu'$.

We say that a pattern P' is a *reduction* of a pattern P, if P' can be obtained from P by replacing a subformula $(P_1 \text{ OPT } P_2)$ of P by P_1, that is, if P' is obtained by deleting some optional part of P. For example, $P' = (t_1 \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)))$ is a reduction of $P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)))$ since P' can be obtained from P by replacing $(t_1 \text{ OPT } t_2)$ by t_1. The reflexive and transitive closure of the reduction relation is denoted by $\trianglelefteq$. Thus, for example, if $P'' = (t_1 \text{ AND } t_2)$, then $P'' \trianglelefteq P$ since P'' is a reduction of P' and P' is a reduction of P. We note that if $P' \trianglelefteq P$ and P is well designed, then P' is well designed. If $P' \trianglelefteq P$ and $P' \neq P$, then we say that P' is a proper reduction of P, denoted by $P' \triangleleft P$.

We can now state the result that formalizes the intuition mentioned at the beginning of this section.

Lemma 4.3. *Let P be a* UNION*-free well-designed graph pattern, and P' a pattern such that $P' \trianglelefteq P$. Then $[\![P']\!]_D \sqsubseteq [\![P]\!]_D$ for every dataset D.*

Before proving the lemma, we show that it does not hold for patterns that are not well designed. Consider dataset $D = \{(1, \mathsf{a}, 1), (2, \mathsf{a}, 2), (3, \mathsf{a}, 3)\}$ and nonwell-designed pattern

$$P = ((?X, \mathsf{a}, 1) \text{ OPT } ((?Y, \mathsf{a}, 2) \text{ OPT } (?X, \mathsf{a}, 3))).$$

The evaluation of P results in the set $\{\{?X \to 1\}\}$. By deleting the optional part $(?X, \texttt{a}, 3)$ of P, we obtain the reduction $P' = ((?X, \texttt{a}, 1) \text{ OPT } (?Y, \texttt{a}, 2))$ of P. The evaluation of P' results in the set $\{\{?X \to 1, ?Y \to 2\}\}$. Thus, we have that $[\![P']\!]_D \not\sqsubseteq [\![P]\!]_D$.

Proof of Lemma 4.3. The proof is by induction on the structure of P. In the basis case, we have that P is a triple pattern and, thus, $P' = P$ and the property trivially holds. For the inductive step, we need to consider three cases. In all these cases, we need to show that if $P' \trianglelefteq P$ and $\mu' \in [\![P']\!]_D$, then there exists $\mu \in [\![P]\!]_D$ such that $\mu' \sqsubseteq \mu$.

(1) If $P = (P_1 \text{ AND } P_2)$, then $P' = (P'_1 \text{ AND } P'_2)$ for $P'_1 \trianglelefteq P_1$ and $P'_2 \trianglelefteq P_2$. Thus, $\mu' = \mu'_1 \cup \mu'_2$ with $\mu'_1 \in [\![P'_1]\!]_D$ and $\mu'_2 \in [\![P'_2]\!]_D$. By induction hypothesis, there exist mappings $\mu_1 \in [\![P_1]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$ such that $\mu'_1 \sqsubseteq \mu_1$ and $\mu'_2 \sqsubseteq \mu_2$. Next we show that μ_1 is compatible with μ_2, which implies that $\mu'_1 \cup \mu'_2 \sqsubseteq \mu_1 \cup \mu_2$ and, hence, $\mu' \sqsubseteq \mu$ for $\mu = \mu_1 \cup \mu_2 \in [\![P]\!]_D$.
Let $?X$ be a variable in $\text{dom}(\mu_1) \cap \text{dom}(\mu_2)$. We need to show that $\mu_1(?X) = \mu_2(?X)$. Given that $?X \in \text{var}(P_1)$ and $?X \in \text{var}(P_2)$, we have that for every pattern $O \trianglelefteq P_1$, it is the case that $?X \in \text{var}(O)$ (otherwise we would obtain a contradiction with the fact that $P = (P_1 \text{ AND } P_2)$ is well designed). Then given that $P'_1 \trianglelefteq P_1$ we also have that $?X \in \text{var}(O)$ for every pattern $O \trianglelefteq P'_1$. Therefore, by using the following claim, we conclude that $?X \in \text{dom}(\mu'_1)$.

Claim 4.4. *Given a dataset D, a well-designed pattern Q, and a variable $?X \in \text{var}(Q)$, if for every pattern $O \trianglelefteq Q$ we have that $?X \in \text{var}(O)$, then $?X \in \text{dom}(\mu)$ for every $\mu \in [\![Q]\!]_D$.*

The proof of this claim can be found in the Appendix. Using a similar argument, we obtain that $?X \in \text{dom}(\mu'_2)$ and, thus, given that μ'_1 is compatible with μ'_2, $\mu'_1 \sqsubseteq \mu_1$ and $\mu'_2 \sqsubseteq \mu_2$, we have that $\mu_1(?X) = \mu'_1(?X) = \mu'_2(?X) = \mu_2(?X)$.

(2) Suppose that $P = (P_1 \text{ OPT } P_2)$. We need to consider two cases. First, assume that $P' \trianglelefteq P_1$. By induction hypothesis, we have that $[\![P']\!]_D \sqsubseteq [\![P_1]\!]_D$. Thus, if $\mu' \in [\![P']\!]_D$, then there exists $\mu_1 \in [\![P_1]\!]_D$ such that $\mu' \sqsubseteq \mu_1$, and we conclude that there exists $\mu \in [\![P]\!]_D$ such that $\mu' \sqsubseteq \mu$ since $[\![P_1]\!]_D \sqsubseteq [\![(P_1 \text{ OPT } P_2)]\!]_D$. Second, assume that $P' = (P'_1 \text{ OPT } P'_2)$ with $P'_1 \trianglelefteq P_1$ and $P'_2 \trianglelefteq P_2$. If $\mu' \in [\![P']\!]_D$, then $\mu' \in [\![(P'_1 \text{ AND } P'_2)]\!]_D$ or $\mu' \in ([\![P'_1]\!]_D \smallsetminus [\![P'_2]\!]_D)$. In the former case, we conclude that there exists $\mu \in [\![P]\!]_D$ such that $\mu' \sqsubseteq \mu$ since from Eq. (1) we know that $[\![(P'_1 \text{ AND } P'_2)]\!]_D \sqsubseteq [\![(P_1 \text{ AND } P_2)]\!]_D$. Furthermore, if $\mu' \in ([\![P'_1]\!]_D \smallsetminus [\![P'_2]\!]_D)$, then $\mu' \in [\![P'_1]\!]_D$. By induction hypothesis, we have that $[\![P'_1]\!]_D \sqsubseteq [\![P_1]\!]_D$ and, hence, there exists $\mu_1 \in [\![P_1]\!]_D$ such that $\mu' \sqsubseteq \mu_1$. We conclude that there exists $\mu \in [\![P]\!]_D$ such that $\mu' \sqsubseteq \mu$ since $[\![P_1]\!]_D \sqsubseteq [\![(P_1 \text{ OPT } P_2)]\!]_D$.

(3) If $P = (P_1 \text{ FILTER } R)$, then $P' = (P'_1 \text{ FILTER } R)$ for $P'_1 \trianglelefteq P_1$. Let $\mu' \in [\![P']\!]_D$. Given that P is well designed, we have that P' is well designed. Recall that a well-designed pattern is safe by definition and, thus, for every $?X \in \text{var}(R)$ and $O \trianglelefteq P'_1$, it is the case that $?X \in \text{var}(O)$. Therefore, by using Claim 4.4, we conclude that $\text{var}(R) \subseteq \text{dom}(\mu')$. By induction hypothesis,

we know that $[\![P'_1]\!]_D \sqsubseteq [\![P_1]\!]_D$ and, hence, there exists $\mu_1 \in [\![P_1]\!]_D$ such that $\mu' \sqsubseteq \mu_1$ (since $[\![P']\!]_D \subseteq [\![P'_1]\!]_D$). Furthermore, given that $\mu' \models R$ and $\mathrm{var}(R) \subseteq \mathrm{dom}(\mu')$, we conclude that $\mu_1 \models R$. Thus, $\mu_1 \in [\![P]\!]_D$ and, therefore, there exists $\mu \in [\![P]\!]_D$ such that $\mu' \sqsubseteq \mu$. This concludes the proof of the lemma. □

We have mentioned that, when evaluating an optional part of a pattern, one is trying to extend mappings with optional information. Another intuition behind the OPT operator is that, when a pattern has several optional parts, one wants to extend the solutions *as much as possible*, that is, one does not want to lose information when the information is present. We formalize this intuition with the notion of *partial solution* for a pattern. Informally, a partial solution for a pattern P is a mapping that is an *exact match* for some P' such that $P' \trianglelefteq P$. We show then, in Proposition 4.5, that the evaluation of a well-designed graph pattern P is exactly the set of *maximal partial solutions* for P with respect to $\sqsubseteq$, that is, the solutions that retrieve as much information as possible. This proposition gives us an alternative characterization of the evaluation of well-designed graph patterns.

Given a pattern P, define $\mathrm{and}(P)$ to be the pattern obtained from P by replacing every OPT operator in P by an AND operator. For example, if $P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)))$, we have that $\mathrm{and}(P) = ((t_1 \text{ AND } t_2) \text{ AND } (t_2 \text{ AND } (t_3 \text{ AND } t_4)))$. Notice that, by the semantics of the OPT operator, for every (not necessarily well designed) pattern P and every dataset D, we have that $[\![\mathrm{and}(P)]\!]_D \subseteq [\![P]\!]_D$.

A mapping μ is a *partial solution* for a pattern P over a dataset D if $\mu \in [\![\mathrm{and}(P')]\!]_D$, for some $P' \trianglelefteq P$. Partial solutions and the notion of subsumption of mappings give us the following characterization of the evaluation of well-designed graph patterns.

PROPOSITION 4.5. *Given a* UNION-*free well-designed graph pattern P, a dataset D, and a mapping μ, we have that $\mu \in [\![P]\!]_D$ if and only if μ is a maximal (with respect to $\sqsubseteq$) partial solution for P over D.*

PROOF. (⇒) The proof goes by induction on P. For the basis case, suppose P is a triple pattern t. Given that $\mathrm{dom}(\mu) = \mathrm{var}(t)$ for every $\mu \in [\![P]\!]_D$, we have that each $\mu \in [\![P]\!]_D$ is a maximal (with respect to $\sqsubseteq$) partial solution for P over D. For the inductive step, we need to consider three cases.

(1) Suppose that $P = (P_1 \text{ AND } P_2)$. If $\mu \in [\![P]\!]_D$, then $\mu = \mu_1 \cup \mu_2$ for $\mu_1 \in [\![P_1]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$. By induction hypothesis, μ_1 and μ_2 are maximal partial solutions for P_1 and P_2, respectively. Thus, μ is a partial solution for $P = (P_1 \text{ AND } P_2)$, and we only need to show that μ is maximal. Suppose that there is a partial solution μ' for P such that $\mu \sqsubseteq \mu'$. We need to show that $\mu = \mu'$ to prove that μ is maximal. Given that μ' is a partial solution for P, we have that $\mu' \in [\![(\mathrm{and}(P'_1) \text{ AND } \mathrm{and}(P'_2))]\!]_D$ for some $P'_1 \trianglelefteq P_1$ and $P'_2 \trianglelefteq P_2$, and $\mu' = \mu'_1 \cup \mu'_2$ for μ'_1 and μ'_2 partial solutions for P_1 and P_2, respectively. By maximality of μ_1 and μ_2, we have that $\mu'_1 \sqsubseteq \mu_1$ and $\mu'_2 \sqsubseteq \mu_2$, and, hence, $\mu' = \mu'_1 \cup \mu'_2 \sqsubseteq \mu_1 \cup \mu_2 = \mu$, which implies that $\mu' = \mu$.

(2) Suppose that $P = (P_1 \text{ OPT } P_2)$. We need to consider two subcases.

(a) Assume that $\mu \in \llbracket (P_1 \text{ AND } P_2) \rrbracket_D$. Then $\mu = \mu_1 \cup \mu_2$ for $\mu_1 \in \llbracket P_1 \rrbracket_D$ and $\mu_2 \in \llbracket P_2 \rrbracket_D$. By induction hypothesis, μ_1 and μ_2 are maximal partial solutions for P_1 and P_2, respectively, implying that μ is a partial solution for P. To prove the maximality of μ, suppose that there is a partial solution μ' for P such that $\mu \sqsubseteq \mu'$. We need to prove that $\mu = \mu'$. Now, we have that either $\mu' \in \llbracket (\text{and}(P_1') \text{ AND } \text{and}(P_2')) \rrbracket_D$ for $P_1' \trianglelefteq P_1$ and $P_2' \trianglelefteq P_2$, or $\mu' \in \llbracket \text{and}(P_3') \rrbracket_D$ for some $P_3' \trianglelefteq P_1$. In the former case, we proceed exactly as in (1) to prove that $\mu' = \mu$. For the latter case, we have that μ' is a partial solution for P_1 and then by the maximality of μ_1 we obtain $\mu' \sqsubseteq \mu_1 \sqsubseteq \mu$, which also implies that $\mu' = \mu$.

(b) Assume that $\mu \in (\llbracket P_1 \rrbracket_D \smallsetminus \llbracket P_2 \rrbracket_D)$. Since $\mu \in \llbracket P_1 \rrbracket_D$, by induction hypothesis we know that μ is a maximal partial solution for P_1. Furthermore, since $P_1 \trianglelefteq P$, we obtain that μ is a partial solution for P. Now, suppose that there is a partial solution μ' for P such that $\mu \sqsubseteq \mu'$. Again we need to show that $\mu = \mu'$ to prove the maximality of μ. We have that, either $\mu' \in \llbracket \text{and}(P_1') \rrbracket_D$ or $\mu' \in \llbracket (\text{and}(P_1') \text{ AND } \text{and}(P_2')) \rrbracket_D$ for some $P_1' \trianglelefteq P_1$ and $P_2' \trianglelefteq P_2$. In the former case, we have that μ' is a partial solution for P_1 and then by maximality of μ we obtain $\mu' \sqsubseteq \mu$. Next we show that the latter case leads to a contradiction. Suppose that $\mu' = \mu_1' \cup \mu_2'$ for μ_1' and μ_2' such that $\mu_1' \in \llbracket \text{and}(P_1') \rrbracket_D$ and $\mu_2' \in \llbracket \text{and}(P_2') \rrbracket_D$. Then $\mu \sqsubseteq \mu_1' \cup \mu_2'$, which implies that μ is compatible with both μ_1' and μ_2'. Given that $\mu_2' \in \llbracket \text{and}(P_2') \rrbracket_D$ and $\llbracket \text{and}(P_2') \rrbracket_D \subseteq \llbracket P_2' \rrbracket_D$ and P_2 is well designed, we have by Lemma 4.3 that there is a mapping $\nu \in \llbracket P_2 \rrbracket_D$ such that $\mu_2' \sqsubseteq \nu$. Furthermore, given that $\mu \in (\llbracket P_1 \rrbracket_D \smallsetminus \llbracket P_2 \rrbracket_D)$, we know that μ is not compatible with any mapping in $\llbracket P_2 \rrbracket_D$ and, in particular, μ is not compatible with ν. This implies that there is a variable $?X \in \text{dom}(\mu) \cap \text{dom}(\nu)$ such that $\mu(?X) \neq \nu(?X)$. Given that μ_2' is compatible with both μ and ν, we have that $?X \notin \text{dom}(\mu_2')$ and, therefore, we have that $?X \notin \text{var}(P_2')$ since $\text{dom}(\mu_2') = \text{var}(P_2')$ (given that $\mu_2' \in \llbracket \text{and}(P_2') \rrbracket_D$). Since $\nu \in \llbracket P_2 \rrbracket_D$, we know that $?X \in \text{var}(P_2)$, and we conclude that there is a subpattern $(O_1 \text{ OPT } O_2)$ of P_2 such that $?X \in \text{var}(O_2)$ and $?X \notin \text{var}(O_1)$. But this contradicts the fact that P is well designed since $?X \in \text{var}(P_1)$ (given that $?X \in \text{dom}(\mu)$ and $\mu \in \llbracket P_1 \rrbracket_D$).

(3) Suppose that $P = (P_1 \text{ FILTER } R)$. If $\mu \in \llbracket P \rrbracket_D$, then $\mu \in \llbracket P_1 \rrbracket_D$ and $\mu \models R$. By induction hypothesis, μ is a maximal partial solution for P_1 and, hence, μ is a partial solution for P. Now suppose that there is a partial solution μ' for P such that $\mu \sqsubseteq \mu'$, again we need to show that $\mu = \mu'$. Now, $\mu' \in \llbracket \text{and}(P_1') \rrbracket_D$ for $P_1' \trianglelefteq P_1$, that is, μ' is a partial solution for P_1. Thus, by maximality of μ as a partial solution for P_1, we conclude that $\mu' \sqsubseteq \mu$.

($\Leftarrow$) If μ is a maximal partial solution for P, then $\mu \in \llbracket \text{and}(P') \rrbracket_D$ for some $P' \trianglelefteq P$. Thus, $\mu \in \llbracket P' \rrbracket_D$ and, hence, by Lemma 4.3 we conclude that there is a mapping $\nu \in \llbracket P \rrbracket_D$ such that $\mu \sqsubseteq \nu$. By the other direction of the proof, we have that ν is a partial solution for P and, hence, by maximality of μ we obtain that $\mu = \nu$, which implies that $\mu \in \llbracket P \rrbracket_D$. This concludes the proof of the proposition. □

Using the characterization of the evaluation of well-designed graph patterns from Proposition 4.5, we can prove the main result of this section.

Theorem 4.6. Evaluation *is coNP-complete for* UNION*-free well-designed graph pattern expressions.*

Proof. For the membership of Evaluation in coNP, consider the problem of determining given a well-designed pattern P, a dataset D, and a mapping μ, whether μ is not in $\llbracket P \rrbracket_D$. To test this condition, we use the negation of the characterization in Proposition 4.5, that is, we use the fact that $\mu \notin \llbracket P \rrbracket_D$ if and only if (1) μ is not a partial solution for P or, (2) μ is a partial solution for P but it is not maximal. We first show that it can be tested in polynomial time whether a mapping is a partial solution for a pattern.

Claim 4.7. *Given a* UNION*-free well-designed pattern P, a dataset D, and a mapping μ, it can be tested in polynomial time whether μ is a partial solution for P over D.*

The proof of the claim can be found in the Appendix.

We now provide a nondeterministic polynomial-time algorithm to test whether $\mu \notin \llbracket P \rrbracket_D$. The algorithm first checks in polynomial time whether μ is not a partial solution for P over D (condition (1)), and if this is the case, it returns *true*. Otherwise, the algorithm guesses a mapping μ', and then checks, in polynomial time, whether μ' is also a partial solution for P over D such that $\mu \sqsubseteq \mu'$ and $\mu' \not\sqsubseteq \mu$ (condition (2)). If this is the case, then the algorithm returns *true* since μ is not a maximal partial solution for P.

To complete the proof, we show that Evaluation is coNP-hard for well-designed graph patterns. We provide here a simple polynomial-time reduction from the complement of the 3-Colorability problem for graphs. An instance of 3-Colorability is an undirected graph $G = (N, E)$, and the problem is to verify whether there exists a function $f : N \to \{0, 1, 2\}$ such that, for every $(u, v) \in E$ it holds that $f(u) \neq f(v)$. If such a function exists for G, then we say that G is 3-colorable. It is well known that 3-Colorability is an NP-complete problem [Garey and Johnson 1979].

In the reduction we use a fixed dataset.

$$D = \{(\mathtt{a}, \mathtt{b}, \mathtt{a}), (0, \mathtt{c}, 1), (0, \mathtt{c}, 2), (1, \mathtt{c}, 0), (1, \mathtt{c}, 2), (2, \mathtt{c}, 0), (2, \mathtt{c}, 1)\}$$

Let $G = (N, E)$ be an undirected graph and assume that $N = \{1, 2, \dots, n\}$ and $E = \{e_1, e_2, \dots, e_m\} \subseteq N \times N$. First, we construct a graph pattern P_G such that $\llbracket P_G \rrbracket_D \neq \emptyset$ if and only if G is 3-colorable. For every $k \in [1, m]$, define a triple pattern t_k as $(?X_i, \mathtt{c}, ?X_j)$ if $e_k = (i, j)$. Now, let $P_G = (t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_m)$. It is straightforward to prove that there is a mapping $\mu \in \llbracket P_G \rrbracket_D$ if and only if G is 3-colorable. Consider now the well-designed graph pattern

$$P = ((?X, \mathtt{b}, ?X) \text{ OPT } P_G).$$

It is easy to see that mapping $\mu = \{?X \to \mathtt{a}\}$ is in$\llbracket P \rrbracket_D$ if and only if$\llbracket P_G \rrbracket_D = \emptyset$.

Therefore, we have that $\mu = \{?X \to \mathtt{a}\}$ is in $[\![P]\!]_D$ if and only if G is not 3-colorable. □

In the hardness part of the previous proof, we use a pattern including only AND and OPT operators. Thus, we also have the following result.

COROLLARY 4.8. EVALUATION *is coNP-complete for well-designed graph pattern expressions constructed by using only* AND *and* OPT *operators.*

The characterization of the evaluation of well-designed graph patterns in Proposition 4.5 can be extended to patterns in UNION normal form. For a well-designed pattern in UNION normal form $P = (P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n)$, a mapping μ, and a dataset D, it holds that $\mu \in [\![P]\!]_D$ if and only if μ is a maximal partial solution (with respect to $\sqsubseteq$) for some P_i $(1 \leq i \leq n)$. Then the evaluation problem for well-designed patterns in UNION normal form is still in coNP: To test whether μ is not in $[\![P]\!]_D$, we only have to check whether μ is not a partial solution for every P_i (condition (1) in the proof of Theorem 4.6), and for every i such that this test fails, we have to guess a mapping μ_i that shows that μ is not a maximal partial solution for P_i (condition (2) in the proof of Theorem 4.6). Thus, we have the following corollary.

COROLLARY 4.9. EVALUATION *is coNP-complete for well-designed graph pattern expressions in* UNION *normal form.*

4.2 Optimization of Well-Designed Patterns

Due to the evident similarity between certain operators of SPARQL and relational algebra, a natural question is whether the classical results of normal forms and optimization for relational algebra are applicable in the SPARQL context. The answer is not straightforward, at least for the case of optional patterns and its relational counterpart, the left-outer join. The classical results about outer-join query reordering and optimization by Galindo-Legaria and Rosenthal [1997] are not directly applicable in the SPARQL context because they assume constraints on the relational queries that are rarely satisfied in SPARQL. The first and most problematic issue is the assumption on predicates used for joining/outer-joining relations to be *null-rejecting* [Galindo-Legaria and Rosenthal 1997]. A predicate p is null-rejecting if it evaluates to *false* (or *undefined*) whenever a null value is used in p. In SPARQL, those predicates are implicit in the variables that graph patterns share and, by the definition of compatible mappings, they are never *null-rejecting*. In fact, people who have developed algorithms for translating SPARQL queries into relational algebra and SQL queries (e.g. Cyganiak [2005]) have used `NULL` to represent unbound variables, `IS NULL` in predicates for joining/outer-joining, and `COALESCE` for merging the values of different columns into a single column. These features are explicitly prohibited in Galindo-Legaria and Rosenthal [1997] since they may imply a violation of the null-rejecting requirement (see the related work for further discussion on these topics). Since the application of classical results is not straightforward, it would be desirable to develop specific techniques in the SPARQL context. In what follows, we show that the property of being well designed has important consequences for the study of normalization and optimization for SPARQL.

PROPOSITION 4.10. *Let P_1, P_2, and P_3 be graph pattern expressions and R a built-in condition. Consider the rewriting rules.*

$$((P_1 \text{ OPT } P_2) \text{ FILTER } R) \longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2) \tag{7}$$
$$(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) \longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) \tag{8}$$
$$((P_1 \text{ OPT } P_2) \text{ AND } P_3) \longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) \tag{9}$$

Let P be a UNION*-free well-designed pattern, and assume that P' is a pattern obtained from P by applying either rule* (7)*, or rule* (8)*, or rule* (9)*. Then P' is a* UNION*-free well-designed pattern equivalent to P.*

PROOF. It is straightforward to prove that the application of rules (7), (8), and (9) preserve the property of being well designed. Thus, we only show here that the application of these rules does not affect the evaluation of the pattern.

Rule (7). We show that $[\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D = [\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D$ for every dataset D. We show first that $[\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D \subseteq [\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D$. Let μ be a mapping in $[\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D$. Then $\mu \in [\![(P_1 \text{ OPT } P_2)]\!]_D$ and $\mu \models R$. We need to consider two cases: (a) $\mu = \mu_1 \cup \mu_2$ for compatible mappings $\mu_1 \in [\![P_1]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$, and (b) $\mu \in [\![P_1]\!]_D$ and μ is not compatible with any mapping $\mu_2 \in [\![P_2]\!]_D$. Consider first case (b). We have that $\mu \in [\![P_1]\!]_D$, $\mu \models R$, and μ is not compatible with any mapping $\mu_2 \in [\![P_2]\!]_D$, which imply that $\mu \in [\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D$. Consider now case (a). We know that $\mu_1 \cup \mu_2 \models R$ (since $\mu = \mu_1 \cup \mu_2$ and $\mu \models R$). Next we show that also $\mu_1 \models R$, and then $\mu = \mu_1 \cup \mu_2 \in [\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D$. Given that $((P_1 \text{ OPT } P_2) \text{ FILTER } R)$ is well designed, if a variable $?X$ occurs in R and in P_2, then $?X$ also occurs in P_1. Moreover, $?X$ occurs in every pattern $O \trianglelefteq P_1$, and then, by Claim 4.4, it holds that $?X \in \text{dom}(\mu_1)$ for every $\mu_1 \in [\![P_1]\!]_D$. We conclude that, for every $?X$ such that $?X \in \text{var}(R)$ and $?X \in \text{dom}(\mu_2)$, it holds that $?X \in \text{dom}(\mu_1)$. Therefore, from $\mu_1 \cup \mu_2 \models R$, we obtain that $\mu_1 \models R$.

We show now that $[\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D \subseteq [\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D$. Let μ be a mapping in $[\![((P_1 \text{ FILTER } R) \text{ OPT } P_2)]\!]_D$. We need to consider two cases: (a) $\mu = \mu_1 \cup \mu_2$ for compatible mappings $\mu_1 \in [\![(P_1 \text{ FILTER } R)]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$, and (b) $\mu \in [\![(P_1 \text{ FILTER } R)]\!]_D$ and μ is not compatible with any mapping $\mu_2 \in [\![P_2]\!]_D$. Consider first case (b). We have that $\mu \in [\![P_1]\!]_D$, μ is not compatible with any mapping $\mu_2 \in [\![P_2]\!]_D$ and $\mu \models R$, which imply that $\mu \in [\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D$. Consider now case (a). We know that $\mu_1 \in [\![P_1]\!]_D$ and $\mu_1 \models R$. Next we show that it is also the case that $\mu \models R$, and, thus, $\mu \in [\![((P_1 \text{ OPT } P_2) \text{ FILTER } R)]\!]_D$. As in the previous part of the proof, using the fact that the pattern $((P_1 \text{ OPT } P_2) \text{ FILTER } R)$ is well designed and Claim 4.4, it can be shown that, for every variable $?X$ such that $?X \in \text{var}(R)$ and $?X \in \text{dom}(\mu_2)$, it holds that $?X \in \text{dom}(\mu_1)$. Then given that μ_1 and μ_2 are compatible and $\mu_1 \models R$, we conclude that $\mu_1 \cup \mu_2 \models R$ and, thus, $\mu \models R$.

Rule (8) We show that $[\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D = [\![((P_1 \text{ AND } P_2) \text{ OPT } P_3)]\!]_D$ for every dataset D. We show first that $[\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D \subseteq [\![((P_1 \text{ AND } P_2) \text{ OPT } P_3)]\!]_D$. Let μ be a mapping in $[\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D$. Then, $\mu = \mu_1 \cup \mu'$ for compatible mappings $\mu_1 \in [\![P_1]\!]_D$ and $\mu' \in [\![(P_2 \text{ OPT } P_3)]\!]_D$. We need to consider two cases: (a) $\mu' \in [\![(P_2 \text{ AND } P_3)]\!]_D$, and (b) $\mu' \in [\![P_2]\!]_D$ and μ' is not compatible with any mapping $\mu_3 \in [\![P_3]\!]_D$. In case (a), we obtain that $\mu = \mu_1 \cup \mu' \in [\![(P_1 \text{ AND } (P_2 \text{ AND } P_3))]\!]_D$, and, therefore, $\mu \in [\![((P_1 \text{ AND } P_2) \text{ OPT } P_3)]\!]_D$. In case (b), given that $\mu' \in [\![P_2]\!]_D$, we obtain that $\mu = \mu_1 \cup \mu' \in [\![(P_1 \text{ AND } P_2)]\!]_D$. Furthermore, given that μ' is not compatible with any mapping $\mu_3 \in [\![P_3]\!]_D$, we have that μ is not compatible with any mapping $\mu_3 \in [\![P_3]\!]_D$, and, thus, $\mu \in [\![((P_1 \text{ AND } P_2) \text{ OPT } P_3)]\!]_D$.

We show now that $[\![((P_1 \text{ AND } P_2) \text{ OPT } P_3)]\!]_D \subseteq [\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D$. Let μ be a mapping in $[\![(P_1 \text{ AND } P_2) \text{ OPT } P_3))]\!]_D$. We need to consider two cases: (a) $\mu \in [\![((P_1 \text{ AND } P_2) \text{ AND } P_3)]\!]_D$, and (b) $\mu \in [\![(P_1 \text{ AND } P_2)]\!]_D$ and for every $\mu_3 \in [\![P_3]\!]_D$, μ is not compatible with μ_3. In case (a), we obtain that $\mu \in [\![(P_1 \text{ AND } (P_2 \text{ AND } P_3))]\!]_D$ and, thus, $\mu \in [\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D$. In case (b), we have that $\mu = \mu_1 \cup \mu_2$ for compatible mappings $\mu_1 \in [\![P_1]\!]_D$ and $\mu_2 \in [\![P_2]\!]_D$, and $\mu = \mu_1 \cup \mu_2$ is not compatible with any mapping $\mu_3 \in [\![P_3]\!]_D$. Next we show that μ_2 is not compatible with any mapping $\mu_3 \in [\![P_3]\!]_D$, which implies that $\mu_2 \in [\![(P_2 \text{ OPT } P_3)]\!]_D$ and, therefore, $\mu = \mu_1 \cup \mu_2 \in [\![(P_1 \text{ AND } (P_2 \text{ OPT } P_3))]\!]_D$. Let $\mu_3 \in [\![P_3]\!]_D$. Given that $\mu = \mu_1 \cup \mu_2$ is not compatible with μ_3, we have that either μ_2 is not compatible with μ_3, or μ_1 is not compatible with μ_3. If μ_2 is not compatible with μ_3, then there is nothing to prove. Thus, assume that μ_1 is not compatible with μ_3. Then there exists a variable $?X$ such that $?X \in \text{dom}(\mu_1)$, $?X \in \text{dom}(\mu_3)$, and $\mu_1(?X) \neq \mu_3(?X)$. This last statement implies that $?X \in \text{var}(P_1)$ and $?X \in \text{var}(P_3)$. Given that pattern $(P_1 \text{ AND } (P_2 \text{ OPT } P_3))$ is well designed, and given that $?X \in \text{var}(P_1)$ and $?X \in \text{var}(P_3)$, we obtain that $?X \in \text{var}(P_2)$. Furthermore, we know that for every pattern $O \trianglelefteq P_2$, it is the case that $?X \in \text{var}(O)$, otherwise we would obtain a contradiction with the fact that $(P_1 \text{ AND } (P_2 \text{ OPT } P_3))$ is well designed. Hence, by applying Claim 4.4, we conclude that $?X \in \text{dom}(\mu_2)$. Thus, given that $?X \in \text{dom}(\mu_1)$, μ_1 and μ_2 are compatible mappings and $\mu_1(?X) \neq \mu_3(?X)$, we obtain that μ_2 is not compatible with μ_3. That was to be shown.

Rule (9) Follows from rule (8) and the commutativity of the AND operator. □

It is worth mentioning that the previous rules are not applicable to nonwell-designed graph patterns. For example, consider dataset $D = \{(1, \mathsf{a}, 1), (2, \mathsf{a}, 2), (3, \mathsf{a}, 3)\}$ and nonwell-designed pattern

$$P = ((?X, \mathsf{a}, 1) \text{ AND } ((?Y, \mathsf{a}, 2) \text{ OPT } (?X, \mathsf{a}, 3))).$$

The evaluation of P results in the empty set of mappings. If we apply rule (8) to P, we obtain pattern $P' = (((?X, \mathsf{a}, 1) \text{ AND } (?Y, \mathsf{a}, 2)) \text{ OPT } (?X, \mathsf{a}, 3))$. The evaluation of P' results in the set $\{\{?X \to 1, ?Y \to 2\}\}$ and, thus, we have that $[\![P]\!]_D \neq [\![P']\!]_D$.

We say that a UNION-free graph pattern P is in OPT *normal form* if either: (1) P is constructed by using only the AND and FILTER operators, or (2) $P = (O_1 \text{ OPT } O_2)$, with O_1 and O_2 patterns in OPT normal form. For example, consider a pattern:

$$P = ((((t_1 \text{ AND } t_2) \text{ FILTER } R_1) \text{ OPT } (t_3 \text{ OPT } ((t_4 \text{ FILTER } R_2) \text{ AND } t_5))) \\ \text{OPT } (t_6 \text{ FILTER } R_3)),$$

where every t_i is a triple pattern, and every R_j is a built-in condition. Then P is in OPT normal form. The following theorem shows that for every well-designed graph pattern, an equivalent pattern in OPT normal form can be efficiently obtained.

THEOREM 4.11. *For every* UNION*-free well-designed pattern P, an equivalent pattern in* OPT *normal form can be obtained after $O(|P|^2)$ applications of rules* (7) through (9).

PROOF. Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be an enumeration of the subpatterns of P of the form either (P_1 AND P_2), or (P_1 FILTER R), or (P_1 OPT P_2). For instance, let P be pattern

$$((((t_1 \text{ OPT } t_2) \text{ AND } (t_3 \text{ AND } (t_4 \text{ AND } (t_1 \text{ OPT } t_2))))) \text{ FILTER } R),$$

where t_1, t_2, t_3, t_4 are triple patterns and R is a built-in condition. Then the following sequence is an enumeration of the subpatterns of P of the form mentioned before: $\alpha_1 = (t_1 \text{ OPT } t_2)$, $\alpha_2 = (t_1 \text{ OPT } t_2)$, $\alpha_3 = (t_4 \text{ AND } (t_1 \text{ OPT } t_2))$, $\alpha_4 = (t_3 \text{ AND } (t_4 \text{ AND } (t_1 \text{ OPT } t_2)))$, $\alpha_5 = (((t_1 \text{ OPT } t_2) \text{ AND } (t_3 \text{ AND } (t_4 \text{ AND } (t_1 \text{ OPT } t_2)))))$, and $\alpha_6 = P$. Consider, for every subpattern α of the form (O_1 OPT O_2), a measure $d(\alpha)$ defined as follows: $d(\alpha)$ is the number of patterns β of the form either (Q_1 AND Q_2), or (Q_1 FILTER R), such that α is a subpattern of β in P. In other words, $d(\alpha)$ represents the number of AND and FILTER operators in a traversal of the parse tree of P from α to the root. In the preceding example, $d(\alpha_1) = 2$ and $d(\alpha_2) = 4$.

For the entire pattern P, define measure $D(P)$ as the sum of the measures $d(\alpha)$ for every OPT subpattern α of P. In the example previous, we have that $D(P) = 6$. By a simple induction argument, it can be shown that P is in OPT normal form if and only if $D(P) = 0$. Also notice that $d(\alpha)$ is linear in the size of P, and then, measure $D(P)$ is quadratic in the size of P.

Now, the application of every one of the rules (7) through (9) to a pattern P decreases $D(P)$. Specifically, if P' is obtained from P by applying one of the rules (7) through (9), then $D(P') < D(P)$. From this last argument plus the fact that $D(P)$ is quadratic in the size of P, we conclude that a pattern $P^\star$ such that $P \equiv P^\star$ and $D(P^\star) = 0$, can be obtained from P after $O(|P|^2)$ applications of rules (7) through (9). This concludes the proof of the theorem. □

The application of rules (7) through (9) may have a considerable impact in the cost of evaluating graph patterns. One can measure this impact by analyzing the intermediate sizes of the sets of mappings produced when evaluating a pattern. By the semantics of the OPT operator, when evaluating an expression of the form (P_1 OPT P_2) over a dataset D, the number of mappings obtained is at least the number of mappings obtained when evaluating P_1 over D. In other words, the application of the OPT operator never implies a reduction in the size of the intermediate results in the evaluation of a graph pattern expression. In contrast, it is clear that operators AND and FILTER may imply a reduction in the size of intermediate results. Thus, for optimization purposes, it would be convenient to perform all the AND and FILTER operations first, delaying the OPT operations to the last step of the evaluation. A pattern in OPT normal form has its operators ordered in a way that the bottom-up evaluation of the pattern follows exactly this strategy: All AND and FILTER operations are executed prior to the execution of the OPT operations.

5. RELATED WORK

As we mentioned before, this article is an extended and revised version of Pérez et al. [2006a]. Our first goal in Pérez et al. [2006a] was to introduce a formal semantics for SPARQL. At the time when Pérez et al. [2006a] was published, there were two main proposals for the semantics of graph pattern expressions. The first was an operational semantics, consisting essentially in the execution of a depth-first traversal of parse trees of graph patterns, and the use of the intermediate results to avoid some computations. At that time, this approach was followed by ARQ [ARQ 2006] (a language developed by HPLabs), and by the W3C when evaluating graph pattern expressions containing nested optionals [Seaborne 2006]. For instance, the computation of the mappings satisfying (A OPT (B OPT C)) was done by first computing the mappings that match A, then checking which of these mappings match B, and for those who match B checking whether they also match C [Seaborne 2006]. The second approach, compositional in spirit and the one advocated in Pérez et al. [2006a], extended classical conjunctive query evaluation [Gutierrez et al. 2004], and was based on a bottom-up evaluation of parse trees of graph pattern expressions, borrowing notions of relational algebra evaluation [Cyganiak 2005; Harris and Shadbolt 2005] plus some additional features.

Our second goal in Pérez et al. [2006a] was to study some fundamental properties of the compositional semantics proposed in that paper. In particular, we provided in Pérez et al. [2006a] some complexity bounds for the evaluation problem for graph pattern expressions, and we compared our compositional semantics with the depth-first traversal semantics mentioned earlier. In fact, the notion of well-designed pattern was proposed in Pérez et al. [2006a] as a simple condition under which the semantics mentioned before coincide.

Currently, the official specification of SPARQL [Prud'hommeaux and Seaborne 2008], endorsed by the W3C, formalizes a semantics based on our work [Pérez et al. 2006a, 2006b; Arenas et al. 2007]. Thus, the motivation of this article is different from the motivation of Pérez et al. [2006a]. In the first place, a more thorough analysis of the complexity of the evaluation problem for graph pattern expressions is given in this article. Special attention is paid to the proofs of the complexity lower bounds, which were not provided in Pérez et al. [2006a]. These proofs help in identifying cases where graph pattern expressions are difficult to evaluate. In the second place, since graph patterns tend to satisfy the well-designed condition introduced in Pérez et al. [2006a], in this article we study the complexity of the evaluation problem for this class of graph pattern expressions. In particular, we prove that the evaluation problem for well-designed graph pattern expressions is coNP-complete. It should be noticed that no complexity bounds for this problem were provided in Pérez et al. [2006a]. The coNP upper bound shows that the notion of well-designed pattern is a promising direction for further research, as these expressions can be evaluated more efficiently than general graph patterns (assuming that coNP $\neq$ PSPACE). In fact, in this article we also show that well-designed patterns are suitable for reordering and optimization, and provide several rewriting rules whose application may have a considerable impact in the cost of evaluating SPARQL queries.

In the rest of this section, we continue with the review of the related work. In Section 5.1, we discuss previous work on the semantics of SPARQL. In Section 5.2, we consider other proposals for query languages for RDF. Finally, in Section 5.3, we consider the related work on complexity and optimization of classical query languages.

5.1 Semantics of SPARQL

Cyganiak [2005] presents a relational model of SPARQL. The author uses modified versions of the standard relational algebra operators (join, left-outer join, projection, selection, etc.) to model SPARQL `SELECT` clauses. The central idea in Cyganiak [2005] is to make a correspondence between SPARQL queries and relational algebra queries over a single relation *Triple*(*subject*, *predicate*, *object*), that stores RDF graphs in the form of triples. The author discusses some drawbacks of using classical relational algebra operators to define the semantics of SPARQL and identifies cases in which his formalization does not match the SPARQL official specification. Additionally, a translation system between SPARQL and SQL is also outlined in Cyganiak [2005]. The system extensively use `COALESCE` and `IS NULL`/`IS NOT NULL` operators to accurately resemble some SPARQL features. With different motivations, but similar philosophy, Harris and Shadbolt [2005] present an implementation of a simple fragment of SPARQL in a relational database engine. They use relational algebra operators similar to the ones used in Cyganiak [2005].

As noted in Cyganiak [2005], the treatment of null values is the major problem encountered when trying to specify the semantics of SPARQL using standard relational algebra. Since mappings must be modeled as relational tuples, null values need to be used to model unbounded variables in graph pattern evaluation. Zaniolo introduces in Zaniolo [1984] a formal algebra to deal with null values in relational databases. The author interprets null values as standing for "no information," as opposed with the more complex "unknown" and "nonexistent" interpretations [Imielinski and Lipski 1984]. In Zaniolo [1984], a *relation with null values* is defined as a set of tuples of not necessarily the same arity that possibly contain null values in some of its components. The author then defines operations over those relations with nulls that generalize the standard relational algebra operations. The treatment of null values in Zaniolo [1984] matches the treatment of unbounded variables in SPARQL. Thus, the operators over sets of mappings introduced in Section 2 can be easily modeled within the framework of Zaniolo [1984]. Although the formalization in Zaniolo

[1984] can be used to define the SPARQL semantics, we follow a simplified approach formalizing only what is strictly necessary in the SPARQL context, and thus simplifying the subsequent study of the language.

de Bruijn et al. [2005] study the semantics of the conjunctive fragment of SPARQL (graph patterns using only the AND operator, plus SELECT clause) from a logical point of view. It slightly differs from the definition in Prud'hommeaux and Seaborne [2008] on the issue of blank nodes. Although de Bruin et al.'s definition allows blank nodes in graph patterns, it is similar to our definition (where blanks nodes are not allowed in patterns). In their approach, blanks play the role of "nondistinguished" variables, that is, variables that are not presented in the answer.

In Polleres [2007], Polleres studies the problem of translating SPARQL queries into Datalog queries. Based on our previous work [Pérez et al. 2006a], the author proposes three different semantics: (1) bravely-joining, (2) cautiously-joining, and (3) strictly-joining semantics. These semantics are obtained by strengthening the notion of compatible mappings, and thus strengthening the conditions under which unbound variables are joined. Strictly-joining semantics essentially resembles the inner-join condition of SQL, allowing a simple translation into Datalog. Bravely-joining semantics coincide with the semantics presented in Section 2. To translate bravely-joining semantics into Datalog, a special predicate $BOUND(\cdot)$ is needed to test whether a variable is bounded to a nonnull value. As a result, the translation generates a program with disjunctions in the bodies of the rules that extensively uses $\neg BOUND(\cdot)$. The program is then transformed into Datalog by using standard techniques [Polleres 2007].

5.2 Semantics of RDF Query Languages

There are several works on the semantics of RDF query languages which tangentially touch the issues addressed in the definition of SPARQL. Gutierrez et al. [2004] discuss the basic issues of the semantics and complexity of a conjunctive query language for RDF, which underlies the basic evaluation approach of SPARQL. Haase et al. [2004] present a comparison of functionalities of pre-SPARQL query languages, many of which served as inspiration for the constructs of SPARQL. Another in-depth comparison of RDF query languages (including SPARQL), in terms of their constructs and evaluation methods, can be found in Furche et al. [2006]. Nevertheless, neither Haase et al. [2004] nor Furche et al. [2006] presents formal semantics for the languages compared.

The idea of having an algebraic query language for RDF is not new. In fact, there are several proposals. Chen et al. [2005] present a set of operators for manipulating RDF graphs, Frasincar et al. [2004] study algebraic operators on the lines of the RQL query language [Karvounarakis et al. 2002], and Robertson [2004] introduces an algebra of triadic relations for RDF. Although these authors show the power of having an algebraic approach for querying RDF, it is not clear how to model SPARQL using the frameworks introduced by them.

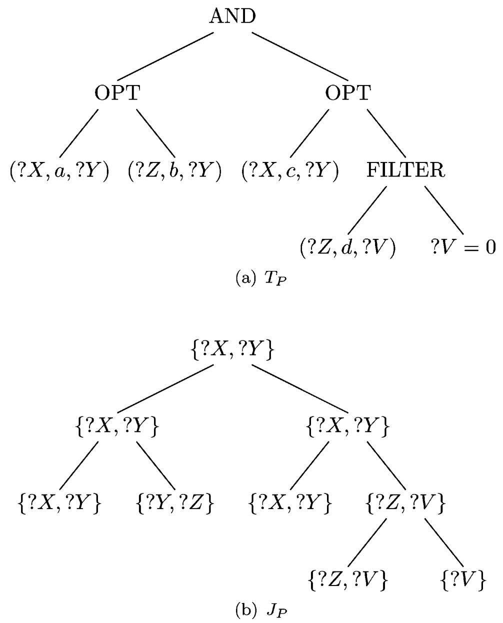
Finally, Serfiotis et al. [2005] study RDFS query fragments using a logical framework, presenting results on the classical database problems of containment and minimization of queries for a model of RDFS. They concentrate on patterns using the RDFS vocabulary of classes and properties in conjunctive queries, making the overlap with our fragment and approach almost empty.

5.3 Complexity and Optimization

A natural question is whether some of the complexity lower bounds of Section 3 can be obtained by first mapping a known query language $\mathcal{Q}$ into SPARQL graph patterns, and then using complexity lower bounds for the evaluation of $\mathcal{Q}$-queries. The first natural candidate is relational algebra/SQL. To the best of our knowledge, there is no complexity study of the fragment of relational algebra/SQL including outer-join, outer-union, and `IS NULL` (`IS NOT NULL`) in join, outer-join, and selection predicates. Thus, we cannot directly obtain complexity bounds from relational algebra. Although the complexity of querying incomplete databases has been extensively studied for different settings and interpretations of null values (refer to Abiteboul et al. [1991]), to the best of our knowledge, there is no complexity study for the fragment of the relational algebra with null values introduced in Zaniolo [1984] that corresponds to the fragment of SPARQL studied in this article. Another natural question is whether one can obtain the PSPACE-hard lower bounds of Theorems 3.3 and 3.4 directly from the lower bound for the evaluation of first-order logic queries. The answer again is negative since, in the fragment of SPARQL we consider, we do not allow *projection* and then existential quantification cannot be simulated. On the contrary, the complexity bounds for SPARQL presented in Section 3 can be used to derive results for extensions (or fragments) of known query languages (e.g., relational algebra/SQL plus outer-join and outer-union, and the relational algebra with null values presented in Zaniolo [1984]).

We now discuss previous work related with well-designed graph patterns. The notion of *join-tree* [Beeri et al. 1983] has been widely used for studying optimization of conjunctive queries. Let $T = (N, E)$ be a tree such that every node $a \in N$ is labeled with set of variables $V(a)$. Tree T is a join-tree if and only if, for every path $(a_1, a_2, \ldots, a_n)$ in T, and every $?X \in V(a_1) \cap V(a_n)$, it holds that $?X \in V(a_i)$ for every $i \in \{1, \ldots, n\}$. It turns out that the join-tree condition can be used to test whether a SPARQL graph pattern is well designed. Consider the parse tree T_P of a UNION-free graph pattern P. For example, Figure 1(a) shows the parse tree of $P = (((?X, a, ?Y)\ \text{OPT}\ (?Y, b, ?Z))\ \text{AND}\ ((?X, c, ?Y)\ \text{OPT}\ ((?Z, d, ?V)\ \text{FILTER}\ ?V = 0)))$. Now, consider tree J_P constructed by recursively labeling the nodes of T_P as follows. For a subpattern P' of P, its label $V(P')$ is:

(1) var(P') if P' is either a triple pattern t or a built-in condition R;
(2) $V(P_1) \cup V(P_2)$ if $P' = (P_1\ \text{AND}\ P_2)$;
(3) $V(P_1) \cup V(R)$ if $P' = (P_1\ \text{FILTER}\ R)$;
(4) $V(P_1)$ if $P' = (P_1\ \text{OPT}\ P_2)$.



(a) T_P

(b) J_P

Fig. 1. Trees T_P and J_P for pattern $P = (((?X, a, ?Y)$ OPT $(?Y, b, ?Z))$ AND $((?X, c, ?Y)$ OPT $((?Z, d, ?V)$ FILTER $?V = 0)))$. Tree J_P is not a join-tree and, hence, P is not well designed.

It is straightforward to show that a pattern P is well designed if and only if P is safe and its associated labeled tree J_P is a join-tree. For example, Figure 1(b) shows the labeled tree J_P for the pattern P of Figure 1(a). J_P is not a join-tree since both paths from the node with label $\{?Y, ?Z\}$ to the nodes with label $\{?Z, ?V\}$, violate the condition for join trees. Therefore, P is not well designed. The well-designed condition for graph patterns was initially motivated by a variation of this join-tree condition. We decide to keep Definition 4.1 for simplicity.

Although the notions of well-designed pattern and join-tree are related, the optimization results for the evaluation of relational join queries based on join-trees cannot be used for general well-designed patterns. The optimization of relational join queries via join-trees rests in the fact that joins are associative and commutative, and thus queries can be reordered in several ways without affecting the output. These properties are violated by well-designed patterns since these patterns are constructed by using AND, OPT, and FILTER operators, and, for example, OPT is neither associative nor commutative.

There is an important amount of work on the problem of optimizing the evaluation of conjunctive queries [Chandra and Merlin 1977; Yannakakis 1981; Beeri et al. 1983; Gottlob et al. 2001] (see also Abiteboul et al. [1995]). All these results can be almost directly used to optimize the evaluation of the conjunctive fragment of SPARQL graph patterns, that is, patterns constructed by using only the AND operator. As we have mentioned, the OPT operator can be simulated by using the relational algebra/SQL outer-join operator plus some additional features to deal with null values. What is not straightforward in this case is the application of classical outer-join optimization results to OPT graph patterns. The classical results about outer-join query reordering and optimization by Galindo-Legaria and Rosenthal [1997] are not directly applicable in the SPARQL context. In Galindo-Legaria and Rosenthal [1997], two restrictions are imposed on queries containing outer-joins: (a) all predicates used for joining and outer-joining relations must be *null-rejecting*, (b) predicates used in outer-joins must mention only two relations. A predicate p is null-rejecting if it evaluates to *false* (or *undefined*) whenever a null value is used in p. For example, a predicate like `A = B OR A IS NULL`, used in the `WHERE` clause of an SQL query, is not null-rejecting since it evaluates to *true* if `A` is a null value. Another type of queries that violate the null-rejecting condition are “queries that coalesce values from multiple sources into a single column and then compute predicates on the coalesced column” [Galindo-Legaria and Rosenthal 1997]. Both `IS NULL` and `COALESCE` are explicitly prohibited in Galindo-Legaria and Rosenthal [1997], although both features are needed to accurately map SPARQL into relational algebra/SQL. Therefore, in general, SPARQL queries violate restriction (a) just given. Regarding restriction (b), the predicates used for outer-joining graph patterns in SPARQL are implicit in the variables that the patterns share, and then several parts of a graph pattern could be mentioned when performing an outer-join. In Bhargava et al. [1995], Braghava et al. relax restriction (b) by using hypergraphs to study the reorderability of outer-join queries. However, the restriction of predicates being null-rejecting is still imposed in Bhargava et al. [1995]. Then a possible way to optimize SPARQL queries is to map SPARQL graph patterns into relational algebra/SQL queries, check whether some of the `IS NULL` and `COALESCE` operations generated are not necessary or do not violate the null-rejection condition, and then apply the optimization rules in Galindo-Legaria and Rosenthal [1997] and Bhargava et al. [1995]. In our work, we follow an alternative approach by developing specific rules for reordering and optimizing the fragment of well-designed SPARQL graph patterns.

6. CONCLUSIONS

The query language SPARQL has been in the process of standardization since 2004. In this process, the semantics of the language has played a key role. A formalization of a semantics is beneficial on several grounds: help identify relationships among the constructors that stay hidden in the use cases, identify redundant and contradicting notions, study the expressiveness and complexity of the language, help in optimization, etc.

In this article, we provide such a formal semantics for the graph pattern-matching facility, which is the core of SPARQL. We isolate a fragment which is rich enough to present the main issues and favor a good formalization. Furthermore, we study several properties of SPARQL. We present a thorough analysis of the complexity of the evaluation of graph patterns showing, among other results, that unlimited use of optional parts in graph patterns could lead to high complexity, namely PSPACE. In this respect, we introduce the class of well-designed graph patterns, where the query evaluation problem can be solved more efficiently. This class is defined by imposing a simple and natural syntactic restriction on optional parts. We also prove that well-designed patterns are suitable for reordering and optimization, and provide several rewriting rules whose application may have a considerable impact in the cost of evaluating SPARQL queries.

APPENDIX

A. PROOFS

A.1 Proof of Lemma 3.7

(1) To prove that $((P_1 \text{ UNION } P_2) \text{ OPT } P_3)) \equiv ((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))$, we consider two cases. First, we show that for every RDF dataset D, we have that $[\![((P_1 \text{ UNION } P_2) \text{ OPT } P_3)]\!]_D \subseteq [\![((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))]\!]_D$. Let D be an RDF dataset and assume that $\mu \in [\![((P_1 \text{ UNION } P_2) \text{ OPT } P_3)]\!]_D$. Then either (a) there exist $\mu_1 \in [\![(P_1 \text{ UNION } P_2)]\!]_D$ and $\mu_2 \in [\![P_3]\!]_D$ such that μ_1 and μ_2 are compatible and $\mu = \mu_1 \cup \mu_2$, or (b) $\mu \in [\![(P_1 \text{ UNION } P_2)]\!]_D$ and there is no $\mu_3 \in [\![P_3]\!]_D$ such that μ and μ_3 are compatible. In case (a), if $\mu_1 \in [\![P_1]\!]_D$, then $\mu = \mu_1 \cup \mu_2 \in [\![(P_1 \text{ OPT } P_3)]\!]_D$. In case (a), if $\mu_1 \in [\![P_2]\!]_D$, then $\mu = \mu_1 \cup \mu_2 \in [\![(P_2 \text{ OPT } P_3)]\!]_D$. In case (b), if $\mu \in [\![P_1]\!]_D$, then $\mu \in [\![(P_1 \text{ OPT } P_3)]\!]_D$ since μ is not compatible with any $\mu_3 \in [\![P_3]\!]_D$. In case (b), if $\mu \in [\![P_2]\!]_D$, then $\mu \in [\![(P_2 \text{ OPT } P_3)]\!]_D$ since μ is not compatible with any $\mu_3 \in [\![P_3]\!]_D$. In any of the previous cases, we conclude that $\mu \in [\![((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))]\!]_D$. Second, we show that for every RDF dataset D, we have that $[\![((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))]\!]_D \subseteq [\![((P_1 \text{ UNION } P_2) \text{ OPT } P_3))]\!]_D$. Let D be an RDF dataset and assume that $\mu \in [\![((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))]\!]_D$. Without loss of generality, we assume that $\mu \in [\![(P_1 \text{ OPT } P_3)]\!]_D$. Then either (a) there exists $\mu_1 \in [\![P_1]\!]_D$ and $\mu_2 \in [\![P_3]\!]_D$ such that μ_1 and μ_2 are compatible and $\mu = \mu_1 \cup \mu_2$, or (b) $\mu \in [\![P_1]\!]_D$ and there is no $\mu_3 \in [\![P_3]\!]_D$ such that μ and μ_3 are compatible. In case (a), we have that $\mu_1 \in [\![(P_1 \text{ UNION } P_2)]\!]_D$ and, hence, $\mu = \mu_1 \cup \mu_2 \in [\![((P_1 \text{ UNION } P_2) \text{ OPT } P_3)]\!]_D$. In case (b), we have that $\mu \in [\![(P_1 \text{ UNION } P_2)]\!]_D$ and, therefore, $\mu \in [\![((P_1 \text{ UNION } P_2) \text{ OPT } P_3)]\!]_D$ since μ is not compatible with any $\mu_3 \in [\![P_3]\!]_D$. This concludes the proof of the equivalence of $((P_1 \text{ UNION } P_2) \text{ OPT } P_3)$ and $((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))$.

(2) Clearly, for every RDF dataset D and built-in condition R, we have that $\{\mu \in [\![P_1]\!]_D \mid \mu \models R\} \subseteq \{\mu \in [\![(P_1 \text{ UNION } P_2)]\!]_D \mid \mu \models R\}$ and $\{\mu \in [\![P_2]\!]_D \mid \mu \models R\} \subseteq \{\mu \in [\![(P_1 \text{ UNION } P_2)]\!]_D \mid \mu \models R\}$ since $[\![P_1]\!]_D \subseteq [\![(P_1 \text{ UNION } P_2)]\!]_D$ and $[\![P_2]\!]_D \subseteq [\![(P_1 \text{ UNION } P_2)]\!]_D$. Thus, we only need to show that for every RDF dataset D and built-in condition R, it is the case that $[\![((P_1 \text{ UNION } P_2) \text{ FILTER } R)]\!]_D \subseteq [\![((P_1 \text{ FILTER } R) \text{ UNION } (P_2 \text{ FILTER } R))]\!]_D$. Assume that $\mu \in$

$[\![((P_1 \text{ UNION } P_2) \text{ FILTER } R)]\!]_D$. Then $\mu \in [\![(P_1 \text{ UNION } P_2)]\!]_D$ and $\mu \models R$. Thus, if $\mu \in [\![P_1]\!]_D$, then $\mu \in [\![(P_1 \text{ FILTER } R)]\!]_D$, and if $\mu \in [\![P_2]\!]_D$, then $\mu \in [\![(P_2 \text{ FILTER } R)]\!]_D$. Therefore, we conclude that $\mu \in [\![((P_1 \text{ FILTER } R) \text{ UNION } (P_2 \text{ FILTER } R))]\!]_D$.

A.2 Proof of Claim 4.4

The proof is by induction on the structure of pattern Q. If Q is a triple pattern, then the property trivially holds. For the inductive step, first suppose that $Q = (Q_1 \text{ AND } Q_2)$. Since $?X \in \text{var}(Q')$ for every $Q' \trianglelefteq Q$, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \trianglelefteq Q_1$, or $?X \in \text{var}(Q'_2)$ for every $Q'_2 \trianglelefteq Q_2$. Assume, without loss of generality, that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \trianglelefteq Q_1$. Then by induction hypothesis, we have that $?X \in \text{dom}(\mu_1)$ for every $\mu_1 \in [\![Q_1]\!]_D$, from which we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in [\![(Q_1 \text{ AND } Q_2)]\!]_D$. Second, suppose that $Q = (Q_1 \text{ OPT } Q_2)$. Since $?X \in \text{var}(Q')$ for every $Q' \trianglelefteq Q$, and since $Q_1 \trianglelefteq Q$, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \trianglelefteq Q_1$. By induction hypothesis, $?X \in \text{dom}(\mu)$ for all $\mu \in [\![Q_1]\!]_D$ and, hence, by the definition of OPT, we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in [\![(Q_1 \text{ OPT } Q_2)]\!]_D$. Third, suppose that $Q = (Q_1 \text{ FILTER } R)$. Since Q is safe, if $?X \in \text{var}(Q)$ then $?X \in \text{var}(Q_1)$, and, thus, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \trianglelefteq Q_1$. By induction hypothesis, we have that $?X \in \text{dom}(\mu)$ for every $\mu \in [\![Q_1]\!]_D$ and thus, given that $[\![(Q_1 \text{ FILTER } R)]\!]_D \subseteq [\![Q_1]\!]_D$, we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in [\![(Q_1 \text{ FILTER } R)]\!]_D$.

A.3 Proof of Claim 4.7

To prove the claim, we show that the following properties hold for every mapping μ, pattern P, and dataset D. Let $R_\mu = \{P' \trianglelefteq P \mid \text{dom}(\mu) = \text{var}(P')\}$.

(a) If R_μ is not empty, then there exists an element $P_\mu \in R_\mu$ such that $P_\mu \trianglelefteq P'$ for every $P' \in R_\mu$, that is, P_μ is a minimum (with respect to $\trianglelefteq$) element of R_μ.

(b) Deciding whether R_μ is empty can be done in polynomial time in the size of P and μ. Moreover, if R_μ is not empty, pattern P_μ can be constructed in polynomial time in the size of P and μ.

(c) Mapping μ is a partial solution for P over D if and only if $R_\mu \neq \emptyset$ and $\mu \in [\![\text{and}(P_\mu)]\!]_D$.

Using these properties and Theorem 3.1, it is straightforward to see that we can test in polynomial time whether μ is a partial solution for P over D: we first check whether R_μ is not empty, and if this is the case, we construct P_μ and check in polynomial time (Theorem 3.1) whether $\mu \in [\![\text{and}(P_\mu)]\!]_D$.

Throughout the proof, we use the following terminology. Given a pattern P, we say that $\alpha_1, \ldots, \alpha_k$ is an OPT-enumeration for P if $\alpha_1, \ldots, \alpha_k$ is an enumeration of all the subpatterns of P of the form $(P_1 \text{ OPT } P_2)$. For example, if $P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_1 \text{ OPT } t_2)))$, then $\alpha_1 = (t_1 \text{ OPT } t_2)$, $\alpha_2 = (t_2 \text{ OPT } (t_1 \text{ OPT } t_2))$ and $\alpha_3 = (t_1 \text{ OPT } t_2)$ is an OPT-enumeration for P. Fix an OPT-enumeration $\alpha_1, \ldots, \alpha_k$ for a pattern P. We say that $I \subseteq \{1, \ldots, k\}$ is consistent with $\alpha_1, \ldots, \alpha_k$ if for every $i, j \in \{1, \ldots, k\}$, if $i \in I$,

$\alpha_j = (P_1 \text{ OPT } P_2)$ and α_i is a subpattern of P_2, then $j \in I$. Given $I \subseteq \{1, \dots, k\}$ consistent with $\alpha_1, \dots, \alpha_k$, define P_I as a reduction of P obtained by replacing every subpattern $\alpha_j = (P_1 \text{ OPT } P_2)$ by P_1 if $j \in \{1, \dots, k\} \setminus I$. Intuitively, the indexes in I represent the OPT subpatterns of P that were not reduced when obtaining P_I. Following the previous example, the set $\{2, 3\}$ is consistent with the OPT-enumeration $\alpha_1 = (t_1 \text{ OPT } t_2)$, $\alpha_2 = (t_2 \text{ OPT } (t_1 \text{ OPT } t_2))$ and $\alpha_3 = (t_1 \text{ OPT } t_2)$ of $P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_1 \text{ OPT } t_2)))$, and $P_{\{2,3\}} = (t_1 \text{ AND } (t_2 \text{ OPT } (t_1 \text{ OPT } t_2)))$. Similarly, we have $P_{\{1,2\}} = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } t_1))$, and $P_{\{1\}} = ((t_1 \text{ OPT } t_2) \text{ AND } t_2)$. We note that $\{1, 3\}$ is not consistent with the previous OPT-enumeration.

For every $P' \trianglelefteq P$, there exists a unique $I_{P'} \subseteq \{1, \dots, k\}$ such that $P' = P_{I_{P'}}$. Furthermore, given two reductions P_1 and P_2 of P, we have that $P_1 \trianglelefteq P_2$ if and only if $I_{P_1} \subseteq I_{P_2}$, and, if $P_1 \sqcap P_2$ denotes the maximal (with respect to $\trianglelefteq$) element of $\{P' \mid P' \trianglelefteq P_1 \text{ and } P' \trianglelefteq P_2\}$, then $I_{P_1 \sqcap P_2} = I_{P_1} \cap I_{P_2}$. For instance, in the example of the previous paragraph we have that $P_{\{2,3\}} \sqcap P_{\{1,2\}} = P_{\{2\}} = (t_1 \text{ AND } (t_2 \text{ OPT } t_1))$.

We now proceed with the proof. Let D be a dataset, P a well-designed pattern, and μ a mapping, and let R_μ and P_μ be as defined earlier.

(a) For the sake of contradiction, assume that $R_\mu \neq \emptyset$ and R_μ does not have a minimum element with respect to $\trianglelefteq$. Then given that $R_\mu \neq \emptyset$ and R_μ is finite, we know that R_μ has at least two distinct minimal elements $P_1, P_2 \in R_\mu$, that is, for every $P' \in R_\mu$, we have that $P' \ntriangleleft P_1$ and $P' \ntriangleleft P_2$. In particular, we have that $P_1 \ntriangleleft P_2$ and $P_2 \ntriangleleft P_1$. Next we show that $\text{var}(P_1 \sqcap P_2) \subsetneq \text{var}(P_1)$. On the contrary, assume that $\text{var}(P_1 \sqcap P_2) = \text{var}(P_1)$. Then given that $\text{var}(P_1 \sqcap P_2) = \text{var}(P_1) = \text{dom}(\mu)$, we have that $P_1 \sqcap P_2 \in R_\mu$, which contradicts the minimality of P_1 since $P_1 \sqcap P_2 \triangleleft P_1$ (given that $P_1 \sqcap P_2 \trianglelefteq P_2$ and P_1, P_2 are distinct minimal elements of R_μ). In the same way, we conclude that $\text{var}(P_1 \sqcap P_2) \subsetneq \text{var}(P_2)$. Therefore, given that $\text{var}(P_1) = \text{dom}(\mu) = \text{var}(P_2)$, there exists a variable $?X$ such that $?X \in \text{dom}(\mu)$ and $?X \notin \text{var}(P_1 \sqcap P_2)$. Hence, if $\alpha_1, \dots, \alpha_k$ is an OPT-enumeration for P, then there exist $i, j \in \{1, \dots, k\}$ such that (1) $\alpha_i = (O_1 \text{ OPT } O_2)$, $i \in I_{P_1}$, $i \notin I_{P_1 \sqcap P_2}$, $?X \in \text{var}(O_2)$ and $?X \notin \text{var}(O_1)$, and (2) $\alpha_j = (Q_1 \text{ OPT } Q_2)$, $j \in I_{P_2}$, $j \notin I_{P_1 \sqcap P_2}$, $?X \in \text{var}(Q_2)$ and $?X \notin \text{var}(Q_1)$. Given that $i \in I_{P_1}$ and $i \notin I_{P_1 \sqcap P_2}$, we have that $i \notin I_{P_2}$ (since $I_{P_1 \sqcap P_2} = I_{P_1} \cap I_{P_2}$), and given that $j \in I_{P_2}$ and $j \notin I_{P_1 \sqcap P_2}$, we have that $j \notin I_{P_1}$. Thus, we conclude that $i \neq j$. Next we show that the existence of i and j leads to a contradiction.

First note that α_j is not a subpattern of α_i: (1) α_j is not a subpattern of O_1 since $?X \notin \text{var}(O_1)$ and $?X \in \text{var}(Q_2)$, and (2) α_j is not a subpattern of O_2 since I_{P_2} is consistent with the OPT-enumeration of P and $j \in I_{P_2}$ but $i \notin I_{P_2}$. Moreover, we also have that α_i is not a subpattern of α_j: (1) α_i is not a subpattern of Q_1 since $?X \notin \text{var}(Q_1)$ and $?X \in \text{var}(O_2)$, and (2) α_i is not a subpattern of Q_2 since I_{P_1} is consistent with the OPT-enumeration of P and $i \in I_{P_1}$ but $j \notin I_{P_1}$. Therefore, we only need to consider the case where α_i is not a subpattern of α_j and α_j is not a subpattern of α_i. In this case, we conclude that P is not well designed since $\alpha_i = (O_1 \text{ OPT } O_2)$, $?X \in \text{var}(O_2)$,

$?X \notin \text{var}(O_1)$ and $?X \in \text{var}(\alpha_j)$, which contradicts the hypothesis of the claim. This concludes the proof of part (a).

(b) In order to prove this part of the claim, we present an algorithm that has as input a pattern P and a mapping μ, and returns P_μ if R_μ is not empty, and *null* otherwise. To simplify the exposition of the algorithm, we need to introduce some terminology. Let $\alpha_1, \dots, \alpha_k$ be an OPT-enumeration for P, and assume that the pattern $\alpha_i = (P_1 \text{ OPT } P_2)$ is such that P_2 does not contain OPT operators. Then we denote by P^{-i} the reduction of P obtained by replacing $(P_1 \text{ OPT } P_2)$ by P_1. Notice that for such an i, the set $\{1, \dots, k\} \setminus \{i\}$ is consistent with $\alpha_1, \dots, \alpha_k$, and $P^{-i} = P_{\{1,\dots,k\}\setminus\{i\}}$.

Algorithm 2. FindMin (μ: mapping, P: pattern)

```
$P^\star := P$
let $E := \alpha_1, \dots, \alpha_k$ be an OPT-enumeration for $P^\star$
while  there exists $\alpha_i = (P_1 \text{ OPT } P_2)$ in $E$, with $P_2$ not containing OPT operators
       and $\text{dom}(\mu) \subseteq \text{var}((P^\star)^{-i})$   do
    $P^\star := (P^\star)^{-i}$
    update $E$ to be an OPT-enumeration for $P^\star$
if $\text{var}(P^\star) = \text{dom}(\mu)$ return $P^\star$
else return null
```

Algorithm 2 shows the procedure **FindMin** that given a pattern P and a mapping μ, returns P_μ if R_μ is not empty, and *null* otherwise. To prove the correctness of **FindMin**, we show first that, if $R_\mu \neq \emptyset$, the property $P_\mu \trianglelefteq P^\star$ is an *invariant* of the **while** loop. It is straightforward to prove that before the first iteration the property holds. Now, assume that $P_\mu \trianglelefteq P^\star$ and that $\alpha_1, \dots, \alpha_k$ is an OPT-enumeration for $P^\star$. Let $\alpha_i = (P_1 \text{ OPT } P_2)$ be an OPT subpattern of $P^\star$ such that P_2 does not contain OPT operators and $\text{dom}(\mu) \subseteq \text{var}((P^\star)^{-i})$. We have to show that $P_\mu \trianglelefteq (P^\star)^{-i}$. First note that, since $P_\mu \trianglelefteq P^\star$ and $\text{dom}(\mu) \subseteq \text{var}((P^\star)^{-i})$ we obtain that $P_\mu \neq P^\star$ (otherwise P_μ is not minimum in R_μ) and, thus, $P_\mu \triangleleft P^\star$. Moreover, from $P_\mu \triangleleft P^\star$ we know that there exists a set $I_\mu \subsetneq \{1, \dots, k\}$ consistent with $\alpha_1, \dots, \alpha_k$ such that $P_\mu = (P^\star)_{I_\mu}$. Now we show by contradiction that $P_\mu \trianglelefteq (P^\star)^{-i}$. Assume then that $P_\mu \ntrianglelefteq (P^\star)^{-i}$. Thus, since $P_\mu \triangleleft P^\star$ and P_2 does not contain OPT operators, we have that P_2 is necessarily a subpattern of P_μ. This last fact implies that $i \in I_\mu$. Consider now the set $I_\mu \setminus \{i\}$ and the pattern $(P^\star)_{I_\mu\setminus\{i\}}$. For the sake of simplicity, and slightly abusing notation, call $(P_\mu)^{-i}$ to the pattern $(P^\star)_{I_\mu\setminus\{i\}}$. It is clear that $(P_\mu)^{-i} \triangleleft P_\mu$, and then by the minimality of P_μ in R_μ, we have that $\text{var}((P_\mu)^{-i}) \subsetneq \text{dom}(\mu)$. Thus we have that there exists a variable, say $?X$, such that $?X \in \text{dom}(\mu) = \text{var}(P_\mu)$, $?X \in \text{var}(P_2)$, but $?X \notin \text{var}((P_\mu)^{-i})$. Now, since we are assuming that $\text{dom}(\mu) \subseteq \text{var}((P^\star)^{-i})$, we have that $?X \in \text{var}((P^\star)^{-i})$. Moreover, since $?X \in \text{var}((P^\star)^{-i})$, $?X \notin \text{var}((P_\mu)^{-i})$, and $P_\mu = (P^\star)_{I_\mu}$, there necessarily exists $j \in \{1, \dots, k\}$ such that $j \notin I_\mu$ and $\alpha_j = (O_1 \text{ OPT } O_2)$, $?X \in \text{var}(O_2)$ but $?X \notin \text{var}(O_1)$. Next we show that the existence of j leads to our desired contradiction.

First note that α_i is not a subpattern of α_j: (1) α_i is not a subpattern of O_1 since $?X \notin \text{var}(O_1)$ and $?X \in \text{var}(P_2)$, and (2) α_i is not a subpattern of O_2 since I_μ is consistent with the OPT-enumeration of $P^\star$ and $j \notin I_\mu$. Moreover, since P_2 does not contain OPT operators, we also have that α_j is not a subpattern of P_2. Therefore, we only need to consider two cases.

—α_j is a subpattern of P_1. In this case. we conclude that $P^\star$ is not well designed since $?X \in \text{var}(O_2)$, $?X \notin \text{var}(O_1)$ and $?X \in \text{var}(P_2)$.

—α_j is not a subpattern α_i, and α_i is not a subpattern of α_j. In this case, we conclude that $P^\star$ is not well designed since $?X \in \text{var}(O_2)$, $?X \notin \text{var}(O_1)$ and $?X \in \text{var}(\alpha_i)$.

Since $P^\star \trianglelefteq P$, in both cases we obtain a contradiction with the fact that P is well designed.

We have shown that $P_\mu \trianglelefteq P^\star$ is an invariant provided that $R_\mu \neq \emptyset$. We now use this invariant to prove the correctness of **FindMin**. We show first that, when the loop halts, if $R_\mu \neq \emptyset$ then $P^\star = P_\mu$. On the contrary, assume that $P^\star \neq P_\mu$. Given that $R_\mu \neq \emptyset$, we know that the invariant $P_\mu \trianglelefteq P^\star$ holds. Thus, given that $P^\star \neq P_\mu$, we have that $P_\mu \vartriangleleft P^\star$ which contradicts the termination of the loop. We show now that when the loop halts, if $R_\mu = \emptyset$ then $\text{dom}(\mu) \neq \text{var}(P^\star)$. On the contrary, assume that $\text{dom}(\mu) = \text{var}(P^\star)$, then $P^\star \in R_\mu$ which contradicts the fact that $R_\mu = \emptyset$. We have shown that, when the loop halts, if $R_\mu \neq \emptyset$ then $P^\star = P_\mu$, and if $R_\mu = \emptyset$ then $\text{dom}(\mu) \neq \text{var}(P^\star)$. Then **FindMin** returns P_μ if $R_\mu \neq \emptyset$, and returns *null* otherwise.

(c) ($\Leftarrow$) If $R_\mu \neq \emptyset$ and $\mu \in [\![\text{and}(P_\mu)]\!]_D$, then μ is a partial solution for P since $P_\mu \trianglelefteq P$.

($\Rightarrow$) Assume that μ is a partial solution for P. Then there exists $P' \trianglelefteq P$ such that $\mu \in [\![\text{and}(P')]\!]_D$. Thus, since P and P' are safe we have that $\text{var}(P') = \text{dom}(\mu)$ and, hence, R_μ is not empty. Therefore, from (a) we have that R_μ has a minimum (with respect to $\trianglelefteq$) element P_μ. In particular, we have that $P_\mu \trianglelefteq P'$, which implies that $[\![\text{and}(P')]\!]_D \subseteq [\![\text{and}(P_\mu)]\!]_D$, since $\text{var}(P') = \text{dom}(\mu) = \text{var}(P_\mu)$, the only operators in $\text{and}(P')$ and $\text{and}(P_\mu)$ are AND and FILTER, and every filter condition in $\text{and}(P_\mu)$ is also present in $\text{and}(P')$. We deduce that $\mu \in [\![\text{and}(P_\mu)]\!]_D$.

ACKNOWLEDGMENTS

We are grateful to the anonymous referees for their careful reading of the article, and for providing many useful comments. We would also like to thank M. Schmidt for many helpful comments on earlier versions of this article.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

ABITEBOUL, S., KANELLAKIS, P., AND GRAHNE, G. 1991. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci. 78,* 1, 158–187.

ANGLES, R. AND GUTIERREZ, C. 2008. The expressive power of sparql. In *Proceedings of the International Semantic Web Conference (ISWC)*, 114–129.

ARENAS, M., GUTIERREZ, C., PARSIA, B., PÉREZ, J., POLLERES, A., AND SEABORNE, A. 2007. SPARQL—Where are we? Current state, theory and practice. Full day tutorial, *European Semantic Web Conference*.

ARQ, 2006. A SPARQL processor for Jena, version 1.3 March 2006, Hewlett-Packard Development Company. http://jena.sourceforge.net/ARQ.

BEERI, C., FAGIN, R., MAIER, D., AND YANNAKAKIS, M. 1983. On the desirability of acyclic database schemes. *J. ACM 30,* 3, 479–513.

BHARGAVA, G., GOEL, P., AND IYER, B. 1995. Hypergraph based reorderings of outer join queries with complex predicates. In *Proceedings of the SIGMOD International Conference of Management of Data*, 304–315.

CHANDRA, A. AND MERLIN, P. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, 77–90.

CHEN, L., GUPTA, A., AND KURUL, E. 2005. A semantic-aware RDF query algebra. In *Proceedings of the International Conference on Management of Data (COMAD)*.

CYGANIAK, R. 2005. A relational algebra for sparql. Tech. rep. HPL-2005-170, HP-Labs. http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html.

DE BRUIJN, J., FRANCONI, E., AND TESSARIS, S. 2005. Logical reconstruction of normative RDF. In *Proceedings of the OWL—Experiences and Directions Workshop (OWLED)*.

DURST, M. AND SUIGNARD, M. 2005. Rfc 3987, internationalized resource identifiers (IRIS). http://www.ietf.org/rfc/rfc3987.txt.

FRASINCAR, F., HOUBEN, G.-J., VDOVJAK, R., AND BARNA, P. 2004. RAL: An algebra for querying RDF. *World Wide Web 7,* 1, 83–109.

FURCHE, T., LINSE, B., BRY, F., PLEXOUSAKIS, D., AND GOTTLOB, G. 2006. RDF querying: Language constructs and evaluation methods compared. In *Reasoning Web*, 1–52.

GALINDO-LEGARIA, C. A. AND ROSENTHAL, A. 1997. Outerjoin simplification and reordering for query optimization. *ACM Trans. Datab. Syst. 22,* 1, 43–73.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

GOTTLOB, G., LEONE, N., AND SCARCELLO, F. 2001. The complexity of acyclic conjunctive queries. *J. ACM 48,* 3, 431–498.

GUTIERREZ, C., HURTADO, C. A., AND MENDELZON, A. O. 2004. Foundations of semantic Web databases. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 95–106.

HAASE, P., BROEKSTRA, J., EBERHART, A., AND VOLZ, R. 2004. A comparison of RDF query languages. In *Proceedings of the International Semantic Web Conference (ISWC)*, 502–517.

HARRIS, S. AND SHADBOLT, N. 2005. SPARQL query processing with conventional relational database systems. In *Proceedings of the WISE Workshops*, 235–244.

IMIELINSKI, T. AND LIPSKI, W. 1984. Incomplete information in relational databases. *J. ACM 31,* 4, 761–791.

KARVOUNARAKIS, G., ALEXAKI, S., CHRISTOPHIDES, V., PLEXOUSAKIS, D., AND SCHOLL, M. 2002. RQL: A declarative query language for RDF. In *Proceedings of the International Conference on World Wide Web (WWW)*, 592–603.

KLYNE, G., CARROLL, J. J., AND MCBRIDE, B. 2004. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation. http://www.w3.org/TR/rdf-concepts/.

MANOLA, F. AND MILLER, E. 2004. RDF primer, W3C recommendation. http://www.w3.org/TR/rdf-concepts/.

MARIN, D. 2004. A formalization of RDF (applications de la logique á la sémantique du Web). Tech. rep., École Polytechnique, Universidad de Chile. Department of Computer Science, Universidad de Chile, TR/DCC-2006-8.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2006a. Semantics and complexity of SPARQL. In *Proceedings of the International Semantic Web Conference (ISWC)*. 30–43.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2006b. Semantics of SPARQL. Tech. rep., Universidad de Chile. Department of Computer Science, Universidad de Chile, TR/DCC-2006-17.

POLLERES, A. 2007. From SPARQL to rules (and back). In *Proceedings of the International Conference on World Wide Web (WWW)*, 787–796.

PRUD'HOMMEAUX, E. AND SEABORNE, A. 2008. SPARQL query language for RDF. W3C recommendation. `http://www.w3.org/TR/rdf-sparql-query/`.

ROBERTSON, E. L. 2004. Triadic relations: An algebra for the semantic Web. In *Proceedings of the International Workshop on Semantic Web and Databases (SWDB)*, 91–108.

SCHMIDT, M., MEIER, M., AND LAUSEN, G. 2008. Foundations of SPARQL query optimization. `http://arxiv.org/abs/0812.3788`.

SEABORNE, A. 2006. Personal communication.

SERFIOTIS, G., KOFFINA, I., CHRISTOPHIDES, V., AND TANNEN, V. 2005. Containment and minimization of RDF/S query patterns. In *Proceedings of the International Semantic Web Conference (ISWC)*, 607–623.

VARDI, M. Y. 1982. The complexity of relational query languages (extended abstract). In *Proceedings of the Symposium on the Theory of Computing (STOC)*, 137–146.

YANNAKAKIS, M. 1981. Algorithms for acyclic database schemes. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 82–94.

ZANIOLO, C. 1984. Database relations with null values. *J. Comput. Syst. Sci. 28,* 1, 142–166.

Received April 2008; revised December 2008; accepted June 2009